

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních
technologií

HABILITAČNÍ PRÁCE



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY

A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

OPTIMALIZACE STRUKTUR ALGORITMŮ
Z HLEDISKA ARCHITEKTURY SIGNÁLOVÝCH
PROCESORŮ

OPTIMIZATION OF ALGORITHM STRUCTURES IN TERMS OF DIGITAL SIGNAL
PROCESSOR ARCHITECTURE

HABILITAČNÍ PRÁCE

HABILITATION THESIS

AUTOR PRÁCE

AUTHOR

Ing. Petr Sysel, Ph.D.

BRNO 2017

OBSAH

Seznam zkratk	8
Seznam symbolů	11
Úvod	14
1 Současný stav řešené problematiky	16
1.1 Vnější popis číslicového systému	16
1.2 Vnitřní popis číslicového systému	18
1.2.1 Druhá přímá forma	18
1.2.2 Transponovaná druhá přímá forma	20
1.2.3 Vazební struktura	21
1.2.4 Křížová struktura	22
1.3 Přehled používaných formátů čísel	25
1.4 Kvantování v případě pevné řádové čárky	29
1.5 Porovnání vlastností signálových procesorů	31
2 Cíle habilitační práce	34
3 Přehled architektur procesorů	35
3.1 Von Neumannova architektura	35
3.2 Harvardská architektura	36
3.2.1 Vybrané procesory s harvardskou architekturou	37
3.2.2 Jednotka generace adres	40
3.2.3 Aritmeticko logická jednotka	48
3.2.4 Paralelní přenosy a další operace	52
3.2.5 Instrukční soubor a strojový kód	55
3.3 Architektura s velmi dlouhým instrukčním slovem	56
3.3.1 Vybrané procesory s velmi dlouhým instrukčním slovem	58
3.3.2 Jednotka generace adres	61
3.3.3 Datová aritmetická-logická jednotka	66
3.3.4 Instrukční soubor a strojový kód	69
3.4 Superskalární architektura	74
3.5 Architektura paralelních systémů	75
3.6 Shrnutí vlastností architektur	78
4 Implementace číslicových systémů v signálových procesorech	80
4.1 Číslicové systémy s konečnou impulsní charakteristikou	80
4.1.1 Signálové procesory s harvardskou architekturou	81
4.1.2 Využití symetrie impulsní charakteristiky	85
4.1.3 Signálové procesory s velmi dlouhým instrukčním slovem	89
4.2 Číslicové systémy s nekonečnou impulsní charakteristikou	97
4.2.1 Signálové procesory s harvardskou architekturou	98

4.2.2	Signálové procesory s velmi dlouhým instrukčním slovem	102
4.3	Shrnutí postupů optimalizace	114
5	Vliv omezené délky slova na číslicové zpracování	116
5.1	Kvantování koeficientů číslicového systému	116
5.1.1	Kvantování koeficientů systémů typu FIR	116
5.1.2	Kvantování koeficientů systémů typu IIR	116
5.2	Rozdělení na sekce 2. řádu	122
5.2.1	Rozdělení do sériového spojení sekcí 2. řádu	122
5.2.2	Rozdělení do paralelního zapojení sekcí 2. řádu	126
5.3	Shrnutí vlivu omezené délky slova	128
6	Závěr	130
	Literatura	133

SEZNAM OBRÁZKŮ

1.1	Základní typy hran grafů signálových toků.	17
1.2	Nekanonická vnější struktura realizující algoritmus číslicového systému 3. řádu typu IIR.	17
1.3	Kanonická vnitřní struktura systému IIR.	19
1.4	Graf signálových toků 1. kanonické stavové struktury číslicového systému typu IIR 3. řádu.	21
1.5	Graf signálových toků vazební struktury čistě rekurzivního číslicového systému 2. řádu.	22
1.6	Křížová struktura pro implementaci systému s konečnou impulsní charakteristikou.	23
1.7	Graf signálových toků křížové struktury pro realizaci čistě rekurzivního systému.	25
1.8	Ilustrační zobrazení rozsahu různých formátů vyjádření čísel a vzniku zkreslení.	26
1.9	Ukázka usekávání (modře) a zaokrouhlování (zeleně) při kvantování signálu.	30
3.1	Von Neumannova koncepce architektury číslicových počítačů.	35
3.2	Harvardská architektura signálových procesorů.	36
3.3	Blokové schéma signálového procesoru DSC56F8367.	38
3.4	Blokové schéma signálového procesoru TMS320C5510.	39
3.5	Blokové schéma signálového procesoru ADSP2191.	40
3.6	Zobrazení lineární vyrovnávací paměti realizované pomocí běžného pole jazyka C.	41
3.7	Zobrazení principu kruhového zásobníku pro realizaci zpoždění signálu.	41
3.8	Blokové schéma jednotky generace adres procesoru DSC56F8367.	44
3.9	Blokové schéma jednotky generace adres u signálového procesoru ADSP2191.	47
3.10	Blokové schéma aritmetické logické jednotky procesoru DSC56F8367.	48
3.11	Datová aritmeticko logická jednotka procesoru TMS320C5510.	50
3.12	Blokové schéma DALU procesoru ADSP2191.	52
3.13	Příklad kódování instrukcí do strojového kódu procesoru DSC56F8367.	55
3.14	Příklad formátu instrukčního slova architektury LIW	57
3.15	Příklad mikroprocesoru s architekturou typu VLIW.	58
3.16	Blokové schéma signálových procesorů řady TMS320C6200 a TMS320C6700.	59
3.17	Blokové schéma jádra SC140 (StarCore).	60
3.18	Struktura modifikačního registru AMR.	62
3.19	Blokové schéma jednotky generace adres jádra StarCore.	63
3.20	Struktura modifikačního registru MCTL jednotky AGU jádra SC140.	65
3.21	Příklad formátů registrů jádra VelociTI.	67
3.22	Blokové schéma aritmetické-logické jednotky DALU jádra SC140.	67
3.23	Reprezentace operandů v doplňkovém kódu.	68
3.24	Formát a členění datových registrů D0 až D15 jednotky DALU.	69
3.25	Příklad kódování instrukce SMPY do strojového kódu.	70
3.26	Sdružování instrukcí do instrukčních paketů.	71
3.27	Sériové a předponové (prefixové) sdružování instrukcí do instrukčních paketů.	73

3.28	Princip spojení vstupních datových operandů pro paralelní zpracování dat metodou typu SIMD.	77
3.29	Příklad formátů datových paketů.	77
4.1	Graf signálových toků systému s konečnou impulsní charakteristikou. . . .	80
4.2	Symbolické znázornění dílčích operací při výpočtu lineární konvoluce podle obr. 4.1 a jejich vzájemný vztah.	81
4.3	Ilustrační rozložení paměti při implementaci systému typu FIR v signálovém procesoru s harvardskou architekturou.	82
4.4	Graf signálových toků realizující algoritmus číslicového systému se symetrickou nebo antisymetrickou konečnou impulsní charakteristikou.	86
4.5	Symbolické znázornění implementace lineární konvoluce s využitím symetrie.	87
4.6	Symbolické znázornění implementace lineární konvoluce v signálovém procesoru TMS320C55xx s využitím dvou MAC.	89
4.7	Symbolické znázornění způsobu výpočtu lineární konvoluce na procesoru s jádrem StarCore.	90
4.8	Symbolické skládání několika iterací výpočtu konvoluce paralelně.	92
4.9	Symbolické znázornění implementace lineární konvoluce v signálovém procesoru TMS320C64xx.	92
4.10	Symbolické znázornění dvojité implementace lineární konvoluce v signálovém procesoru TMS320C64xx.	93
4.11	Symbolické skládání několika iterací výpočtu dvojité konvoluce paralelně.	96
4.12	Symbolické znázornění závislostí dílčích operací při výpočtu první kanonické formy podle (4.12).	98
4.13	Ilustrační rozložení paměti při implementaci 1. kanonické formy v signálovém procesoru s harvardskou architekturou.	99
4.14	Ilustrační zobrazení uložení koeficientů pro implementaci 1. kanonické formy s využitím principu SIMD.	106
5.1	Zobrazení možných poloh kořenů polynomu 2. řádu pro první a druhou kanonickou strukturu.	119
5.2	Zobrazení možných poloh kořenů polynomu 2. řádu u vazební struktury.	120
5.3	Porovnání možného umístění kořenů polynomu 2. řádu pro různé struktury.	121
5.4	Sériové zapojení sekcí druhého řádu pro realizaci celkové přenosové funkce.	122
5.5	Modulové kmitočtové charakteristiky dílčích sekcí a celkové modulové kmitočtové charakteristiky při sériovém spojení sekcí.	126
5.6	Paralelní zapojení sekcí druhého řádu pro realizaci celkové přenosové funkce.	127
5.7	Modulové kmitočtové charakteristiky dílčích sekcí 2. řádu při paralelních zapojení.	128

SEZNAM TABULEK

1.1	Porovnání hlavních vlastností nejčastěji používaných formátů vyjádření čísel.	27
1.2	Vyjádření 4bitových zlomkových čísel v přímém kódu, jednotkovém doplňku a dvojkovém doplňku.	28
1.3	Porovnání vlastností některých procesorů s pevnou řádovou čárkou a harvardskou architekturou.	32
1.4	Porovnání vlastností některých procesorů s pevnou a pohyblivou řádovou čárkou s architekturou s velmi dlouhým instrukčním slovem.	33
3.1	Možnosti nepřímého adresování datové paměti u procesoru DSC56F8367. .	45
3.2	Možnosti nepřímého adresování datové paměti u procesoru TMS320C5510.	46
3.3	Možnosti nepřímého adresování paměti u procesoru ADSP2191.	47
3.4	Příklady formátů a rozsahů hodnot registrů DALU procesoru DSC56F8367.	49
3.5	Povolené kombinace dvou paralelních přesunů u DSC56F8367.	54
3.6	Způsoby nepřímého adresování u architektury VelociTI.	61
3.7	Možnosti nepřímého adresování paměti u procesoru StarCore.	64
3.8	Nastavení adresového módu spodní banky registrů R0 až R7.	65
4.1	Porovnání výpočetní náročnosti procesu kódování a dekódování v signálovém procesoru TMS320C6416 pro vybrané řečové kodeky před a po optimalizaci.	115
5.1	Možné variace uspořádání nulových bodů a pólů systému 6. řádu do sekcí 2. řádu.	124

SEZNAM VÝPISŮ

4.1	Příklad implementace systému typu FIR v signálovém procesoru s hardvardskou architekturou.	83
4.2	Příklad využití symetrie při implementaci systému typu FIR v signálovém procesoru s hardvardskou architekturou.	87
4.3	Příklad využití dvou jednotek MAC při implementaci systému typu FIR v signálovém procesoru s hardvardskou architekturou.	89
4.4	Příklad sériové implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.	91
4.5	Příklad paralelní implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.	94
4.6	Příklad dvojité sériové implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.	95
4.7	Příklad jádra dvojité paralelní implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.	97
4.8	Příklad implementace 1. kanonické formy v signálovém procesoru s hardvardskou architekturou.	99
4.9	Příklad sekvenční implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW.	102
4.10	Příklad jádra paralelní implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW.	105
4.11	Příklad sekvenční implementace 1. kanonické formy s využitím principu SIMD.	107
4.12	Příklad jádra paralelní implementace 1. kanonické formy s využitím principu SIMD.	108
4.13	Příklad sériové implementace 4. kanonické formy v procesoru s architekturou typu VLIW.	111
4.14	Příklad paralelní implementace 4. kanonické formy v procesoru s architekturou typu VLIW.	112

SEZNAM ZKRATEK

AAU	Address Arithmetic Unit aritmická jednotka jednotky generace adres
AGU	Address Generator Unit jednotka pro generování adres
ALU	Arithmetic-Logic Unit aritmická-logická jednotka
AMR	Addressing Mode Register registr pro nastavení způsobu adresování
ARM	Advanced RISC Machine architektura mikroprocesorů se zaměřením na nízkou spotřebu
BFU	Bit Field Unit jednotka pro práci s bitovými poli
BMU	Bit Manipulating Unit jednotka pro práci s bitovými poli
CARRY	příznakový bit, že u předchozí operace byl využit přenos z nejvýznamnějšího bitu
CCOP	Cyclic-code Co-Processor pomocný procesor pro podporu implementace cyklických kódů
CPU	Central Processor Unit centrální řídicí jednotka
DALU	Data Arithmetic Logic Unit datová aritmická-logická jednotka
DARAM	Dual Access RAM paměť RAM s možností dvou současných přístupů
DF1	Direct Form I první přímá forma
DF2	Direct Form II druhá přímá forma
DF2T	Direct Form II Transposed transponovaná druhá přímá forma
DMA	Direct Memory Access přímý přístup do paměti
DSC	Digital Signal Controllers signálový mikrokontrolér
DSP	Digital Signal Processing číslicové zpracování signálů

EDVAC	Electronic Discrete Variable Computer číslicový elektronkový počítač s proměnným programem
EMR	Exception and Mode Register registr nastavení výjimek a módu
ENIAC	Electronic Numerical Integrator and Calculator elektronkový číslicový integrátor a počítač
ESP	Exception Stack Pointer ukazatel zásobníku při obsluze přerušení/výjimky
FCOP	Filter Co-Processor pomocný procesor pro realizaci číslicové filtrace
FFT	Fast Fourier Transform rychlá Fourierova transformace
FIR	Finite Impulse Response konečná impulsní charakteristika
GSM	Global System for Mobile communications globální systém pro mobilní komunikaci
IIR	Infinite Impulse Response nekonečná impulsní charakteristika
ITU	International Telecommunication Union mezinárodní telekomunikační unie
LIW	Long Instruction Word dlouhé instrukční slovo
LSB	Least Significant Bit nejméně významný bit
MAC	Multiply and Accumulate násobení s akumulací
MCTL	Modifier Control Register modifikační registr
MMAC	Million Multiplies and Accumulate milión operací násobení s akumulací
MMX	Multi Media Extension rozšíření instrukčního souboru zaměřené na zpracování multimédií
MSB	Most Significant Bit nejvíce významný bit
NSP	Normal Stack Pointer ukazatel zásobníku v normálním režimu

OMR	Operating Mode Register registr operačního módu
PC	Program Counter programový čítač – registr obsahující adresu právě zpracovávané instrukce
PSEQ	Program Sequencer Unit jednotka sekvenčního zpracování programu
PWM	Pulse Width Modulation pulsně šířková modulace
SARAM	Single Access RAM paměť pro čtení i zápis s možností jediného přístupu v daný okamžik
SIMD	Single Instruction Multiple Data jeden programový tok a více datových toků
SP	Stack Pointer ukazatel zásobníku – adresa vrcholu zásobníku
SQNR	Signal to Quantization Noise Ratio poměr signálu ku kvantizačnímu šumu
SR	Status Register stavový registr
SSE	Streaming SIMD Extensions rozšíření instrukčního souboru zaměřené na využití paralelního zpracování typu SIMD
VBA	Vector Base Address adresa tabulky vektorů obsluhy přerušení
VCOP	Viterbi Co-Processor pomocný procesor pro realizaci algoritmu Viterbiho dekodování
VLES	Variable Length Execution Set instrukční soubor s proměnnou délkou instrukčního slova
VLIW	Very Long Instruction Word velmi dlouhé instrukční slovo
VLSI	Very Large-Scale Integration integrované obvody s velmi vysokou mírou integrace

SEZNAM SYMBOLŮ

$(\cdot)_{10}$	číslo v desítkové soustavě
$(\cdot)_{16}$	číslo v šestnáctkové soustavě
$(\cdot)_2$	číslo ve dvojkové soustavě
$(\cdot)_8$	číslo v osmičkové soustavě
$\%$	operace zbytek po dělení (modulo)
A	nekvantovaná hodnota čísla
$\lfloor A \rfloor$	největší celé číslo menší nebo maximálně rovné A
$\mathbf{A, B, C, D}$	přenosové matice systému
A_{\max}^+	největší kladné vyjádřitelné číslo v daném formátu
A_{\min}^+	nejmenší kladné vyjádřitelné číslo v daném formátu
A_Q	kvantovaná hodnota čísla
A_R	zaokrouhlená hodnota
A_T	hodnota zkrácená vypuštěním nejméně významných bitů (useknutá)
β_k	váhový koeficient pro zabránění přetečení výstupního signálu dílčí sekce
c_0, c_1, c_2	koeficienty čitatele přenosové funkce v normalizovaném tvaru
d_1, d_2	koeficienty jmenovatele přenosové funkce v normalizovaném tvaru
$\text{diag}()$	hlavní diagonála matice
$\Delta(z)$	determinant grafu signálových toků
$\Delta_k(z)$	subdeterminant grafu signálových toků pro k . přímou cestu
e_Q	kvantovací šum $e_Q = A_Q - A$
e_T	kvantovací šum $e_T = A_T - A$ v případě usekávání
e_R	kvantovací šum $e_R = A_R - A$ v případě zaokrouhlení
\mathcal{F}	Fourierova transformace
f	kmitočet
f_c	kmitočet hodinových impulsů
f_{\max}	maximální kmitočet obsažený v signálu
f_r	relativní kmitočet
f_{vz}	vzorkovací kmitočet
$H(z)$	přenosová funkce v \mathcal{Z} transformaci
$H_k(z)$	přenosová funkce druhého řádu k . dílčí sekce v \mathcal{Z} transformaci
$h[n]$	impulsní charakteristika diskrétního systému
$H(p)$	přenosová funkce v Laplaceově transformaci

$h(t)$	impulsní charakteristika spojitého systému
\forall	všechny prvky množiny
$\inf_v A(v)$	minimální prvek množiny $A(v)$ přes $\forall v$
\mathcal{L}	Laplaceova transformace
\log_{10}	logaritmus se základem 10
μ	střední hodnota
\mathbb{N}	přirozená čísla
N_b	celkový počet bitů číselného formátu $N_b = N_b^+ + N_b^-$
N_b^+	počet bitů číselného formátu vlevo od řádové čárky
N_b^-	počet bitů číselného formátu vpravo od řádové čárky
p	komplexní parametr Laplaceovy transformace
$P_k(z)$	přenos k . přímé cesty grafu signálových toků
q	kvantovací krok $q = 2^{-N_b^-}$
$Qm.n$	formát čísel v pevné řádové čárce; m udává počet bitů N_b^+ vlevo od řádové čárky, n udává počet bitů N_b^- vpravo o řádové čárky
R_{dB}	dynamický rozsah
\mathbb{R}	reálná čísla
σ	rozptyl
sinc	funkce $\text{sinc}(x) = \frac{\sin(x)}{x}$
$S[k]$	k . složka diskrétní Fourierovy transformace
$s(t)$	spojité (analogové) signály
$\text{Im}\{\cdot\}$	imaginární část komplexního parametru
$\text{Re}\{\cdot\}$	reálná část komplexního parametru
$\sup_v A(v)$	maximální prvek množiny $A(v)$ přes $\forall v$
$S_k(z)$	přenos k . smyčky grafu signálových toků
T	vzorkovací perioda
$v_1[n], v_2[n]$	stavové proměnné číslicového systému
ω	úhlový kmitočet
ω_r	relativní úhlový kmitočet

$x[n], y[n]$ posloupnost diskretních hodnot (číslicový signál)
 $X(z), Y(z)$ obraz číslicového signálu v \mathcal{Z} transformaci

\mathbb{Z} celá čísla

\mathcal{Z} \mathcal{Z} transformace

z komplexní parametr \mathcal{Z} transformace

ÚVOD

Základní architektonické koncepce mikroprocesorových obvodů byly postupně přebírány ze struktur číslicových počítačů. Potřeba různorodosti zpracování (současné zpracování zvukových a obrazových signálů, řešení paralelních úloh apod.) vyvolala nejprve modifikace a pak i nové myšlenky v koncepci jednotlivých složek architektury mikroprocesorů [12, 27, 29, 30]. Základy architektury číslicových počítačů a pak i mikroprocesorů formulovala ve čtyřicátých letech minulého století skupina vědců, kterou vedl matematik John von Neumann [45]. V dnešním technickém světě, který je propojen celosvětovou počítačovou sítí – internetem, jsou stanoveny požadavky velmi rychlého společného zpracování zvukového a obrazového (multimediálního) signálu, vysokorychlostního přenosu dat apod. Za této situace prvotní koncepce počítačového nebo mikroprocesorového systému již zastarala a je zde vidět snaha vývojářů změnit architekturu mikroprocesorů s cílem podstatně zvýšit jejich výpočetní výkon.

Návrh nového mikroprocesorového integrovaného obvodu nebo mikroprocesorového systému závisí na tradici, úrovni technologie, fantazii návrhářů a v neposlední řadě i na inspiraci v okolním světě např. ve vlastnostech živých organismů. Soustředíme-li se na mikroprocesory pro realizaci algoritmů číslicového zpracování signálů, tak od jejich vzniku až dosud lze architektury těchto mikroprocesorových obvodů rozdělit takto:

- von Neumannova architektura (viz část 3.1),
- harvardská architektura (viz část 3.2),
- architektury s velmi dlouhým instrukčním slovem (označovaná jako VLIW (Very Long Instruction Word), resp. VLES (Variable Length Execution Set), viz část 3.3),
- superskalární architektura (viz část 3.4),
- architektura paralelních systémů (viz část 3.5).

Pro další zvýšení výkonu signálového procesoru v určitých aplikacích jsou někdy k jádru signálového procesoru přidávány další jednotky jako jsou koprocesory, vstupně výstupní jednotky nebo jiné speciální obvody. Tyto jednotky jsou přizpůsobeny konkrétní aplikaci, pro kterou je signálový procesor určen. Tak se vlastně signálový procesor s koprocesory stal víceprocesorovým systémem, neboť koprocesory pracují zpravidla nezávisle na základním jádře signálového procesoru, se kterým pouze sdílí část paměti. Příkladem může být signálový procesor s pevnou řádovou čárkou Motorola DSP56305 [34], který má tři koprocesory pro podporu systému GSM (Global System for Mobile communications). Filtrační koprocesor FCOP (Filter Co-Processor) může realizovat komplexní číslicový filtr typu FIR (Finite Impulse Response) až 21. řádu s přesností na 16 bitů. V systému GSM se může tento koprocesor využít pro realizaci vzájemné korelace přijaté posloupnosti dat s předem definovanou trénovací posloupností. Koprocesor VCOP (Viterbi Co-Processor) lze v systému GSM použít pro konvoluční kódování a dekodování a kompenzaci nevhodných vlastností přenosového kanálu (ekvalizaci). Poslední z koprocesorů CCOP (Cyclic-code Co-Processor) podporuje kódování pomocí cyklických kódů pro zabezpečení, opravy chyb a šifrování přenášených dat v systému GSM. Podobně lze doplnit koprocesory pro spektrální analýzu s algoritmem FFT (Fast Fourier Transform), řízení elektrických strojů pomocí PWM (Pulse Width Modulation) apod.

Signálové procesory s harvardskou architekturou byly původně navrženy pro aplikace, u nichž záleželo na malém příkonu, nízké ceně a malém rozměru čipu. Tato architektura signálových procesorů prováděla čtení jedné instrukce během jednoho hodinového cyklu (Single-Issue Processing) a byla tím jednodušší pro implementaci než architektury, které čtou více instrukcí během jednoho hodinového cyklu. K tomu takováto architektura signálového procesoru zabrala méně plochy na čipu a spotřebovala méně příkonu. Velikost programové paměti mikroprocesoru, potřebná k uložení programu, ovlivňuje jeho cenu, plochu čipu a příkon, a proto instrukční sada signálových procesorů byla navržena tak, aby umožnila vytváření vysoce úsporných programů. To vede k jednoduššímu a tím i levnějšímu vývoji nového zařízení. S drobnými změnami to platí i pro nejmodernější architekturu typu VLIW. Z těchto důvodů se signálové procesory využívají v mnoha průmyslových aplikacích. Vysoký výpočetní výkon signálových procesorů použitých v telefonních ústřednách umožňuje zpracování, kodování nebo dekodování velkého množství kanálů [93, 98, 99]. Signálové procesory se uplatní i v aplikacích, které sice nevyžadují vysoký výpočetní výkon, ale využijí jednoduchost návrhu a konstrukce. Typicky se jedná o vestavená (embedded) zařízení [74, 70]. Na druhou stranu kompaktní strojový kód s velkým množstvím výjimek způsobuje potíže při překladu programu z vyšších programovacích jazyků a komplikuje optimalizaci zdrojových kódů.

Proto je předložená habilitační práce zaměřena na optimalizaci algoritmů číslicového zpracování signálů z hlediska architektury signálových procesorů. Uvažovány jsou pouze lineární časově invariantní systémy. První kapitola se věnuje stručnému popisu nejčastěji používaných struktur realizace číslicových systémů. Dále jsou shrnuty vlastnosti formátů čísel používaných v technických prostředcích číslicového zpracování. Důraz je kladen na formát s pevnou řádovou čárkou ve dvojkovém doplňku, který je v současnosti nejvíce využíván pro své výhodné vlastnosti z hlediska přetečení rozsahu použitého formátu vyjádření čísel. V závěru kapitoly porovnány vlastnosti konkrétních vybraných signálových procesorů. Kapitola má především sjednotit použitou terminologii.

Kapitola 3 obsahuje podrobný popis jednotlivých architektur signálových procesorů. Důraz je kladen především na klasickou harvardskou architekturu a moderní architektury s velmi dlouhým instrukčním slovem. Snahou je nalézt společné vlastnosti architektur užívanými různými výrobci signálových procesorů, které by mohly být použity pro optimalizaci struktur výpočetních algoritmů.

Stěžejní část habilitační práce je kapitola 4, která obsahuje podrobný návrh optimalizace struktur algoritmů číslicového zpracování signálů z hlediska architektury signálových procesorů. Kapitola je rozdělena na dvě velké podkapitoly: *optimalizace číslicových systémů s konečnou impulsní charakteristikou* a *optimalizace číslicových systémů s nekonečnou impulsní charakteristikou*. Obě pak obsahují vždy část věnovanou optimalizaci pro harvardskou architekturu a část věnovanou optimalizaci pro architekturu s velmi dlouhým instrukčním slovem.

Neméně důležitou částí habilitační práce je poslední kapitola 5 věnovaná vlivu aritmetiky v pevné řádové čarce na vlastní realizaci výpočetních algoritmů. Zejména u vestavených (embedded) systémů je stále vyžadována implementace v pevné řádové čarce a analýza vlivu kvantování musí být nedílnou součástí návrhu a optimalizace algoritmů.

1 SOUČASNÝ STAV ŘEŠENÉ PROBLEMATIKY

1.1 Vnější popis číslicového systému

Vnější popis lineárního číslicového systému s konstantními koeficienty je dán lineární diferenční rovnicí obecně s -tého řádu s konstantními koeficienty. Tento popis u lineárních časově invariantních systémů s jedním vstupem a jedním výstupem udává vztah mezi vstupním signálem $x[n]$ a výstupním signálem $y[n]$. Přenosová funkce systému je definována vztahem

$$H(z) = \frac{Y(z)}{X(z)}, \quad (1.1)$$

kde $Y(z)$ je obraz výstupního signálu a $X(z)$ je obraz vstupního signálu v \mathcal{Z} transformaci.

Pro jednoduchost je zvolen řád systému $s = 3$. Odvození pro obecný s -tý řád přenosové funkce je možné nalézt např. v pramenu [97, 32, 47]. Přenosová funkce systému 3. řádu bude mít obecný tvar

$$H(z) = \frac{b_3 z^3 + b_2 z^2 + b_1 z + b_0}{a_3 z^3 + a_2 z^2 + a_1 z + a_0} = \frac{c_0 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3}}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}}, \quad (1.2)$$

kde vztahy mezi jednotlivými koeficienty v obou tvarech je roven

$$c_0 = \frac{b_3}{a_3}, \quad c_1 = \frac{b_2}{a_3}, \quad c_2 = \frac{b_1}{a_3}, \quad c_3 = \frac{b_0}{a_3}, \quad d_1 = \frac{a_2}{a_3}, \quad d_2 = \frac{a_1}{a_3}, \quad \text{a} \quad d_3 = \frac{a_0}{a_3}. \quad (1.3)$$

Při hledání diferenční rovnice číslicového systému, která by realizovala zadanou přenosovou funkci $H(z)$, lze definici přenosové funkce (1.1) dosadit do obecného tvaru (1.2)

$$\frac{Y(z)}{X(z)} = \frac{c_0 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3}}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}}.$$

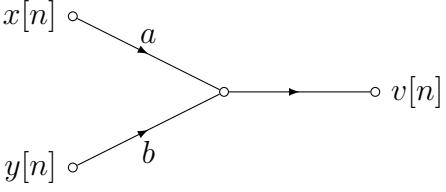
Poté se osamostatní obraz výstupního signálu $Y(z)$ v \mathcal{Z} transformaci

$$\begin{aligned} Y(z)(1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}) &= X(z)(c_0 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3}), \\ Y(z) &= c_0 X(z) + c_1 X(z)z^{-1} + c_2 X(z)z^{-2} + c_3 X(z)z^{-3} \\ &\quad - d_1 Y(z)z^{-1} - d_2 Y(z)z^{-2} - d_3 Y(z)z^{-3}. \end{aligned} \quad (1.4)$$

Provedením zpětné \mathcal{Z} transformace lze získat rekurentní rovnice 3. řádu pro výpočet výstupní hodnoty $y[n]$

$$\begin{aligned} y[n] &= c_0 x[n] + c_1 x[n-1] + c_2 x[n-2] + c_3 x[n-3] \\ &\quad - d_1 y[n-1] - d_2 y[n-2] - d_3 y[n-3]. \end{aligned} \quad (1.5)$$

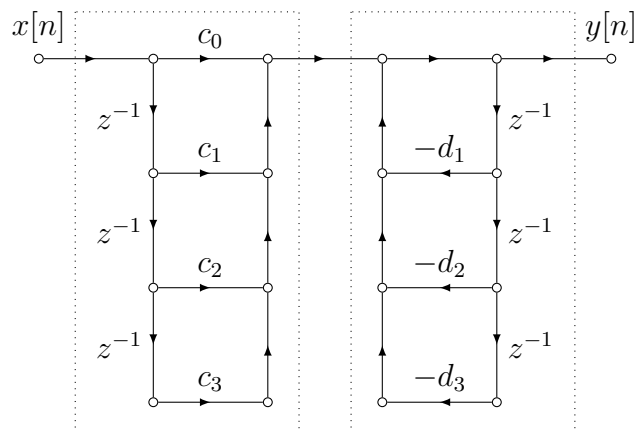
Pro popis diferenčních rovnic lze použít *graf signálových toků* [47, 97, 68], což je orientovaný graf s prvky podle obr. 1.1. Uzly představují vzorky vstupního nebo výstupního signálu, eventuálně vnitřních stavových proměnných. Hrany grafu pak určují vzájemný vztah mezi nimi. Obrázek 1.2 ukazuje zápis rekurentní diferenční rovnice (1.5) pomocí

$x[n] \circ \xrightarrow{a} \circ y[n]$	násobení konstantou $y[n] = a \cdot x[n]$, pokud a není uvedeno, předpokládá se $a = 1$
	součet $v[n] = a \cdot x[n] + b \cdot y[n]$
$x[n] \circ \xrightarrow{z^{-1}} \circ x[n-1]$	zpoždění o jeden vzorek

Obr. 1.1: Základní typy hran grafů signálových toků.

grafu signálových toků. Jedná se o 1. přímou strukturu DF1 (Direct Form I). Označení vychází z přímého vztahu mezi diferenční rovnicí a grafem signálových toků.

Graf signálových toků názorně ukazuje, jak bude výpočet probíhat např. v mikroprocesoru. Uzly si lze představit jako registry v datové paměti a v grafu je možné rozpoznat násobičky a sčítačky. Tato struktura se nazývá vnější struktura, neboť popisuje pouze souvislost mezi vstupním signálem $x[n]$ a výstupním signálem $y[n]$. Jedná se o *nekanonickou strukturu*, protože počet zpožďovacích bloků je větší než je řád diferenční rovnice a tomu odpovídající řád přenosové funkce, kterou struktura realizuje. V tomto případě má struktura 6 zpožďovacích bloků s z^{-1} , tj. dvakrát tolik než je řád $s = 3$ diferenční rovnice (1.5). Zavedením vnitřních (stavových) proměnných lze však získat kanonickou strukturu.



Obr. 1.2: Nekanonická vnější struktura realizující algoritmus číslicového systému 3. řádu typu IIR, který je definován diferenční rovnicí (1.5).

1.2 Vnitřní popis číslicového systému

Oproti vnějšímu popisu definuje vnitřní popis lineárního číslicového systému s konstantními koeficienty vztah mezi vstupními, výstupními a stavovými veličinami číslicového systému. Stavové proměnné definují u diskrétních systémů stav paměti daného systému, tj. stav paměti mikroprocesoru, který číslicový systém realizuje. Z toho důvodu je nutné znát blíže vnitřní strukturu diskrétního systému nebo si vytvořit její model. Stavový popis lze získat například přímo z vnějšího popisu vhodnou úpravou přenosové funkce $H(z)$. [97, 32, 47]

1.2.1 Druhá přímá forma

Stavové proměnné lze zavést rozšířením přenosové funkce (1.2) jedničkou ve tvaru $\frac{V(z)}{V(z)}$

$$H(z) = \frac{c_0 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3}}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}} = \frac{Y(z)}{X(z)} \frac{V(z)}{V(z)} .$$

Přenosovou funkci $H(z)$ rozdělíme na dvě části

$$\frac{V(z)}{X(z)} = \frac{1}{1 + d_1 z^{-1} + d_2 z^{-2} + d_3 z^{-3}} ,$$
$$\frac{Y(z)}{V(z)} = c_0 + c_1 z^{-1} + c_2 z^{-2} + c_3 z^{-3} .$$

Roznásobením a úpravou lze získat tyto obrazové rovnice

$$V(z) = X(z) - d_1 V(z) z^{-1} - d_2 V(z) z^{-2} - d_3 V(z) z^{-3} , \quad (1.6)$$

$$Y(z) = c_0 V(z) + c_1 V(z) z^{-1} + c_2 V(z) z^{-2} + c_3 V(z) z^{-3} . \quad (1.7)$$

Pomocí zpětné transformace \mathcal{Z} lze odvodit stavovou diferenční rovnici

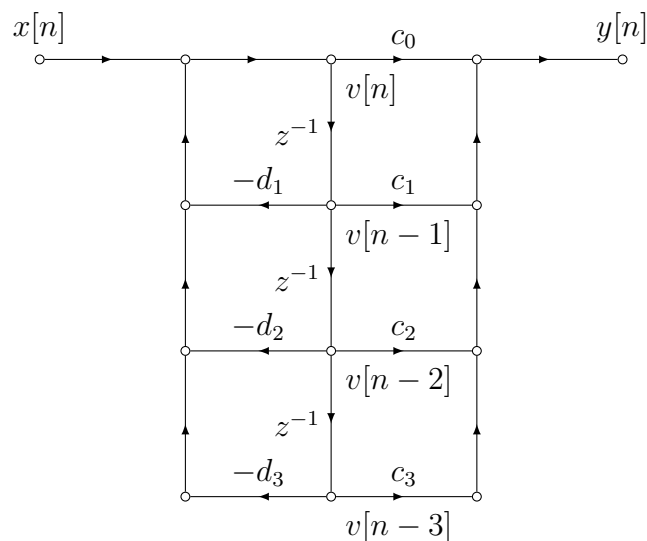
$$v[n] = x[n] - d_1 v[n-1] - d_2 v[n-2] - d_3 v[n-3] \quad (1.8)$$

a výstupní diferenční rovnici

$$y[n] = c_0 v[n] + c_1 v[n-1] + c_2 v[n-2] + c_3 v[n-3] . \quad (1.9)$$

Z vyjádření rovnice (1.8) a (1.9) pomocí grafu signálových toků na obr. 1.3, je zřejmé, že se jedná o *kanonickou strukturu*. Kanonická struktura má stejný počet zpožďovacích bloků jako je řád výchozí přenosové funkce (1.2). Zároveň je to nejnižší možný počet zpožďovacích bloků nutný pro realizaci daného systému. Graf signálových toků je velice podobný grafu na obr. 1.2. V podstatě došlo pouze k záměně pořadí tečkovaných bloků a sloučení bloků z^{-1} , které po záměně zpožďují stejný signál. Vzhledem k tomu, že realizovaný číslicový systém je lineární a časově invariantní, záměna pořadí operací nemá vliv na celkovou přenosovou funkci. Systém se pak označuje jako 2. přímá forma DF2 (Direct Form II) nebo 2. kanonická forma.

Podobně jako je tomu u spojitých systémů, je výhodnější rozepsat jednu stavovou diferenční rovnici s -tého řádu na s stavových diferenčních rovnic 1. řádu. Jestliže opět



Obr. 1.3: Kanonická vnitřní struktura realizující algoritmus číslicového systému 3. řádu typu IIR, který je definován diferenčními rovnicemi (1.8) a (1.9).

pro větší názornost bude použit číslicový systém 3. řádu, který je definován přenosovou funkcí (1.2), pak stavové diferenční rovnice lze získat tímto předpisem

$$\begin{aligned}
 v_1[n] &= v[n-3], \\
 v_2[n] &= v_1[n+1] = v[n-2], \\
 v_3[n] &= v_2[n+1] = v[n-1], \\
 v_3[n+1] &= v[n].
 \end{aligned} \tag{1.10}$$

Dosažením proměnných $v_1[n]$, $v_2[n]$ a $v_3[n]$ z rovnic (1.10) do vztahu (1.8) a po úpravě lze odvodit soustavu tří stavových rovnic

$$\begin{aligned}
 v_3[n+1] &= x[n] - d_1 v_3[n] - d_2 v_2[n] - d_3 v_1[n], \\
 v_2[n+1] &= v_3[n], \\
 v_1[n+1] &= v_2[n].
 \end{aligned} \tag{1.11}$$

Stavové rovnice (1.11) je možné vyjádřit maticovou rovnicí

$$\begin{bmatrix} v_3[n+1] \\ v_2[n+1] \\ v_1[n+1] \end{bmatrix} = \begin{bmatrix} -d_1 & -d_2 & -d_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} v_3[n] \\ v_2[n] \\ v_1[n] \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x[n], \tag{1.12}$$

nebo s využitím vektorových proměnných jako

$$\mathbf{v}[n+1] = \mathbf{A}\mathbf{v}[n] + \mathbf{B}x[n], \tag{1.13}$$

kde $\mathbf{v}[n+1] = [v_3[n+1] \ v_2[n+1] \ v_1[n+1]]^T$. Symbol T značí operaci transponování matice. Podobně vektor $\mathbf{v}[n] = [v_3[n] \ v_2[n] \ v_1[n]]^T$. Matice \mathbf{A} a \mathbf{B} jsou rovny

$$\mathbf{A} = \begin{bmatrix} -d_1 & -d_2 & -d_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}. \quad (1.14)$$

Po dosazení stavových proměnných $v_1[n], v_2[n]$ a $v_3[n]$ do vztahu (1.9) lze odvodit výstupní diferenční rovnici

$$y[n] = c_0 v_3[n+1] + c_1 v_3[n] + c_2 v_2[n] + c_3 v_1[n]. \quad (1.15)$$

Záměrem je však získat výstupní maticovou rovnici

$$y[n] = \mathbf{C}\mathbf{v}[n] + \mathbf{D}x[n]. \quad (1.16)$$

K tomu účelu je nutné dosadit do vztahu (1.15) proměnnou $v_3[n+1]$ a rovnici upravit

$$\begin{aligned} y[n] &= c_0(x[n] - d_1 v_3[n] - d_2 v_2[n] - d_3 v_1[n]) \\ &\quad + c_1 v_3[n] + c_2 v_2[n] + c_3 v_1[n], \\ y[n] &= c_0 x[n] + (c_1 - c_0 d_1) v_3[n] + (c_2 - c_0 d_2) v_2[n] \\ &\quad + (c_3 - c_0 d_3) v_1[n]. \end{aligned}$$

Maticový zápis pak je

$$y[n] = \begin{bmatrix} c_1 - c_0 d_1 & c_2 - c_0 d_2 & c_3 - c_0 d_3 \end{bmatrix} \begin{bmatrix} v_3[n] \\ v_2[n] \\ v_1[n] \end{bmatrix} + [c_0] x[n]. \quad (1.17)$$

Srovnáním zápisů (1.16) a (1.17) je zřejmé, čemu jsou rovny matice \mathbf{C} a \mathbf{D}

$$\mathbf{C} = [c_1 - c_0 d_1 \quad c_2 - c_0 d_2 \quad c_3 - c_0 d_3], \quad \mathbf{D} = [c_0]. \quad (1.18)$$

1.2.2 Transponovaná druhá přímá forma

Druhá kanonická forma není jediná možnost realizace. Pokud v diferenční rovnici (1.5) budou sloučeny členy se stejným zpožděním

$$\begin{aligned} y[n] &= c_0 x[n] + c_1 x[n-1] - d_1 y[n-1] + \\ &\quad c_2 x[n-2] - d_2 y[n-2] + \\ &\quad c_3 x[n-3] - d_3 y[n-3] \end{aligned}$$

je možné provést substituci

$$\begin{aligned}
 v_3[n+1] &= c_3x[n] - d_3y[n], \\
 v_2[n+1] &= c_2x[n] - d_2y[n] + v_3[n], \\
 v_1[n+1] &= c_1x[n] - d_1y[n] + v_2[n], \\
 y[n] &= c_0x[n] + v_1[n].
 \end{aligned}
 \tag{1.19}$$

Odpovídající graf signálových toků je na obr. 1.4. Opět je velice podobný grafu signálových toků 2. kanonické formy na obr. 1.3, graf je pouze transponován. *Transponování grafu* je operace, při níž se zamění uzly vstupní a výstupní veličiny a otočí se do protisměru šipky přenosu všech větví. Tato struktura se proto označuje jako transponovaná 2. přímá forma DF2T (Direct Form II Transposed) nebo také jako 1. kanonická forma. Z teorie grafů signálových toků je známé, že prostřednictvím transponování grafu je možné pro stejnou přenosovou funkci získat další struktury číslicových systémů.

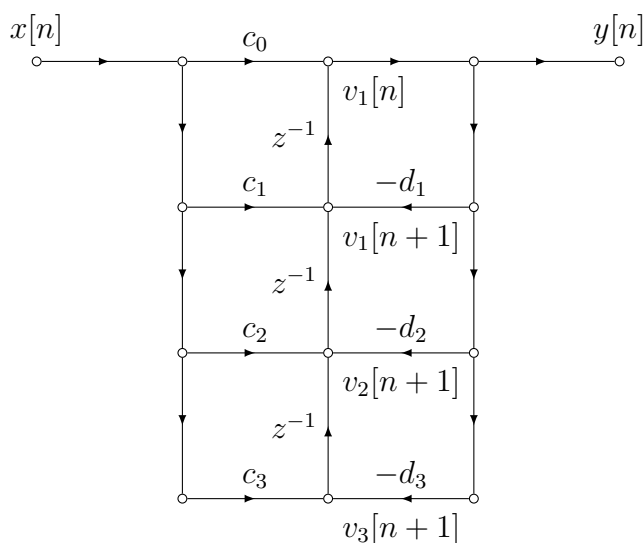
Maticový zápis lze získat z rovnic (1.13) a (1.16) po úpravě, když do prvních tří stavových rovnic bude dosazena rovnice výstupní. Matice **A**, **B**, **C** a **D** pro 1. kanonickou strukturu jsou rovny

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & -d_3 \\ 1 & 0 & -d_2 \\ 0 & 1 & -d_1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} c_3 - d_3c_0 \\ c_2 - d_2c_0 \\ c_1 - d_1c_0 \end{bmatrix}, \quad \mathbf{C} = [0 \quad 0 \quad 1], \quad \mathbf{D} = [c_0]. \tag{1.20}$$

1.2.3 Vazební struktura

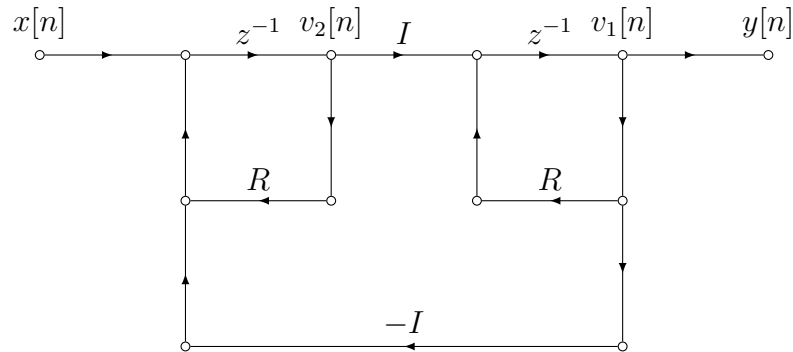
V případě čistě rekurzivního číslicového systému 2. řádu s přenosovou funkcí

$$H(z) = \frac{Iz^{-2}}{1 - 2Rz^{-1} + (R^2 + I^2)z^{-2}} \tag{1.21}$$



Obr. 1.4: Graf signálových toků 1. kanonické stavové struktury číslicového systému typu IIR 3. řádu.

lze použít vazební strukturu [47] (označovanou také jako *coupled*) s grafem signálových toků na obr. 1.5. Vlastností vazební struktury je, že koeficienty R a I jsou reálná a imaginární složka komplexně sdružené dvojice pólů přenosové funkce. Proto se tato struktura často používá pro realizaci rezonátorů s póly na jednotkové kružnici. V takovém případě jsou stavové proměnné $v_1[n]$ a $v_2[n]$ fázově posunuty o $\frac{\pi}{2}$ a generují tak kvadraturní signál.



Obr. 1.5: Graf signálových toků vazební struktury čistě rekurzivního číslicového systému 2. řádu.

Stavové rovnice pak budou mít tvar

$$\begin{aligned} v_2[n+1] &= x[n] + Rv_2[n] - Iv_1[n], \\ v_1[n+1] &= Iv_2[n] + Rv_1[n], \\ y[n] &= v_1[n]. \end{aligned} \quad (1.22)$$

Ze stavových rovnic je přímo patrný maticový zápis. Matice \mathbf{A} , \mathbf{B} , \mathbf{C} a \mathbf{D} pro vazební strukturu jsou

$$\mathbf{A} = \begin{bmatrix} R & -I \\ I & R \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{C} = [1 \ 0], \quad \mathbf{D} = [0]. \quad (1.23)$$

1.2.4 Křížová struktura

Pro systémy s konečnou impulsní charakteristikou

$$y[n] = \sum_{m=0}^{M-1} h[m]x[n-m] \quad (1.24)$$

lze použít i křížovou (lattice) strukturu [47, 32]. Základem struktury je jeden blok na obr. 1.6 vyznačený čárkovaně. Vstupní signál je rozdělen do dvou cest

$$\begin{aligned} f_0[n] &= x[n], \\ g_0[n] &= x[n]. \end{aligned}$$

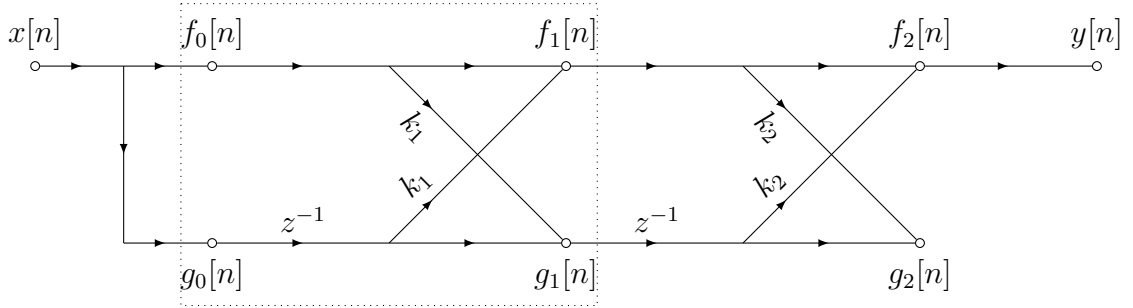
Systém 1. řádu realizují výstupy prvního bloku

$$\begin{aligned} f_1[n] &= f_0[n] + k_1g_0[n-1] = x[n] + k_1x[n-1], \\ g_1[n] &= k_1f_0[n] + g_0[n-1] = k_1x[n] + x[n-1]. \end{aligned}$$

Provedením \mathcal{Z} transformace lze odvodit přenosové funkce

$$A_1(z) = \frac{F_1(z)}{X(z)} = 1 + k_1 z^{-1},$$

$$B_1(z) = \frac{G_1(z)}{X(z)} = k_1 + z^{-1}.$$



Obr. 1.6: Křížová struktura pro implementaci systému s konečnou impulsní charakteristikou.

Systém 2. řádu lze získat opakováním bloku

$$\begin{aligned} f_2[n] &= f_1[n] + k_2 g_1[n-1] = x[n] + k_1 x[n-1] + k_2 (k_1 x[n-1] + x[n-2]) \\ &= x[n] + k_1 (1 + k_2) x[n-1] + k_2 x[n-2], \\ g_2[n] &= k_2 f_1[n] + g_1[n-1] = k_2 (x[n] + k_1 x[n-1]) + k_1 x[n-1] + x[n-2] \\ &= k_2 x[n] + k_1 (1 + k_2) x[n-1] + x[n-2]. \end{aligned}$$

Přenosové funkce budou mít tvar

$$\begin{aligned} A_2(z) &= A_1(z) + k_2 z^{-1} B_1(z) = 1 + k_1 z^{-1} + k_2 z^{-1} (k_1 + z^{-1}) \\ &= 1 + k_1 (1 + k_2) z^{-1} + k_2 z^{-2}, \\ B_2(z) &= k_2 A_1(z) + z^{-1} B_1(z) = k_2 (1 + k_1 z^{-1}) + z^{-1} (k_1 + z^{-1}) \\ &= k_2 + k_1 (1 + k_2) z^{-1} + z^{-2}. \end{aligned}$$

Zobecněním lze odvodit rekurentní rovnice

$$\begin{aligned} A_0(z) &= 1, \\ B_0(z) &= 1, \\ A_m(z) &= A_{m-1}(z) + k_m z^{-1} B_{m-1}(z), \\ B_m(z) &= k_m A_{m-1}(z) + z^{-1} B_{m-1}(z), \end{aligned} \tag{1.25}$$

pro $m = 1, 2, \dots, M-1$, kde $M-1$ je řád přenosové funkce realizovaného systému. Je zřejmé, že mezi přenosovými funkcemi $A_m(z)$ a $B_m(z)$ platí vztah

$$\begin{aligned} B_m(z) &= z^{-m} A_m(z^{-1}) \\ &= z^{-m} \left(A_{m-1}(z^{-1}) + k_m (z^{-1})^{-1} B_{m-1}(z^{-1}) \right) \\ &= z^{-1} z^{-m+1} A_{m-1}(z^{-1}) + k_m z^{-m+1} B_{m-1}(z^{-1}) \\ &= z^{-1} B_{m-1}(z) + k_m A_{m-1}(z). \end{aligned} \tag{1.26}$$

Oba polynomy mají tedy stejné hodnoty koeficientů, ale v opačném pořadí

$$\begin{aligned} f_m[n] &= \sum_{k=0}^m a_k x[n-k], \\ g_m[n] &= \sum_{k=0}^m a_{m-k} x[n-k]. \end{aligned} \quad (1.27)$$

První větev $f_m[n]$ se proto někdy označuje jako přímý prediktor (*forward predictor*), zatímco druhá větev $g_m[n]$ jako zpětný prediktor (*backward predictor*).

Dosazením (1.26) do (1.25) lze odvodit vztahy pro rekurentní výpočet koeficientů a_k přenosové funkce A_m řádu m z koeficientů k_m

$$\begin{aligned} a_0^{(m)} &= 1, \\ a_m^{(m)} &= k_m, \\ a_k^{(m)} &= a_k^{(m-1)} + k_m a_{m-k}^{(m-1)}, \quad k \in \langle 1; m-1 \rangle. \end{aligned} \quad (1.28)$$

Vztah (1.28) je podobný rekurentním vztahům pro Levinson Durbinův algoritmus [47]. Koeficienty k_m se pak označují jako koeficienty odrazu (*reflection coefficient*).

Ze vztahu (1.25) lze odvodit i opačný postup pro zjištění koeficientů odrazu k_m z koeficientů a_k přenosové funkce m . řádu.

$$\begin{aligned} A_m(z) &= A_{m-1}(z) + k_m z^{-1} B_{m-1}(z) \\ &= A_{m-1}(z) + k_m (B_m(z) - k_m A_{m-1}(z)) \\ A_{m-1}(z) &= \frac{A_m(z) - k_m B_m(z)}{1 - k_m^2}. \end{aligned} \quad (1.29)$$

Jednotlivé koeficienty k_m lze tedy získat rekurentním snižováním řádu $m = M-1, M-2, \dots, 1$ polynomu podle vztahů

$$\begin{aligned} k_m &= a_m^{(m)}, \\ a_0^{(m-1)} &= 1, \\ a_k^{(m-1)} &= \frac{a_k^{(m)} - k_m a_{m-k}^{(m)}}{1 - k_m^2}, \quad k \in \langle 1; m-1 \rangle. \end{aligned} \quad (1.30)$$

Ze srovnání (1.27) a (1.24) přitom vyplývá, že ze vzorků impulsní charakteristiky lze získat koeficienty původní přenosové funkce $(M-1)$. řádu podle vztahu

$$a_k = \frac{h[k]}{h[0]}, \quad (1.31)$$

kde $h[k]$ jsou vzorky impulsní charakteristiky realizovaného systému typu FIR (Finite Impulse Response).

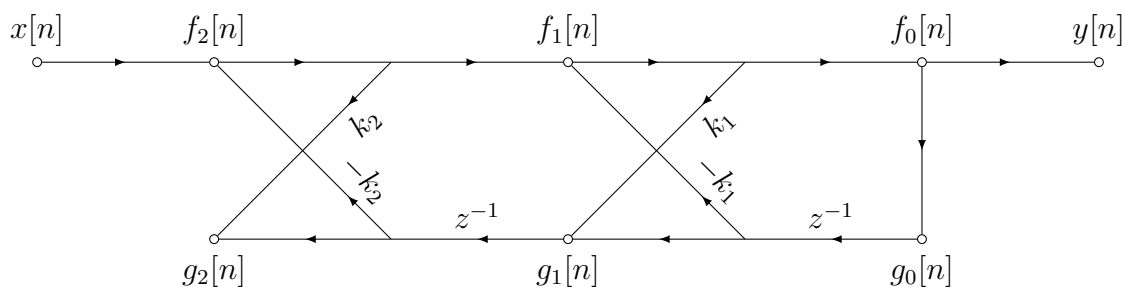
Křížovou strukturu lze použít i pro realizaci čistě rekurzivních systémů

$$\begin{aligned} y[n] &= x[n] - \sum_{m=1}^{M-1} a_m y[n-m], \\ y[n] + \sum_{m=1}^{M-1} a_m y[n-m] &= x[n]. \end{aligned} \quad (1.32)$$

Ze srovnání (1.32) a (1.24) plyne, že rovnice mají podobný tvar, ale jsou vzájemně zaměněny signály $x[n]$ a $y[n]$. Pro realizaci tak lze použít strukturu na obr. 1.7, která

připomíná zrcadlově převrácenou strukturou z obr. 1.6. Odvození jednotlivých přenosových funkcí je podobné jako v případě realizace systému s konečnou impulsní charakteristikou a pro výpočet koeficientů odrazu lze použít stejné rekurentní rovnice (1.29). Vztahy jsou podobné jako u Schur-Cohnova testu stability přenosové funkce [47, 32]. Aby všechny kořeny polynomu $A_m(z)$ ležely uvnitř jednotkové kružnice a systém byl stabilní, musí být všechny koeficienty odrazu v absolutní hodnotě menší než 1

$$|k_m| < 1. \quad (1.33)$$



Obr. 1.7: Graf signálových toků křížové struktury pro realizaci čistě rekurzivního systému.

1.3 Přehled používaných formátů čísel

Technické prostředky pro zpracování číslicových signálů používají pro vyjádření hodnot vzorků nebo koeficientů omezený počet diskretních hodnot určený konečným počtem bitů použitých pro jejich číselné vyjádření. To nepříznivě ovlivňuje *rozsah hodnot* a *přesnost* jejich vyjádření. Oba uvedené parametry silně závisí na konkrétním tvaru formátu použitého pro vyjádření čísel, přičemž nejběžněji používané formáty můžeme rozdělit do dvou hlavních kategorií:

pevná řádová čárka: pozice řádové čárky se ve formátu nemůže měnit,

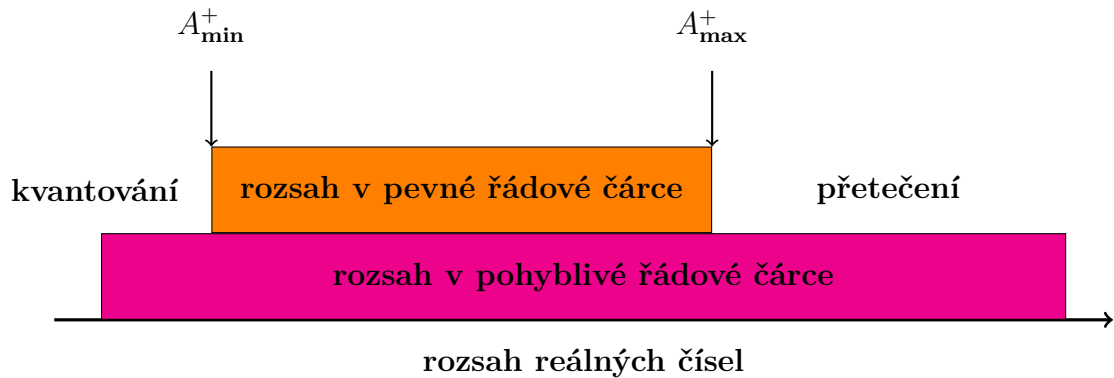
pohyblivá řádová čárka: pozice řádové čárky se ve formátu může měnit.

Rozsah hodnot je zjednodušeně definován maximální A_{\max}^+ a minimální A_{\min}^+ kladnou hodnotou, které lze v daném formátu zobrazit. Udává se většinou v podobě dynamického rozsahu, který je definován jako podíl uvedených dvou hodnot

$$R_{\text{dB}} = 20 \log_{10} \frac{A_{\max}^+}{A_{\min}^+} \text{ [dB]}. \quad (1.34)$$

Formát je nutné zvolit tak, aby jeho rozsah odpovídal rozsahu hodnot zpracovávaného signálu. Pokud signál obsahuje hodnoty v absolutní hodnotě větší než je maximální zobrazitelná hodnota, dojde k přetečení a k chybám zpracování. Pokud signál obsahuje hodnoty v absolutní hodnotě menší, než je minimální kladná zobrazitelná hodnota, budou tyto hodnoty vlivem kvantování vynulovány a opět dojde k chybám při zpracování.

Ilustrační zobrazení rozsahu jednotlivých formátů je na obr. 1.8. Formáty s pevnou řádovou čárkou mají obecně menší rozsah zobrazitelných hodnot. Polohu řádové čárky je



Obr. 1.8: Ilustrační zobrazení rozsahu různých formátů vyjádření čísel a vzniku zkrselení.

možné zvolit a tak přizpůsobit rozsahu hodnot zpracovávaného signálu, během zpracování se však poloha nemění. Při volbě pozice řádové čárky (celá, zlomková, smíšená čísla) se grafické vyjádření rozsahu formátu v pevné řádové čárce na obr. 1.8 tedy může posunovat podél reálné osy a tím zvolit, jaké hodnoty je možné ve formátu zobrazit. V případě formátů s pohyblivou řádovou čárkou se může pozice řádové čárky měnit i během zpracování, což má za důsledek mnohem větší rozsah zobrazitelných hodnot.

Přesnost formátu je možné definovat jako nejmenší rozdíl mezi dvěma sousedními hodnotami, které leží v rozsahu zobrazitelných čísel. Oproti zažitému názoru, že formáty s pohyblivou řádovou čárkou jsou výrazně přesnější než s pevnou řádovou čárkou, jsou ve skutečnosti mezi formáty jen malé rozdíly. Např. 32bitový formát s pohyblivou řádovou čárkou podle standardu IEEE754 používá pro uložení zlomkové části mantisy 23 bitů [23]. Nejmenší rozdíl mezi dvěma sousedními čísly je tak 2^{-23} . Signálové procesory běžně pracují se 16bitovými zlomkovými čísly s pevnou řádovou čárkou, u kterých je nejmenší rozdíl roven 2^{-15} . Střadače jsou však 32bitové a nejmenší rozdíl je pak roven 2^{-31} . Při stejném počtu bitů tedy může být přesnější formát v pevné řádové čárce, pokud všechny hodnoty budou ve stejném rozsahu.

Velké rozdíly mohou také plynout z formátu vyjádření záporných čísel, kde postupně vznikaly a nyní se používají formáty:

přímý kód: je vyjádřena absolutní hodnota čísla, kdy v případě záporné hodnoty je znaménkový bit (většinou nejvýznamnější bit MSB) nastaven na 1, a v případě kladné hodnoty na 0,

jednotkový doplněk: je vyjádřena absolutní hodnota čísla, kdy pro případ záporné hodnoty jsou poté všechny bity invertovány,

dvojkový doplněk: je vyjádřena absolutní hodnota čísla, kdy pro případ záporného čísla jsou poté invertovány všechny bity a k nejméně významnému bitu LSB je přičtena 1.

Různé formáty záporných čísel se chovají různě při přetečení, které vzniká při vyjádření hodnoty v absolutní hodnotě větší než A_{\max}^+ .

Hlavní vlastnosti nejběžnějších formátů jsou uvedeny v tab. 1.1.

Tab. 1.1: Porovnání hlavních vlastností nejčastěji používaných formátů vyjádření čísel.

Formát	Rozsah čísel	Dynamický rozsah	Rozlišení
Pevná řádová čárka s počtem bitů 16			
Celé Q16.0 BZ	0 až 65 535	96 dB	1
Celé Q16.0 SZ	−32 768 až 32 767	90 dB	1
Zlomkové Q0.16 BZ	0 až 0,99998474	96 dB	2^{-16}
Zlomkové Q1.15 SZ	−1 až 0,99998474	90 dB	2^{-15}
Pohyblivá řádová čárka dle IEEE754			
Základní přesnost	$-3,40 \cdot 10^{38}$ až $3,40 \cdot 10^{38}$	1 529 dB	2^{-23}
Dvojitá přesnost	$-1,80 \cdot 10^{308}$ až $1,80 \cdot 10^{308}$	12 318 dB	2^{-52}
BZ – bez znaménka			
SZ – se znaménkem			

V případě číslicového zpracování signálů se nejčastěji používá formát zlomkových čísel ve dvojkovém doplňku. Od celočíselných formátů se liší pouze umístěním řádové čárky před MSB (Most Significant Bit) nebo za LSB (Least Significant Bit). V případě formátu se znaménkem je řádová čárka umístěna za znaménkovým bitem. Formálně lze přemístění řádové čárky před MSB provést vynásobením čísla hodnotou 2^{-N_b+1}

$$A = 2^{-N_b+1} \left(-b_{N_b-1} 2^{N_b-1} + \sum_{k=0}^{N_b-2} b_k 2^k \right) = -b_{N_b-1} 2^0 + \sum_{k=0}^{N_b-2} b_k 2^{k-N_b+1}, \quad (1.35)$$

kde N_b je počet bitů zvoleného formátu, a b_k je hodnota (0 nebo 1) bitu s indexem k . Pokud v případě umístění řádové čárky za LSB (celočíselného formátu) jsou váhy jednotlivých bitů

b_3	b_2	b_1	b_0
-2^3	2^2	2^1	2^0

potom v případě umístění řádové čárky za znaménkový bit MSB (zlomkový formát) se váha jednotlivých bitů změní na

$b_3 \cdot b_2$	b_1	b_0
$-2^0 \cdot 2^{-1}$	2^{-2}	2^{-3}

Odpovídajícím způsobem se změní i rozsah z $\langle -2^{N_b-1}; 2^{N_b-1} - 1 \rangle$ v případě celočíselného formátu na $\langle -1; 1 - 2^{-N_b+1} \rangle$ v případě zlomkového formátu. Příklady vyjádření 4bitového zlomkového čísla ve formátu dvojkového doplňku Q1.3 jsou uvedeny v tab. 1.2.

Tab. 1.2: Vyjádření 4bitových zlomkových čísel v přímém kódu, jednotkovém doplňku a dvojkovém doplňku. V binárním kódu je poloha řádové čárky vyjádřena symbolem \cdot .

$(\cdot)_2$	přímý kód	jednotkový doplněk	dvojkový doplněk
0 . 000	0,0000	0,0000	0,0000
0 . 001	0,1250	0,1250	0,1250
0 . 010	0,2500	0,2500	0,2500
0 . 011	0,3750	0,3750	0,3750
0 . 100	0,5000	0,5000	0,5000
0 . 101	0,6250	0,6250	0,6250
0 . 110	0,7500	0,7500	0,7500
0 . 111	0,8750	0,8750	0,8750
1 . 000	0,0000	-0,8750	-1,0000
1 . 001	-0,1250	-0,7500	-0,8750
1 . 010	-0,2500	-0,6250	-0,7500
1 . 011	-0,3750	-0,5000	-0,6250
1 . 100	-0,5000	-0,3750	-0,5000
1 . 101	-0,6250	-0,2500	-0,3750
1 . 110	-0,7500	-0,1250	-0,2500
1 . 111	-0,8750	-0,0000	-0,1250

Výhodou oproti celočíselnému formátu je menší riziko vzniku přetečení, protože při běžných operacích neroste jejich rozsah. Např. při násobení:

$$\begin{array}{r} (0,50000)_{10} \quad (0 \cdot 1000)_2 \\ \times (0,18750)_{10} \quad \times (0 \cdot 0011)_2 \\ \hline (0,09375)_{10} \quad (0 \cdot 0001 \ 1000)_2 \\ \hline (0,06250)_{10} \quad (0 \cdot 0001)_2 \end{array}$$

má výsledek dvojnásobný počet bitů podobně jako u celočíselného formátu. Pokud ale oddělíme část se stejnou pozicí řádové čárky (v případě zlomkových čísel horní polovinu slova) nedojde k přetečení jako u celočíselného formátu. Z důvodu zkrácení délky slova dojde k chybě jiného typu, a to ke *kvantování* výsledku. Chyba nebude nulová, ale bude výrazně menší než v případě přetečení celočíselných formátů. V tomto konkrétním případě $e_Q = 0,06250 - 0,09375 = -0,03125$.

Přetečení při násobení však není zcela vyloučeno. Z důvodu nesymetrického rozsahu dvojkového doplňku může dojít k přetečení při násobení:

$$\begin{array}{r} (-1)_{10} \quad (1 \cdot 000)_2 \\ \times (-1)_{10} \quad \times (1 \cdot 000)_2 \\ \hline = (1)_{10} \quad (01 \cdot 000 \ 000)_2 = (-1)_{10} \end{array}$$

Chyba je rovna $e_Q = -1 - 1 = -2$ a je tedy větší než 100%. Zároveň je z příkladu patrné posunutí řádové čárky, která ve výsledku není umístěna za nejvýznamnějším bitem MSB, ale až za druhým nejvýznamnějším bitem. Je to z toho důvodu, že oba činitelé mají 3 bity za řádovou čárkou a proto výsledek bude mít $3 + 3 = 6$ bitů za řádovou čárkou. Pro správnou pozici řádové čárky je nutné provést bitový posun vlevo o 1 bit. Posunutí musí

být provedeno tak, aby se zabránilo přetečení při násobení $(-1)_{10} \cdot (-1)_{10}$, jak bylo uvedeno v předchozím příkladu. Posunutí je také jediný rozdíl mezi operacemi v celočíselném a zlomkovém formátu. Proto u některých procesorů najdeme dvě různé instrukce pro násobení: např. `mult` nebo `MPY` pro celočíselné násobení a `fmult` nebo `SMPY` pro násobení zlomkových čísel ve dvojkovém doplňku. Operace sčítání a odečítání je možné provádět stejně bez rozdílu, pozice řádové čárky zůstává stejná.

1.4 Kvantování v případě pevné řádové čárky

Jak bylo naznačeno v předchozí části po operaci násobení dojde ke zdvojnásobení délky slova. Pro další zpracování je nutné provést zkrácení na stejnou délku jako měl původní vzorek. K omezení chyb postupně vznikly tři možné způsoby zkrácení slova:

usekávání: ponechání pouze N_b nejvýznamnějších bitů, tj. nahrazení největší hodnotou v daném formátu, která je menší nebo maximálně rovnou původní hodnotě,

zaokrouhlení: přičtení 1 k nejvýznamnějšímu bitu spodní poloviny a následné useknutí, tj. nahrazení nejbližší kvantovanou hodnotou,

konvergentní zaokrouhlení: adaptivní nahrazení nejbližší kvantovanou hodnotou.

V případě usekávání (*truncation*) je použito pouze N_b nejvýznamnějších bitů. Ve formátu dvojkového doplňku to znamená, že hodnota je vyjádřena nejbližší nižší kvantovací hladinou

$$A_T = \lfloor A \rfloor. \quad (1.36)$$

Kvantovacího šum nabývá hodnot z intervalu $e_T(A) = A_T - A \in (-q; 0)$. Za předpokladu, že kvantovací šum je nekorelovaný, tj. všechny hodnoty kvantovacího šumu jsou stejně pravděpodobné (hustota pravděpodobnosti má rovnoměrné rozložení), je střední hodnota kvantovacího šumu rovna

$$\mu_T = \int_{-q}^0 e_T p(e_T) de_T = -\frac{q}{2}, \quad (1.37)$$

kde q je hodnota kvantovacího kroku (rozdíl mezi sousedními kvantovacími hladinami) a $p(e_T)$ je hustota pravděpodobnosti. Rozptyl kvantovacího šumu je roven

$$\sigma_T^2 = \int_{-q}^0 (e_T - \mu_T)^2 p(e_T) de_T = \frac{q^2}{12}. \quad (1.38)$$

Příklad vzniku kvantovacího šumu při usekávání je na obr. 1.9 zobrazen modrou barvou.

Při výpočtu střední energie P_{e_T} upravíme definiční vztah pro rozptyl

$$\begin{aligned} \sigma_T^2 &= \int_{-q}^0 (e_T - \mu_T)^2 p(e_T) de_T = \\ &= \int_{-q}^0 (e_T^2 p(e_T) - 2\mu_T e_T p(e_T) + \mu_T^2 p(e_T)) de_T = \\ &= \int_{-q}^0 e_T^2 p(e_T) de_T - 2\mu_T \int_{-q}^0 e_T p(e_T) de_T + \int_{-q}^0 \mu_T^2 p(e_T) de_T \\ P_{e_T} = \int_{-q}^0 e_T^2 p(e_T) dx &= \sigma_T^2 + \mu_T^2 = \frac{q^2}{12} + \left(-\frac{q}{2}\right)^2 = \frac{q^2}{3} \end{aligned} \quad (1.39)$$

Střední energie kvantovacího šumu při usekávání je rovna součtu rozptylu σ_T^2 a druhé mocniny střední hodnoty μ_T^2 .

V případě zaokrouhlení (*rounding*) je hodnota signálu vyjádřena nejbližší kvantovací hladinou. U formátu dvojkového doplňku to lze provést jako useknutí hodnoty, ke které je předtím přičtena polovina kvantovacího kroku

$$A_R = \lfloor A + 0,5 \rfloor. \quad (1.40)$$

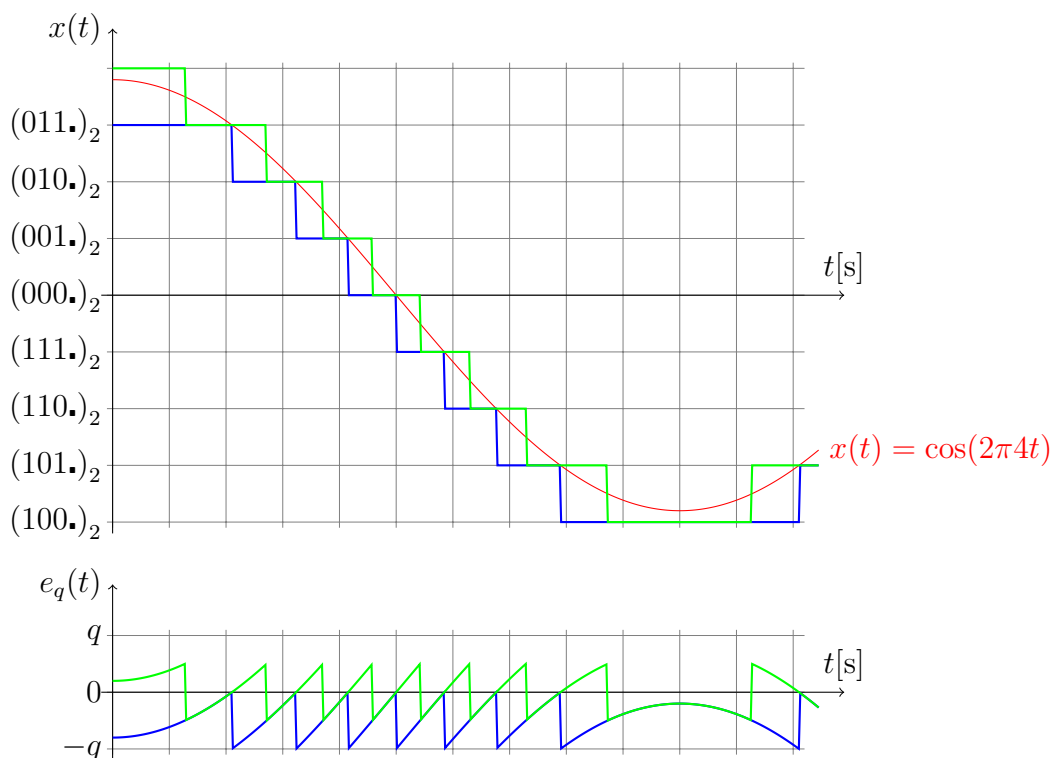
Kvantovací šum nabývá hodnot z intervalu $e_R = A_R - A \in (-\frac{q}{2}; \frac{q}{2})$. Za předpokladu, že kvantovací šum je nekorelovaný s hodnotami signálu, je střední hodnota kvantovacího šumu rovna

$$\mu_R = \int_{-\frac{q}{2}}^{\frac{q}{2}} e_R p(e_R) de_R = 0 \quad (1.41)$$

a rozptyl kvantovacího šumu je roven

$$\sigma_R^2 = \int_{-\frac{q}{2}}^{\frac{q}{2}} (e_R - \mu_R)^2 p(e_R) de_R = \frac{q^2}{12}. \quad (1.42)$$

Příklad vzniku kvantovacího šumu při zaokrouhlování je na obr. 1.9 zobrazen zelenou barvou.



Obr. 1.9: Ukázka usekávání (modře) a zaokrouhlování (zeleně) při kvantování signálu.

Hodnota rozptylu kvantovacího šumu v případě zaokrouhlování je rovna hodnotě rozptylu kvantovacího šumu při usekávání. Rozdíl je však ve střední hodnotě kvantovacího šumu, která je v případě zaokrouhlení rovna 0. Po dosazení do vztahu pro střední energii (1.39)

$$P_{e_R} = \sigma_R^2 + \mu_R^2 = \sigma_R^2 = \frac{q^2}{12} \quad (1.43)$$

bude střední energie menší než v případě usekávání. Z tohoto důvodu se zaokrouhlování používá častěji než usekávání. V některých případech však může způsobit problémy přičítání hodnoty 0,5 v rovnici (1.40) ke všem hodnotám signálu. Důsledkem může být vznik mezních cyklů [47, 32] při zpracování.

Pro omezení vlivu přičítání hodnoty 0,5 při zaokrouhlování bylo zavedeno tzv. *konvergentní zaokrouhlení* nebo-li zaokrouhlení k nejbližší sudé hodnotě, které některé signálové procesory hardwarově podporují. V případě konvergentního zaokrouhlování se v mezním případě, kdy část vpravo od místa zkrácení je rovna přesně ,500... (tj. ve spodní polovině slova je nastaven pouze nejvýznamnější bit) provede přičtení hodnoty 0,5 pouze tehdy, pokud je bit vlevo od místa zkrácení (tj. nejméně významný bit horní poloviny slova) nastaven. Pokud je bit nulový, tzn. celá část je sudá, přičtení se neprovede. Např. (symbol \wr vyznačuje místo zkrácení)

$$\begin{array}{lll} \text{nekvantovaná hodnota} & (10 \wr 5)_{10} = (1010 \wr 100 \dots)_2 & (9,5)_{10} = (1001 \wr 100 \dots)_2 \\ \text{hodnota před zkrácením} & (10 \wr 1)_{10} = (1010 \wr 000 \dots)_2 & (10,0)_{10} = (1010 \wr 000 \dots)_2 \\ \text{zaokrouhlená hodnota} & (10 \wr 0)_{10} = (1010 \wr 000 \dots)_2 & (10,0)_{10} = (1010 \wr 000 \dots)_2 \end{array}$$

Jako veličina pro porovnání vlivu kvantování se nejčastěji používá *poměr signálu ku kvantovacímu šumu* SQNR (Signal to Quantization Noise Ratio), který je definován jako poměr střední energie zpracovávaného signálu P_s vztažený ke střední energii kvantovacího šumu P_{e_q} a vyjadřuje se většinou v dB

$$SQNR = 10 \log_{10} \frac{P_s}{P_{e_q}} \text{ [dB]}. \quad (1.44)$$

1.5 Porovnání vlastností signálových procesorů

I když signálové procesory s harvardskou architekturou byly v minulosti vyráběny i s pohyblivou řádovou čárkou (např. řada DSP96xxxx), v současnosti se téměř neobjevují. Jejich výpočetní výkon je totiž nižší než výkon nejrůznějších ARM (Advanced RISC Machine) procesorů, které mají i podobné nároky na spotřebu. Jiná situace je v oblasti procesorů s pevnou řádovou čárkou, kde výkon signálových procesorů v algoritmech číslicového zpracování signálu především díky paralelním přesunům a instrukci násobení s akumulací přesahuje výkon procesorů pro všeobecné použití. Kmitočet hodinového signálu se pohybuje řádově ve stovkách MHz, výpočetní výkon v milionech operací násobení s akumulací MMAC (Million Multiplies and Accumulate) je poloviční (architektury 2x se dvěma hodinovými cykly na instrukci), odpovídá hodinovému kmitočtu (architektury 1x s jediným hodinovým cyklem na instrukci) nebo je až dvojnásobný (díky dvěma paralelním jednotkám MAC (Multiply and Accumulate) u řady TMS320C55xx). Cena se odvíjí především podle vybavení procesoru pamětí a dalšími perifériemi, případně koprocesory, a pohybuje se řádově do desítek dolarů při odběru řádově tisíců kusů. Základní parametry několika procesorů s harvardskou architekturou jsou shrnuty v tab. 1.3.

Poněkud jiná situace je v oblasti signálových procesorů s architekturou s velmi dlouhým instrukčním slovem. Díky rozdělení jádra na paralelně pracující jednotky se výkon signálových procesorů s velmi dlouhým instrukčním slovem podstatně zvýšil v porovnání s harvardskou architekturou. A i když výkon v pohyblivé řádové čárce procesorů pro

Tab. 1.3: Porovnání vlastností některých procesorů s pevnou řádovou čárkou a harvardskou architekturou.[5, 6, 33, 83, 79, 83]

Processor	Cena [\$]	Kmitočet [MHz]	Výkon [MMACS]	Program [Kslov]	Data [Kslov]
ADSP-21992	34,40	160	160	32	16
ADSP-2191M	25,74	160	160	32	32
DSP56F803	9,46	80	40	32	2
DSP56F801	4,71	80	40	8	1
TMS320VC5402A	11,85	160	160	16	16
TMS320VC5501	4,91	300	600	16	4

všeobecné použití, především nejvýkonnější řady Core i7 firmy Intel se superskalární architekturou, stále převyšuje výkon signálových procesorů, signálové procesory mají nižší spotřebu a jednodušší návrh řešení včetně desky plošných spojů. Proto v případě architektury s velmi dlouhým instrukčním slovem zejména v oblasti vestavěných (embedded) zařízeních se signálové procesory s pohyblivou řádovou čárkou stále používají.

Parametry několika signálových procesorů s velmi dlouhým instrukčním slovem jsou shrnuty v tab. 1.4. Zajímavé je, že oproti minulosti, kdy procesory s pohyblivou řádovou čárkou byly výrazně dražší než procesory s pevnou řádovou čárkou, se dnes ceny u procesorů obou kategorií příliš neliší. Větší rozdíly jsou opět při rozdílné výbavě periferiemi nebo kapacitě paměti. Důvodem může být zvládnutí technologie vyšší integrace. Jednotka pro operace v pohyblivé řádové čárce je totiž přirozeně složitější a zabírá větší místo na čipu. Díky vyšší integraci ale poměr plochy jednotky vůči ostatním blokům výrazně poklesl a tím mohla klesnout i cena. V čem se však procesory s pohyblivou řádovou čárkou výrazně odlišují od procesorů s pevnou řádovou čárkou je naopak nižší dosažitelný kmitočet hodinového signálu a tedy i nižší výpočetní výkon. Důvodem může být opět složitější provádění aritmetických operací s čísly v pohyblivé řádové čárce oproti pevné řádové čárce, což vede ke komplikovanější DALU (Data Arithmetic Logic Unit) neschopné pracovat na vyšším hodinovém kmitočtu. Z tohoto pohledu má (a zřejmě i do budoucna bude mít) pořád význam zabývat se úpravou algoritmů pro pevnou řádovou čárku, kde postačuje méně výkonný signálový procesor nebo při stejném hodinovém kmitočtu je možné současně zpracovat více kanálů signálu.

Tab. 1.4: Porovnání vlastností některých procesorů s pevnou a pohyblivou řádovou čárkou s architekturou s velmi dlouhým instrukčním slovem.[1, 2, 20, 21, 90, 7, 85, 87]

Procesor	Cena [\\$]	Kmitočet [MHz]	Výkon [MMACS]	Paměť [KB]
s pevnou řádovou čárkou				
ADSP-BF561	22,44	600	2400	328
ADSP-BF609	21,81	500	2000	328
MSC8151	75,81	1000	4000	512
MSC8251	67,69	1000	8000	512
TMS320C6416	209,78	1000	4000	4096
TMS320C6412	50,30	720	2400	2048
s pohyblivou řádovou čárkou				
ADSP-21587	26,25	450	1800	256
ADSP-21584	22,82	450	1800	256
TMS320C6701	87,00	167	334	1024
TMS320C6720	7,09	350	700	256

2 CÍLE HABILITAČNÍ PRÁCE

Cílem habilitační práce je rozvinout problematiku optimalizace struktur číslicového zpracování signálů z hlediska architektury moderních signálových procesorů. Přitom jsou uvažovány pouze lineární časově invariantní systémy. V současné době jsou stále nejčastěji používány struktury, jejichž matematický základ vznikl v druhé polovině minulého století. V oblasti zpracování řeči je to především křížová struktura, v oblasti systémů s konečnou impulsní charakteristikou lineární konvoluce a v oblasti obecných číslicových systémů především kaskádní zapojení dílčích sekcí 2. řádu realizovaných v 1. kanonické formě. V době odvození těchto struktur byly k dispozici pouze signálové procesory s harvardskou architekturou, jejichž výpočetní výkon již nedostačuje rostoucím požadavkům a které jsou nahrazovány moderními signálovými procesory s velmi dlouhým instrukčním slovem, jejichž architektura je založena na jiných principech. Cílem je tedy prozkoumat možnosti optimalizace uvedených struktur pro moderní architekturu signálových procesorů s velmi dlouhým instrukčním slovem, eventuálně pro paralelní systémy typu SIMD (Single Instruction Multiple Data).

Jak vyplývá z předchozí kapitoly, signálové procesory s pevnou řádovou čárkou dosahují stále výrazně vyššího výpočetního výkonu než procesory s pohyblivou řádovou čárkou. Proto je stále výhodné algoritmy optimalizovat pro realizaci v pevné řádové čárce. Zároveň z předchozí kapitoly plyne, že operace zaokrouhlení (ale i usekávání nebo přetečení) jsou nelineární operace a proto záleží na pořadí jejich provádění a optimalizace může ovlivnit citlivost algoritmů na přesnost kvantování vzorků nebo přesnost vyjádření koeficientů systému. Dalším cílem habilitační práce je proto analýza citlivosti struktur algoritmů na kvantovací jevy a vzájemné porovnání jednotlivých struktur.

V pedagogické oblasti je cílem habilitační práce především aktualizace uvedených témat ve výukových učebnicích, přednáškových materiálech a laboratorních úlohách předmětů věnovaných tematice číslicového zpracování signálů. Jedná se především o předměty *Číslicové filtry* a *Signálové procesory*, dále pak *Číslicové zpracování signálů*.

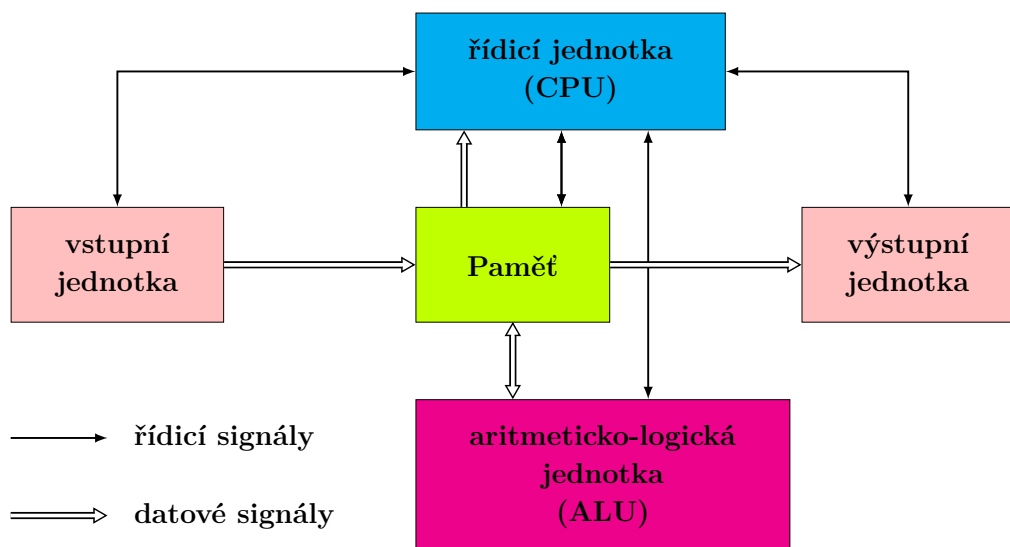
3 PŘEHLED ARCHITEKTUR PROCESORŮ

Signálové procesory prošly mnohaletým vývojem, během kterého se měnily výkonnostní požadavky i nároky na spotřebu, jednoduchost konstrukce, apod. Výsledkem vývoje je několik generací a řad signálových procesorů, jejichž architektury se podobají nebo jsou naprosto odlišné. Vlastnosti architektury signálového procesoru přitom ovlivňují, jaká struktura výpočetního algoritmu bude pro daný signálový procesor optimální z hlediska výpočetní náročnosti. Podrobná analýza jednotlivých architektur signálových procesorů a pochopení koncepce výpočetních algoritmů je proto základem pro jejich optimalizaci.

3.1 Von Neumannova architektura

Základem von Neumannovy architektury byly myšlenky, které vznikly při vývoji prvních počítačů (např. EDVAC (Electronic Discrete Variable Computer)) a formuloval je v padesátých letech minulého století matematik John von Neumann [12, 45]. Některé z jeho myšlenek jsou zde uvedeny:

- Počítač (mikroprocesor) se skládá ze čtyř základních bloků (obr. 3.1): paměti, řadiče, ALU (Arithmetic-Logic Unit) a ze vstupních a výstupních jednotek.
- Struktura počítače je nezávislá na typu řešení úlohy a je řízena programem uloženým v paměti.
- Instrukce a operandy jsou uloženy v téže paměti.
- Paměť je rozdělena do buněk stejné velikosti, jejichž pořadová čísla se používají jako adresy.
- Program je tvořen posloupností elementárních příkazů (instrukcí), v nichž zpravidla není obsažena hodnota operandu (uvádí se pouze jeho adresa), takže program se při změně dat nemění.



Obr. 3.1: Von Neumannova koncepce architektury číslicových počítačů.

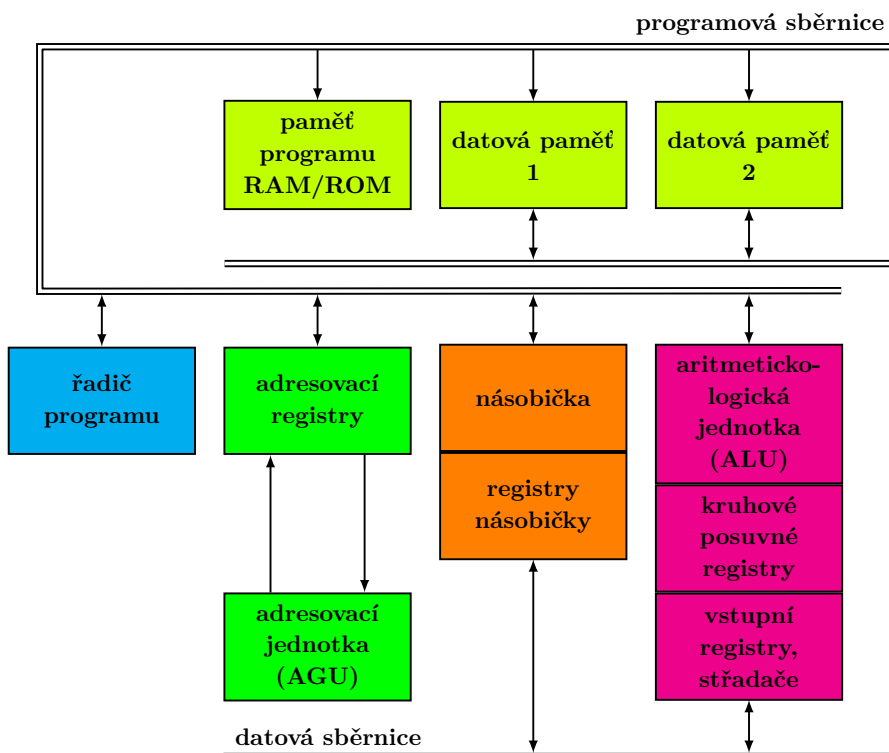
- Instrukce se provádějí jednotlivě v pořadí, v němž jsou zapsány do paměti.
- Změna provádění instrukcí se vyvolá instrukcí podmíněného a nepodmíněného skoku.
- Pro reprezentaci instrukcí a čísel (operandů, výsledků, adres apod.) se používají dvojkové signály a dvojková číselná soustava.

Tato koncepce ovlivnila vývoj číslicových počítačů a pak i mikroprocesorů až do osmdesátých let minulého století, kdy se objevil první signálový procesor s harvardskou architekturou.

3.2 Harvardská architektura

Architektura byla původně navržena Howardem Aikenem v třicátých letech minulého století na Harvardské univerzitě ve Spojených státech při vývoji reléového počítače Harvard Mark 1 (1944). Dále byla využita na Pensylvánské univerzitě pro elektronkový počítač ENIAC (Electronic Numerical Integrator and Calculator). Již tehdy byla moderní koncepcí, ale technické prostředky v té době neumožnily její realizaci, a proto byla zavržena a používána koncepce von Neumannova. Později asi po čtyřiceti letech dosáhla technologie výroby integrovaných obvodů takového stupně, že mohla být harvardská architektura realizována. Základní principy harvardské architektury (obr. 3.2) lze shrnout následovně:

- Paměť je rozdělena na paměť programu a jednu nebo dvě paměti dat tak, aby současně mohly dva operandy vstupovat do aritmeticko-logické jednotky ALU.
- Součin dvou operandů v jednom instrukčním cyklu provede hardwarová násobička a



Obr. 3.2: Harvardská architektura signálových procesorů.

výsledek součinu je přičten k obsahu střadače (např. operace typu MAC (Multiply and Accumulate) u signálových procesorů firmy Motorola).

- Pro zvýšení výpočetní výkonnosti se používá zřetězené zpracování instrukcí (*pipelining*).
- Pro současnou práci se dvěma operandy je zvýšen počet datových a adresových sběrnic. Přímý přístup do paměti je prováděn vícenásobným kanálem typu DMA (Direct Memory Access).
- Řízení jádra signálového procesoru je odděleno od řízení vstupních a výstupních jednotek. Architektura typu 1X a 2X znamená, že jedna instrukce je provedena v jednom nebo ve dvou hodinových cyklech (taktech).

Harvardská architektura je dnes běžně používána v signálových procesorech zvláště pro méně výpočetně náročné aplikace, ale s potřebou nízkého příkonu. Používají se v telekomunikacích, při zpracování hudby, řeči a obrázků, v medicíně, v průmyslu pro řízení a měření apod. Typickým představitelem této architektury je rodina signálových procesorů firmy Motorola s označením DSP56002, DSP568xx nebo DSP563xx. [13, 36, 35, 40] Firma Texas Instruments má rodinu signálových procesorů s harvardskou architekturou označenou jako TMS320C50xx nebo TMS320C55xx [79, 83, 86] a firma Analog Devices používá pro rodinu signálových procesorů s harvardskou architekturou označení ADSP21xx [3, 5, 4]. I když u všech signálových procesorů těchto firem se jedná o harvardskou architekturu, tak se jejich uspořádání funkčních jednotek, pamětí a sběrnic mnohdy dosti liší. Při optimalizaci výpočetních algoritmů se musí k těmto odlišnostem přihlídnout.

3.2.1 Vybrané procesory s harvardskou architekturou

Signálové procesory řady DSP56xxx

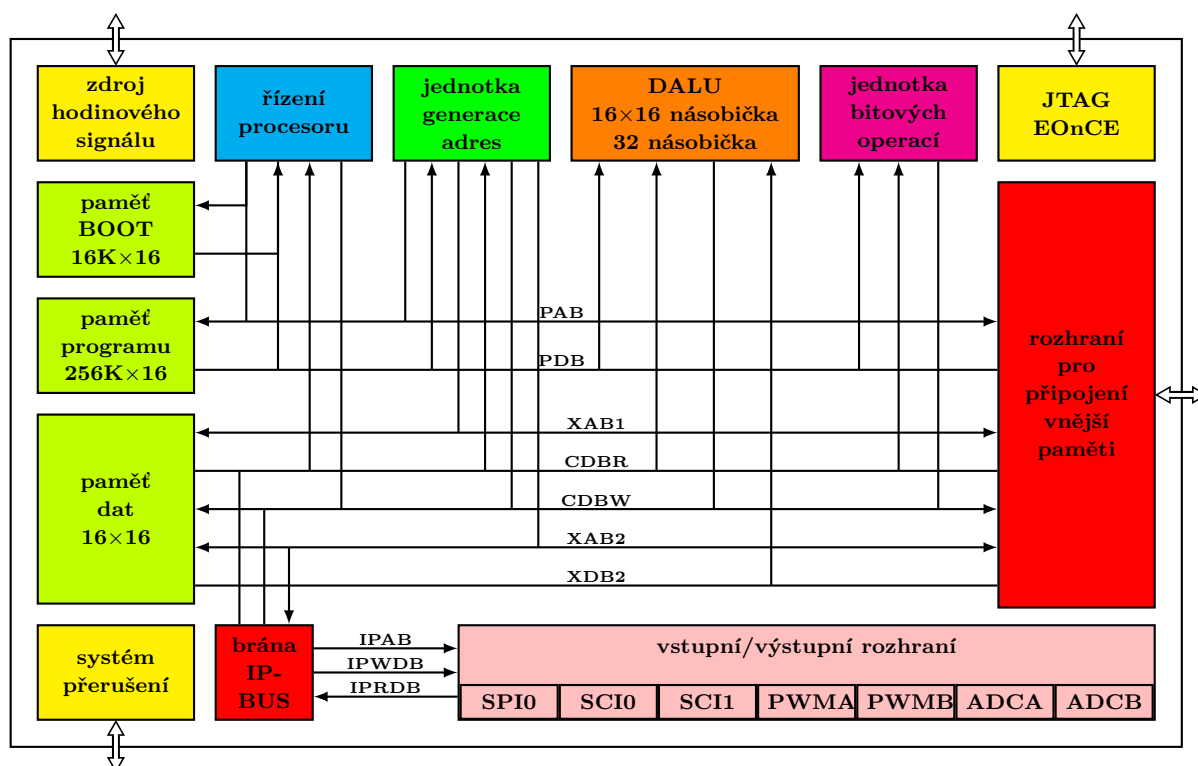
Mezi typické zástupce procesorů s harvardskou architekturou patří řada DSP56xxx [13]. Původně tuto řadu vyvíjela firma Motorola, později část vývoje signálových procesorů oddělila do dceřinné společnosti Freescale. V současné době udržování a vývoj nových procesorů zajišťuje společnost NXP.

Základní řada DSP560xx již splňovala všechny podmínky harvardské architektury. Dokonce realizovala tzv. dvojitou harvardskou architekturu, kdy paměť dat je zdvojená. Lze tak v jednom okamžiku provádět dva přenosy z paměti: jeden z paměti dat označené jako X: (např. vzorek signálu) a druhé označené jako Y: (např. vzorek impulsní charakteristiky). Velkou výhodou byly 24bitové datové registry i ALU, takže procesor umožňoval zpracovávat signály s větším dynamickým rozsahem než 16bitové signálové procesory běžné v té době. Naopak nevýhodou bylo použití architektury 2X, kdy instrukce byly zpracovány během dvou hodinových cyklů.

Na úspěchy řady navázala vylepšená řada DSP563xx [36, 39, 34, 37]. Podstatným rozdílem byl přechod na architekturu 1X, kdy většina instrukcí je vykonána během jediného hodinového cyklu. Ostatní vlastnosti (24bitová aritmetika, zdvojení datové paměti) zůstaly zachovány. Díky velké dynamice aritmetiky a relativně velkému výpočetnímu výkonu je řada často využívána v profesionálních systémech pro zpracování hudby (např. ProTools, Chameleon, apod.).

Nejnovější řada DSP568xxx se od předchozích významně liší [35, 40, 38]. Při jejím návrhu byly uvažovány výpočetně méně náročné průmyslové aplikace. Hlavním rozdílem je zkrácení datového slova na 16 bitů a doplnění některých principů typických spíše pro mikrokontroléry. Procesory této řady jsou také někdy označovány jako DSC (Digital Signal Controllers) místo DSP (Digital Signal Processing).

Jako příklad architektury je na obr. 3.3 zobrazeno blokové schéma signálového procesoru DSC56F8367. Je patrné oddělení paměti programu a paměti dat. Oproti původní řadě (DSP560xx, DSP563xx) je paměť dat jen jedna, nicméně je rozdělena na bloky, ke kterým lze přistupovat nezávisle. Paměť s podobnou organizací je označována jako DARAM (Dual Access RAM) a podobný přístup používají i procesory Texas Instruments [80]. Ke každému bloku paměti dat jsou multiplexovány dvě 24bitové adresové (XAB1, XAB2) sběrnice a tři datové sběrnice (32bitová CDBR a 16bitová XDB2 pouze pro čtení a 32bitová CDBW pouze pro zápis). Přístup do paměti je řízen jednotkou generace adres AGU (Address Generator Unit) nezávislou na datové ALU, takže současně s prováděním aritmetické instrukce lze číst až dvě hodnoty z paměti dat nebo jednu hodnotu do paměti dat zapisovat. Čtení z paměti programu je prováděno pomocí 21bitové adresové sběrnice PAB a 16bitové datové PDB. Vně procesoru vychází pouze jediná adresová a jediná datová sběrnice, takže vnitřní sběrnice musí být přepínány. Čtení z vnější paměti tak může výrazně zpomalit provádění programu. V blokovém schématu jsou i další bloky zajišťující řízení, systém přerušení, sériové rozhraní pro komunikaci s okolím, apod.

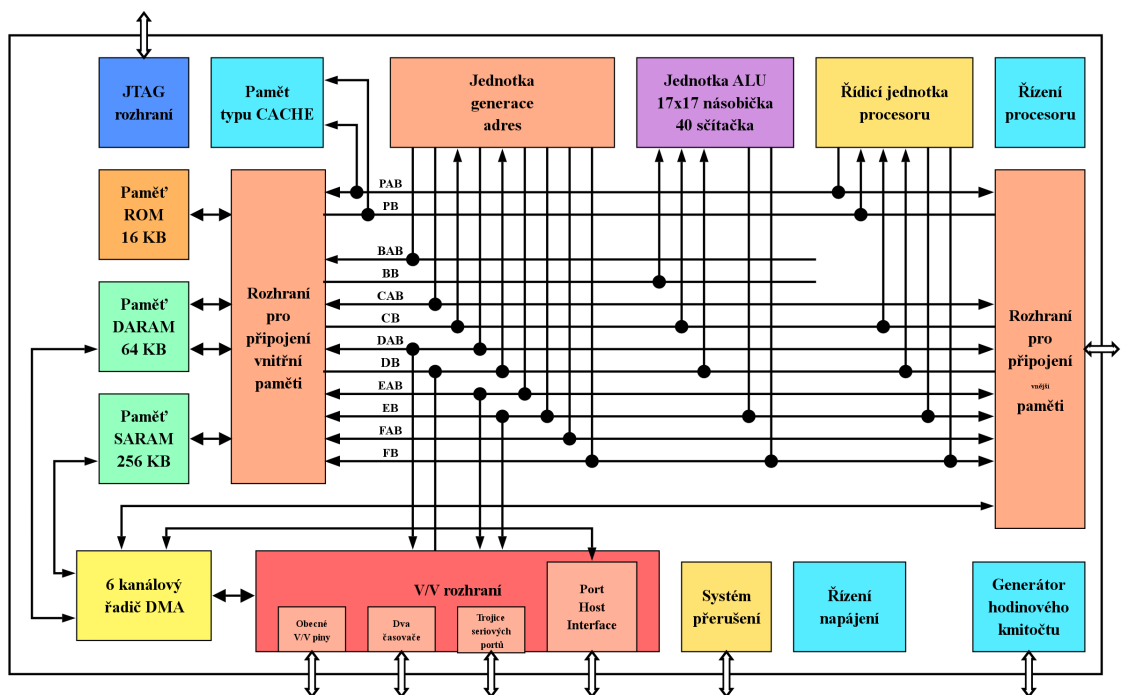


Obr. 3.3: Blokové schéma signálového procesoru DSC56F8367.

Signálové procesory řady TMS320C5xxx

Firma Texas Instruments vyrábí signálové procesory s harvardskou architekturou pod označením TMS320C5xxx [79, 83]. Mezi typické zástupce této řady patří signálový procesor TMS320C5510, jehož blokové schéma je na obr. 3.4 [86]. Paměť je rozdělena nikoliv na paměť programu a dat, ale podle možnosti přístupu do paměti. Nicméně program je většinou uložen v paměti označené jako SARAM (Single Access RAM), která umožňuje pouze jeden současný přístup. Naproti tomu data je vhodné ukládat do paměti DARAM, která je rozdělena na bloky velikosti 8 KB. Ke každému bloku lze přes multiplexor připojit jednu z adresových a datových sběrnic. Lze tak provádět současně více přístupů, pokud jsou to přístupy k různým blokům paměti DARAM.

Pro čtení instrukcí je vyhrazena 24bitová adresová sběrnice PAB a 32bitová datová sběrnice PB. Čtení dat je prováděno trojicí 23bitových adresových sběrnic BAB, CAB, DAB a 16bitových datových sběrnic BB, CB, DB. Je tak možné provádět čtení až tří hodnot současně. Zápis probíhá pomocí 23bitových adresových sběrnic EAB, FAB a 16bitových datových sběrnic EB, FB. Je tak možné současně provádět až dva zápisy do paměti. Vně procesoru vystupuje pouze jediná adresová a datová sběrnice, takže vnitřní sběrnice musí být přepínány. Čtení z vnější paměti tak může výrazně snížit výpočetní výkon. V blokovém schématu na obr. 3.4 jsou i další bloky zajišťující řízení, komunikaci s okolím, apod.

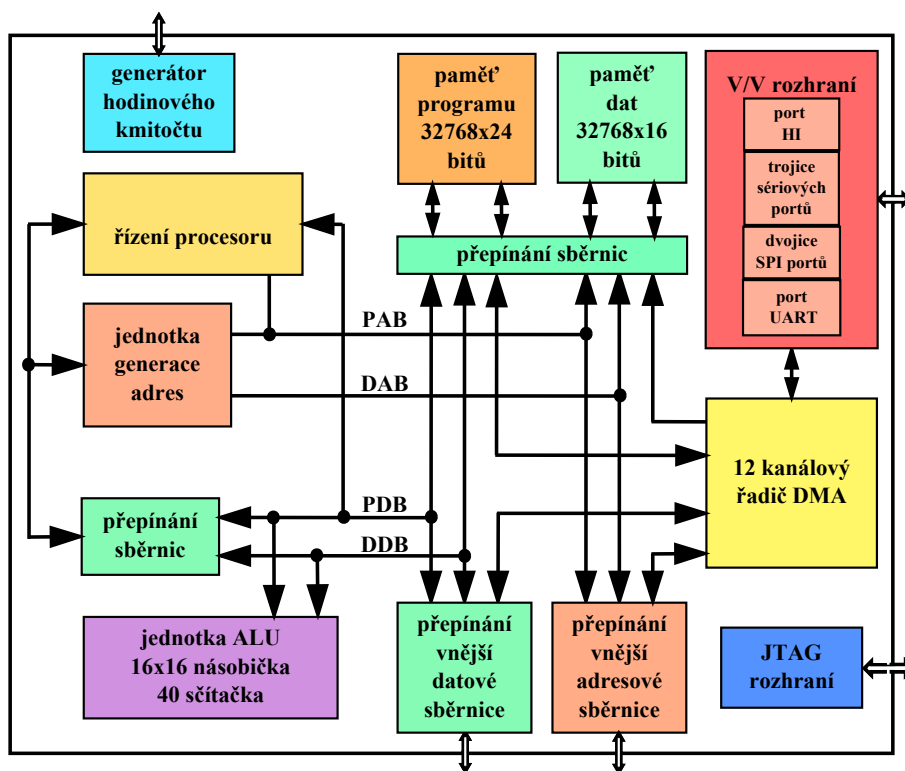


Obr. 3.4: Blokové schéma signálového procesoru TMS320C5510.

Signálové procesory řady ADSP21xx

Posledním významným výrobcem signálových procesorů je firma Analog Devices, která harvardskou architekturu používá u řady ADSP21xx [3, 5]. Jako zástupce této řady je na

obr. 3.5 zobrazeno blokové schéma signálového procesoru ADSP2191 [6]. Opět je patrné oddělení paměti programu a dat typické pro harvardskou architekturu. Přístup do paměti je realizován dvojicí 24bitových adresových sběrnic PAB, DAB a datovou 24bitovou sběrnicí PDB, resp. 16bitovou DDB. Paměť je rozdělena na 4 bloky velikosti 32 KB, které umožňují současný přístup. To je umožněno i oddělením datové ALU a jednotky generace adres AGU. Vně procesoru vystupuje jediná adresová a jediná datová sběrnice, takže vnitřní sběrnice musí být přepínány. Použití vnější paměti tak může vést ke snížení výpočetního výkonu. V blokovém schématu jsou i další jednotky pro systém přerušování, sériové rozhraní, apod.



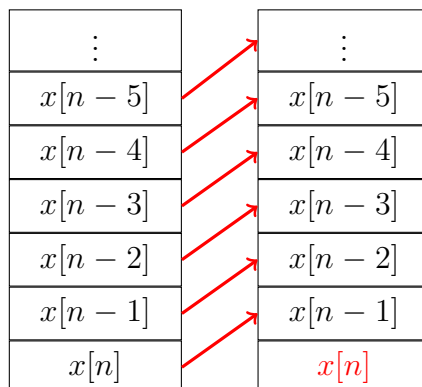
Obr. 3.5: Blokové schéma signálového procesoru ADSP2191.

3.2.2 Jednotka generace adres

Kromě oddělení paměti programu a dat je společným rysem všech signálových procesorů s harvardskou architekturou oddělená jednotka generace adres AGU s nezávislou ALU, která podporuje několik způsobů nepřímé adresace. Jedním z módů adresace, který podporuje většina signálových procesorů je *modulo adresování* používané při realizaci nejrůznějších zpoždění a realizaci vyrovnávacích pamětí.

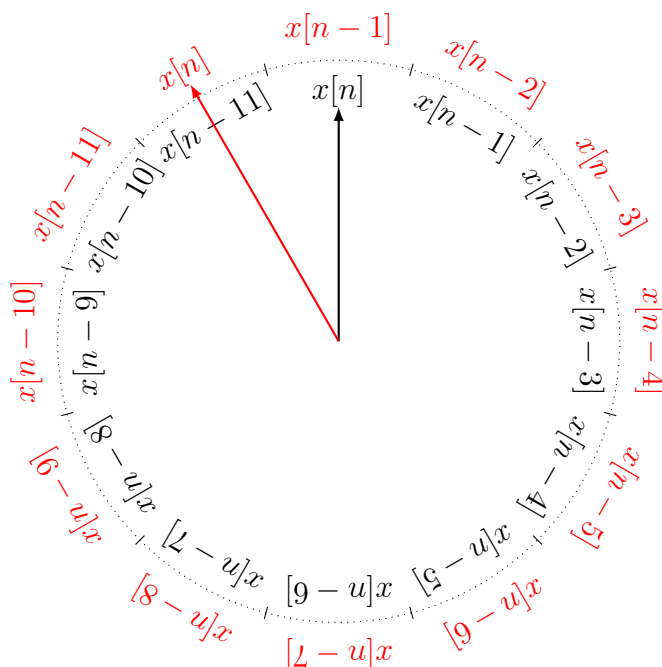
Na obr. 3.6 je naznačena realizace zpoždění pomocí běžného datového pole v paměti. Vzorky signálu jsou v paměti umístěny za sebou. Pokud přijde nový vzorek $x[n]$ signálu, je nutné pro něj nejdříve vytvořit místo překopírováním všech vzorků o adresu výše. Dojde tak k logickému zpoždění vzorků, vzorek $x[n]$ přejde na $x[n-1]$, $x[n-1] \rightarrow x[n-2]$, atd. Na uvolněné první místo v paměti je uložen nově příchozí vzorek $x[n]$. Poslední vzorek

$x[n - N + 1]$ kopírován nebude a pouze se přepíše vzorkem $x[n - N + 2]$. Tento způsob realizace je značně neefektivní, protože vyžaduje přesun téměř všech vzorků v paměti a tedy příliš mnoho paměťových operací, které jsou navíc často velmi pomalé.



Obr. 3.6: Zobrazení lineární vyrovnávací paměti realizované pomocí běžného pole jazyka C.

Výhodnější je využití tzv. kruhové (nebo též modulo) paměti, kdy jsou buňky logicky uspořádány do kruhu a je zaznamenáno umístění začátku paměti, tj. adresa aktuálního vzorku $x[n]$ (viz obr. 3.7). V okamžiku příchodu nového vzorku $x[n]$ se pozice začátku kruhové paměti posune o 1. Tím dojde k logickému zpoždění všech vzorků $x[n] \rightarrow x[n-1]$, $x[n-1] \rightarrow x[n-2]$, atd. Začátek kruhové paměti ukazuje na nejstarší vzorek $x[n - N + 1]$, který již není ke zpracování potřeba. Je tak možné ho přepsat novým vzorkem $x[n]$ a začátek kruhové paměti opět ukazuje na nejnovější vzorek. Čtení probíhá obdobně.



Obr. 3.7: Zobrazení principu kruhového zásobníku pro realizaci zpoždění signálu.

Realizace kruhové paměti je možná několika způsoby. Nejpřímochařejší je využití modulu operace % (zbytek po celočíselném dělení), podle které je kruhová paměť označována jako modulo. Uvažujme následující příklad:

```
#define LENGTH (5)

int buffer[LENGTH] = {
    0, 2, 6, 8, 12
}

.
.
int x, ind = 0;

while( 1){
    x = buffer[ind];
    ind = (ind+1) % LENGTH;
}
```

Výsledná hodnota proměnné x bude v jednotlivých iteracích rovna:

iterace	0	1	2	3	4	5	6	7	8	9	10	11	12
x	0	2	6	8	12	0	2	6	8	12	0	2	6

Modulo operace je však velmi výpočetně náročná. Navíc na některých systémech se může lišit implementace modulu operace pro záporná čísla.

Pokud je velikost kruhové paměti omezena na celočíselnou mocninu hodnoty 2^K , $K \in \mathbb{N}$, pak lze kruhovou paměť realizovat mnohem efektivněji. Modulo operaci lze nahradit bitovým součinem s maskou, která má pouze K spodních bitů nastavených na hodnotu 1, ostatní jsou nulové.

Uvažujme, že velikost paměti bude $4 = 2^2$. Zdrojový kód modulu adresování by pak měl tvar:

```
#define LENGTH (4)

int buffer[LENGTH] = {
    0, 2, 6, 8
}

.
.
int ind = 0;
int x;

while( 1){
    x = buffer[ind];
    ind = (ind+1) & (LENGTH-1);
}
```

Výsledkem zpracování by pak byly hodnoty proměnné x :

iterace	0	1	2	3	4	5	6
	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$
AND	$(011)_2$	$(011)_2$	$(011)_2$	$(011)_2$	$(011)_2$	$(011)_2$	$(011)_2$
ind	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(000)_2$	$(001)_2$	$(010)_2$
x	0	2	6	8	0	2	6

Omezení velikosti paměti na celočíselnou mocninu 2 většinou nevádí. Pokud požadavek na velikost paměti plynoucí z daného typu zpracování tuto podmínku nesplňuje, je možné bez ztráty funkčnosti velikost paměti zvolit jako nejbližší vyšší mocninu 2. Pouze se zvýší paměťové nároky.

Podobnou implementaci kruhové paměti často podporuje přímo hardware signálových procesorů. Pokud je nastaveno modulo adresování, pak se u adresy může měnit pouze K nejméně významných bitů. Ostatní bity zůstávají nezměněny. Tím je zaručeno, že adresa zůstává v rámci bloku délky 2^K .

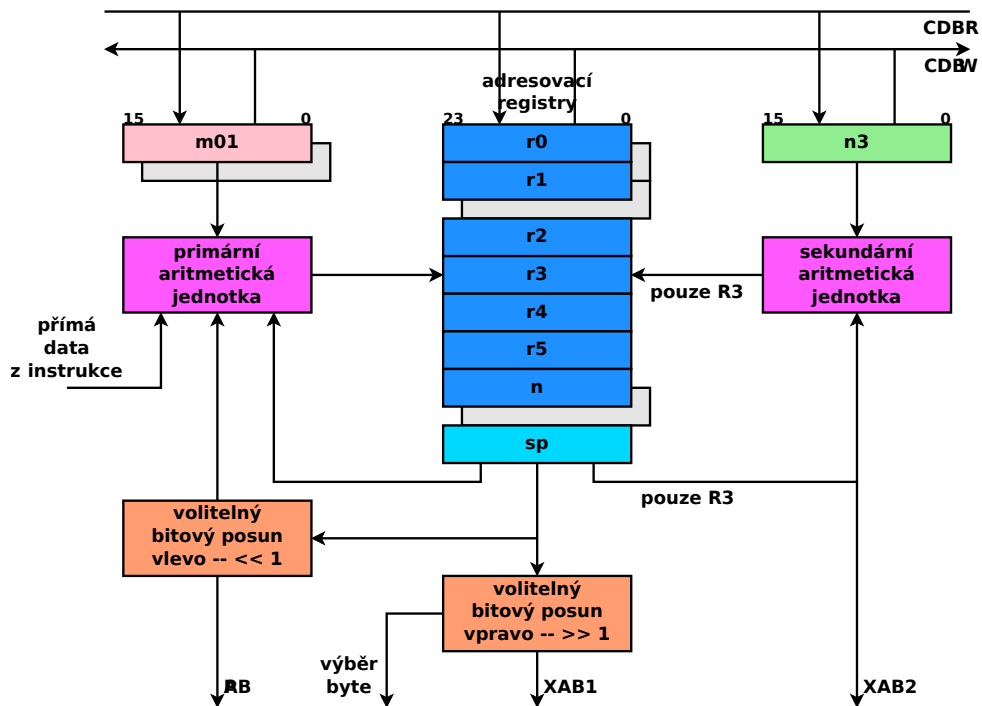
Signálové procesory řady DSP56xxx

Blokové schéma jednotky generace adres je na obr. 3.8 [40]. Jednotka obsahuje celkem šest 24bitových adresovacích registrů $r0$ až $r5$, tzv. registry posunu n (lze je použít i jako adresovací registr) a $n3$ a modifikační registr $m01$. Kromě nich obsahuje také speciální adresovací registr sp , který se využívá jako ukazatel zásobníku SP (Stack Pointer). Původní řada DSP563xx obsahovala 8 adresovacích registrů $r0$ až $r7$ a každý měl příslušný registr posunu $n0$ až $n7$ a modifikační registr $m0$ až $m7$. Registry $r0$, $r1$, n a $m01$ jsou zde zdvojeny. Druhá dvojice $shadow$ s registrů se automaticky používá během obsluhy přerušeni nebo lze její použití přepnout pomocí instrukce `swap shadows`. Podobně jako AGU původní řady jednotka obsahuje dvě ALU, přičemž druhá může pracovat pouze s registry $r3$ a $n3$. A také na adresovou sběrnici $XAB2$ lze generovat adresu pouze z registru $r3$. To je další omezení, neboť u původní řady mohla jedna ALU pracovat s polovinou registrů $r0$ až $r3$ (a odpovídající $n0$ až $n3$, resp. $m0$ až $m3$), zatímco druhou polovinu $r4$ až $r7$ (a odpovídající $n4$ až $n7$, resp. $m3$ až $m7$) mohla používat druhá ALU.

Adresace je prováděna po 16bitových slovech, což je i délka datových registrů, pomocí skupiny instrukcí `move.w` (Move Word). Pokud je potřeba adresovat jeden byte, je nutné použít speciální instrukci `move.bp` (Move Byte Pointer) pro přenos 8bitových hodnot. Adresa je přitom posunuta vpravo o jeden bit (viz volitelný bitový posun na obr. 3.8), z takto získané adresy se přečte celé 16bitové slovo a podle nejméně významného bitu původní adresy se použije buď dolní nebo horní bajt. Přenos 32bitových hodnot provádí instrukce `move.l` (Move Long Word).

Možnosti nepřímé adresace zůstaly přibližně zachovány jako u předešlých řad a jsou shrnuty v tab. 3.1. Základem je nepřímá adresace pomocí adresovacího registru, kdy se hodnota adresovacího registru použije jako adresa do paměti. Např.

```
move.l    #01234a , r2
move.w    x : (r2) , r3
```



Obr. 3.8: Blokové schéma jednotky generace adres procesoru DSC56F8367.

První instrukce naplní registr r_2 přímou hodnotou (uvozeno znakem #) $(01234a)_{16}$ v hexadecimální soustavě (uvozeno znakem \$). Druhá instrukce pak hodnotu z registru použije jako adresu a do registru r_3 umístí 16bitovou hodnotu z adresy $0x01234a$.

Adresa může být současně s přenosem modifikována jedním ze způsobů z tab. 3.1. Např. instrukce

```
move.l  # $01234a, r2
move.l  x:(r2)-, r3
```

naplní registr r_2 přímo zadanou hodnotou $(01234a)_{16}$. Druhá instrukce použije obsah registru r_2 jako adresu a do registru r_3 vloží 32bitovou hodnotu z paměti na adrese $0x01234a$. Po přenosu se hodnota adresovacího registru r_2 sníží o 2, protože došlo k přenosu dvou 16bitových slov. Registr tak obsahuje adresu $(012348)_{16}$ 32bitového slova níže před přečtenou adresou. Podobné je to v případě použití registru posunu, kdy instrukce

```
move.l  # $20, n
move.l  # $01234a, r2
move.w  x:(r2)+n, r3
```

vloží do registru r_3 16bitovou hodnotu z adresy $0x01234a$ a poté k registru r_2 přičte hodnotu registru posunu n . Registr r_2 tak bude mít výslednou hodnotu $0x01236a$.

Oproti tomu při indexaci se hodnota adresovacího registru nemění, změněná hodnota se použije pouze pro adresaci. Např.

```
move.l  # $01234a, sp
move.w  x:(sp-#2), x0
```

Tab. 3.1: Možnosti nepřímého adresování datové paměti u procesoru DSC56F8367.

beze změny	$x:(rn)$	
se změnou	post inkrementace	post dekrementace
o 1	$x:(rn)+$	$x:(rn)-$
o n	$x:(rn)+n$	
pouze r3	$x:(r3)+n3$	
indexování	pouze registry rn	pouze sp
o n	$x:(rn+n)$	$x:(sp+n)$
o 3bitovou hodnotu	$x:(rn+x)$	$x:(sp-x)$
o 6bitovou hodnotu		$x:(sp-xx)$
o 16bitovou hodnotu	$x:(rn+xxxx)$	$x:(sp+xxxx)$
o 24bitovou hodnotu	$x:(rn+xxxxxx)$	$x:(sp+xxxxxx)$

nejprve naplní registr `sp` přímou hodnotu `0x01234a`. Poté hodnotu z registru `sp` vezme, odečte přímo zadaná data 2 a do datového registru `x0` vloží 16bitovou hodnotu z paměti na adrese `0x012348`. Hodnota registru `sp` zůstane nezměněna. Je to typický příklad adresování lokálních proměnných na zásobníku, kdy registr `sp` obsahuje adresu vrcholu zásobníku a k lokálním proměnným se přistupuje pouze relativními adresami k vrcholu zásobníku.

Jednotka generace adres podporuje i modulo adresování, ale pouze pro registry `r0` a `r1`. Do registru `m01` se vloží hodnota délky kruhové paměti $N - 1$, přičemž délka bloku N se nemusí rovnat celočíselné mocnině 2. Počáteční adresa kruhové paměti však musí být násobkem 2^K , kde K je nejmenší celé číslo splňující $2^K \geq N, K \in \mathbb{N}$. Adresa začátku kruhové paměti tak musí být na násobku celočíselné mocniny. Jinými slovy adresa začátku má K nejméně významných bitů nulových. Pokud je nejvýznamnější bit registru `m01` vynulován, modulo adresování bude použito pouze pro registr `r0`. Pokud je nejvýznamnější bit registru `m01` nastaven, modulo adresování platí pro registr `r0` i `r1`. Hodnota $(FFFF)_{16}$ odpovídá vypnutí modulo adresování a adresování bude probíhat jako standardní lineární adresování.

Signálové procesory řady TMS320C5xxx

Podobné možnosti nabízí jednotka generace adres u procesoru TMS320C5510. Jako adresovací lze použít 8 registrů `AR0`, `AR1`, až `AR7` a registr `CDP`. Registry jsou 16bitové, zatímco adresa je 23bitová. Před adresací je tak nutné 7 nejvýznamnějších bitů adresy vložit do příslušných stránkovacích registrů `AR0H`, `AR1H` až `AR7H` a `CDPH`. [86]

Možnosti nepřímého adresování uvedené v tab. 3.2 jsou velice podobné možnostem procesoru DSC56F8367. Jako registry posunu lze použít registry `T0` a `T1`, resp. registr `AR0`. Dále přibyla možnost změny před adresováním, např.

```
mov  *-AR1, AC0
mov  AC1, **AR2(#10)
```

Tab. 3.2: Možnosti nepřímého adresování datové paměti u procesoru TMS320C5510.

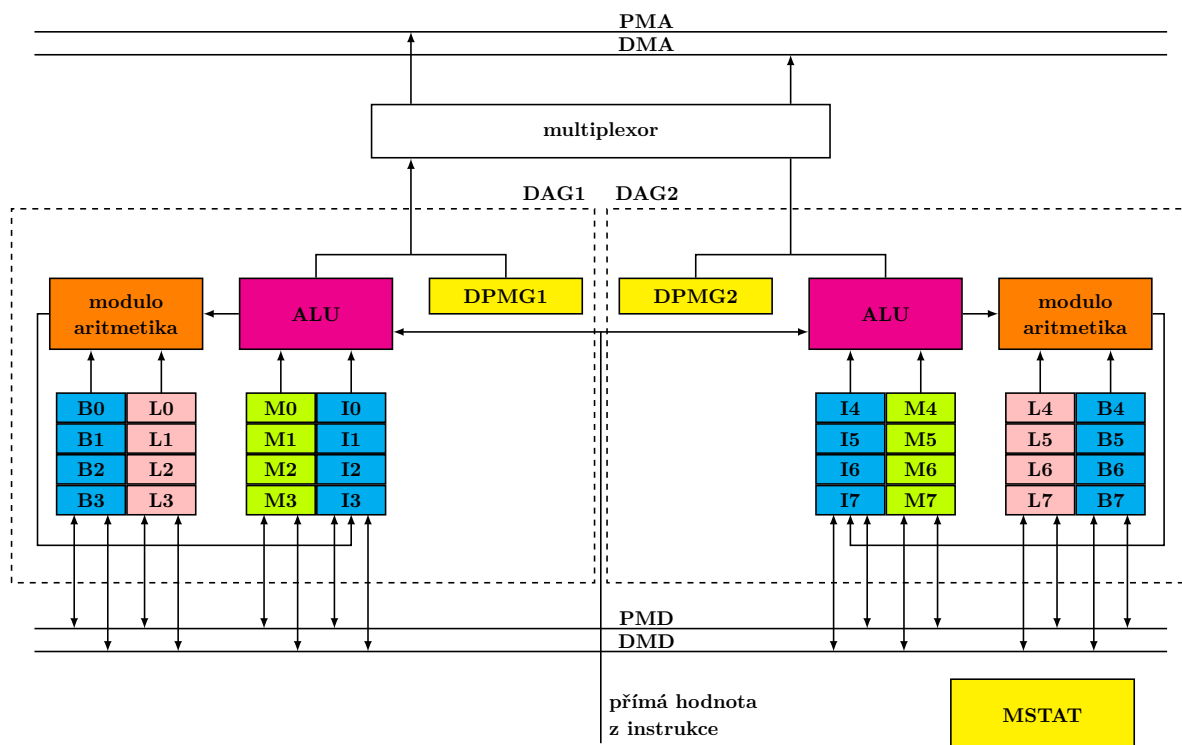
beze změny	*ARn	
se změnou	post inkrementace	post dekrementace
o 1	*ARn+	*ARn-
o T0	*(ARn + T0)	*(ARn - T0)
o T1	*(ARn + T1)	*(ARn - T1)
o ARO	*(ARn + ARO)	*(ARn - ARO)
o TOB bitově reverzní	*(ARn + TOB)	*(ARn - TOB)
o AROB bitově reverzní	*(ARn + AROB)	*(ARn - AROB)
se změnou	pre inkrementace	pre dekrementace
o 1	*+ARn	*-ARn
o 16bitovou hodnotu	*+ARn (#xxxx)	
indexování		
o T0	*ARn (T0)	
o T1	*ARn (T1)	
o ARO	*ARn (ARO)	
o 16bitovou hodnotu	*ARn (#xxxx)	

První instrukce nejprve sníží hodnotu adresovacího registru AR1 o 1. Poté hodnotu registru ARO použije jako adresu a z paměti přenesení hodnotu do registru AC0. Druhá instrukce nejprve zvýší hodnotu registru AR2 o přímo zadanou hodnotu 10 a na takto získanou adresu uloží obsah registru AC1.

Jednotka podporuje i modulo adresování. Délka bloku N nemusí být celočíselnou mocninou 2, ale počáteční adresa musí být násobkem $2^K > N$, $K \in \mathbb{N}$, podobně jako u řady DSC56F8367. Pro registry AR0 až AR3 se délka bloku nastavuje společně v registru BK03. Počáteční adresa je pak nastavována vždy pro dvojice registrů AR0 a AR1 v registru BSA01, AR2 a AR3 v registru BSA23. Podobně v registru BK47 je nastavena délka bloku pro registry AR4 až AR7, registry AR4 a AR5 přitom mají společnou počáteční adresu BSA45, zatímco počáteční adresa pro registry AR6 a AR7 je uložena v registru BSA67. Povolit nebo vypnout modulo adresování je pak možné pro jednotlivé registry individuálně vynulováním (lineární) nebo nastavením (modulo) příslušného bitu ARnLC v řídicím registru ST2_55.

Signálové procesory řady ADSP21xx

Jednotka generace adres u signálového procesoru ADSP2191 (viz obr. 3.9) má strukturu ještě více podobnou procesorům řady DSP56xxx a podobné jsou i možnosti adresace. Jako adresovací lze použít 16bitové registry I0 až I7, rozdělené do dvou skupin. Adresa je 24 bitová a horních 8 bitů adresy je uloženo ve stránkovacích registrech DMPG1 a DMPG2 společných vždy pro čtveřici adresovacích registrů. Registry M0 až M7 slouží jako registry posunu podobně jako registr n3 u DSC56F8367. Možnosti modifikace během adresování jsou shrnuty v tab. 3.3. [5]



Obr. 3.9: Blokové schéma jednotky generace adres u signálového procesoru ADSP2191.

Jednotka generace adres podporuje i modulo adresování. Délka bloku opět nemusí být mocnina dvou a je nastavena v příslušném modifikačním registru Ln. Pokud $L_n = 0$, pak je modulo adresování vypnuto a paměť je adresována standardně lineárně. Počáteční adresa bloku je uložena v registru Bn. Modulo adresování je možné realizovat pouze při post modifikaci registru. Při adresování pomocí indexace nefunguje. Modulo adresování platí také v rámci jedné stránky paměti nastavené registrem DMPGn, pokud při modifikaci adresy dojde k překročení hranice stránky, nemusí fungovat správně.

Tab. 3.3: Možnosti nepřímého adresování paměti u procesoru ADSP2191.

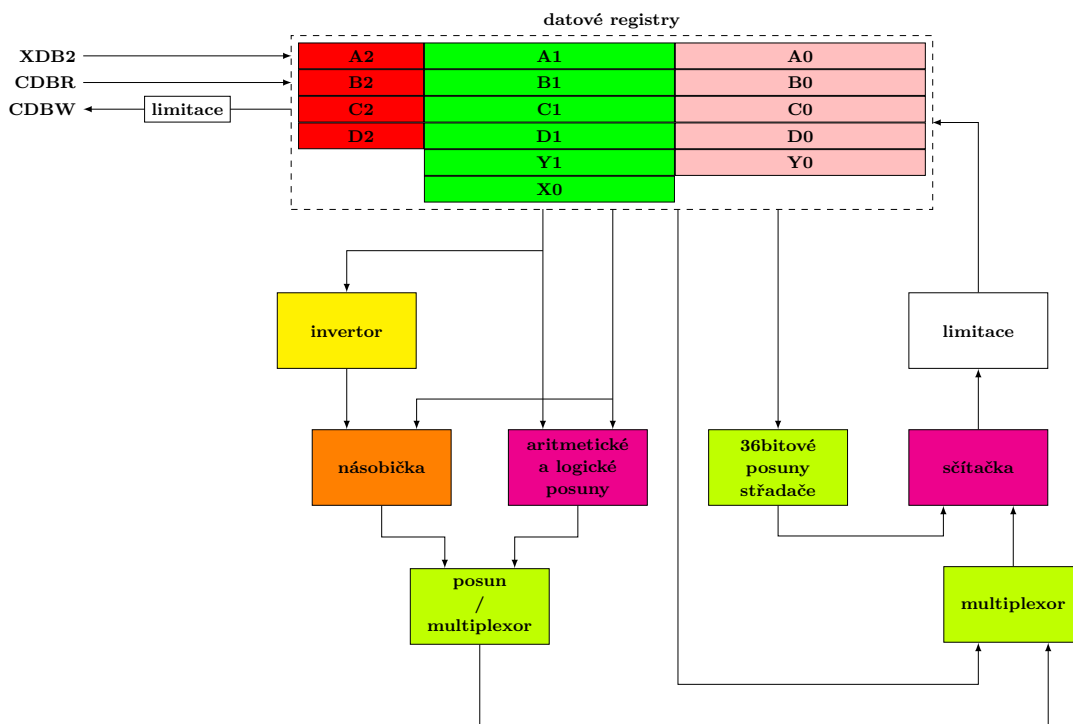
adresace pomocí	DMA, DMD	PMA, PMD
beze změny	$DM(In + 0)$	$PM(In + 0)$
se změnou po		
o Mn	$DM(In += Mn)$	$PM(In += Mn)$
o 8bitovou konstantu	$DM(In += xx)$	$PM(In += xx)$
indexování		
o Mn	$DM(In + Mn)$	$PM(In + Mn)$
o 8bitovou konstantu	$DM(In + xx)$	$PM(In + xx)$

3.2.3 Aritmeticko logická jednotka

Dalším společným znakem signálových procesorů s harvardskou architekturou je, že datová aritmeticko logická jednotka (DALU (Data Arithmetic Logic Unit)), která je optimalizována pro operaci násobení s akumulací.

Signálové procesory řady DSP56xxx

Blokové schéma aritmeticko logické jednotky procesoru DSC56F8367 je na obr. 3.10 [33]. Obsahuje celkem jedenáct 16bitových registrů A0, A1, B0, B1, C0, C1, D0, D1, Y0, Y1, X0. Ty lze používat samostatně nebo složit do registrových párů A10, B10, C10, D10 a Y s dvojnásobnou délkou 32 bitů. Tím lze zvýšit přesnost vyjádřených čísel z formátu Q0.15 na Q0.31. Registry A2, B2, C2 a D2 jsou pouze 4 bitové. Opět je lze použít samostatně. Častěji se však používají pro zvýšení rozsahu registrových párů a krátkodobé ošetření možného přetečení. Vzniknou tak 36bitové registry A, B, C, D vždy složené z odpovídajících částí A2, A1, A0. Formát takto sestavených střadačů je pak Q4.31. Příklady formátů a orientační rozsahy hodnot jsou uvedeny v tab. 3.4.



Obr. 3.10: Blokové schéma aritmetické logické jednotky procesoru DSC56F8367.

Registry X0, Y0, Y1 jsou primárně určeny pro ukládání vstupních operandů. Naopak registry A, B, C a D nebo jejich části jsou používány jako střadače pro ukládání výsledků operací. Jednotka obsahuje samostatnou plně čtyřkvadrantovou násobičku, která umožňuje provádět násobení celých nebo zlomkových čísel:

```
MPY      Y0, X0, A    ; násobení zlomkových čísel Y0 X0 do A
IMPY.L  Y0, X0, A    ; násobení celých čísel Y0 X0 do A
```

Tab. 3.4: Příklady formátů a rozsahů hodnot registrů DALU procesoru DSC56F8367.

registr	zlomkový		celočíslný	
	formát	rozsah	formát	rozsah
A0, A1, B0, B1, C0, B1, C0, B1, Y0, Y1, X0	Q0.15	$\langle -1; 1 \rangle$	Q15.0	$\langle -32768; 32767 \rangle$
A10, B10, C10, D10, Y	Q0.31	$\langle -1; 1 \rangle$	Q31.0	$\approx (-2,147 \cdot 10^9; 2,147 \cdot 10^9)$
A, B, C, D	Q4.31	$\langle -16; 16 \rangle$	Q35.0	$\approx (-34,36 \cdot 10^9; 34,36 \cdot 10^9)$

V obou případech jsou vynásobeny dvě 16bitové hodnoty registrů Y0 a X0 a 32bitový výsledek je uložen do střadače A. V případě násobení zlomkových čísel MPY je před uložením proveden bitový posun vlevo o jeden bit, aby byla správně umístěna řádová čárka (viz část 1.3). Díky invertoru u jednoho vstupu násobičky je možné změnit znaménko prvního operandu. Je to možné pouze u vybraných instrukcí násobení a s konkrétním vstupním registrem. Omezení není způsobeno ani tak hardwarově, jako spíš problémy kódování instrukcí do strojového kódu procesoru (viz část 3.2.5). Např.

```
MPY    -C1, Y1, A    ; násobení zlomkových čísel -C1 Y1 do A
MAC    -C1, Y0, B    ; násobení s akumulací -C1 Y0 + B do B
```

Druhá z předchozích instrukcí provádí násobení s akumulací, tzn. že výsledek součinu neukládá do střadače, ale k hodnotě ve střadači ho přičte. To je umožněno tím, že výstup z násobičky nevstupuje přímo do banky střadačů, ale může být přes multiplexor přiveden jako jeden ze vstupů sčítačky. Druhým vstupem je pak v uvedeném případě střadač B a hodnota součinu je k hodnotě střadače přičtena. Instrukce MAC tak provádí dvě operace

```
MPY    -C1, Y0, A    ; násobení zlomkových čísel -C1 Y0 do A
ADD    A, B          ; přičtení A + B do B
```

Jak je patrné z předchozího příkladu, sčítačka může pracovat i samostatně. S vybranými instrukcemi a vstupními operandy je možné jeden vstup i invertovat přímo během provádění instrukce

```
ADD    A, B          ; přičtení A + B do B
ADD    -A, B         ; odečtení B - A do B
```

Výstup sčítačky vstupuje do samostatné jednotky limitace. Výsledek operace je tak možné automaticky zaokrouhlit

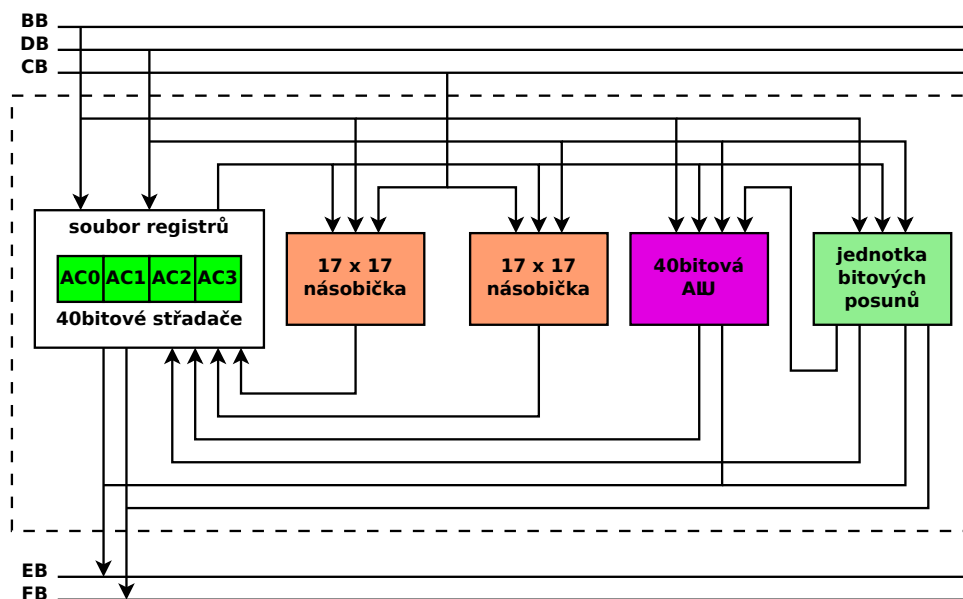
```
MACR   -Y1, X0, A    ; násobení zlomkových čísel -Y1 X0
                    ; přičtení hodnoty střadače A
                    ; zaokrouhlení na 16 bitů ve Q0.15
                    ; uložení do střadače :
```

nebo provést automatickou saturaci podle nastavení konfiguračního bitu SA v registru operačního módu OMR (Operating Mode Register). Automatická saturace může být také prováděna při přenosu střadače do paměti přes datovou sběrnici CDBW. Naopak čtení

z paměti probíhá po dvojici sběrnic CDBR a XDB2, takže je možné provádět až dvě čtení paralelně, viz část 3.2.2.

Signálové procesory řady TMS320C5xxx

Podobnou strukturu má i DALU procesoru TMS320C5510 na obr. 3.11. Jako střadače lze použít 40bitové registry AC0, AC1, AC2, AC3 ve zlomkovém formátu Q8.31 nebo celočíselném Q39.0. Vstupní registry nejsou potřeba, neboť instrukce mohou používat přímo hodnoty čtené z paměti. Lze však použít pomocné 16bitové registry T0, T1, T2, T3. [86]



Obr. 3.11: Datová aritmeticko logická jednotka procesoru TMS320C5510.

Hlavní blokem jednotky je 40bitová ALU, která umožňuje provádět sčítání, odčítání, logické operace, saturaci a zaokrouhlení. Jako příklad lze uvést operaci

```
ADD    AC1, AC0      ; přičtení AC1 + AC0 do AC0
ADD    *AR3+, AC0    ; přičtení hodnoty z adresy AR3 + AC0 do AC0
```

Druhá instrukce provede čtení z paměti na adrese, která je uložena v registru AR3, tuto hodnotu připočte k obsahu střadače AC0 a výsledek uloží do střadače AC0, a současně hodnotu adresovacího registru AR3 zvýší o 1. Při operaci může dojít i k bitovému posunu jednoho z operandů

```
ADD    AC0<<T0, AC1  ; vezmi hodnotu AC0
                        ; proved' bitový posun o hodnotu T0
                        ; výsledek přičti k hodnotě AC1
```

Násobení zajišťují dvě jednotky násobení s akumulací MAC. Jednotky umožňují provádět plně čtyřdvadrantové 17bitové násobení a 40bitovou akumulaci výsledku. Jednotka je zdvojená, takže v jednom okamžiku je možné provádět dvě taková násobení. Podmínkou je, že jeden vstupní operand obou násobení je společný, např. instrukce

```
MAC *AR1+, *CDP+, AC0
::MAC *AR3+, *CDP+, AC1
```

Společný operand je v tomto případě hodnota v paměti na adrese z adresovacího registru CDP. Hodnota je násobena hodnotou na adrese AR1 a součin je připočten ke střadači AC0. Současně je hodnota z adresy CDP vynásobena hodnotou z adresy AR3 a součin je přičten ke střadači AC1. Všechny adresovací registry AR1, AR3, CDP budou rovněž zvýšeny o 1. Dvě násobení současně lze využít např. pokud stejným algoritmem zpracováváme dva různé signály (dva kanály) nebo při implementaci číslicových filtrů typu FIR (Finite Impulse Response) se symetrickou nebo antisymetrickou impulsní charakteristikou (viz část 4.1.2).

Dalším blokem je jednotka bitových posunů, která zajišťuje provádění bitových posunů.

Signálové procesory řady ADSP21xx

Podobnou strukturu DALU používá i řada ADSP21xx. Jednotka obsahuje čtyři 16bitové registry AX0, AX1, AY0, AY1, které slouží primárně pro vstupní operandy ALU, a dva 16bitové registry AR a AF, které slouží primárně pro uložení výstupu ALU. Dále čtyři 16bitové registry MX0, MX1, MY0, MY1, které slouží primárně pro vstupní operandy MAC a tři 16bitové registry MR0, MR1, MR2, které slouží primárně pro uložení výstupu MAC. Posledními registry jsou 16bitový registr SI sloužící primárně pro vstupní operand jednotky bitových posunů a 16bitové registry SR0, SR1, SR2 sloužící pro uložení výstupu posunu. Speciální registry SB a SE jsou sice 16bitové, ale často je využíváno pouze 5 nejméně významných bitů ve smyslu exponentu pro bitové posuny. Registry je možné skládat do registrových párů MX (MX1, MX0), MY (MY1, MY0), AX (AX1, AX0), AY (AY1, AY0) ve formátu Q0.31, nebo trojic MR (MR2, MR1, MR0), SR (SR2, SR1, SR0) ve formátu Q16.31. Registry lze v některých případech použít i k jiným než uvedeným účelům. [5]

Blok SHIFTER slouží pro provádění bitových posunů, normalizaci a/nebo stanovení exponentu. Např. instrukce

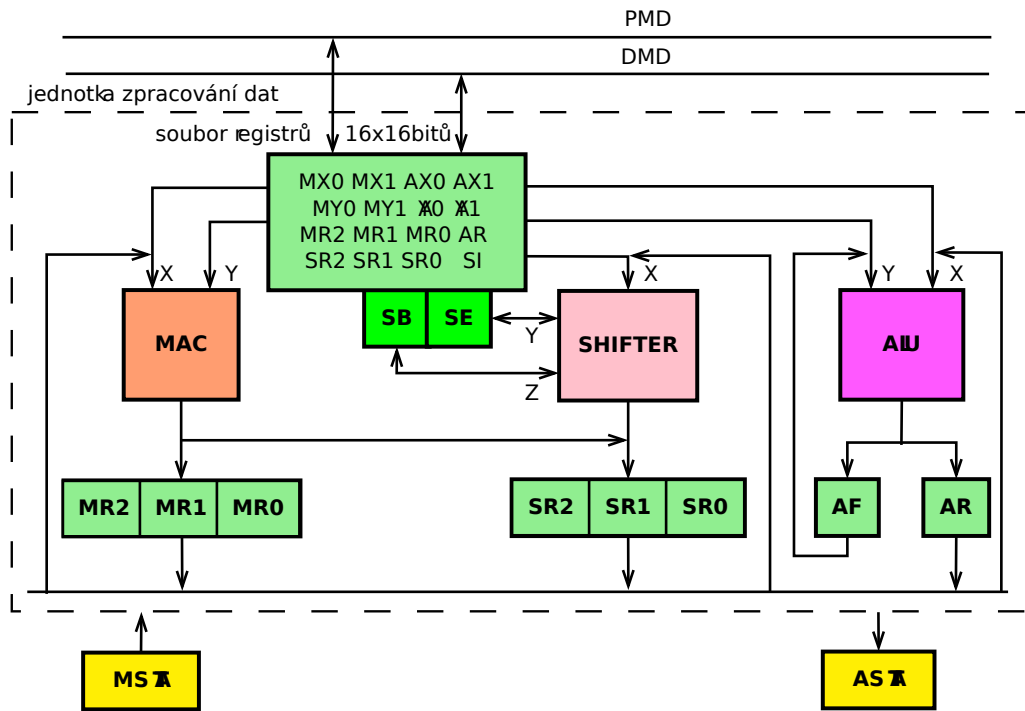
```
SR = ASHIFT SI
SR = SR OR LSHIFT AX0 BY -3
```

První instrukce provede aritmetický posun registru SI o hodnotu v registru SE a výsledek uloží do registru SR. Zatímco druhá instrukce provede logický posun registru AX0 vpravo o 3 bity (přímo zadaná hodnota -3) a výsledek bitově přičte k registru SR (operace OR).

Druhý blok MAC realizuje hardwarovou násobičku včetně akumulace. Např. instrukce

```
MR = MX0 * MY0 (SU)
MR = MR + MX0 * MY1 (SS)
```

První instrukce provede násobení registru MX0 jako číslo se znaménkem registrem MY0 jako číslo bez znaménka (SU – Signed Unsigned) a výsledek uloží do registru MR. Druhá instrukce vynásobí registr MX0 a MY1, oba jako číslo se znaménkem (SS – Signed Signed), a součin připočte k registru MR. V instrukci není rozlišeno, zda se provádí celočíselné nebo zlomkové násobení. Rozlišení se děje pomocí bitu M_MODE v řídicím registru MSTAT. Pokud je bit vynulován M_MODE = 0, pak se provádí celočíselné násobení ve formátu Q15.0. Pokud je bit nastaven M_MODE = 1, pak se provádí zlomkové násobení ve formátu Q0.15.



Obr. 3.12: Blokové schéma DALU rocesoru ADSP2191.

Ostatní operace (součet, rozdíl, logické operace) provádí blok ALU. Výsledek operace může být přes registr AF okamžitě použit jako vstup další ALU operace. Případně je výsledek uložen do registru AR, který může sloužit jako vstup všem blokům DALU. Např. instrukce

$$\begin{aligned} AR &= SR1 + AF \\ AF &= MR0 + SR0 + C \end{aligned}$$

První instrukce sečte hodnoty registrů SR1 a AF a výsledek uloží do registru AR. Druhá instrukce sečte hodnoty registrů MR0 a SR0, k výsledku připočte příznak CARRY a výsledek uloží do registru AF.

3.2.4 Paralelní přenosy a další operace

Z uvedeného vyplývá, že instrukce signálového procesoru mohou realizovat několik operací současně – typicky instrukce násobení a součtu. U většiny signálových procesorů lze však spolu s aritmetickými instrukcemi provádět současně přenosy mezi pamětí a registry, eventuálně provádět současně modifikaci adresovacích registrů apod.

Signálové procesory řady DSP56xxx

U procesorů řady DSP56xxx lze s některými vybranými aritmetickými instrukcemi provádět jeden nebo dva paralelní přenosy. Pokud je u procesoru DSC56F8367 prováděn jeden paralelní přenos, pak se může jednat o čtení z paměti nebo zápis do paměti některého z registrů A, B, C, A1, X0, Y0, B1, Y1. Pro adresaci je možné použít pouze nepřímou adresaci s modifikací (rn)+ nebo (rn)+n. Ostatní kombinace nejsou povoleny. [35]

Např. instrukce

```
mac  -c1, y0, a    x: (r0)+, y0
```

vynásobí registry `c1` a `y0`, součin odečte od hodnoty registru `a` a výsledek uloží do registru `a`. Současně registr `y0` naplní novou hodnotou z paměti na adrese `r0` a hodnotu `r0` zvýší o 1. Naproti tomu instrukce

```
mac  -c1, y0, a    a, x: (r0)
```

není dovolena. V tomto případě lze použít

```
mac  -c1, y0, a    a, x: (r0)+n
```

která vynásobí registry `c1` a `y0`, součin odečte od hodnoty registru `a` a uloží jej do registru `a`. Původní hodnotu registru `a` uloží do paměti na adresu `r0` a zvýší hodnotu `r0` o hodnotu `n`. Pokud je registr `n` nastaven na 0, hodnota `r0` se nezmění, jak bylo požadováno.

Protože jsou k dispozici dvě sběrnice pro čtení z paměti, je možné s některými aritmetickými instrukcemi provést dvě paralelní čtení z paměti. Povolené kombinace jsou uvedeny v tab. 3.5. Povolená je např. instrukce tvořící základ diskretní konvoluce

```
mac  y0, x0, a    x: (r0)+, y0    x: (r3)+, x0
```

Instrukce vynásobí registr `y0` (hodnotu vzorku signálu $x[n]$) registrem `x0` (hodnotou vzorku impulsní charakteristiky $h[0]$) a výsledek připočte k hodnotě střadače `a`. Dále do registru `y0` přesune z paměti hodnotu na adrese `r0` (nový vzorek $x[n-1]$) a registr `r0` zvýší o 1 (takže obsahuje adresu dalšího vzorku $x[n-2]$). V rámci druhého paralelního přesunu do registru `x0` vloží hodnotu na adrese `r3` (nový koeficient $h[1]$) a zvýší adresu `r3` o 1 (takže obsahuje adresu dalšího vzorku $h[2]$). Opakováním této instrukce realizujeme lineární konvoluci

$$y[n] = \sum_{m=0}^{N-1} x[n-m]h[m]$$

Musí však být dodrženy uvedené registry, protože např. instrukce

```
mac  y0, x0, a    x: (r0)+, y0    x: (r2)+, x0
```

není podle tab. 3.5 povolena.

Signálové procesory řady TMS320C5xxx

U procesorů řady TMS320C5xxx nejsou paralelní přesuny při výpočtech tak významné. Hodnoty z paměti totiž není nutné nejprve přenést do registrů, ale mohou přímo vstupovat jako operandy instrukcí. Příkladem může být instrukce uvedená při popisu DALU v části 3.2.3

```
MAC  *AR1+, *CDP+, AC0  
::MAC *AR3+, *CDP+, AC1
```

která ve skutečnosti není dvojicí paralelně vykonávaných instrukcí, ale instrukcí jedinou. Přenosy z paměti jsou vykonány paralelně s provedením této jediné instrukce.

Protože ale DALU procesoru obsahuje oddělené bloky pro násobení s akumulací, dále jednotku bitových posunů a ALU, může zpracovávat několik instrukcí opravdu paralelně a nemusí se jednat pouze o přenosy. [80]

Tab. 3.5: Povolené kombinace dvou paralelních přesunů u DSC56F8367.

první přenos		druhý přenos	
zdroj	cíl	zdroj	cíl
x: (r0)+ x: (r0)+n x: (r1)+ x: (r1)+n	y0 y1	x: (r3)+ x: (r3)-	x0
x: (r4)+ x: (r4)+n	y0	x: (r3)+ x: (r3)+n	x0
x: (r0)+ x: (r0)+n x: (r4)+ x: (r4)+n	y1	x: (r3)+ x: (r3)+n3	c

Např. instrukce

```
MPYM  *AR1 - , *CDP , AC1
|| XOR AR2 , T1
```

Instrukci na první řádku provádí násobička. Hodnotu na adrese AR1 vynásobí hodnotou na adrese CDP a součin uloží do registru AC1. Současně AGU sníží hodnotu registru AR1 o 1. Znak paralelního zpracování || na začátku druhého řádku znamená, že tato instrukce má být prováděna současně s předchozí. Paralelně s násobičkou tak blok ALU provede exkluzivní bitový součin registru AR2 a registru T1 a výsledek uloží do registru T1.

Podobně jako u řady DSP56xxx existuje spousta omezení, které instrukce lze provádět paralelně a které ne. Základní pravidlo je, že instrukce musí používat různé bloky jádra a nesmí výsledek ukládat do stejného registru.

Signálové procesory řady ADSP21xx

Naopak možnosti paralelních přesunů u procesorů řady ADSP21xx jsou podobné jako u řady signálových procesorů DSC56xxx. Paralelně lze provádět tyto operace [4]:

- výpočet se dvěma paralelními čteními,
- dvě paralelní čtení,
- výpočet s paralelním čtením nebo zápisem,
- výpočet s přesuny mezi registry.

Lze tak provádět jádro lineární konvoluce jedinou instrukcí podobně jako u DSP56xxx

$$MR = MR + MX0 * MY0, \quad MX0 = DM(I0 += M0), \quad MY0 = PM(I4 += M4)$$

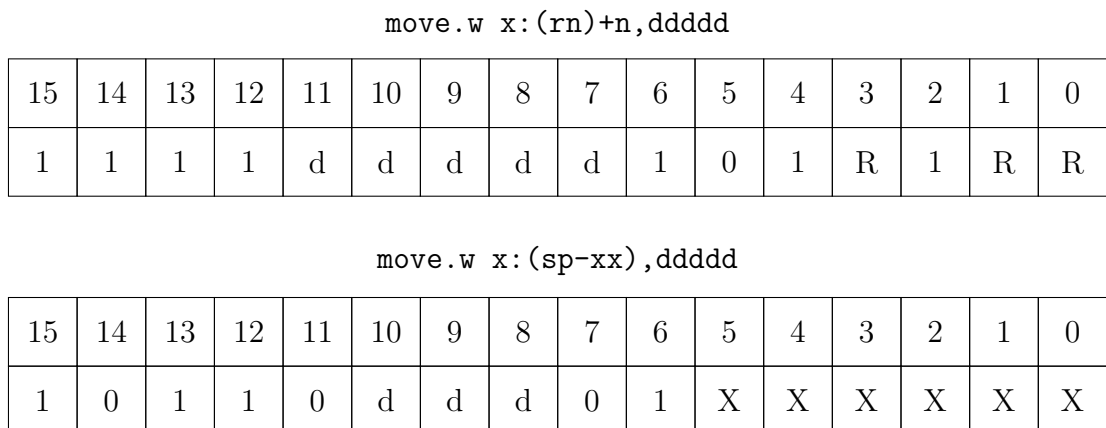
Instrukce vynásobí obsahy registrů MX0 a MY0, součin připočte k registru MR a výsledek uloží do registru MR. Paralelně do registru MX0 vloží hodnotu z adresy I0 a poté k registru I0 připočte M0. V rámci druhého paralelního přesunu vloží do registru MY0 hodnotu na adrese I4 a poté zvýší hodnotu I4 o M4.

Mezi omezení paralelních přenosu patří, že nelze použít indexování, pouze post modifikaci adresovacího registru.

3.2.5 Instrukční soubor a strojový kód

Z popisu jednotlivých prvků architektury signálových procesorů a především z předchozí části je patrné, že nelze libovolný registr použít v libovolné instrukci. Takový instrukční soubor je označován jako *neortogonální*. Je to způsobeno částečně vnitřní hardwarovou strukturou jádra procesoru, ale částečně i omezenými možnostmi strojového kódu procesoru. Signálové procesory s harvardskou architekturou totiž používají relativně krátké instrukční slovo, což značně redukuje počet instrukcí a kombinací jejich operandů. Problém lze ilustrovat na signálových procesorech řady DSP56xxx, u ostatních řad nebo výrobců jsou problémy podobné.

Příklad kódování instrukcí do strojového kódu je naznačen na obr. 3.13 [40]. Instrukce `move.w x:(rn)+n,d` umožňuje adresovací registr zakódovat 3 bity (na obrázku bity označené jako R), což umožňuje $2^3 = 8$ kombinací. Je to dost kombinací pro zakódování libovolného adresovacího registru jednotky generace adres. Podobně pro zakódování cílového registru strojový kód obsahuje dokonce 5 bitů (na obrázku označené jako d), což umožňuje $2^5 = 32$ kombinací. To je dost kombinací na to, aby cílovým registrem mohl být primárně datový registr DALU, ale i adresovací registr AGU.



Obr. 3.13: Příklad kódování instrukcí do strojového kódu procesoru DSC56F8367.

Naproti tomu instrukce `move.w x:(sp+xx), d` potřebuje 6 bitů (na obrázku označeny X) pro zakódování přímé 6 bitové hodnoty. Hodnota se nepředává v dalším 16bitovém slově, což by bylo značně neefektivní a pomalé. Ve strojovém kódu tak nejsou k dispozici bity pro zakódování adresovacího registru a je možné použít pouze jediný registr `sp`. Dokonce není možné pro zakódování cílového registru použít 5 bitů jako v předchozím případě, ale pouze 3. To značně omezí počet možných kombinací a nelze tak přenášet do adresovacího registru, ale jsou upřednostněny datové registry DALU. Je pravděpodobnější, že hodnota čtená ze zásobníku bude dále zpracovávána a tedy musí být uložena do zdrojových registrů DALU. Problém nastane, pokud hodnota na zásobníku je ukazatel

na zpracovávaná data. V takovém případě je nutné nejprve hodnotu načíst do datového registru, z něho teprve přesunout do adresovacího registru a nakonec provést čtení z dané adresy.

Uvedené omezení se netýká pouze instrukcí přesunu, ale takřka všech instrukcí nebo paralelních přenosů (viz omezení v tab. 3.5) signálového procesoru s harvardskou architekturou. Je to významná nevýhoda signálových procesorů s harvardskou architekturou, která komplikuje automatizovaný překlad, plánování využívání registrů a optimalizaci kódu. Na druhou stranu výhodou je kompaktní strojový kód, který má malé nároky na velikost paměti programu.

3.3 Architektura s velmi dlouhým instrukčním slovem

Jak bylo naznačeno v předchozí části harvardská architektura narazila na mez svých možností. Další zvýšení výpočetní výkonnosti mikroprocesorů zvýšením paralelizace nebylo možné s ohledem na omezenou délku strojového kódu instrukcí. Pokud by byl zachován stejný přístup, bylo by zvýšení počtu paralelně pracujících funkčních jednotek možné pouze za cenu prodlužování délky instrukčního slova. Tato architektura byla nejprve označována jako architektura typu LIW (Long Instruction Word). Prvním příkladem využití tohoto principu byl signálový procesor s pohyblivou řádovou čárkou firmy Texas Instruments s označením TMS320C30. Sdružení samostatných operací bylo vyznačeno symbolem `||`. Např. instrukce

```
MPYI vstup 2, vstup 1, výstup 1
|| STI vstup 3, výstup2
```

provádí současně dvě operace. První operací je celočíselné násobení `výstup 1 = vstup 1 × vstup 2` (symbol `MPY` – multiply – násobení, `I` – integer – celočíselná aritmetika) a druhou je uložení celočíselné hodnoty do paměti `výstup 2 = vstup 3` (symbol `ST` – store – ukládání). Skutečný zápis této instrukce v assembleru signálového procesoru by mohl vypadat například takto:

```
MPYI3  *++AR0(1), R5, R7
|| STI   R2, *-AR3(1)
```

Instrukce vyjadřovala skupinu operací:

```
registr R7 = registr R5 × paměť [adresa AR0 + 1]
paměť [adresa AR3 - 1] = registr R2
změna adresy AR0 = AR0 + 1
```

Architektura typu LIW má ve struktuře jeden programový čítač a v každém instrukčním cyklu načítá jedinou instrukci, ale několik dílčích částí jedné instrukce řídí několik funkčních jednotek. Tento způsob má základ v horizontálním programování a byl již využit u mikroprocesorových systémů sestavených z mikroprocesorových řezů (jako byla např. řada 3000). Instrukční slovo signálového procesoru TMS320C30 má 32 bitů a jeho formát je vidět na obr. 3.14.

Pokroky v technologii VLSI (Very Large-Scale Integration) umožnily dále zvyšovat počet funkčních jednotek. Spolu s tím se ale objevily problémy s efektivním využíváním

operační kód	výstup 1	vstup 1	vstup 3	výstup 2	vstup 2
7 bitů	3 bity	3 bity	3 bity	8 bitů	8 bitů

Obr. 3.14: Příklad formátu instrukčního slova signálového procesoru TMS320C30 s architekturou typu LIW.

kapacity programové paměti. Dlouhé instrukční slovo zvyšovalo velikost programu, zejména pokud většina operací nemohla být sdružována do paralelních instrukcí. Bylo nutné nalézt nový koncept skládání operačních kódů jednotlivých funkčních jednotek, který by se přizpůsoboval možnostem paralelizace algoritmů. Pokud je možné daný algoritmus paralelizovat, pak je žádoucí co nejvíce operací sdružit i za cenu dlouhého instrukčního slova. Naopak v případě, že algoritmus nebo jeho část je sekvenční a nelze ho paralelizovat, je žádoucí zachovat krátké instrukční slovo.

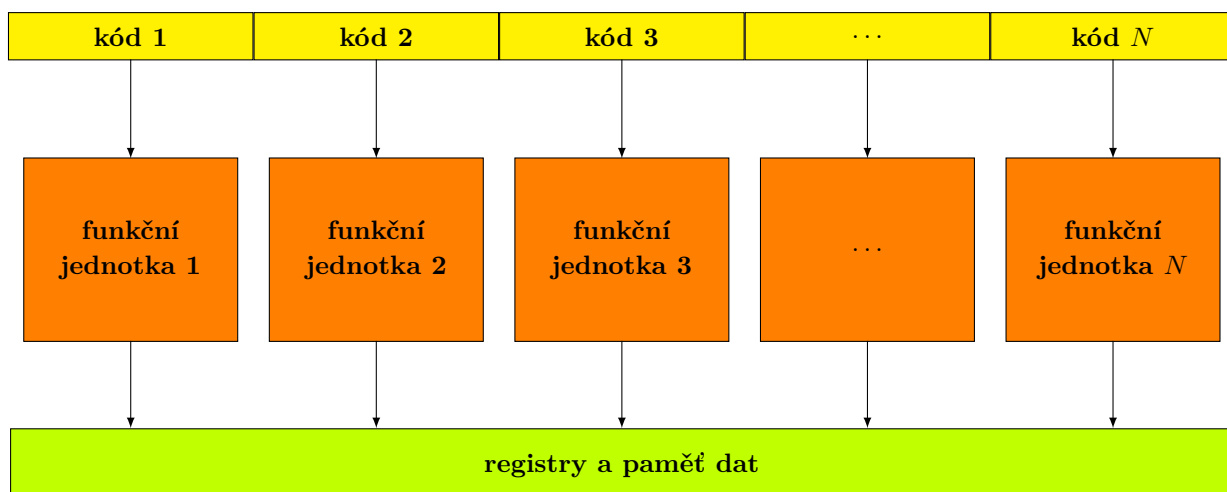
Tak se objevily nové typy signálových procesorů *s velmi dlouhým instrukčním slovem* typu VLIW (Very Long Instruction Word) [12]. Jejich architektura je určena velkým počtem vzájemně propojených funkčních jednotek, jako jsou aritmeticko-logické jednotky, násobičky, jednotky přístupu do paměti, jednotky pro manipulace s bity apod. Příkladem nového přístupu v oblasti architektury signálových procesorů byly mikroprocesory uvedené na trh v roce 1997 firmou Texas Instruments. Byl to signálový procesor s pevnou řádovou čárkou TMS320C6201 a signálový procesor s pohyblivou řádovou čárkou TMS320C6701 (VelociTI [48]). Každý ze signálových procesorů těchto řad má ve struktuře 8 funkčních jednotek.

Mírně odlišný přístup skládání dílčích instrukcí používá architektura *s proměnnou délkou instrukčního slova* VLES (Variable Length Execution Set). Představitelem je signálový procesor StarCore SC140 (MSC8101), vyvinutý společně firmami Motorola a Agere Systems, který má 15 funkčních jednotek [17].

Instrukce architektury typu VLIW se zpracovávají po jedné – stejně jako ve von Neumannově počítači. Rozdíl ovšem je v tom, že každá programová instrukce (označovaná jako *instrukční paket*) je podle pevně stanovených pravidel rozdělena do skupiny polí (dílčích instrukcí), v nichž jsou zapsány operační kódy, které ovládají každou funkční jednotku samostatně. Současně jsou také ovládány přenosové cesty (datové a adresové sběrnice) a přenosy dat po nich. Všechny operace předepsané v jednom instrukčním paketu jsou prováděny paralelně. Blokové schéma architektury mikroprocesoru typu VLIW je znázorněno na obr. 3.15.

Výhody architektury typu VLIW lze shrnout stručně do těchto bodů:

- Architektura typu VLIW dosahuje vysokého výpočetního výkonu s relativně jednoduchou architekturou mikroprocesoru ve srovnání s architekturou paralelních systémů. Například signálový procesor TMS320C64xx z řady VelociTI firmy Texas Instruments má při kmitočtu hodinových impulsů $f_{\text{clock}} = 1 \text{ GHz}$ výpočetní rychlost 8 000 MIPS (neboli 1 000 VLIW MIPS).
- U architektury typu VLIW je možné poměrně snadno přizpůsobit výpočet dané aplikaci. Snahou výrobců je dosáhnout co největší ortogonalita architektury. Or-



Obr. 3.15: Příklad mikroprocesoru s architekturou typu VLIW.

togonalita architektury znamená, že libovolný registr ze souboru registrů může být operandem v libovolné instrukci. Úplnou ortogonalitu architektury zde však nelze dosáhnout.

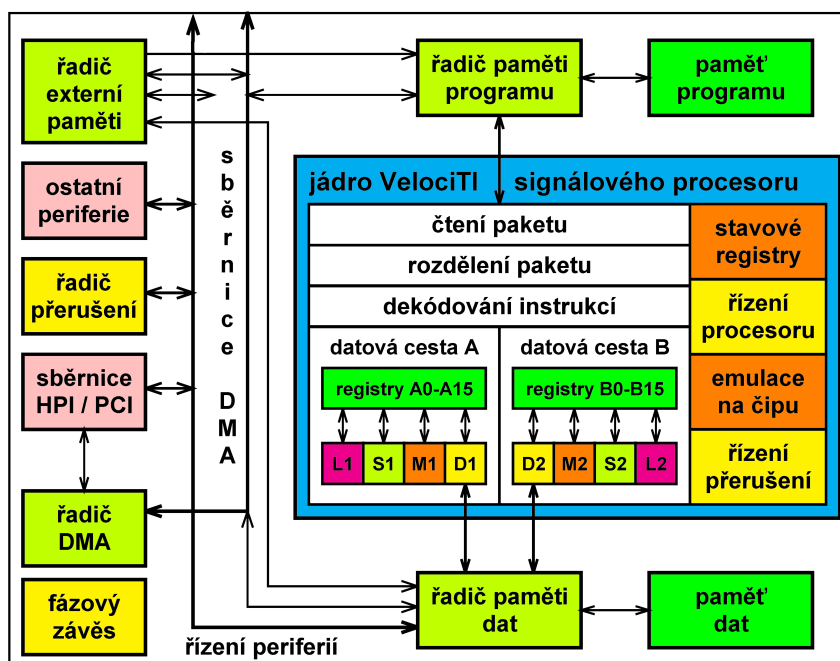
- Analýza a plánování paralelního zpracování je provedeno během překladu zdrojového kódu programu. To snižuje nároky na složitost čipu (hardware), protože dílčí části instrukčního paketu jsou funkčním jednotkám přiřazovány podle přesně daných jednoduchých pravidel. Operační kód dílčích polí je možné volit z omezené instrukční sady, to znamená, že se vlastně jedná o dílčí instrukce typu RISC (Reduced Instruction Set Computer).
- Přestože architektura typu VLIW je podstatně složitější než harvardská architektura, tak překlad z vyšších programovacích jazyků (jazyk C, C++ apod) je méně komplikovaný. Jak již bylo zmíněno, je to tak dáno tím, že dílčí instrukce mají tvar instrukcí typu RISC a je částečně dodržována ortogonalita architektury. Tím lze optimalizovat překladače z jazyka C a C++ a dosáhnout vysoké účinnosti generovaného kódu, která nebyla zatím dosažena u klasických signálových procesorů s harvardskou architekturou. Zvláště dva současné paralelní přenosy, které používají například signálové procesory s harvardskou architekturou firmy Motorola, značně komplikují postup překladu.

3.3.1 Vybrané procesory s velmi dlouhým instrukčním slovem

Signálové procesory s jádrem VelociTI

Firma Texas Instruments používá pro jádro signálových procesorů s architekturou VLIW označení VelociTI [48]. Existují dvě základní skupiny: *signálové procesory s pevnou řádovou čárkou* (původní řada TMS320C62xx a vylepšená řada TMS320C64xx) a *signálové procesory s pohyblivou řádovou čárkou* (TMS320C67xx). Nejnovější řada TMS320C66xx je více jádrová, a lze dokonce kombinovat jádra s pevnou a pohyblivou řádovou čárkou v jednom procesoru.

Jádro VelociTI je rozděleno do dvou symetrických datových cest, která každá obsahuje 4 paralelní funkční jednotky L1, S1, M1, D1, resp. L2, S2, M2, D2. Současně je tak možné zpracovávat až 8 instrukcí sdružených do instrukčního paketu. Každá cesta obsahuje šestnáct 32bitových registrů označených A0 až A15, resp. B0 až B15. U řady TMS320C64xx byl počet registrů v každé datové cestě zvýšen na 32. Jádro signálového procesoru je spojeno s okolím dvojicí adresových a datových sběrnic (viz obr. 3.16). Všechny sběrnice mají šířku 32 bitů, u řady TMS320C64xx je šířka datových sběrnic zvýšena na 64 bitů. Vně procesoru však vystupuje jediný pár adresové a datové sběrnice, na které jsou vnitřní sběrnice přepínány. Vnitřní paměť integrovanou na čipu lze konfigurovat jako paměť typu RAM nebo jako jednoúrovňovou paměť typu cache. U signálových procesorů TMS320C621x a TMS320C671x je použita dvojúrovňová organizace paměti cache.



Obr. 3.16: Blokové schéma signálových procesorů řady TMS320C6200 a TMS320C6700.

Každá z funkčních jednotek v dané datové cestě je specializována na vykonávání operací určitého typu. Jedná se o tyto funkční jednotky: *jednotka pro realizaci aritmetických operací L*, *jednotka pro bitové operace S*, *jednotka pro přístup do paměti D* a *hardwarová násobička M*. Všech osm funkčních jednotek jádra signálového procesoru je řízeno nezávisle a všechny mohou pracovat současně.

Funkční jednotky mohou zapisovat výsledky operací pouze do registrů té datové cesty, do které přísluší. Jednotky L1, S1, M1, D1 tak mohou zapisovat pouze do registrů datové cesty A, naopak jednotky L2, S2, M2, D2 mohou zapisovat pouze do registrů datové cesty B. Vstupní operandy, které jsou využity pro danou operaci, jsou opět čteny ze skupiny registrů datové cesty příslušející dané funkční jednotce. Jedna vstupní hodnota v datové cestě A však může být čtena z registrů datové cesty B po křížové sběrnici 1x a současně jedna vstupní hodnota v datové cestě B může být čtena z registrů datové cesty A po křížové sběrnici 2x. Výjimkou je operace čtení z paměti nebo zápis do paměti, kdy funkční jednotky D mohou přistupovat k registrům opačné datové cesty přímo.

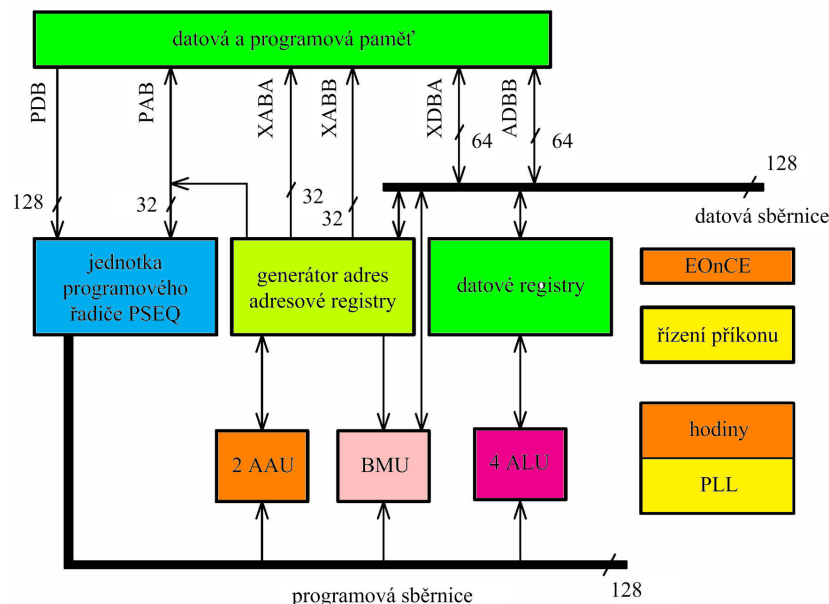
Signálové procesory s jádrem StarCore

Podobný, ale mírně odlišný, přístup k řešení je označován jako VLES. Zástupcem je řada signálových procesorů MSC81xx vyvinutá původně firmou Motorola, dnes nabízených společností NXP. Jádro je označováno jako StarCore140 a prvním signálovým procesorem s tímto jádrem je integrovaný obvod MSC8101 [19]. Jádro StarCore140 (SC140) tvoří 3 paralelně pracující jednotky (obr. 3.17):

Datová aritmetická-logická jednotka DALU Tato jednotka obsahuje šestnáct 40bitových datových registrů D0-D15, 4 aritmeticko-logické jednotky ALU a 8 jednotek pro ošetření saturace a provedení vážení (Data Shifter/Limiter).

Jednotka pro generování adres AGU Jednotka AGU sestává z registrů pro nepřímé adresování R0-R15, z registrů posunu N0-N3, modifikačních registrů M0-M3 a řídicího modifikačního registru MCTL (Modifier Control Register) a ukazatele na zásobník SP. Výpočty provádí dvě 32bitové adresovací aritmetické jednotky AAU1 (Address Arithmetic Unit) a AAU2 a jednotka pro maskování bitů BMU (Bit Manipulating Unit).

Programová řídicí jednotka PSEQ Jednotka PSEQ (Program Sequencer Unit) se skládá ze tří částí: jednotky dekódování programu PDU (Program Dispatch Unit), řídicí jednotky PCU (Program Control Unit) a generátoru adres PAG (Program Address Generator). Má k dispozici tyto registry: programový čítač PC (Program Counter), stavový registr SR (Status Register), registry počáteční adresy cyklu SA0-SA3, registr pro čítače cyklů LC0-LC3, registr módu EMR (Exception and Mode Register) a registr počáteční adresy tabulky vektorů přerušení VBA (Vector Base Address).



Obr. 3.17: Blokové schéma jádra SC140 (StarCore).

3.3.2 Jednotka generace adres

Signálové procesory s jádrem VelociTI

Jednotku generace adres pro čtení nebo zápis dat do paměti zastupuje v signálovém procesoru s jádrem VelociTI funkční jednotka D. Tato jednotka kromě aritmetických operací podporuje také lineární a modulo adresování (pouze s délkou bloku rovnou celočíselné mocnině 2) a přenos 8bitových (**byte**), 16bitových (**half word**), 32bitových (**word**) hodnot mezi pamětí a souborem registrů. Délka přenášených dat je v instrukci čtení (**LD** - load) nebo zápisu (**ST** - store) do paměti vyjádřena připojeným písmenem (**B** - **byte**, **H** - **half word**, **W** - **word**). Adresa je generována jednou ze dvou jednotek D1 nebo D2 na jednu z adresových sběrnic DA1 nebo DA2. Pokud je adresa generována na adresovou sběrnici DA1, pak čtení z paměti probíhá po datové sběrnici LD1 a zápis do paměti po datové sběrnici ST1. V obou případech jako cíl nebo zdroj dat musí být registry datové cesty A. Pokud je adresa generována na adresovou sběrnici DA2, pak čtení z paměti probíhá po datové sběrnici LD2 a zápis do paměti po datové sběrnici ST2. V tomto případě jako cíl nebo zdroj dat musí být registry datové cesty B. Obě jednotky D mohou přistupovat do paměti současně za předpokladu, že používají odlišné adresové sběrnice. Adresové i datové sběrnice mají šířku 32 bitů, u řady TMS320C64xx jsou datové sběrnice 64bitové.

Jednotka D podporuje pouze nepřímé adresování, což znamená, že adresa pro přístup do paměti musí být uložena v jednom z adresovacích registrů. Jako adresovací registr R může sloužit kterýkoli z datových registrů, rozlišení se děje znakem '*' před označení registru v symbolickém zápisu instrukce. Při nepřímém adresování je možné adresu nebo obsah registru změnit podle tab. 3.6. Jako registr posunu N je opět možné použít libovolný datový registr. Adresování beze změny obsahu registru odpovídá indexování u harvardské architektury. Hodnota o kolik se má adresa změnit, neudává počet bajtů, ale počet přenášených datových slov. Hodnota změny bude násobena délkou přenášených dat v bajtech (pro B - 1, H - 2, W - 4) a teprve tento součin bude přičten nebo odečten od hodnoty adresy. Indexování se změnou o 15bitovou konstantu lze použít pouze ve spojení s registry

Tab. 3.6: Způsoby nepřímého adresování u architektury VelociTI.

nepřímé adresování	beze změny obsahu registru	změna před adresací	změna po adresaci
beze změny	*R		
se změnou o jedničku		*--R *++R	*R++ *R--
se změnou max. o 5bitovou konstantu	*+R[konst] *-R[konst]	*++R[konst] *--R[konst]	*R[konst]++ *R[konst]--
se změnou max. o 15bitovou konstantu ¹	*+R[konst]		
se změnou o hodnotu uloženou v registru posunu N	*+R[N] *-R[N]	*++R[N] *--R[N]	*R++[N] *R--[N]

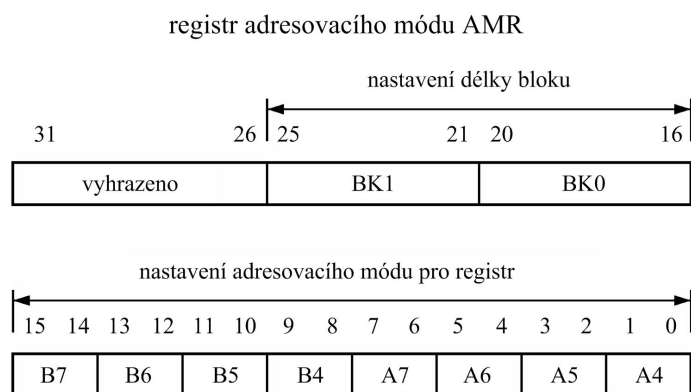
Pozn.

¹ Změnu o 15bitovou konstantu lze použít pouze pro registry B14, B15.

B14 a B15. Proto se tyto registry používají jako ukazatele na velká pole dat, u kterých je nutná možnost přistupovat náhodně k jednotlivým prvkům. Příkladem takového pole může být softwarový zásobník (*stack*) nebo tzv. softwarová „halda“ (*heap*).

Modulo adresování lze nastavit pouze pro registry A4 až A7, B4 až B7. Typ adresování je určen nastavením modifikačního registru AMR (Addressing Mode Register). Modifikační registr AMR se skládá ze dvou částí (obr. 3.18). V horní 16bitové polovině jsou uloženy dvě 5bitová čísla BK0, BK1, která určují velikost bloku pro adresování v režimu modulo. Velikost bloku v bajtech je vždy celočíselná mocnina dvou a pohybuje se v rozmezí 2 až 2^{32} . Velikost bloku je rovna hodnotě 2^{K+1} , kde K je hodnota uložená v části BK0 nebo BK1. Pokud je K rovno 31, je délka bloku rovna velikosti adresovacího prostoru 4 GB ($2^{32} = 4\,294\,967\,296$) a adresování typu modulo má stejnou funkci jako lineární adresování. Spodní adresa bloku musí být celistvým násobkem délky bloku, tzn. že má $K + 1$ nejméně významných bitů rovno nule. Není podmínkou, aby adresovací registr použitý pro nepřímé adresování začínal na spodní adrese, ale musí ležet kdekoli uvnitř bloku. Modulová aritmetika potom zaručí, že hodnoty adres zůstanou uvnitř bloku. Spodní 16bitová část registru AMR je rozdělena na osm 2bitových hodnot, které nastavují adresovací mód pro každý z osmi registrů A4 až A7, B4 až B7. Podle nastavení hodnoty těchto dvou bitů je adresovací mód registru nastaven takto:

- 00 - lineární adresování,
- 01 - adresování typu modulo s délkou bloku uloženou v části BK0,
- 10 - adresování typu modulo s délkou bloku uloženou v části BK1,
- 11 - vyhrazeno pro pozdější použití.



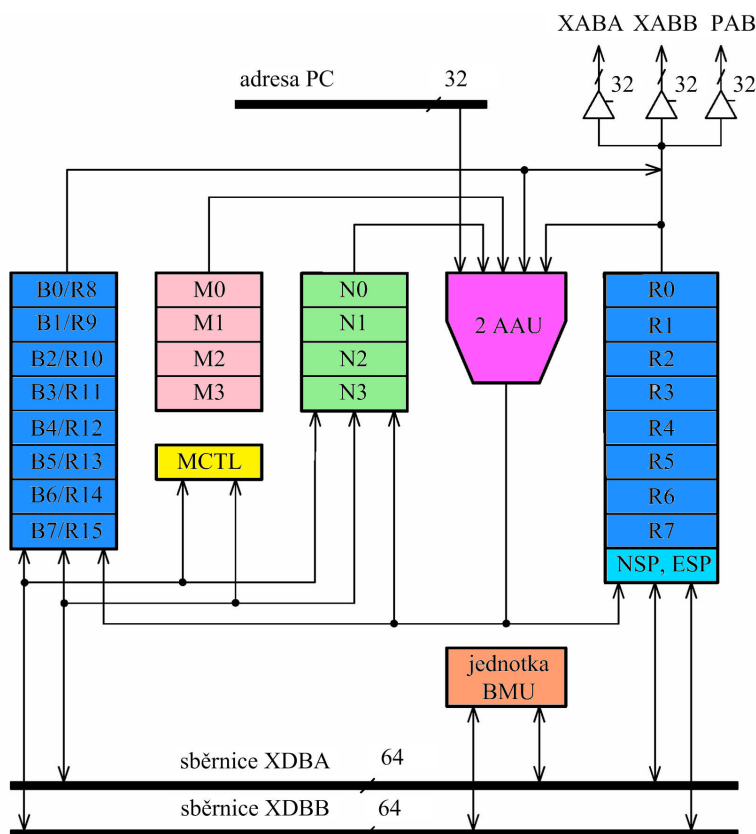
Obr. 3.18: Struktura modifikačního registru AMR pro nastavení adresovacího módu pro registry A4 až A7, B4 až B7.

Při počátečním nastavení signálového procesoru (RESET) jsou všechny bity stavového registru AMR nastaveny na nulu. Pro registry A4 až A7, B4 až B7 je tak nastaveno lineární adresování. [82]

Signálové procesory s jádrem StarCore

U jádra StarCore se provádí efektivní výpočet adres pro přenos dat mezi registry a paměť, který je uskutečněn 64bitovými sběrnici XDBA a XDBB, jednotka AGU. Blokové schéma jednotky AGU je na obr. 3.19. Šestnáct 32bitových registrů R0-R15 slouží pro nepřímé adresování datových operandů v paměti nebo je možné je využít jako univerzální registry. Tato skupina registrů je rozdělena na dvě banky po 8 registrech.

Registry posunu N0 až N3 jsou 32bitové registry, v nichž je uložena hodnota, o kterou se zvýší nebo sníží adresa v registru nepřímého adresování R0 až R15. Libovolný registr posunu může být spojen s libovolným registrem nepřímého adresování. Registry posunu lze také využít jako univerzální registry.



Obr. 3.19: Blokové schéma jednotky generace adres jádra StarCore.

Ukazatel zásobníku SP je tvořen dvěma 32bitovými registry, které mají označení jako NSP (Normal Stack Pointer) a jako ESP (Exception Stack Pointer). Aktivní však může být vždy jen jeden z nich. Rozlišení se děje bitem EXP stavového registru SR. Pokud je bit vynulován, pracuje signálový procesor v normálním stavu a jako ukazatel na zásobník SP se používá registr NSP. Pokud je bit nastaven na 1, signálový procesor obsluhuje některou z výjimek a jako ukazatel na zásobník SP používá registr ESP. Pojem výjimka představuje obecný termín pro veškerá přerušení výkonu programu vyvolaná hardwarovým přerušením (interním i externím) nebo softwarovým přerušením. Pojem přerušení se používá pouze

pro vnější přerušování. Oba registry musí být inicializovány na začátku programu. Data ukládaná do zásobníku během obsluhy výjimek jsou tak oddělena od dat ukládaných během normální činnosti.

V případě nepřímého adresování je adresa operandu uložena v registrech R0 až R15. Přitom může dojít k modifikaci adresy nebo registru podle tab. 3.7. Při modifikaci se uvažuje i velikost přenášených dat. Pokud přenášíme data délky jednoho bajtu, je k registru přičtena nebo odečtena hodnota registru posunu. Pokud přenášíme data délky 2 bajty, je hodnota registru posunu nejprve násobena 2 a pak je přičtena k adresovému registru. Podobně, pokud přenášíme data délky 8 bajtů, je hodnota registru posunu nejprve násobena osmi. Pokud použijeme pro změnu adresového registru přímou hodnotu, bude adresový registr modifikován podobně. Modifikace adresy je prováděna pomocí celočíselné aritmetiky dvou aritmetických jednotek AAU1 a AAU2. Během jednoho instrukčního cyklu mohou dvě jednotky AAU (Address Arithmetic Unit) generovat 32bitovou adresu pro programovou paměť na programovou adresovou sběrnici PAB nebo dvě 32bitové adresy pro datové sběrnice XABA a XABB.

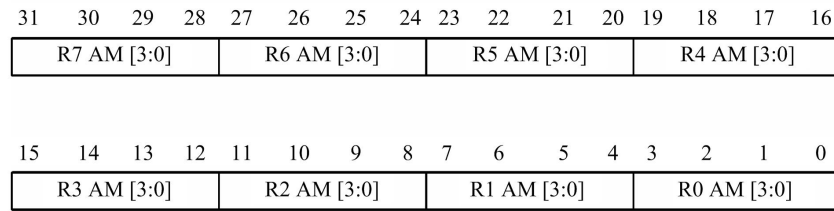
Tab. 3.7: Možnosti nepřímého adresování paměti u procesoru StarCore.

beze změny	(Rn)	
se změnou	post inkrementace	post dekrementace
o 1	(Rn)+	(Rn)-
o Nm	(Rn)+Nm	
indexování	pouze registry Rn	pouze SP
o Rm	(Rn+Rm)	
o 4bitovou hodnotu	(Rn+x)	
o 8bitovou hodnotu		(SP-xx)
o 16bitovou hodnotu	(Rn+xxxx)	(SP+xxxx)

U nepřímého adresování jednotka podporuje čtyři adresovací režimy:

- lineární adresování,
- modulo adresování s libovolnou délkou bloku,
- modulo adresování s délkou bloku rovnou celočíselné mocnině 2,
- bitově-reverzované adresování.

Typ adresování je určen nastavením bitů AM modifikačního řídicího registru MCTL (viz obr. 3.20) příslušejících danému adresovacímu registru Rn. Adresování typu modulo lze nastavit pouze pro registry R0 až R7 spodní banky. Důvodem je, že při adresování typu modulo jsou registry horní banky použity jako bazové adresové registry B0 až B7 pro adresové registry R0 až R7 spodní banky, přičemž přiřazení je pevně dáno (registr R8 se používá jako bazový registr B0 pro adresový registr R0, registr R9 jako B1 pro R1, atd.). Pokud je pro některý z registrů dolní banky nastaveno modulo adresování, jeho sdružený registr v horní bance je možné použít pouze jako bazový registr. Není možné ho použít pro nepřímé adresování. Pokud je nastaveno modulo adresování s libovolnou délkou bloku, pak je délka nastavena v jednom z 32bitových modifikačních registrů M0 až M3. Počáteční



Obr. 3.20: Struktura modifikačního registru MCTL jednotky AGU jádra SC140.

adresa bloku SA může být také libovolná a je uložena v přidruženém básovém registru Bn. V případě nastavení modulu adresování s délkou rovnou celočíselné mocnině 2 je v jednom z modifikačních registrů uložena *délkabloku* − 1, tj. $2^K - 1$, $K \in \mathbb{N}$. Počáteční adresa bloku pak musí být celočíselným násobkem délky bloku, tj. musí mít K nejméně významných bitů nulových, a je opět uložena v přidruženém básovém registru Bn. Nastavení bitů AM v řídicím modifikačním registru MCTL pro adresovací režim a přiřazení modifikačního registru k registru nepřímého adresování je definováno v tab. 3.8. Modifikační registry lze použít opakovaně, tzn. že jeden modifikační registr může být přiřazen více adresovým registrům.

Tab. 3.8: Nastavení adresového módu spodní banky registrů R0 až R7.

AM3	AM2	AM1	AM0	adresový mód
0	0	0	0	lineární adresování
0	0	0	1	bitově reverzované adresování
1	0	0	0	adresování modulo, využit registr M0
1	0	0	1	adresování modulo, využit registr M1
1	0	1	0	adresování modulo, využit registr M2
1	0	1	1	adresování modulo, využit registr M3
1	1	0	0	adresování modulo s mocninou 2, využit registr M0
1	1	0	1	adresování modulo s mocninou 2, využit registr M1
1	1	1	0	adresování modulo s mocninou 2, využit registr M2
1	1	1	1	adresování modulo s mocninou 2, využit registr M3

Poslední blok BMU umožňuje jednoduché provádění operací s jednotlivými bity nebo se skupinami bitů ve všech registrech R0 až R15, D0 až D15, v paměti a všech řídicích registrech EMR, VBA, PCTL0, PCTL1, SR a MCTL. Je možné provádět nastavení, nulování, testování a změnu bitů podle zvolené masky. Tato maska definuje, jak mají být bity měněny a je uvedena v instrukci jako konstanta.

Nejmenší adresovatelnou položkou v paměti je 1 bajt (1 B). Jádro SC140 podporuje ovšem ukládání a čtení hodnot s různým počtem bajtů. Počet bajtů je označen přímo v instrukci MOVE. Například instrukce MOVE.B uvažuje přesun jednoho bajtu. Chceme-li

přenášet slova o 2 bajtech, pak použijeme instrukci `MOVE.W` nebo `MOVE.F`. Instrukce `MOVE.W` přenáší 16bitová celá čísla (WORD) a instrukce `MOVE.F` přenáší 16bitová zlomková čísla (FRACTIONAL). Rozdíl je v uložení do cílového registru. Pokud je přenesen zlomkový operand typu FRACTIONAL ve formátu Q0.15, tak je zapsán do části Dn.h datového registru (viz obr. 3.24), část Dn.l je doplněna nulami, část Dn.e je využita pro znaménko a bit Ln je nastaven na 0. Celočíslný operand typu WORD ve formátu Q15.0 je přenesen do části Dn.l a části Dn.h a Dn.e jsou využity pro znaménko. Bit Ln je opět nastaven na nulu. Podobně se pracuje s ostatními operandy různé délky v celočíselném a zlomkovém formátu doplňkového kódu. Pro přenos dat o 4 bajtech (LONG) je možné použít instrukce `MOVE.L` (LONG), `MOVE.2F` (TWO FRACTIONAL) nebo `MOVE.2W` (TWO WORD). Pro data o šířce 8 bajtů (TWO LONG) se používají instrukce přesunu `MOVE.2L`, `MOVE.4F` (FOUR FRACTIONAL) nebo `MOVE.4W` (FOUR WORD). Při zápisu obsahu datového registru (formát popsáný níže) do paměti je nejprve testován bit L. Je-li roven 1, pak dochází k ošetření saturace.

3.3.3 Datová aritmetická-logická jednotka

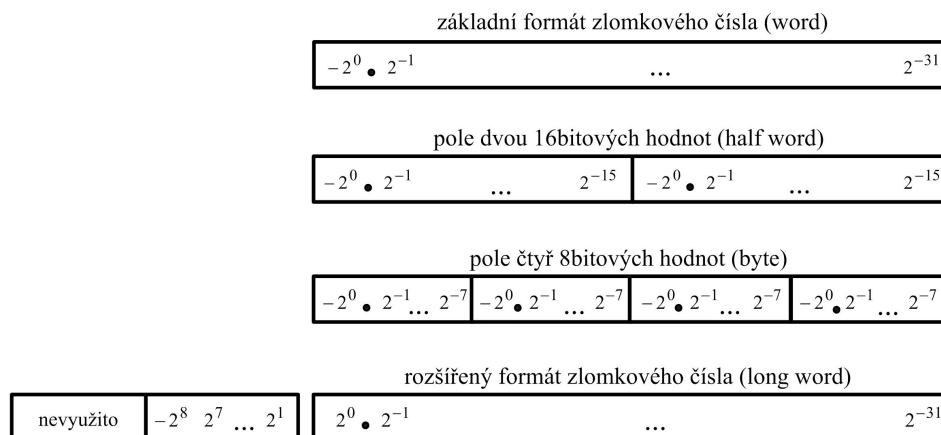
Signálové procesory s jádrem VelociTI

Jádro VelociTI neobsahuje DALU podobnou jádru řady TMS320C5xxx s harvardskou architekturou. Každá z 8 funkčních jednotek jádra je specializována na určitý druh operací. Násobení provádí pouze jednotka M1 nebo M2, která ale nemůže provádět bitové operace nebo součty. Podobně čtení a zápis do paměti provádí pouze jednotka D1 nebo D2, kterou lze využít i pro jednoduché aritmetické operace bez saturace. Jednotky S1 a S2 jako jediné provádí instrukce skoku a některé bitové operace. Mohou ale provádět aritmetické operace se saturací podobně jako jednotky L1 nebo L2. Oproti harvardské architektuře žádná z jednotek nepodporuje operaci násobení s akumulací MAC. [82]

Operandy jsou přitom uloženy v datových registrech (A0, A1, ... až A31, B0, B1, ... až B31). V registrech lze ukládat čísla s pevnou řádovou čárkou buď jako čísla celá (formát Q15.0, Q31.0, Q39.0) nebo jako čísla zlomková (formát Q0.15, Q0.31, Q8.31). Poslední formát v každé skupině délky 40 bitů je uložen do registrového páru. Ten vznikne spojením sudého a vyššího lichého registru (A1:A0, A3:A2, ... , A31:A30 a B1:B0, B3:B2, ... , B31:B30) s délkou 64 bitů. Z této délky je však využito pouze 40 bitů (8+16+16). Je to provedeno tak, že sudý registr z registrového páru je využit celý a z lichého registru je použito pouze 8 nejméně významných bitů. Těchto 8 bitů slouží pro krátkodobé ošetření přetečení a u zlomkového formátu Q8.31 zvyšuje povolený rozsah čísel na $\langle -256; 256 - 2^{-31} \rangle$.

Záleží na dané instrukci v jakém formátu (celé nebo zlomkové) a v jakém rozsahu (32 bitů - word, 16 bitů - half word, 8 bitů - byte nebo 40 bitů - long word) je číslo do datového registru uloženo. Příkladem může být instrukce `MPY`, která provádí násobení 16bitových celých čísel, zatímco instrukce `SMPY` provádí násobení 16bitových zlomkových čísel, resp. spodních polovin zdrojových registrů. Lze provádět násobní i horních polovin instrukcí `SMPYH` nebo vzájemné násobení horní a dolní poloviny `SMPYHL`, resp. `SMPYLH`. Instrukce `ADD` pak sečítá celá i zlomková čísla v rozsahu 32 bitů a instrukce `SADD` provede i saturaci výsledku. Uložení čísel ve zlomkovém formátu pro různé formáty je znázorněno

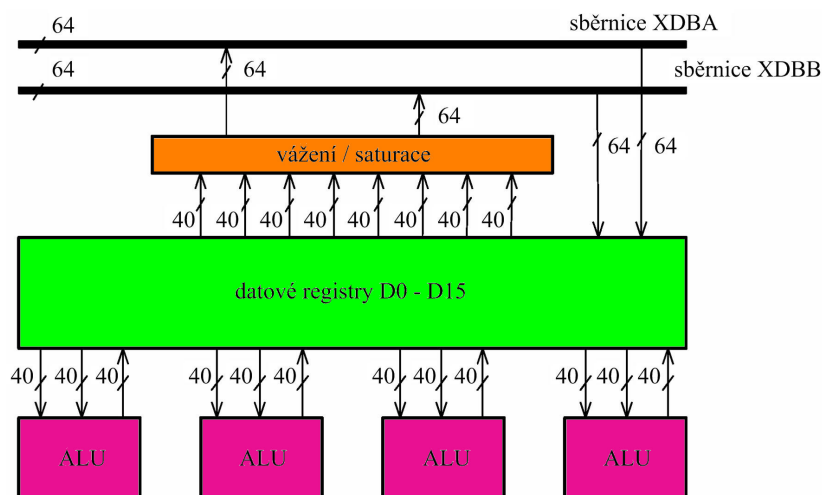
na obr. 3.21. Tečka • ukazuje umístění řádové čárky, která odděluje znaménkový bit od zlomkové části čísla.



Obr. 3.21: Příklad formátů zlomkových čísel v doplňkovém kódu, které lze uložit do datových registrů.

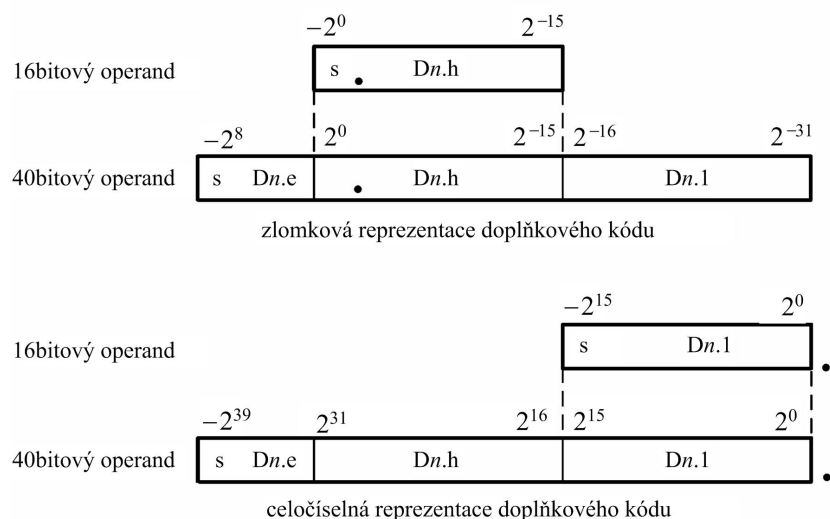
Signálové procesory s jádrem StarCore

Aritmetické a logické operace provádí v jádru SC140 jednotka DALU. Její blokové schéma je vidět na obr. 3.22. Jednotka je z velké části inspirována původní DALU procesorů řady DSP56xxx s harvardskou architekturou. Umožňuje tak většinu operací jako původní DALU včetně násobení s akumulací. Jednotka MAC je součástí každé ze čtyř jednotek



Obr. 3.22: Blokové schéma aritmetické-logické jednotky DALU jádra SC140.

ALU. Jednotka MAC obsahuje násobičku a sčítačku. Provádí násobení 16 x 16 bitů horní nebo dolní poloviny registrů a výsledek má délku 40 bitů. Operandy jsou vyjádřeny v doplňkovém kódu jako celá čísla nebo zlomková čísla bez znaménka a se znaménkem (obr. 3.23). Tečka • ukazuje umístění řádové čárky, která odděluje celou část od zlomkové části čísla. Písmeno *s* zde označuje znaménkový bit. Vstupem i výstupem do každé násobičky může být kterýkoliv z datových registrů D0 až D15. Celkem může tato jednotka realizovat 61 různých operací včetně násobení, násobení s akumulací, sčítání, sčítání s přenosem, výpočet absolutní hodnoty, porovnání dvou čísel, dělení, přenosy dat mezi datovými registry apod.



Obr. 3.23: Reprezentace operandů v doplňkovém kódu jako zlomková čísla nebo jako celá čísla.

Jednotka DALU obsahuje 16 datových registrů D0 až D15, které zároveň plní funkci střadačů a jsou jak vstupními, tak i výstupními registry pro čtyři jednotky ALU. Každý datový registr je rozdělen do tří částí *Dn.e*:*Dn.h*:*Dn.l*, $n = 0, 1, \dots, 15$. Jejich formát je vidět na obr. 3.24. Část *Dn.e* (extension portion) má 8 bitů a slouží k rozšíření rozsahu doplňkového kódu v zlomkovém formátu z přibližně ± 1 na ± 256 . Části *Dn.h* (high portion) a *Dn.l* (low portion) mají každá délku 16 bitů. Celková délka datového registru je tedy 40 bitů. Ke každému datovému registru D0 až D15 je přidružen jeden z bitů L0 až L15, který indikuje překročení povoleného rozsahu zobrazení čísel. Tzn. že je používána část *Dn.e*.

Kromě jednotky MAC obsahuje každá z jednotek ALU i jednotku BFU (Bit Field Unit). Tato jednotka obsahuje paralelní 40bitový obousměrný posuvný registr a logickou jednotku. Provádí 24 operací jako jsou bitové posuny a rotace, logické operace, zápis a čtení prvků bitového pole apod.

Posledními jednotkami, které jsou ve struktuře jednotky DALU (obr. 3.22) jsou jednotky pro vážení a saturaci. Jednotky umožňují provádět saturaci a vážení při přenosu dat z registrů D0-D15 na datové sběrnice XDBA a XDBB a do paměti. Hodnota uložená

	7	0 15	0 15	0	
L0	D0.e	D0.h	D0.1		D0
L1	D1.e	D1.h	D1.1		D1
L2	D2.e	D2.h	D2.1		D2
L3	D3.e	D3.h	D3.1		D3
L4	D4.e	D4.h	D4.1		D4
L5	D5.e	D5.h	D5.1		D5
L6	D6.e	D6.h	D6.1		D6
L7	D7.e	D7.h	D7.1		D7
L8	D8.e	D8.h	D8.1		D8
L9	D9.e	D9.h	D9.1		D9
L10	D10.e	D10.h	D10.1		D10
L11	D11.e	D11.h	D11.1		D11
L12	D12.e	D12.h	D12.1		D12
L13	D13.e	D13.h	D13.1		D13
L14	D14.e	D14.h	D14.1		D14
L15	D15.e	D15.h	D15.1		D15

datové registry

Obr. 3.24: Formát a členění datových registrů D0 až D15 jednotky DALU.

v registru se přitom nezmění, změna se týká pouze dat uložených do paměti. Saturace i vážení se provede pouze v případě instrukcí přesunu `MOVES.F`, `MOVES.2F`, `MOVES.4F` a `MOVES.L`. Pro skupinu instrukcí `MOVE` se operace saturace ani vážení neuplatní. Pro každou sběrnici `XDBA` a `XDBB` jsou určeny 4 jednotky pro vážení a saturaci, tj. celkem je ve struktuře 8 těchto jednotek. Jednotky mají podobnou funkci jako jednotka vážení a saturace u signálových procesorů s harvardskou architekturou (např. řada DSP56300 nebo DSP56800). Pro nastavení operace vážení jsou určeny bity `S0` a `S1` (bity 4 a 5) stavového registru `SR`. Při nastavení bitů `S1 = 0` a `S0 = 1` jsou data před uložením do paměti dělena 2 (násobena 0,5) a při nastavení bitů `S1 = 1` a `S0 = 0` jsou data násobena 2. Beze změny data zůstanou, když `S1 = 0` a `S0 = 0`. Pokud hodnota po vážení překročí povolený rozsah doplňkového kódu v zlomkovém formátu se znaménkem, dojde k saturaci [18].

3.3.4 Instrukční soubor a strojový kód

Signálové procesory s jádrem VelociTI

Celková maximální délka instrukčního paketu jádra VelociTI je 256 bitů, přičemž paket obsahuje až 8 dílčích částí délky 32 bitů pro 8 funkčních jednotek jádra. Délka jedné dílčí instrukce 32 bitů je dostatečná pro zakódování možných kombinací instrukcí a registrů.

Instrukční soubor tak vykazuje mnohem větší ortogonalitu než u procesorů s harvardskou architekturou. To vede k jednodušší alokaci registrů, jednotek a dalších prostředků a ve výsledku k jednoduššímu překladu a optimalizaci kódu. Příklad strojového kódu pro instrukci **SMPY** je na obr. 3.25. Nejvýznamnější tři bity označené jako **creg** jsou vyhrazeny pro zakódování podmiňovacího registru. Lze tak obsáhnout celkem 8 různých kombinací a proto každou instrukci lze podmínit hodnotou v registru A0, B0, A1, B1, A2, B2. Bit 28 označený **z** (**zero**) odlišuje, zda instrukce bude provedena při nulové hodnotě (**[A0]** – nastaven na 1) nebo nenulové hodnotě (**[!A0]** – vynulován) podmiňovacího registru. Kombinace $(0000)_2$ označuje nepodmíněné provedení a kombinace $(0001)_2$ je u některých procesorů využívána pro jednoduché vkládání breakpointů. Zbývající dvě kombinace $(1110)_2$ a $(1111)_2$ jsou vyhrazeny pro budoucí použití.

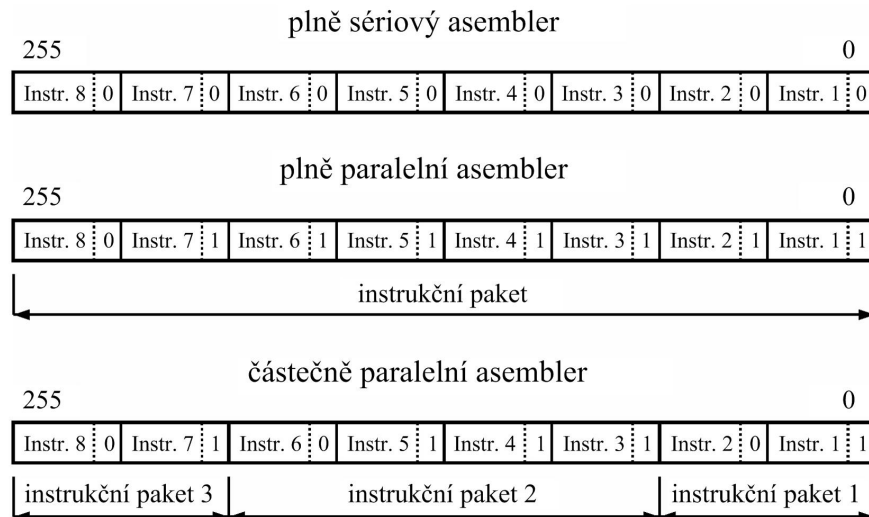
31	29	28	27	23	22	18	17	13	12	11	10	9	8	7	6	5	4	3	2	1	0
creg	z	dst	src1	src2	x	1	1	0	1	0	0	0	0	0	0	0	s	p			

Obr. 3.25: Příklad kódování instrukce **SMPY** do strojového kódu.

Následují bity **dst**, **src2**, **src1**, ve kterých jsou zakódovány operandy. Rozložení se může lišit pro jednotlivé typy instrukcí podle počtu operandů, nicméně 5 bitů pro jeden operand poskytuje $2^5 = 32$ kombinací. To je dostatečný počet pro zakódování libovolného registru A0 až A31, resp. B0 až B31. Ze které banky registrů jsou operandy vybírány je rozlišeno bitem 1 označeným **s** (**side**), který rozlišuje, pro kterou datovou cestu A nebo B je instrukce určena.

Aby bylo možné jednoduše přenášet hodnoty mezi bankami registrů, jeden zdrojový operand může být z opačné datové cesty než přísluší instrukci. V takovém případě se použije jedna z tzv. křížových sběrnice **1x** nebo **2x**, což ve strojovém kódu označuje bit 12 označený **x**.

Nejméně významný bit **p** (**parallel**) označuje, zda instrukce má být zpracovávána současně s následující instrukcí (nastaven) nebo ne (vynulován). Programová řídicí jednotka čte z paměti programu vždy 256bitové programové slovo. Programové slovo obsahuje 8 dílčích 32bitových instrukcí, které mohou být sdruženy do několika instrukčních paketů. Pokud je dílčí instrukce samostatná, tj. není sdružena s žádnou další instrukcí, je bit **p** vynulován. Pokud je instrukce první instrukcí instrukčního paketu, pak je bit **p** nastaven na 1 a nastaven je i u všech dalších instrukcí instrukčního paketu. Vynulován je bit **p** až u poslední instrukce instrukčního paketu. Přčtené programové slovo tak může obsahovat 8 instrukčních paketů složených z jediné instrukce (*plně sériový assembler*), jediný instrukční paket složený z 8 dílčích instrukcí (*plně paralelní assembler*) nebo několik instrukčních paketů složených z různého počtu dílčích instrukcí (*částečně paralelní assembler*), viz obr. 3.26. Instrukční paket však nemůže přesahovat hranice programového slova. To znamená, že poslední instrukce v programovém slově má bit **p** vždy nulový.



Obr. 3.26: Sdružování instrukcí do instrukčních paketů.

Sdružování instrukcí je provedeno ve výpisu assembleru dvojicí znaků „||“ uvedenou před instrukcí. Příkladem může být sdružení dvou instrukcí **SMPY** a **SADD**

```

SMPY .M1 *A8,A9,A10 ; první instrukce instrukčního paketu
|| SADD .L1 A10,A11,A11 ; instrukce sdružená s předchozí instrukcí

```

Současně jsou provedeny tyto operace:

- Vynásobí se obsah registru A8 a A9 jako zlomková čísla (instrukce **SMPY**) a součin se uloží do registru A10,
- současně se hodnota registru A10 přičte k registru A11 se saturací (instrukce **SADD**) a součet je uložen do A11.

Dvojice instrukcí prováděných současně je tak podobná instrukci **MAC** u procesorů s harvardskou architekturou. Je však nutné zdůraznit, že zatímco u **MAC** je ke střadači připočtena hodnota aktuálního součinu okamžitě, v tomto případě je připočtena hodnota předchozího registru A10 (předchozí součin) a nový součin se do registru A10 pouze uloží. Pro přičtení aktuálního součinu je tedy nutné opakovat celou dvojici instrukcí nebo jen instrukci **SADD**.

Příkladem sdružení 8 instrukcí do jednoho instrukčního paketu může být následující zdrojový kód:

```

SADD .L1 A16 , A17 , A7
|| MVK .S1 0x4000 , A16
|| SMPY .M1 A18 , A8 , A18
|| LDH .D1T1 *A9-- , A17
|| [!A0] ZERO .L2 B4
|| ADD .S2x A7 , B7 , B7
|| SMPYH .M2 B6 , B9 , B18
|| SUB .D2 B0 , 5 , B0

```

Při zpracování tohoto paketu se paralelně provedou následující operace:

- Jednotka L1 provede součet se saturací registrů A16 a A17 a výsledek uloží do registru A7.
- Jednotka S1 vloží do registru A16 16bitovou konstantu $(4000)_{16}$ (MVK).
- Jednotka M1 vynásobí ve zlomkovém formátu (SMPY) v doplňkovém kódu spodní 16bitové poloviny registrů A18 a A8 a 32bitový výsledek uloží do registru A18.
- Jednotka D1 přečte z adresy uložené v registru A9 16bitovou hodnotu (LDH) a uloží ji do registru A17.
- Po přenosu sníží hodnotu registru A9 o 2.
- Jednotka L2 vloží do registru B4 nulu (ZERO) ale pouze tehdy, když registr A0 je nulový ($[!A0]$). Pokud registr A0 obsahuje nenulovou hodnotu, jednotka neprovádí žádnou činnost.
- Jednotka S2 provede součet bez saturace (ADD) registrů A7 a B7 a výsledek vloží do registru B7.
- Jednotka M2 vynásobí ve zlomkovém formátu horní 16bitové poloviny (SMPYH) registrů B6 a B9 a 32bitový výsledek vloží do registru B18.
- Jednotka D2 provede rozdíl registru B0 bez saturace (SUB) a 5bitové konstanty $(5)_{10}$ a výsledek uloží do registru B0.

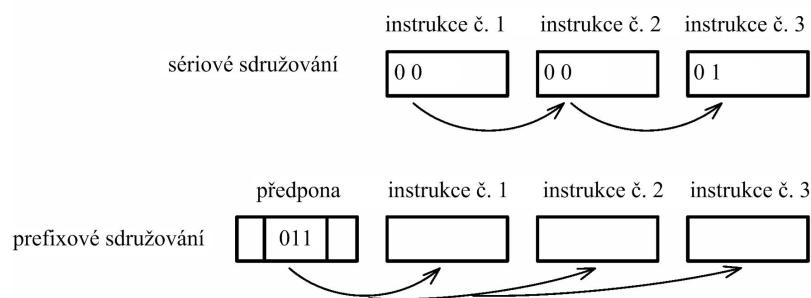
Instrukce nelze do instrukčního paketu sestavovat libovolně, ale existuje několik omezení, která musí být respektována. Omezení je však mnohem méně než v případě signálových procesorů s harvardskou architekturou a platí nezávisle pro všechny instrukce stejně. Všechny vycházejí z architektury signálového procesoru a z omezených zdrojů, zejména sběrnic, které jsou k dispozici [91]. Prvním a základním omezením je skutečnost, že v jednom instrukčním paketu nemohou být dvě instrukce pro stejnou funkční jednotku. Podobně nemohou být v jednom instrukčním paketu dvě instrukce, které zapisují výsledek operace do jednoho výstupního registru. Výjimkou z tohoto pravidla jsou podmíněné instrukce a instrukce, které vlivem zřetěženého zpracování instrukcí ukládají výsledek do registru se zpožděním několika hodinových cyklů. Předpokládejme, že v instrukčním paketu jsou dvě instrukce pro různé jednotky, ale obě zapisují výsledek operace do stejného registru. První instrukce nechtě je podmíněna nenulovou hodnotou některého podmínkového registru a druhá instrukce nechtě je podmíněna nulovou hodnotou téhož podmínkového registru. V tom případě bude provedena vždy jen jedna z této dvojice instrukcí (podle hodnoty podmínkového registru) a nemůže dojít ke konfliktu při zápisu do výstupního registru.

Signálové procesory s jádrem StarCore

Jádro SC140 obsahuje dvě adresovací jednotky AAU a čtyři datové jednotky ALU. Instrukční paket tedy může obsahovat 1 až 6 instrukcí prováděných paralelně. Z tohoto počtu však mohou být maximálně 4 instrukce pro jednotky ALU a maximálně 2 instrukce pro jednotky AAU. Instrukční paket může mít proměnnou délku, ale zakódování ve strojovém kódu se mírně odlišuje od způsobu použitého u jádra VelociTI. Jádro StarCore totiž používá dílčí instrukce délky 16 bitů, aby byla zachována malá paměťová náročnost programů. Nižší počet kombinací 16bitového kódu omezuje možnosti zakódování paralelního provádění, operandů, typu instrukce, atd. Proto je možné provádět ve strojovém kódu sdružování dílčích instrukcí do instrukčního paketu dvěma způsoby (obr. 3.27):

Sériové sdružování Informace o sdružení instrukcí je umístěna do dvou nejvíce významných bitů každé instrukce. Kombinace 00 znamená, že daná instrukce je sdružena s následující instrukcí. Kombinace 01 značí, že jde o poslední instrukci daného instrukčního paketu. Sériové sdružení je efektivnější z hlediska velikosti programového kódu, ale ne vždy je možné sériového sdružení použít. Tento způsob je podobný jako u jádra VelociTI. Z důvodu krátkého instrukčního slova však neumožňuje zakódovat podmínění instrukce nebo používání registrů D8 až D15.

Předponové (prefixové) sdružení Tento způsob sdružení vyžaduje použití dalšího slova na začátku instrukčního paketu. Maximální délka programové instrukce je 128 bitů. Pokud bude instrukční paket obsahovat maximální počet instrukcí, bude mít délku 96 bitů = 6 krát 16 bitů. Zbývá 32 bitů, které se používají pro jedno nebo dvě prefixová slova. Předponové sdružení se používá zvláště, když jsou používány registry D8-D15 a R8-R15, při využití podmíněných instrukcí IFT, IFF a IFA a dále při realizaci hardwarových cyklů. Počet instrukcí, příznaky podmínění instrukcí nebo příznaky hardwarových cyklů jsou zakódovány v prefixových slovech. Předponové sdružování tedy zvyšuje velikost programu, ale přesto je maximální délka programového slova poloviční (128 bitů) než u jádra VelociTI (256 bitů).



Obr. 3.27: Sériové a předponové (prefixové) sdružování instrukcí do instrukčních paketů.

V instrukčním paketu není uvedeno, která instrukce je určena pro kterou jednotku. Rozdělení mezi jednotky provede překladač a zakóduje je do strojového kódu. Nicméně přesto existují omezení, jak lze instrukce sdružovat. Všechna omezení jsou uvedena v pramenu [18].

Ve zdrojovém textu se sdružování provede zapsáním několika instrukcí na stejný řádek zdrojového souboru. Příklad instrukčního paketu s 4 dílčími instrukcemi může vypadat následovně:

```
MACR D0 , D1 , D7   AND D4 , D5       MOVE .L (R0)+NO , R6   ADDA R2 , R3
```

Tento instrukční paket provádí tyto operace:

1. Je proveden součin operandů v registrech D0 a D1 v zlomkovém doplňkovém kódu. Výsledek je odečten od obsahu registru D7. Získané číslo je zaokrouhleno a uloženo do registru D7. Instrukce je určena pro jednotku DALU.

2. S operandy v registrech D4 a D5 je realizována logická operace AND a výsledek je uložen do registru D5 Operaci provádí jednotka DALU.
3. Obsah paměti na adrese určené registrem R0 je uložen do registru R6. Po přesunu je zvýšena adresa v registru R0 o čtyřnásobek hodnoty uložené v registru R0. Operaci provádí jednotka AGU.
4. Obsah registru R2 je přičten k obsahu registru R3 a výsledek je uložen zpět do registru R3. Operaci provádí jednotka AGU.

Další příklad ukazuje sdružení 6 instrukcí do instrukčního paketu. První čtyři instrukce jsou určeny pro jednotku DALU a poslední dvě provádí jednotky AGU.

```
MAC D0,D1,D7  MAC D3,D4,D6  MACR D0,D2,D5  ADR D3,D4  MOVE.L (R0)+R0,R6  MOVE.L D0,R1
```

Řádky s instrukčními pakety s větším počtem instrukcí mohou být velmi dlouhé a jsou špatně čitelné na obrazovce monitoru i při výpisu na tiskárně. Např. v této práci musely být vysázeny menším písmem, aby se vešly na šířku stránky. Zápis dílčích instrukcí instrukčního paketu může být proveden vedle sebe nebo za sebou uzavřený do hranatých závorek:

```
[
MAC      D0 , D1 , D7
MAC      D3 , D4 , D6
MACR     D0 , D2 , D5
ADR      D3 , D4
MOVE.L   (R0)+R0 , D2
MOVE.L   D0 , R1
]
```

V tomto instrukčním paketu jsou provedeny tyto operace:

- Je proveden součin v zlomkovém doplňkovém kódu obsahu registrů D0 a D1 a výsledek je přičten k obsahu registru D7.
- Je provedena stejná operace jako v předchozí dílčí instrukci s registry D3, D4 a D6.
- Je proveden součin v zlomkovém doplňkovém kódu obsahu registrů D0 a D2. Výsledek je přičten k obsahu registru D5. Nakonec je obsah registru D5 zaokrouhlen.
- Je sečten obsah registrů D3 a D4 a po zaokrouhlení je výsledek uložen do registru D4.
- Obsah datové paměti na adrese umístěné v registru R0 je přenesen do registru D2. Po provedení přesunu je obsah R0 zvýšen o čtyřnásobek obsahu registru R0.
- Obsah registru D0 je přenesen do registru R1.

3.4 Superskalární architektura

Jinou možností, než sdružování více operací do jedné instrukce, je použití přístupu běžně používaného u mikroprocesorů pro všeobecné použití, tj. provést jednu operaci v jednom instrukčním cyklu a číst přitom skupinu těchto instrukcí současně (paralelně). Tato metoda využívá paralelního zpracování instrukcí a také se označuje jako metoda čtení s více přístupy (Multi-Issue Processing). Jedna instrukce (instrukční paket) je složena z několika částí, které současně ovládají skupinu funkčních jednotek. Instrukce pro dílčí

funkční jednotky jsou jednoduché a jednoznačné, a proto kompilační programy nejsou tak komplikované jako pro harvardskou architekturu. V předchozí části popsaná architektura typu VLIW (viz obr. 3.15) je jednou ze dvou koncepcí čtení s více přístupy. Druhý typ architektury je nazýván superskalární architektura [12]. Je to koncepce používaná ve většině mikroprocesorů pro všeobecné použití (zvláště firmou Intel při vývoji mikroprocesorů Pentium). U signálových procesorů je využívána jen výjimečně (signálový procesor LSI40xx od firmy LSI Logic Inc., Milpitas Calif.) [15].

Tyto dvě koncepce se hlavně liší tím, jak je provedeno seskupení instrukcí pro paralelní provádění operací. V architektuře typu VLIW musí programátor buď v assembleru nebo při kompilaci z vyššího programovacího jazyka určit, které operace budou realizovány paralelně. Naopak u superskalárních procesorů je navržen speciální hardware uvnitř procesoru (Schedule Unit), který sám sdruží operace zpracovávané paralelně do jedné instrukce. Sdružení závisí na několika okolnostech: zda a které registry lze použít, zda jsou dostupná všechna vstupní data instrukce apod. Tento problém se řeší za běhu programu. Jinými slovy, superskalární procesor přenáší zodpovědnost za paralelní sdružování instrukcí z programátora nebo kompilačního programu přímo na hardware mikroprocesoru.

Superskalární architektura umožňuje binární kompatibilitu mezi generacemi procesorů. Na rozdíl od toho různé generace architektur signálových procesorů typu VLIW z principu nemusí být binárně kompatibilní, neboť informace o sdružování instrukcí se právě vyskytuje v tomto binárním kódu, který je vytvořen až programátorem. Zlepšení vlastností signálového procesoru typu VLIW, které bude podporovat provedení více instrukcí paralelně, bude pravděpodobně vyžadovat opakovanou kompilaci programu pro novou generaci signálových procesorů s novou architekturou.

Vážným nedostatkem superskalární koncepce pro aplikace číslicového zpracování signálů je skutečnost, že programátor přesně neví, které instrukce procesor seskupí pro paralelní zpracování, a tudíž není schopen odhadnout, kolik instrukčních cyklů bude nutné použít pro zpracování daného segmentu programu. Procesor může stejný segment programu seskupit při každém opakování různě, např. když poprvé provádí smyčku, může seskupit instrukce jedním způsobem a podruhé je může pro následné iterace seskupit jiným způsobem. Neznalost odhadu prováděcího času se může stát vážným problémem, jestliže program musí provést časově napjatou operaci v reálném čase. Programátor může ovšem vzít v úvahu ten nejhorší případ a podle toho programovat, ale pak nemusí být plně využit výkon procesoru. Následkem toho, že lze obtížně zaručit předpověď délky trvání daného programu, je obtížné tento program optimalizovat. U aplikací číslicového zpracování signálu, které jsou výpočtově náročné při značném omezení paměti a příkonu, je optimalizace programu kritickým bodem. Programátor, který nemůže snadno odhadnout vliv změny programu na dobu jeho trvání, těžko odhadne, zda optimalizace skutečně přinese zlepšení vlastností. V tom případě se optimalizace stane věcí pokusu a omylu a výsledek bude pravděpodobně dosti vzdálen od optimálního stavu.

3.5 Architektura paralelních systémů

Základním myšlenkou zvyšování výpočetního výkonu je zvětšování podílu paralelního zpracování. Nejznámější třídění paralelních systémů zavedl M. J. Flynn v roce 1966

[16]. Systémy dělí jednak podle počtu programů řešených současně na:

SI (Single Instruction Stream) jeden řešený program,

MI (Multiple Instruction Stream) více řešených programů,

a jednak podle počtu datových souborů zpracovávaných současně na:

SD (Single Data Stream) jeden zpracovávaný tok dat,

MD (Multiple Data Stream) více zpracovávaných toků dat.

Kombinací těchto možností vzniknou čtyři kategorie paralelních systémů:

SISD: Klasický von Neumannův počítač s jedním programem a jedním sériově přiváděným tokem dat.

MISD: Hypotetická kombinace několika programů zpracovávající jeden tok dat. Tento typ se zatím nepoužívá a je nesprávně zaměňován se systémem se zřetězeným zpracováním instrukcí (pipelining). Tato záměna je nedůsledná v tom, že pomocí zřetězeného zpracování instrukcí je zpracováván pouze jeden program.

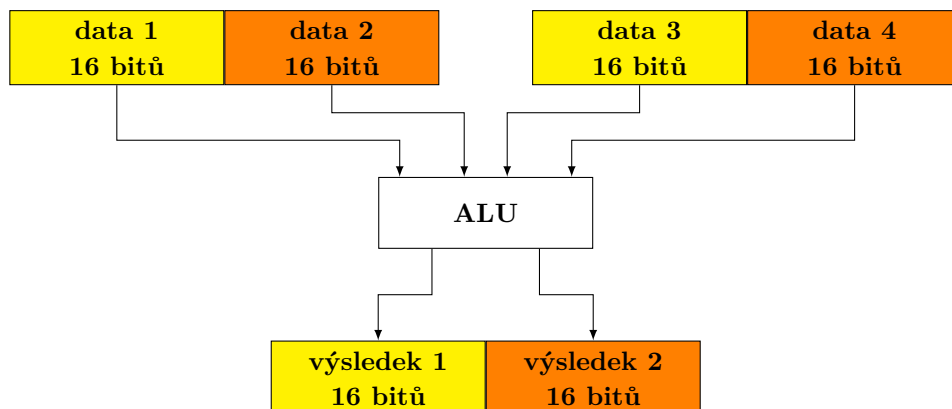
SIMD: Větší počet funkčních jednotek pracuje na řešení téhož programu. Všechny jednotky provádí současně tutéž instrukci, ale každá s jinými daty.

MIMD: Obecný typ paralelního systému, který obsahuje jednotky již tak samostatné, že každá z nich plní samostatný program a přitom zpracovávají jiná data.

U moderních signálových procesorů se uplatňuje především koncepce typu SIMD (Single-Instruction, Multiple-Data). Tento přístup dovolí procesoru provádět stejnou operaci pomocí jedné instrukce pro skupinu nezávislých dat. Procesor na principu SIMD může pracovat s dlouhými registry (např. 64 bitů) jako násobky menších datových slov (např. čtyřikrát 16 bitů), pro něž provádí stejnou operaci a generuje více nezávislých výstupů. Princip typu SIMD (Single Instruction Multiple Data) se stal populární v 90. letech minulého století jako prostředek pro zvýšení výkonu tehdy existujících CPU (Central Processor Unit) architektur pro vektorové operace, které jsou často využívány v multi-mediálních aplikacích a při zpracování signálu v rámci rozšíření instrukční sady MMX (Multi Media Extension) a později SSE (Streaming SIMD Extensions). Tento přístup významně zvyšuje rychlost procesorů pro algoritmy pracující s vektory, kde jsou operace přirozeně přizpůsobené pro paralelní zpracování. Právě tento princip umožnil mikroprocesorům pro všeobecné použití, jako je Intel Pentium III a Motorola PowerPC G4, konkurovat signálovým procesorům s pohyblivou řádovou čárkou co do rychlosti při zpracování algoritmů číselového zpracování signálu. Ačkoliv kořeny principu typu SIMD jsou u mikroprocesorů pro všeobecné použití, využití zpracování typu SIMD v signálových procesorech se nyní rozšířilo a zcela běžně se používá v nejnovějších procesorech. Úroveň podpory pro operace typu SIMD se však stále dosti liší.

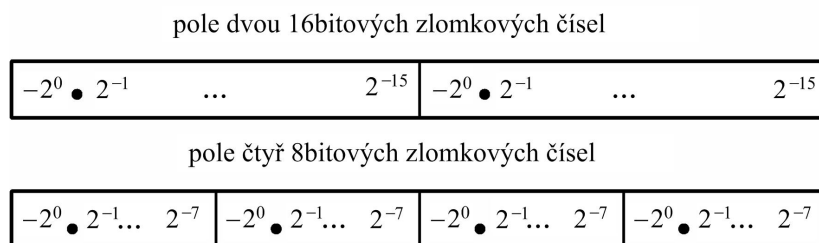
Základní myšlenku využití principu typu SIMD lze demonstrovat na příkladu signálového procesoru firmy Analog Devices. Při tvorbě signálového procesoru ADSP-2116x (s označením Hammerhead) firma modifikovala základní architekturu signálového procesoru s pohyblivou řádovou čárkou ADSP-2106x (Sharc) přidáním druhé skupiny výkonných jednotek, které kopírují původní návrh. Každá z obou skupin v ADSP-2116x obsahuje jednotku MAC, jednotku ALU, jednotku pro posun a každá skupina disponuje vlastní sestavou operačních registrů. Signálový procesor ADSP-2116x může číst jednu instrukci a zpracovat ji paralelně ve dvou datových cestách používajících různá data, čímž u některých algoritmů lze dosáhnout zdvojnásobení výkonu. Způsob výpočtu s využitím

principu typu SIMD je vidět na obr. 3.28. Dvě a dvě vstupní 16bitová slova jsou sdružena do dvou 32bitových operandů. Tyto dva operandy zpracovává daná funkční jednotka a ve výstupním registru je uložen 32bitový výsledek operace. Prvních 16 bitů je výsledek operace pro první dvojici a druhých 16 bitů výsledku odpovídá druhé dvojici vstupních operandů.



Obr. 3.28: Princip spojení vstupních datových operandů pro paralelní zpracování dat metodou typu SIMD.

Ze signálových procesorů s pevnou řádovou čárkou princip SIMD využívá jádro VelocityTI.2 řady TMS320C64xx firmy Texas Instruments. Do 32bitových registrů lze uložit i pole – datový paket dvou 16bitových nebo čtyř 8bitových hodnot. Formát uložení datového paketu do 32bitového registru je zobrazen na obr. 3.29. Tečka • ukazuje umístění řádové čárky, která odděluje znaménkový bit od zlomkové části čísla. Jednotky jádra VelocityTI.2 pak zpracovávají jednotlivé části datového paketu samostatně. Příkladem může být instrukce MPY, která násobí dolní poloviny dvou různých 32bitových datových registrů (tj. 16 × 16 bitů) a 32bitový výsledek ukládá do datového registru. Instrukce MPYH násobí horní 16bitové části dvou 32bitových datových registrů, instrukce MPYHL a MPYLH střídavě násobí horní polovinu jednoho registru s dolní polovinou druhého registru a naopak. Všechny uvedené instrukce provádějí násobení celých čísel. Pokud násobíme



Obr. 3.29: Příklad formátů datových paketů zlomkových čísel v doplňkovém kódu, které lze uložit do datových registrů.

čísla zlomková s pevnou řádovou čárkou musíme použít obdobné instrukce *SMPY*, *SMPYH*, *SMPYHL* a *SMPYLH*, které zaručí správné umístění řádové čárky ve výsledku. Uvedené instrukce nejsou nové pro jádro *VelociTI.2*, ale podporovalo je již jádro *VelociTI* signálových procesorů řady *TMS320C62xx* a *TMS320C67xx*. Byly používány při násobení více bitových čísel (16x32 bitů, 32x32bitů). Pro jádro *VelociTI.2* byly doplněny instrukce pro paralelní zpracování dat, které zpracují všechny části datového paketu současně. Instrukce násobení *MPY2* provádí násobení horních i dolních polovin dvou registrů současně a může tak nahradit dvojici instrukcí *MPY* a *MPYH*. Výsledek v podobě 64bitového datového slova uloží do datového paketu (dvojice 32bitových registrů). V lichém datovém registru bude součin horních polovin, v sudém registru bude součin dolních polovin. Podobně instrukce *MPY4* paralelně vynásobí všechny čtyři dvojice 8bitových čísel a výsledek opět uloží do datového paketu. Dále jednotky *L* a *S* mohou provádět operace rotace o 8 bitů vlevo nebo vpravo a skládání nebo rozkládání dvou 16bitových nebo čtyř 8bitových hodnot do jednoho 32bitového datového paketu. Na takto vytvořeném paketu mohou obě jednotky provádět aritmetické operace podle principu *SIMD*. Jednotka *L* navíc umožňuje nalézt maximum nebo minimum z hodnot v datovém paketu. Může dále zapisovat 5bitovou konstantu do registru. Jednotka *S* je rozšířena o operace komparace datových paketů a o operaci bitového posunu vpravo datového paketu složeného ze dvou 16bitových hodnot. Existují tak instrukce pro sčítání nebo odčítání datových paketů (např. *ADD2*, *ADD4*, *SUB2*, *SUB4*), komparaci (např. *CMPEQ2*, *CMPEQ4*), bitové posuny (*SHR2*), skládání čísel do datového paketu (např. *PACK2*, *PACKL4*) nebo naopak rozklad datového paketu na jednotlivé prvky (*UNPKHU4*, *UNPKLU4*).

Architektura typu *SIMD* má ovšem i svá omezení. Efektivnost využití metody typu *SIMD* záleží na skutečnosti, zda vstupní data lze zpracovat paralelně. U algoritmů, které jsou svou podstatou zpracovatelné sériově – například operace, které využívají výsledky jedné operace jako vstup pro následující operaci – se nedoporučuje princip *SIMD* využít. Programátoři musí navrhnout uspořádání dat v paměti tak, aby zpracování metodou *SIMD* proběhlo plnou rychlostí. Data lze například uspořádat v paměti současně po čtyřech operandech. Často bude při využití tohoto přístupu nutné přepsat stávající algoritmus. Procesor musí totiž provádět další dodatečné instrukce pro přeorganizování a čtení dat připravených pro provedení instrukcí *SIMD* nebo sečtení všech mezivýsledků. Tyto požadavky na přeorganizování dat nebo přepsání algoritmu zmenšují výhody této metody.

3.6 Shrnutí vlastností architektur

Z uvedeného vyplývá, že v případě procesorů s harvardskou architekturou je struktura jádra procesorů různých výrobců velmi podobá. Základem je využití instrukce násobení s akumulací, kterou podporují všechny výrobní řady. Společným rysem jsou také paralelní přesuny prováděné současně s aritmetickou instrukcí díky oddělené jednotce generace adres. Výjimkou je řada signálových procesorů *TMS320C551x* firmy Texas Instruments, u které je možné operandy aritmetické instrukce číst přímo z paměti bez uložení do vstupních registrů *DALU*.

Velmi podobné jsou také možnosti nepřímého adresování. Jednotky generace adres obsahují několik adresovacích registrů, jejichž obsah může být použit jako adresa. Během adresace nebo po ní je možné hodnotu registru zvýšit nebo snížit o 1, konstantu nebo hodnotu registru posunu. Všechny řady procesorů podporují modulo adresování, u většiny ani nemusí být délka bloku rovna celočíselné mocnině 2. Naopak začátek bloku (počáteční adresa) musí být celočíselným násobkem $2^K \geq M$, $K \in \mathbb{N}$, kde M je délka bloku. Jednotka generace adres podporuje často i bitově reverzní adresování využitelné při implementaci rychlého algoritmu Fourierovy transformace.

Z podobných vlastností architektury pak vyplývají podobné postupy při optimalizaci struktur výpočetních algoritmů, které jsou uvedeny v kapitole 4.

Naproti tomu u architektury signálových procesorů s velmi dlouhým slovem je situace zcela odlišná. Někteří výrobci šli cestou několikanásobného zopakování ověřeného jádra harvadské architektury, takže procesory s jádrem StarCore firmy NXP obsahují čtyři paralelní DALU velice podobné DALU procesorů DSP56300 s harvadskou architekturou. Všechny 4 DALU mohou pracovat současně, takže výpočetní výkon teoreticky vzroste také 4násobně. V praktických aplikacích však bude nárůst výpočetní výkonu nižší, záleží na struktuře výpočetního algoritmu. Sekvenční algoritmy je obtížné rozdělit mezi čtyři nezávislé operace pro 4 DALU a výpočetní výkon se v extrémním případě vůbec nezvýší.

Naproti tomu firma Texas Instruments navrhla zcela nové jádro založené na 8 paralelně pracujících jednotkách, podobně jako u jádra StarCore, ale každá z jednotek je optimalizována na jinou činnost. Teoreticky je tak možné paralelizovat i sekvenční algoritmy, k čemuž přispívá mírně odlišný přístup k řetězenému zpracování instrukcí. Postupy při optimalizaci se tak zásadně odlišují od postupů pro výše uvedená jádra, lze např. využít zřetězeného zpracování cyklů (*software pipelining*), které jsou uvedeny v kapitole 4.

4 IMPLEMENTACE ČÍSLICOVÝCH SYSTÉMŮ V SIGNÁLOVÝCH PROCESORECH

Podobně jako se liší postupy optimalizace struktur výpočetních algoritmů pro harvardskou architekturu a architekturu s velmi dlouhým slovem, liší se postupy optimalizace i pro lineární časově invariantní systémy s konečnou a nekonečnou impulsní charakteristikou. Systémy s konečnou impulsní charakteristikou neobsahují zpětné vazby a proto se mnohem jednodušeji rozdělují do paralelních cest. Naproti tomu systémy s nekonečnou impulsní charakteristikou mají vlivem zpětných vazeb povahu sekvenčních algoritmů, které je obtížné paralelizovat.

4.1 Číslicové systémy s konečnou impulsní charakteristikou

Protože systém s konečnou impulsní charakteristikou FIR (Finite Impulse Response) má přenosovou funkci [47]

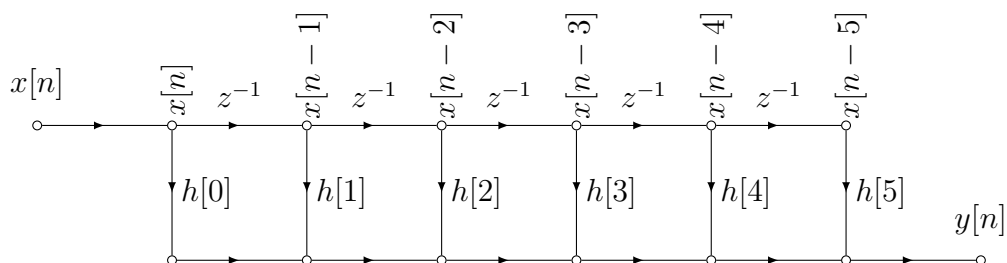
$$H(z) = \frac{\sum_{m=0}^{N-1} b_m z^m}{a_{N-1} z^{N-1}}, \quad (4.1)$$

jsou kanonické struktury implementace jednodušší než u systémů s nekonečnou impulsní charakteristikou IIR (Infinite Impulse Response). Ve strukturách uvedených v části 1.2 budou vynechány zpětnovazební větve (viz obr. 4.1). Díky jedinému členu ve jmenovateli přenosová funkce přejde do tvaru

$$H(z) = \sum_{m=0}^{N-1} \frac{b_m}{a_{N-1} z^{N-1+m}} = \sum_{m=0}^{N-1} h[m] z^{-m} \quad (4.2)$$

což je definiční vztah přímé \mathcal{Z} transformace. Vzorky impulsní charakteristiky jsou pak přímo dány koeficienty přenosové funkce

$$h[m] = \frac{b_{N-1+m}}{a_{N-1}}. \quad (4.3)$$

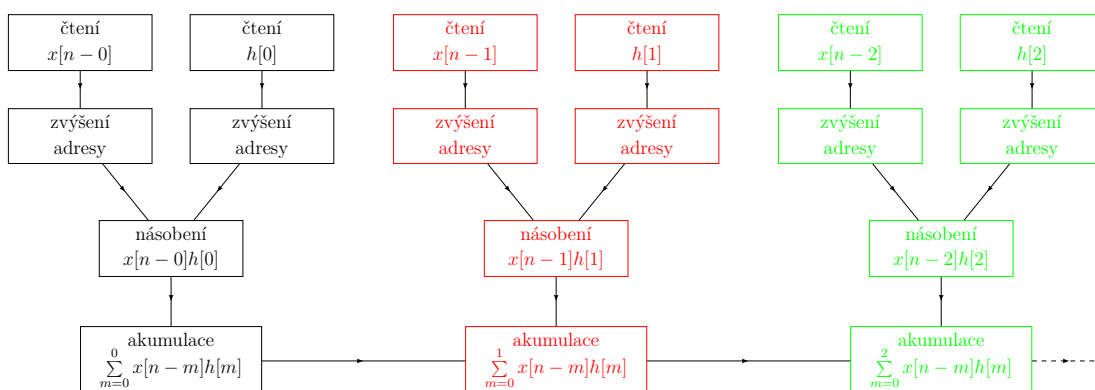


Obr. 4.1: Graf signálových toků realizující algoritmus číslicového systému s konečnou impulsní charakteristikou délky $N = 6$, který je definován diferenční rovnicí (4.4).

Číslicový systém s konečnou impulsní charakteristikou se pak nejčastěji realizuje pomocí lineární konvoluce

$$y[n] = \sum_{m=0}^{N-1} x[n-m]h[m]. \quad (4.4)$$

Celý výpočet lze rozdělit do několika kroků, jejichž symbolické znázornění včetně vzájemných závislostí je na obr. 4.2. Nejprve je nutné přečíst vzorek vstupního signálu $x[n]$ a impulsní charakteristiky $h[0]$ z paměti do DALU (Data Arithmetic Logic Unit). Po přečtení je posunuta adresa pro čtení tak, že ukazuje na následující vzorek $x[n-1]$ a $h[1]$. Přečtené vzorky se vynásobí $x[n]h[0]$ a uloží do střadače. Následuje čtení dalších vzorků $x[n-1]$ a $h[1]$, posun adres, vynásobení $x[n-1]h[1]$ a součin je připočten k předchozímu součinu $\sum_{m=0}^1 x[n-m]h[m]$. Takto se postupuje dokud není přečteno všech N vzorků vstupního signálu a impulsní charakteristiky, nejsou provedeny všechny součiny a součiny sečteny. Je zřejmé, že nelze provádět násobení, dokud nejsou přečteny vzorky vstupního signálu a impulsní charakteristiky, a není možné provádět akumulaci, dokud není proveden součin. Tyto vztahy jsou na obr. 4.2 naznačeny hranami mezi jednotlivými bloky. Zároveň však se čtením dalších vzorků není nutné čekat na dokončení akumulace. Teoreticky je možné zahájit čtení dalších vzorků ihned po zvýšení adresy, pokud budou uloženy do jiných registrů DALU. V jakém pořadí budou jednotlivé operace provedeny závisí na architektuře signálového procesoru.



Obr. 4.2: Symbolické znázornění dílčích operací při výpočtu lineární konvoluce podle obr. 4.1 a jejich vzájemný vztah.

4.1.1 Signálové procesory s harvardskou архитектурou

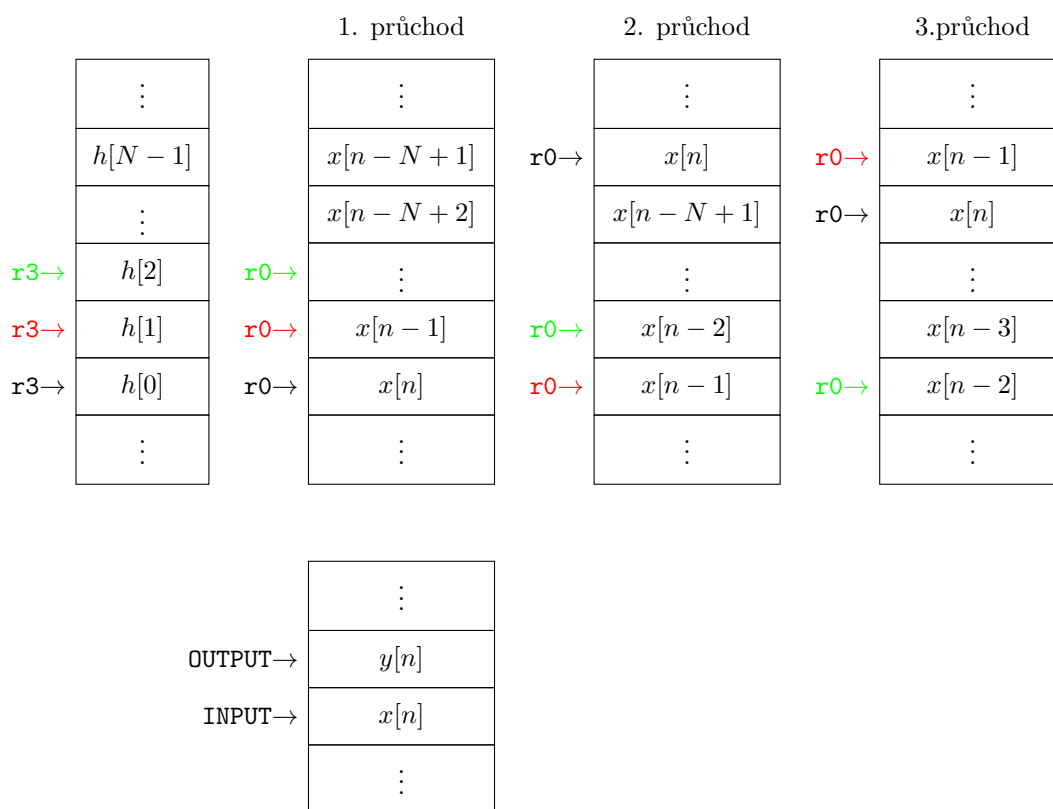
Při implementaci lineární konvoluce v signálovém procesoru s harvardskou архитектурou lze několik operací sloučit dohromady a provádět je paralelně. Především využijeme možností jednotky generace adres (viz část 3.2.2). Zpoždění vzorků signálu realizujeme pomocí kruhové paměti s využitím hardwarové podpory modulo adresování. U některých procesorů musí být délka bloku pro modulo adresování rovna celočíselné mocnině 2, ale délka impulsní charakteristiky tuto podmínku nemusí splňovat. V takovém případě je délka bloku nastavena na hodnotu $2^K > N, K \in \mathbb{N}$ nejmenší celočíselné mocnině 2

větší než délka impulsní charakteristiky. Algoritmus zůstane funkční, pouze se zvýší paměťové nároky. Dále bylo v části 3.2.2 ukázáno, že zvýšení adresy lze provádět současně s přenosem hodnot. Proto první dva bloky (čtení $x[n - m]$ a $h[m]$) a zvýšení adresy lze sloučit do jediné operace.

Podobně díky podpoře násobení s akumulací lze sloučit násobení $x[n - m]h[m]$ a jeho akumulaci do jediné operace typu MAC (Multiply and Accumulate).

A konečně díky paralelním přenosům (viz část 3.2.4) lze operace v obou uvedených blocích také provést současně.

Konkrétní implementace pro signálový procesor DSP56F8367 s harvardskou architekturou tak může vypadat následovně. Vzorok impulsní charakteristiky jsou umístěny v paměti a adresa prvního vzorku $h[0]$ je uložena do adresovacího registru $r3$. Podobně pro zpožděné vzorky vstupního signálu je vyhrazena část paměti a počáteční adresa je uložena do adresovacího registru $r0$. Pro registr je nastaveno modulo adresování délky $2^K > N, K \in \mathbb{N}$. Vstupní vzorek $x[n]$ je v rámci obsluhy přerušování A/D převodníku uložena do paměti na adresu $INPUT$, a v rámci obsluhy přerušování D/A převodníku je výstupní vzorek $y[n]$ přečten z adresy $OUTPUT$. Ilustrační rozmístění vzorků v paměti je na obr. 4.3.



Obr. 4.3: Ilustrační rozložení paměti při implementaci systému typu FIR v signálovém procesoru s harvardskou architekturou.

Na začátku výpisu 4.1 je vstupní vzorek přečten z adresy $INPUT$ a uložen do registru $y1$. Odtud je následně uložen do paměti dat na adresu $r0$. Zdánlivě nesmyslný přesun je nutný, protože kruhová paměť adresovaná $r0$ mění svůj začátek (viz dále a rozložení

Výpis 4.1: Příklad implementace systému typu FIR v signálovém procesoru s hardvardskou architekturou.

```

130  _FILT
131      ; nacteni vstupniho vzorku
132      move.w    X:(INPUT),y1
133          ; y1 <-- x[n]
134      move.w    y1,X:(r0)
135          ; X:(r0) <-- x[n]
136
137      clr      a    X:(r0)+,y1    X:(r3)+,x0
138          ; a <-- 0
139          ; y1 <-- x[n-N+1+m]
140          ; r0 <-- &x[n-N+1+m]
141          ; x0 <-- h[N-1-m]
142          ; r3 <-- &h[N-1-m]

```

paměti při 2. a 3. průchodu na obr. 4.3). Dále je vynulován střadač *a*. V prvním paralelním přenosu je do registru *y1* uložen nový vzorek $x[n]$ z kruhové paměti a je zvýšen registr *r0* o 1. Obsahuje tak adresu následujícího vzorku $x[n-1]$. Současně ve druhém paralelním přenosu je přečten koeficient $h[0]$ a registr *r3* je zvýšen o 1. Obsahuje tak adresu následujícího koeficienty $h[1]$. Změna obsahu adresovacích registrů je na obr. 4.3 znázorněno červeně.

Dále instrukce `rep #ORDER` způsobí opakování následující instrukce `ORDER`krát. Opakování je podporováno hardwarem signálového procesoru, takže čítač, adresa začátku cyklu, apod. jsou uloženy ve speciálních registrech procesoru. Během opakování je dokonce vyřazeno čtení instrukcí z paměti, neboť instrukce je načtena v registru řadiče programu, a zakázáno přerušování.

Instrukce `mac x0,y1,a` vynásobí obsahy registru *x0* ($h[0]$) a *y1* ($x[n]$) a součin připočte ke střadači *a* (0). Střadač *a* tak bude obsahovat hodnotu $x[n]h[0]$. V prvním paralelním přenosu přečte do registru *y1* vzorek $x[n-1]$ a zvýší *r0* o 1, takže obsahuje adresu $x[n-2]$.

Výpis 4.1: Příklad implementace systému typu FIR v signálovém procesoru s hardvardskou architekturou (pokračování).

```

144      rep      #ORDER
145      mac      x0,y1,a    X:(r0)+,y1    X:(r3)+,x0
146          ; a <-- h[m]*x[n-m]
147          ; y1 <-- x[n-N+1+m]
148          ; r0 <-- &x[n-N+1+m]
149          ; x0 <-- h[N-1-m]
150          ; x0 <-- &h[N-1-m]
151  _ENDFILT

```

V druhém paralelním přenosu přečte do registru $x0$ koeficient $h[1]$ a zvýší $r3$ o 1, takže obsahuje adresu $h[2]$. Změna obsahu adresovacích registrů je na obr. 4.3 znázorněno zeleně.

Při dalším opakování instrukce `mac x0,y1,a` se vynásobí obsahy registru $x0$ ($h[1]$) a $y1$ ($x[n-1]$) a součin se připočte k obsahu střadače a ($x[n]h[0]$). Střadač a tak bude obsahovat hodnotu $x[n]h[0]+x[n-1]h[1]$. V prvním paralelním přenosu přečte do registru $y1$ vzorek $x[n-2]$ a zvýší $r0$ o 1, takže obsahuje adresu $x[n-3]$. V druhém paralelním přenosu přečte do registru $x0$ koeficient $h[2]$ a zvýší $r3$ o 1, takže obsahuje adresu $h[3]$. Změna obsahu adresovacích registrů je na obr. 4.3 znázorněna červeně.

Při $N-1$ opakování instrukce `mac x0,y1,a` se vynásobí obsahy registru $x0$ ($h[N-2]$) a $y1$ ($x[n-N+2]$) a součin se připočte k obsahu střadače a . Střadač a tak bude obsahovat hodnotu $\sum_{m=0}^{N-2} x[n-m]h[m]$. V prvním paralelním přenosu přečte do registru $y1$ vzorek $x[n-N+1]$ a zvýší $r0$ o 1, takže by ukazoval za poslední vzorek. Díky modulo adresování však bude registr nastaven na začátek bloku modulo a a ukazuje tedy na vzorek $x[n]$ na začátku modulo bloku. V druhém paralelním přenosu přečte do registru $x0$ koeficient $h[N-1]$ a zvýší $r3$ o 1, takže opět ukazuje za poslední koeficient. U registru $r3$ ale není nastaveno modulo adresování (ani nelze) a tak jeho hodnota zůstává nastavena za adresu posledního koeficientu $h[N-1]$.

Akumulace posledního násobení $x[n-N+1]h[N-1]$ je instrukcí `macr` provedena se zaokrouhlením do formátu $Q0.15$. Střadač a tak obsahuje 16bitový vzorek výstupního signálu.

Výpis 4.1: Příklad implementace systému typu FIR v signálovém procesoru s hardvardskou architekturou (pokračování).

152	<code>macr</code>	<code>x0,y1,a</code>	<code>X:(r0)+n,y1</code>	<code>X:(r3)+n3,c</code>
153	<code>;</code>	<code>a</code>	<code><--</code>	<code>zaokr(\sum h[m]*x[n-m])</code>
154	<code>;</code>			

V prvním paralelním přenosu je k registru $r0$ připočtena hodnota registru posunu, která byla na začátku programu nastavena na $n = -1$. Měl by tak obsahovat adresu před začátkem bloku stavových proměnných. Díky modulo adresování však bude hodnota registru nastavena na adresu konce bloku, tj. adresu vzorku $x[n-N+1]$. Začátek kruhové paměti se virtuálně posune a při dalším opakování bude na řádku 134 nejstarší vzorek $x[n-N+1]$ přepsán novým vzorkem $x[n]$. Zároveň dojde k logickému zpoždění všech dalších vzorků v kruhové paměti $x[n] \rightarrow x[n-1]$, $x[n-1] \rightarrow x[n-2]$, atd. Hodnota přečtená do registru $y1$ se nikdy nepoužije. Čtení je pouze kvůli aktualizaci registru $r0$.

Podobně ve druhém paralelním přenosu je k registru $r3$ připočtena hodnota registru posunu, která byla na začátku programu nastavena na $n3 = -(\text{ORDER}+1) = -N$. Registr $r3$ tak opět obsahuje adresu prvního koeficientu $h[0]$. Aby bylo možné přičítat hodnotu $n3$, musí být čtení provedeno do registru c (viz tab. 3.5). Hodnota přečtená do registru c není nikdy použita. Čtení je provedeno pouze kvůli aktualizaci registru $r3$. Nastavení adresovacích registrů $r0$ a $r3$ před zahájením 2. průchodu programem je na obr. 4.3 naznačeno ve třetí tabulce zleva. Při dokončení dalšího průchodu dojde k posunutí začátku kruhové paměti o další vzorek, jak je naznačeno na obrázku zcela vpravo.

Na závěr je vzorek výstupního signálu ze střadače a uložen do paměti dat na adresu OUTPUT. Odtud by se v rámci obsluhy přerušení D/A převodníku přenesl do převodníku a na výstup. Instrukce `jmp` provede skok na návěští `_FILT` a celý cyklus se opakuje.

Je zřejmé, že hlavní část programu, tj. výpočet konvoluce, se odehrává mezi řádky 132, kdy je přečten vzorek vstupního signálu a 157, kdy je uložen vzorek výstupního signálu. Jádro přitom tvoří instrukce násobení s akumulací a paralelními přenosy na řádku 145, která je opakována $N - 1$ instrukcí `rep`. Tato instrukce maximálně využívá možnosti DALU signálového procesoru. Další snížení výpočetní náročnosti tak nelze očekávat. Odhad celkové náročnosti by byl $(N - 1) + 6$ instrukcí, kde N je délka impulsní charakteristiky.

Výpis 4.1: Příklad implementace systému typu FIR v signálovém procesoru s hardvardskou architekturou (pokračování).

155
156
157

```

    move .w      a , X : ( OUTPUT )
    jmp      _FILT

```

4.1.2 Využití symetrie impulsní charakteristiky

U systémů s konečnou impulsní charakteristikou je impulsní charakteristika často symetrická nebo antisymetrická

$$h[n] = \pm h[N - 1 - n]. \quad (4.5)$$

V takovém případě je totiž skupinové zpoždění konstantní a nedochází k fázovému zkreslení. Ve starší literatuře jsou uváděny struktury grafů signálových toků, které tuto symetrii využívají.

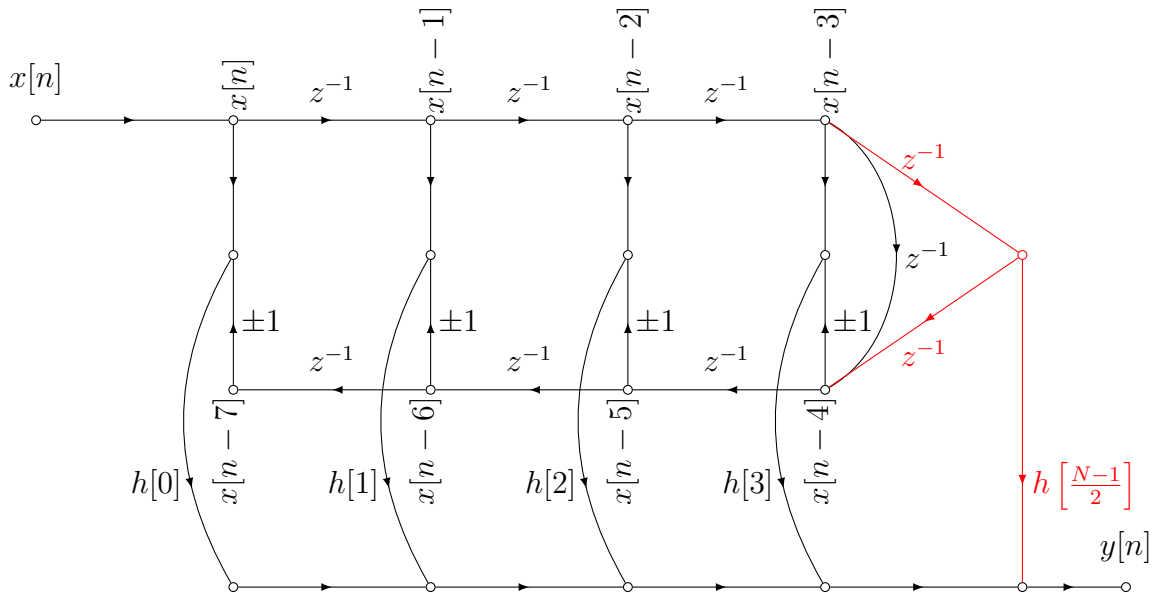
Pokud je impulsní charakteristika symetrická, znamená to, že první vzorek $x[n]$ a poslední vzorek $x[n - N + 1]$ jsou násobeny koeficientem se stejnou hodnotou

$$y[n] = \begin{cases} \sum_{m=0}^{\frac{N-1}{2}} (x[n - m] + x[n - N + 1 + m]) h[m] & \text{pro } N \text{ sudé} \\ x \left[n - \frac{N-1}{2} \right] h \left[\frac{N-1}{2} \right] + \sum_{m=0}^{\frac{N-3}{2}} (x[n - m] + x[n - N + 1 + m]) h[m] & \text{pro } N \text{ liché} \end{cases} \quad (4.6)$$

Proto je možné nejprve sečíst odpovídající vzorky a teprve součet násobit koeficientem (viz obr. 4.4).

Podobně pokud je impulsní charakteristika antisymetrická, znamená to, že první vzorek $x[n]$ a poslední vzorek $x[n - N + 1]$ jsou násobeny koeficientem se stejnou hodnotou ale opačným znaménkem

$$y[n] = \begin{cases} \sum_{m=0}^{\frac{N-1}{2}} (x[n - m] - x[n - N + 1 + m]) h[m] & \text{pro } N \text{ sudé} \\ \sum_{m=0}^{\frac{N-1}{2}} (x[n - m] - x[n - N + 1 + m]) h[m] & \text{pro } N \text{ liché} \end{cases} \quad (4.7)$$



Obr. 4.4: Graf signálových toků realizující algoritmus číslicového systému se symetrickou nebo antisymetrickou konečnou impulsní charakteristikou.

Pro liché N zmizí prostřední člen, protože koeficient $h\left[n - \frac{N-1}{2}\right]$ může antisymetrii splnit pouze v případě $h\left[n - \frac{N-1}{2}\right] = -h\left[n - N + 1 + \frac{N-1}{2}\right] = 0$. Proto je možné nejprve odpovídající vzorky odečíst a teprve součet násobit koeficientem.

V obou případech je místo N násobení prováděno o $\approx \frac{N-1}{2}$ násobení méně, ale o stejnou hodnotu více součtů nebo rozdílů. Násobení je tak zaměněno za součet. Symetrická struktura realizace je tak vhodná pouze u procesorů, kde násobení trvá podstatně delší dobu než součet/rozdíl. V případě signálových procesorů, kdy násobení trvá většinou stejný počet hodinových cyklů jako součet, nemá symetrická struktura význam. Čtení z protilehlých míst kruhové paměti komplikuje provádění paralelních přenosů, při součtu dvou vzorků (bez násobení koeficientem) je vyšší pravděpodobnost přetečení, atd. Úspora poloviny násobení nevyhradí tyto problémy.

Výjimkou je implementace v signálovém procesoru řady TMS320C55xx, který ve struktuře obsahuje dva bloky násobení s akumulací (viz část 3.2.3). Obě jednotky mohou pracovat paralelně, pokud jeden z operandů je společný pro obě násobičky. V tomto případě je společný koeficient $h[m]$, kterým je násoben jednou vzorek $x[n - m]$ a podruhé vzorek $x[n - N + 1 + m]$. Pro antisymetrickou impulsní charakteristiku tak můžeme vztah (4.7) rozdělit na dva

$$y_1[n] = \sum_{m=0}^{\frac{N}{2}} x[n - m]h[m], \quad (4.8)$$

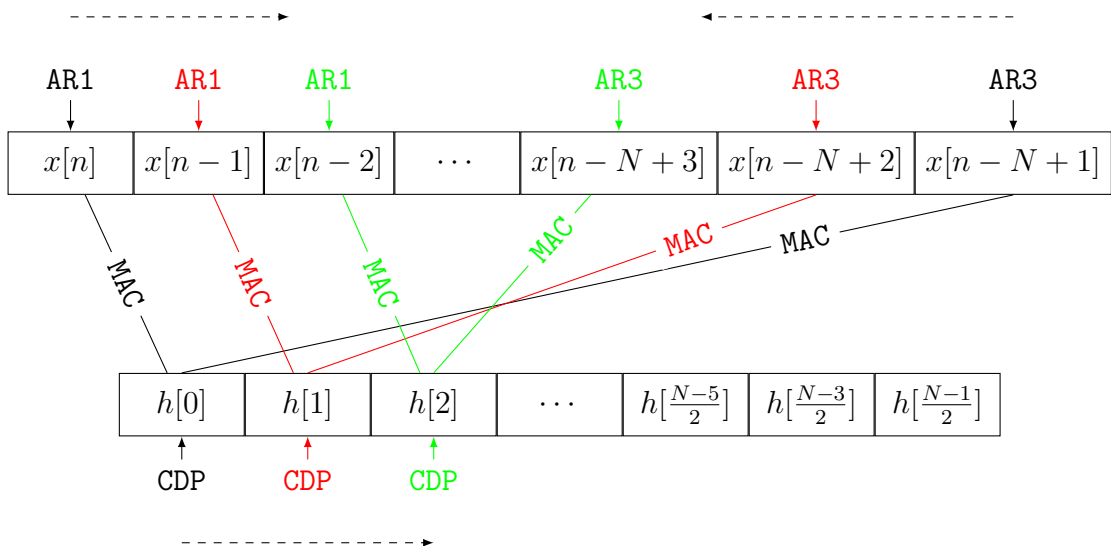
$$y_2[n] = \sum_{m=0}^{\frac{N}{2}} x[n - N + 1 + m]h[m]. \quad (4.9)$$

První signál $y_1[n]$ bude počítat jeden z bloků MAC, druhý signál $y_2[n]$ bude počítat druhý.

Vzorek výstupního signálu bude dán rozdílem

$$y[n] = y_1[n] - y_2[n]. \quad (4.10)$$

V paměti je vytvořena kruhová paměť pro zpožděné vzorky a pole koeficientů jako v předchozím případě. Modulo adresování s délkou N je nastaveno pro registry AR1 a AR3, přičemž registr AR1 je naplněn adresou začátku bloku (tj. vzorek $x[n]$) a AR3 adresou konce bloku (tj. $x[n - N + 1]$). Adresovací registr CDP obsahuje adresu pole koeficientů. Počáteční nastavení adresovacích registrů je ilustrována na obr. 4.5. Instrukce RPT ve výpisu 4.2 opakuje následující instrukci tolikrát, jaká hodnota je v registru CSR (v případě sudé délky je na začátku programu nastaven na hodnotu $\frac{N}{2} - 1$). První polovina instrukce MAC *AR1+, *CDP+, AC0 přenese hodnoty $x[n]$ a $h[0]$, vynásobí je a připočte ke střadači AC0. Zároveň zvýší hodnoty adresovacích registrů AR1 a CDP, takže ukazují na $x[n - 1]$ a $h[1]$. Druhá polovina instrukce MAC *AR3-, *CDP+, AC1 přenese hodnoty $x[n - N + 1]$ a $h[0]$, vynásobí je a připočte ke střadači AC1. Zároveň sníží hodnotu adresovacího registru AR3, takže ukazuje na $x[n - N + 2]$ a $h[1]$. Na obr. 4.5 jsou hodnoty adresovacích registrů a operace naznačeny černě. Při následujícím opakování jsou vynásobeny $x[n - 1]h[1]$ a $x[n - N + 2]h[1]$ a součiny jsou akumulovány (na obr. 4.5 červeně). Zároveň se změní hodnoty adresovacích registrů, takže ukazují na následující vzorky. Po opakování $(\frac{N}{2} - 1)$ krát ukazuje AR1 na $x[n - \frac{N}{2} + 1]$ a AR3 na $x[n - \frac{N}{2}]$. Po vynásobení je ale žádoucí, aby AR1 ukazoval na $x[n - N + 1]$, který se stane novým $x[n]$, a AR3 na $x[n - N + 2]$, který se stane novým $x[n - N + 1]$. Při posledním násobení je tak k AR1 přičtena hodnota $-\frac{N}{2}$, která je na začátku programu vložena do registru T0. Díky modulo adresování bude registr AR1 ukazovat na poslední vzorek kruhové paměti. K AR3 je naopak přičtena hodnota $\frac{N}{2} - 2$, která je na začátku programu vložena do registru T1. Nakonec jsou obě akumulované hodnoty odečteny $y[n] = y_1[n] - y_2[n]$ instrukcí SUB.



Obr. 4.5: Symbolické znázornění implementace lineární konvoluce v signálovém procesoru TMS320C55xx s využitím dvou MAC a symetrie impulsní charakteristiky.

Výpis 4.2: Příklad využití symetrie při implementaci systému typu FIR v signálovém procesoru s hardvardskou architekturou.

```

53  RPT  CSR
54
55  MAC  *AR1+, *CDP+, AC0
56  ::MAC *AR3-, *CDP+, AC1
57
58  MAC  *(AR1+T0), *CDP+, AC0
59  ::MAC *(AR3+T1), *CDP+, AC1
60
61  SUB  AC0, AC1

```

V případě symetrické impulsní charakteristiky se závěrečný rozdíl změni na součet. Analogicky lze postupovat, pokud N bude liché. Jen pokud navíc bude charakteristika symetrická, je nutné přičíst součin prostředního koeficientu a vzorku

$$x \left[n - \frac{N-1}{2} \right] h \left[\frac{N-1}{2} \right].$$

Využití dvou násobiček je možné i jiným způsobem. Vztah (4.4) lze pro dva po sobě následující vzorky výstupního signálu rozepsat jako

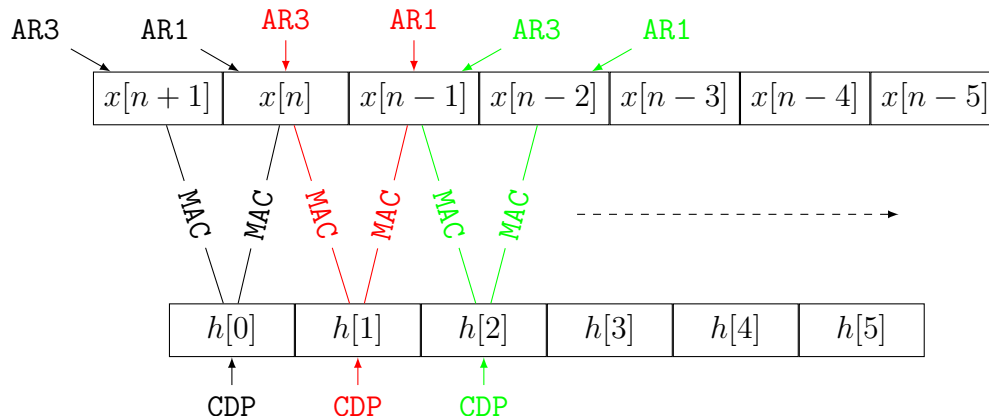
$$\begin{aligned} y[n] &= \sum_{m=0}^{N-1} x[n-m]h[m] \\ y[n+1] &= \sum_{m=0}^{N-1} x[n+1-m]h[m] \end{aligned} \quad (4.11)$$

Vzorek $x[n+1]$ je násoben stejným koeficientem $h[0]$ jako vzorek $x[n]$. Proto lze použít podobnou dvojici instrukcí násobení s akumulací s tím rozdílem, že tentokrát oba nebudou provádět výpočet stejného vzorku výstupního signálu, ale dvou odlišných vzorků.

Adresovací registr CDP je opět nastaven na pole koeficientů impulsní charakteristiky. Pro registry AR1 a AR3 je nastaveno modulo adresování s délkou bloku alespoň $N+1$. Jeden registr AR1 obsahuje adresu začátku bloku $x[n]$, druhý AR3 je o jedničku nižší $x[n+1]$. Nově příchozí vzorek je zapsán na pozici AR1, ale výpočet se nespustí. Teprve v okamžiku příchodu druhého vzorku vstupního signálu, který je uložen na pozici AR3, je spuštěn výpočet konvoluce. Počátečního nastavení adresovacích registrů je naznačeno na obr. 4.6.

Instrukce RPT CSR ve výpisu 4.3 opět opakuje následující instrukce tolikrát, jaká je hodnota v registru CSR. Oproti předchozímu případu je počet opakování nastaven na $N-1$. Počet opakování je tedy dvojnásobný, ale během cyklu jsou počítány dva vzorky výstupního signálu. Celková výpočetní náročnost je tak přibližně stejná jako při symetrické implementaci, a přibližně poloviční, než při využití jediné jednotky MAC.

První polovina MAC *AR1+, *CDP+, AC0 přenesou hodnoty $x[n]$ a $h[0]$, vynásobí je a připočte ke střadači AC0. Zároveň zvýší hodnoty adresovacích registrů AR1 a CDP, takže ukazují na $x[n-1]$ a $h[1]$. Druhá polovina MAC *AR3+, *CDP+, AC1 přenesou hodnoty $x[n+1]$ a $h[0]$, vynásobí je a připočte ke střadači AC1. Na obrázku 4.6 jsou registry a operace znázorněny černě. Zároveň zvýší hodnotu adresovacího registru AR3, takže ukazuje na $x[n]$ a $h[1]$. Na obrázku 4.6 červeně. Po opakování $(N-1)$ krát ukazuje AR1 na $x[n-N+1]$ a



Obr. 4.6: Symbolické znázornění implementace lineární konvoluce v signálovém procesoru TMS320C55xx s využitím dvou MAC.

AR3 na $x[n - N + 2]$. Jsou to poslední vzorky, které je nutné vynásobit a přičíst k akumulátoru. Zároveň je žádoucí, aby před zahájením následujícího cyklu AR1 ukazoval na pozici $x[n + 2]$ a AR3 na $x[n + 3]$, kam budou uloženy nové vzorky $x[n]$, resp. $x[n + 1]$. Proto je při posledním násobení od AR1 i AR3 odečtena hodnota $N + 1$, která byla na začátku programu uložena do registru T1.

Výpis 4.3: Příklad využití dvou jednotek MAC při implementaci systému typu FIR v signálovém procesoru s hardvardskou architekturou.

```

64   RPT CSR
65
66   MAC *AR1+, *CDP+, AC0
67   ::MAC *AR3-, *CDP+, AC1
68
69   MACR *(AR1-T1), *CDP+, AC0
70   ::MACR *(AR3-T1), *CDP+, AC1
71
72   SUB AC0, AC1

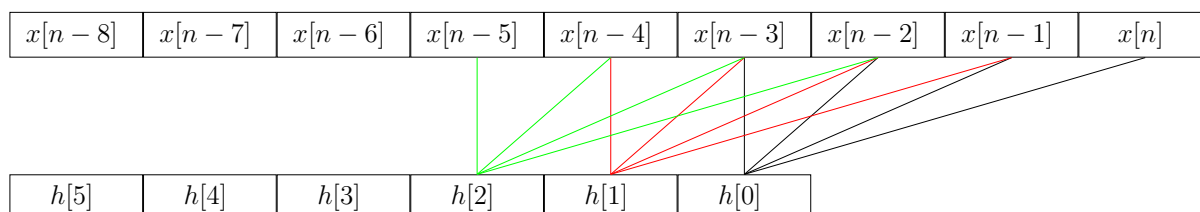
```

Výhodou paralelního zpracování dvou vzorků výstupního signálu je nižší výpočetní náročnost. Nevýhodou je komplikovanější program a větší zpoždění, které je minimálně dva vzorky.

4.1.3 Signálové procesory s velmi dlouhým instrukčním slovem

V případě procesorů s velmi dlouhým instrukčním slovem je implementace lineární konvoluce po jednom vzorku jako v procesorech s harvardskou architekturou (viz část 4.1.1) značně neefektivní. Jednotka DALU totiž obsahuje několik násobiček, např. procesory s jádrem StarCore obsahují čtyři jednotky násobení s akumulací MAC. Využití symetrie jako v případě procesoru TMS320C55xx (viz část 4.1.2) narazí na problémy se čtením z paměti. V jednom okamžiku lze totiž provádět pouze dvě čtení – tj. koeficient $h[m]$ a vzorek

$x[n - m]$. Čtení symetrického vzorku $x[n - N + 1 + m]$ je nemožné nebo pouze s komplikovaným uspořádáním vzorků v kruhové paměti tak, aby vzorky nebyly v přirozeném pořadí, ale byly prokládány symetrickými vzorky $x[n]$, $x[n - N + 1]$, $x[n - 1]$, $x[n - N + 2]$, \dots . Výhodnější je inspirovat se příkladem ze závěru části 4.1.2, vzorky ponechat v přirozeném pořadí, ale rozdělit je na více posloupností. Paralelně je tak možné provádět výpočet 4 výstupních vzorků $y[n]$, $y[n - 1]$, $y[n - 2]$, $y[n - 3]$, podobně jako byl u TMS320C55xx prováděn výpočet 2 výstupních vzorků $y[n]$, $y[n - 1]$. Symbolické znázornění je na obr. 4.7.



Obr. 4.7: Symbolické znázornění způsobu výpočtu lineární konvoluce na procesoru s jádrem StarCore.

U signálových procesorů s jádrem VelociTI je situace odlišná. Jádro VelociTI totiž neobsahuje jednotku násobení s akumulací, ale pouze dvě násobičky. Navíc se jinak chová v případě zřetěženého zpracování instrukcí. Jednotlivé instrukce se mohou skládat z různého počtu fází zřetěženého zpracování instrukcí, podobně jako u jádra StarCore nebo většiny procesorů. V případě instrukcí, jejichž provedení spotřebuje více hodinových cyklů, však většina signálových procesorů (včetně procesorů s jádrem StarCore) nezahájí vykonávání následující instrukce, dokud není vykonání předchozí instrukce dokončeno. Naopak jádro VelociTI na dokončení instrukce nečeká, ale zahájí vykonání další instrukce paralelně s prováděním předchozí instrukce nebo instrukcí. Vykonáním se zde rozumí skutečně provádění instrukce, nikoliv jen fáze čtení instrukce, její dekódování, apod., které jsou prováděny paralelně u všech procesorů podporujících zřetěžené zpracování instrukcí. To umožňuje optimalizaci kódu někdy označovanou jako *zřetěžené zpracování cyklů* (software pipelining), ale na druhou stranu značně znepráhledňuje kód.

Sekvenční implementace

Nejprve uvažujme případ sériové implementace s využitím jediné násobičky. Do registru A2 je vložena délka impulsní charakteristiky N . Pro registr A5 je nastaveno modulo adresování s délkou minimálně N a registr je nastaven na začátek kruhové paměti. Do registru B5 je uložena adresa koeficientů. V první instrukčním paketu ve výpisu 4.4 je dvojicí instrukcí LDH přečten nový vzorek vstupního signálu $x[n - m]$ z adresy A5 a nový koeficient $h[m]$ z adresy B5. Zároveň jsou hodnoty registrů A5 a B5 zvýšeny, takže obsahují adresy následujících vzorků. Vlivem zřetěženého zpracování instrukcí u jádra VelociTI má čtení z paměti zpoždění 4 hodinové cykly. Proto následují 4 instrukce NOP, díky kterým procesor počká na dokončení čtení. Zápis je zkrácen do jednoho řádku NOP s uvedením počtu opakování 4. Čísla na koncích řádků (za středníkem) pak označují číslo instrukčního paketu a zároveň hodinový cyklus, kdy je řádek prováděn. Teprve poté lze přečtený

Výpis 4.4: Příklad sériové implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.

1	<code>_loop</code>				
2	<code>LDH</code>	<code>.D1T1</code>	<code>*A5++,A9</code>	<code>; 0</code>	
3	<code> </code>	<code>LDH</code>	<code>.D2T2</code>	<code>*B5++,B9</code>	<code>; 0</code>
4		<code>NOP</code>	<code>4</code>		<code>; 1 2 3 4</code>
5		<code>SMPY</code>	<code>.M1</code>	<code>A9,B9,A8</code>	<code>; 5</code>
6		<code>NOP</code>			<code>; 6</code>
7		<code>SADD</code>	<code>.L1</code>	<code>A8,A7,A7</code>	<code>; 7</code>
8	<code> [A2]</code>	<code>BDEC</code>	<code>.S1</code>	<code>_loop,A2</code>	<code>; 7</code>
9		<code>NOP</code>	<code>5</code>		<code>; 8 9 10 11 12</code>

koeficient a vzorek vynásobit a součin uložit do registru `A8` instrukcí `SMPY .M1 A9,B9,A8`. Opět vlivem zřetězeného zpracování instrukcí má instrukce násobení `SMPY` zpoždění 1 hodinový cyklus. Proto následuje jedna instrukce `NOP`. V posledním 8. instrukčním paketu je provedena akumulace součinu instrukcí `SADD .L1 A8,A7,A7` do registru `A7`. Pokud je čítač cyklu `A2` nenulový, je provedena instrukce `BDEC .S1 _loop,A2`. Tato instrukce sníží registr `A2` o 1 a provede skok na návěští `_loop`. Instrukce skoku `B` má zpoždění 5 hodinových cyklů. Následuje tak 5 instrukcí `NOP`.

Paralelní implementace

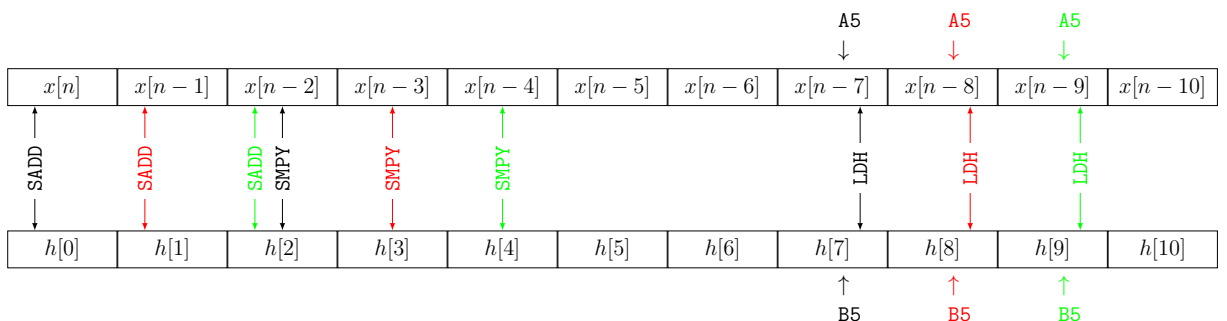
Jednotlivé iterace cyklu lze provádět paralelně, jak lze snadno zjistit, když několik iterací znázorníme vedle sebe (viz obr. 4.8). Jednotlivé sloupce představují jednu iteraci podle výpisu 4.4. Řádky pak odpovídají provedeným hodinovým cyklům. Po prvním hodinovém cyklu může jednotka `D` zahájit další čtení, i když předchozí čtení není dokončeno. Proto v druhém hodinovém cyklu je možné znovu použít instrukci `LDH` pro čtení koeficientu a vzorku vstupního signálu. Podobně je možné zahájit čtení třetího vzorku a koeficientu ve třetím hodinovém cyklu. V dolním indexu instrukcí je vyznačeno, ke které iteraci a kterému instrukčnímu paketu ve výpisu 4.4 patří `INSTRUKCEiterace,paket`. Teprve v šestém hodinovém cyklu je čtení prvních vzorků dokončeno a je možné je vynásobit. Současně s tím, je zahájeno čtení vzorku $x[n-5]$ a $h[5]$ z šesté iterace. Následující hodinový cyklus se čeká na dokončení násobení. Tímto hodinovým cyklem končí sekce nazývaná *prolog*, která inicializuje průběh algoritmu.

Od osmého hodinového cyklu se opakuje jednotný vzor. Je provedena akumulace součinu $x[n]h[0]$, paralelně s tím je provedeno násobení vzorků $x[n-2]h[2]$ a zahájeno čtení vzorků $x[n-7]$ a $h[7]$. Následující hodinový cyklus je provedena akumulace $x[n-1]h[1]$, paralelně s tím je provedeno násobení vzorků $x[n-3]h[3]$ a zahájeno čtení vzorků $x[n-8]$ a $h[8]$. Je zřejmé, že skupina instrukcí z hodinového cyklu 8 se dále opakuje a tvoří tzv. *kernel* algoritmu. V tomto případě je jádro složeno z jediného instrukčního paketu, protože všechny instrukce `LDH`, `SMPY`, `SADD` lze provádět paralelně. Jednotlivé instrukce ale zpracovávají vzorky z různých iterací cyklu, jak je naznačeno na obr. 4.9.

		iterace m									
cyklus		0	1	2	3	4	5	6	7	8	9
0		LDH _{0,0}									
1		NOP _{0,1}	LDH _{1,0}								
2		NOP _{0,2}	NOP _{1,1}	LDH _{2,0}							
3		NOP _{0,3}	NOP _{1,2}	NOP _{2,1}	LDH _{3,0}						
4		NOP _{0,4}	NOP _{1,3}	NOP _{2,2}	NOP _{3,1}	LDH _{4,0}					
5		SMPY _{0,5}	NOP _{1,4}	NOP _{2,3}	NOP _{3,2}	NOP _{4,1}	LDH _{5,0}				
6		NOP _{0,6}	SMPY _{1,5}	NOP _{2,4}	NOP _{3,3}	NOP _{4,2}	NOP _{5,1}	LDH _{6,0}			
7		SADD _{0,7}	NOP _{1,6}	SMPY _{2,5}	NOP _{3,4}	NOP _{4,3}	NOP _{5,2}	NOP _{6,1}	LDH _{7,0}		
8			SADD _{1,7}	NOP _{2,6}	SMPY _{3,5}	NOP _{4,4}	NOP _{5,3}	NOP _{6,2}	NOP _{7,1}	LDH _{8,0}	
9				SADD _{2,7}	NOP _{3,6}	SMPY _{4,5}	NOP _{5,4}	NOP _{6,3}	NOP _{7,2}	NOP _{8,1}	LDH _{9,0}
10					SADD _{3,7}	NOP _{4,6}	SMPY _{5,5}	NOP _{6,4}	NOP _{7,3}	NOP _{8,2}	NOP _{9,1}
11						SADD _{4,7}	NOP _{5,6}	SMPY _{6,5}	NOP _{7,4}	NOP _{8,3}	NOP _{9,2}
12							SADD _{5,7}	NOP _{6,6}	SMPY _{7,5}	NOP _{8,4}	NOP _{9,3}
13								SADD _{6,7}	NOP _{7,6}	SMPY _{8,5}	NOP _{9,4}
14									SADD _{7,7}	NOP _{8,6}	SMPY _{9,5}
15										SADD _{8,7}	NOP _{9,6}
16											SADD _{9,7}

Obr. 4.8: Symbolické skládání několika iterací výpočtu konvoluce paralelně.

Pokud impulsní charakteristika má délku $N = 10$, není v 11. hodinovém cyklu už nutné (a ani možné) zahájit čtení koeficientu $h[10]$ a vzorku $x[n - 10]$. V instrukčním paketu postupně ubývá instrukcí, v obráceném pořadí než v sekci *prolog* přibývaly. Postupně je pouze dokončováno násobení a akumulaci posledních vzorků signálu a koeficientů. Tato část programu se označuje jako *epilog*.



Obr. 4.9: Symbolické znázornění implementace lineární konvoluce v signálovém procesoru TMS320C64xx.

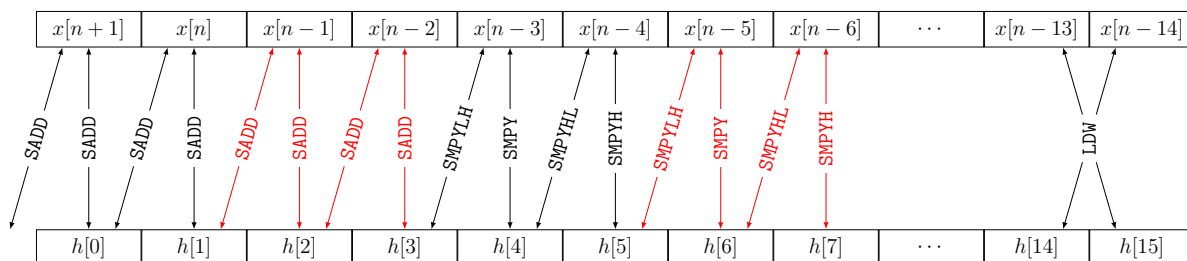
Konkrétní zápis instrukcí optimalizované verze je ve výpisu 4.5. V komentářích na konci řádků je uvedeno číslo iterace m a číslo instrukčního paketu z výpisu 4.4, ke kterým instrukce patří $\langle \text{iterace}, \text{paket} \rangle$. Instrukce BDEC je předsunuta o 5 instrukčních paketů tak, aby v okamžiku zpracování paketu `_loop1` byl již proveden skok a nebylo nutné čekat. Podobně je jeden hodinový signál před jádrem zahájeno násobení vzorků. Jádro se pak skládá z jediného instrukčního paketu, který obsahuje všechny potřebné instrukce. Každá z instrukcí je však pro odlišnou iteraci, což značně znehodnocuje kód a ztěžuje úpravy.

V rámci epilogu je možné dále provádět čtení a násobení dalších vzorků, protože součin již nebude akumulován. Čtení za koncem impulsní charakteristiky tak neovlivní výpočet a epilog může být sloučen s jádrem algoritmu. Postup se označuje jako *spekulativní provádění instrukcí*, kdy výsledek instrukce nemusí být použit. Vedlejším efektem ale je, že po skončení cyklu registr A5 neobsahuje adresu vzorku $x[n - N]$, ale $x[n - N - 8]$. Před dalším opakováním cyklu je žádoucí, aby registr A5 obsahoval adresu posledního vzorku v kruhové paměti $x[n - N + 1]$. Hodnota registru musí být snížena o $N + 8$ a nikoliv pouze N . To je provedeno instrukcemi SUBAH A5, A2, A5, přičemž na začátku programu je do registru A2 uložena hodnota $N + 8$. Všechny operace s adresovacím registrem A5 jsou prováděny v rámci modulu adresování.

Dvojitá paralelní implementace

Instrukční paket v jádru algoritmu obsahuje pouze 5 paralelních instrukcí, přičemž maximální počet je 8. Vůbec není použita násobička M2 a jednotka L2. Větší efektivity lze dosáhnout, pokud budou zpracovávány dva vstupní vzorky současně, podobně jako v případě TMS320C55xx. V jedné cestě, např. A bude zpracováván $x[n]$, v druhé, např. B, bude zpracováván vzorek $x[n + 1]$.

Symbolické znázornění je na obr. 4.10, konkrétní výpis sériové podoby algoritmu pak ve výpisu 4.6. Pro čtení z paměti je použita instrukce LDW, která přenáší 32bitové hodnoty. Při jednom čtení tak přeneseme dvojici 16bitových hodnot. Při organizaci paměti little endian bude v dolní 16bitové polovině registru uložena hodnota na nižší adrese. V tomto případě bude v dolní polovině registru B9 sudý vzorek impulsní charakteristiky $h[0]$, $h[2]$, $h[4]$, atd., resp. v dolní polovině registru A9 bude novější vzorek $x[n + 1]$, $x[n - 1]$, $x[n - 3]$, atd. V horní polovině registru budou hodnoty uloženy na adrese výše. V tomto případě



Obr. 4.10: Symbolické znázornění dvojitě implementace lineární konvoluce v signálovém procesoru TMS320C64xx.

Výpis 4.5: Příklad paralelní implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.

```

1      ; prolog
2      LDH      .D1T1  *A5++,A9      ; <0,0>
3  ||      LDH      .D2T2  *B5++,B9      ; <0,0>
4
5      LDH      .D1T1  *A5++,A9      ; <1,0>
6  ||      LDH      .D2T2  *B5++,B9      ; <1,0>
7
8      LDH      .D1T1  *A5++,A9      ; <2,0>
9  ||      LDH      .D2T2  *B5++,B9      ; <2,0>
10     |[A2] BDEC   .S1      _loop1,A2    ; <0,7>
11
12      LDH      .D1T1  *A5++,A9      ; <3,0>
13  ||      LDH      .D2T2  *B5++,B9      ; <3,0>
14  |[A2] BDEC   .S1      _loop1,A2    ; <1,7>
15
16      LDH      .D1T1  *A5++,A9      ; <4,0>
17  ||      LDH      .D2T2  *B5++,B9      ; <4,0>
18  |[A2] BDEC   .S1      _loop1,A2    ; <2,7>
19
20      LDH      .D1T1  *A5++,A9      ; <5,0>
21  ||      LDH      .D2T2  *B5++,B9      ; <5,0>
22  ||      SMPY    .M1      A9,B9,A8      ; <0,5>
23  |[A2] BDEC   .S1      _loop1,A2    ; <3,7>
24
25      LDH      .D1T1  *A5++,A9      ; <6,0>
26  ||      LDH      .D2T2  *B5++,B9      ; <6,0>
27  ||      SMPY    .M1      A9,B9,A8      ; <1,5>
28  |[A2] BDEC   .S1      _loop1,A2    ; <4,7>
29
30     ; kernel sloučený s epilodem
31  _loop1
32      LDH      .D1T1  *A5++,A9      ; <7,0>
33  ||      LDH      .D2T2  *B5++,B9      ; <7,0>
34  ||      SMPY    .M1      A9,B9,A8      ; <2,5>
35  ||      SADD    .L1      A8,A7,A7      ; <0,7>
36  |[A2] BDEC   .S1      _loop1,A2    ; <5,7>
37
38     ; ukončení cyklu
39      MVK                order+9,A2
40      SUBAH             A5,A2,A5

```

bude v horní polovině registru B9 lichý vzorek impulsní charakteristiky $h[1]$, $h[3]$, $h[5]$, atd., resp. v horní polovině registru A9 starší vzorek $x[n]$, $x[n-2]$, $x[n-4]$, atd.

Po 4 hodinových cyklech je přenos dokončen a je možné provést násobení spodních polovin instrukcí SMPY na řádku 6 výpisu, tj. $x[n+1]h[0]$, $x[n-1]h[2]$, atd. Tato instrukce přísluší výpočtu vzorku $y[n+1]$. Podobně jako instrukce SMPYH na řádku 10, která násobí horní poloviny registrů, tj. $x[n]h[1]$. Pro výpočet $y[n]$ je provedeno násobení $x[n]h[0]$, tj. horní polovina registru A9 a dolní B9 instrukcí SMPYHL na řádku 11. Po načtení nového páru vzorků a koeficientů se situace opakuje. S tím rozdílem, že vzorek $x[n-1]$ je nutné vynásobit koeficient $h[1]$ z předchozího páru. Registr B9 však již obsahuje novou dvojici koeficientů $h[2]$, $h[3]$. Proto je hodnota registru B9 ještě předtím zkopírována instrukcí MV na řádku 8 do registru B4. Někdy se takto zkopírovaný registr označuje jako tzv. *twin register*. Násobení $x[n-1]h[1]$ je pak provedeno instrukcí SMPYLH A9, B4, A8 na řádku 7. Součiny v datové cestě A jsou instrukcí SADD na řádcích 13 a 16 akumulovány v registru A7. Po skončení cyklu registr A7 obsahuje nový výstupní vzorek $y[n+1]$. Podobně součiny v datové cestě B jsou akumulovány v registru B7 na řádcích 14 a 17. Po skončení cyklu registr B7 obsahuje výstupní vzorek $y[n]$. Horní 16bitové poloviny jsou sloučeny do 32bitového slova instrukcí PACKH2. V horní polovině registru A4 bude uložena horní 16bitová polovina prvního registru A7 (tj. $y[n+1]$), v dolní polovině bude uložena horní 16bitová polovina druhého registru B7 (tj. $y[n]$).

Výpis 4.6: Příklad dvojité sériové implementace systému typu FIR v signálovém procesoru s velmi dlouhým instrukčním slovem.

1	_loop	LDW	.D1T1	*A5++, A9	; 0
2		LDW	.D2T2	*B5++, B9	; 0
3		NOP	4		; 1 2 3 4
4					
5		SMPY	.M1	A9, B9, A8	; 5
6		SMPYLH	.M2	A9, B4, B8	; 5
7		MV		B9, B4	; 5
8					
9		SMPYH	.M1	A9, B4, A8	; 6
10		SMPYHL	.M2	A9, B4, B8	; 6
11					
12		SADD	.L1	A8, A7, A7	; 7
13		SADD	.L2	B8, B7, B7	; 7
14					
15		SADD	.L1	A8, A7, A7	; 8
16		SADD	.L2	B8, B7, B7	; 8
17					
18	[B2]	BDEC	.S2	_loop, B2	; 9
19		NOP	5		
20					
21		PACKH2		A7, B7, A4	

Podobně jako v předchozím případě je možné jednotlivé iterace cyklu provádět paralelně. Na obr. 4.11 je znázorněno několik rozvinutých iterací paralelně vedle sebe. Je patrné, že po určitém počtu se opět opakuje jednotný vzor, který tvoří jádro cyklu. Pro zjednodušení jsou instrukční pakety 1, 6, 7, 8, 9 obsahující vždy dvě instrukce vyjádřeny jediným blokem. Proto i když se vzor opakuje zdánlivě po jednom hodinovém cyklu, jádro bude obsahovat 2 instrukční pakety ve výpisu 4.7.

	iterace m										
cyklus	0	1	2	3	4	5	6	7	8	9	10
0	LDW _{0,0}										
1	NOP _{0,1}	LDW _{1,0}									
2	NOP _{0,2}	NOP _{1,1}	LDW _{2,0}								
3	NOP _{0,3}	NOP _{1,2}	NOP _{2,1}	LDW _{3,0}							
4	NOP _{0,4}	NOP _{1,3}	NOP _{2,2}	NOP _{3,1}	LDW _{4,0}						
5	SMPY _{0,5}	NOP _{1,4}	NOP _{2,3}	NOP _{3,2}	NOP _{4,1}	LDW _{5,0}					
6	SMPYH _{0,6}	SMPY _{1,5}	NOP _{2,4}	NOP _{3,3}	NOP _{4,2}	NOP _{5,1}	LDW _{6,0}				
7	SADD _{0,7}	SMPYH _{1,6}	SMPY _{2,5}	NOP _{3,4}	NOP _{4,3}	NOP _{5,2}	NOP _{6,1}	LDW _{7,0}			
8	SADD _{0,8}	SADD _{1,7}	SMPYH _{2,6}	SMPY _{3,5}	NOP _{4,4}	NOP _{5,3}	NOP _{6,2}	NOP _{7,1}	LDW _{8,0}		
9		SADD _{1,8}	SADD _{2,7}	SMPYH _{3,6}	SMPY _{4,5}	NOP _{5,4}	NOP _{6,3}	NOP _{7,2}	NOP _{8,1}	LDW _{9,0}	
10			SADD _{2,8}	SADD _{3,7}	SMPYH _{4,6}	SMPY _{5,5}	NOP _{6,4}	NOP _{7,3}	NOP _{8,2}	NOP _{9,1}	LDW _{10,0}
11				SADD _{3,8}	SADD _{4,7}	SMPYH _{5,6}	SMPY _{6,5}	NOP _{7,4}	NOP _{8,3}	NOP _{9,2}	NOP _{10,1}
12					SADD _{4,8}	SADD _{5,7}	SMPYH _{6,6}	SMPY _{7,5}	NOP _{8,4}	NOP _{9,3}	NOP _{10,2}
13						SADD _{5,8}	SADD _{6,7}	SMPYH _{7,6}	SMPY _{8,5}	NOP _{9,4}	NOP _{10,3}
14							SADD _{6,8}	SADD _{7,7}	SMPYH _{8,6}	SMPY _{9,5}	NOP _{10,4}
15								SADD _{7,8}	SADD _{8,7}	SMPYH _{9,6}	SMPY _{10,5}
16									SADD _{8,8}	SADD _{9,7}	SMPYH _{10,6}
17										SADD _{9,8}	SADD _{10,7}
18											SADD _{10,8}

Obr. 4.11: Symbolické skládání několika iterací výpočtu dvojité konvoluce paralelně.

Porovnání výpočetní náročnosti

Ve všech případech je nutné jádro algoritmu opakovat N -krát, podle délky impulsní charakteristiky. V případě sekvenční implementace podle výpisu 4.4 však jedna iterace trvá 13 hodinových cyklů a celková výpočetní náročnost bude přibližně $13N$ hodinových cyklů pro jeden výstupní vzorek.

V případě paralelní implementace podle výpisu 4.5 prolog algoritmu trvá 7 hodinových cyklů a jádro tvoří jediný instrukční paket s délkou 1 hodinový cyklus. Celková výpočetní náročnost tak bude přibližně $7 + N$ hodinových cyklů. Je zřejmé, že výpočetní náročnost významně poklesla. Nevýhodou je, že paralelní implementaci lze použít pouze pro impulsní charakteristiku délky $N > 8$. Takovouto délku totiž vyžaduje prolog, aby došlo ke správnému zahájení algoritmu. V případě kratší impulsní charakteristiky se paralelní implementace nedá použít.

V případě dvojité paralelní implementace podle výpisu 4.7 počet paketů v jádru sice vzrostl na dvojnásobek, ale jádro zpracovává dvojnásobný počet výstupních vzorků.

Výpis 4.7: Příklad jádra dvojité paralelní implementace systému typu FIR v signálovém procesory s velmi dlouhým instrukčním slovem.

1	_loop	LDW	.D1T1	*A5++,A9	; <0,0>
2		SMPYH	.M1	A9,B4,A8	; <2,6>
3		SMPYHL	.M2	A9,B4,B8	; <2,6>
4		SADD	.L1	A8,A7,A7	; <0,8>
5		SADD	.L2	B8,B7,B7	; <0,8>
6	[B2]	BDEC	.S2	_loop,B2	; <8,9>
7					
8		LDW	.D2T2	*B5++,B9	; <8,0>
9		SMPY	.M1	A9,B9,A8	; <3,5>
10		SMPYLH	.M2	A9,B4,B8	; <3,5>
11		MV		B9,B4	; <3,5>
12		SADD	.L1	A8,A7,A7	; <1,7>
13		SADD	.L2	B8,B7,B7	; <1,7>

Výpočetní náročnost zůstává tedy přibližně stejná. Díky dvěma instrukčním paketům v jádru je možné oddělit obě instrukce LDW – čtení vzorků a koeficientů – do dvou různých hodinových cyklů. To je výhodné, pokud jsou vzorky uloženy ve vnější paměti. V takovém případě je totiž přenos realizován pomocí jediné adresové a datové sběrnice, takže nelze provádět dvě čtení paralelně. Pokud v jednom paketu jsou dvě čtení z vnější paměti, procesor nejprve provede jedno a teprve po jeho skončení druhé. Činnost jádra je během té doby pozastavena, což má za následek až dvojnásobnou dobu trvání algoritmu.

4.2 Číslicové systémy s nekonečnou impulsní charakteristikou

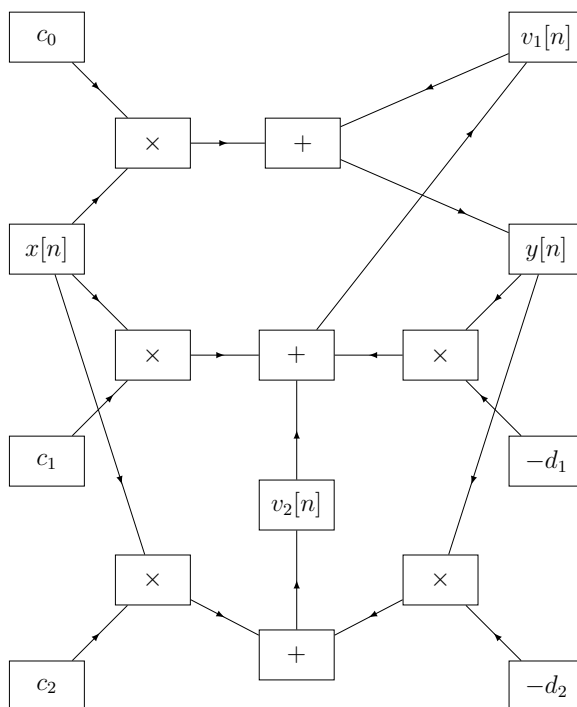
V případě systémů s nekonečnou impulsní charakteristikou je situace složitější, zejména při implementaci v pevné řádové čárce. Vlivem zpětných vazeb jsou tyto systémy mnohem citlivější na vlivy omezené délky slova. A to nejen na kvantování vstupního, výstupního signálu nebo stavových proměnných, ale i na kvantování koeficientů přenosové funkce (viz část 5.1). Proto se systémy vyšších řádů většinou realizují jako sériově nebo paralelně spojené sekce 2. řádu (viz část 5.2).

Dílčí sekce je možné realizovat první nebo druhou kanonickou strukturu. V případě druhé kanonické struktury však často dochází k přetékání stavových proměnných. Proto se více používá 1. kanonická forma, jejíž stavové rovnice pro 2. řád přejdou do tvaru

$$\begin{aligned}
 y[n] &= 2 \left(\frac{c_0}{2} x[n] + \frac{1}{2} v_1[n] \right) \\
 v_1[n+1] &= 2 \left(\frac{c_1}{2} x[n] - \frac{d_1}{2} y[n] + \frac{v_2[n]}{2} \right) \\
 v_2[n+1] &= 2 \left(\frac{c_2}{2} x[n] - \frac{d_2}{2} y[n] \right)
 \end{aligned} \tag{4.12}$$

Závislosti mezi jednotlivými kroky algoritmu jsou symbolicky znázorněny na obr. 4.12. Je zřejmé, že algoritmus je mnohem více sekvenční s menšími možnostmi paralelizace, než

tomu bylo v případě systémů s konečnou impulsní charakteristikou. Paralelně lze opět provádět násobení s akumulací a některé paralelní přenosy. Provádět algoritmus paralelně pro dva vzorky vstupního signálu je však téměř nemožné, protože hodnoty stavových proměnných závisí na předchozím vzorku výstupního signálu.



Obr. 4.12: Symbolické znázornění závislostí dílčích operací při výpočtu první kanonické formy podle (4.12).

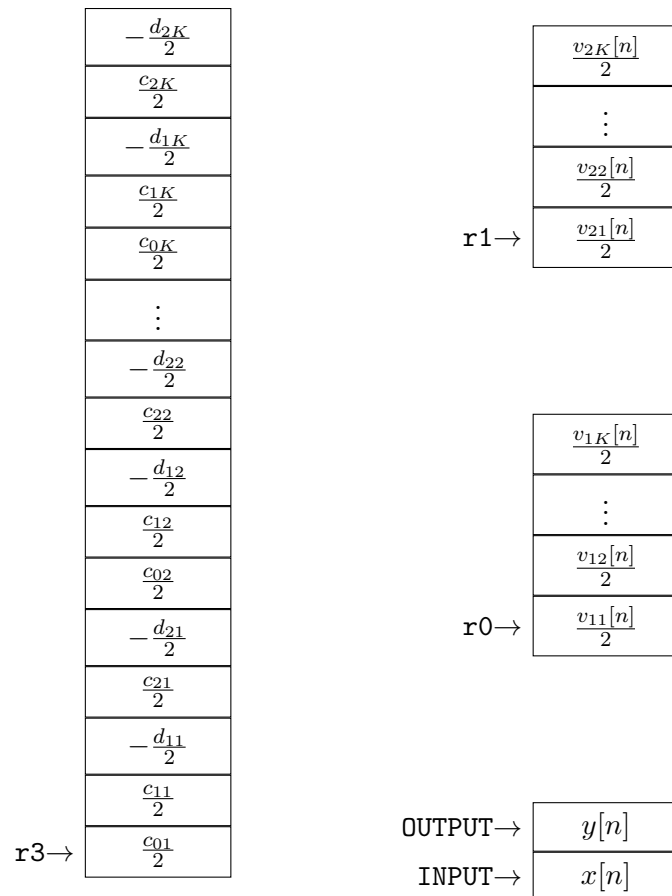
Nejprve je nutné vypočítat vzorek výstupního signálu $y[n]$, protože se vyskytuje ve vztazích pro aktualizaci stavových proměnných. Předtím je však nutné načíst hodnotu vzorku vstupního signálu $x[n]$ a koeficient c_0 , vynásobit je mezi sebou a součín přičíst k hodnotě stavové proměnné $v_1[n]$.

Potom je vhodné aktualizovat hodnotu stavové proměnné $v_1[n + 1]$. Vzorek vstupního signálu $x[n]$ a výstupního signálu $y[n]$ jsou již načteny do DALU. Je ale nutné načíst koeficient c_1 a d_1 . K součinům $c_1 x[n]$ a $-d_1 y[n]$ je připočtena hodnota stavové proměnné $v_2[n]$ a výslednou hodnotou je přepsána stavová proměnná $v_1[n]$. Pro aktualizaci stavové proměnné $v_2[n + 1]$ je nutné načíst koeficienty c_2 a d_2 , provést příslušné součiny $c_2 x[n]$ a $-d_2 y[n]$ a provést součet.

4.2.1 Signálové procesory s harvardskou architekturou

Pro příklad implementace filtrů s nekonečnou impulsní charakteristikou v signálových procesorech s harvardskou architekturou byl opět zvolen procesor DSC56F8367. Při implementaci je možné některé kroky algoritmu (typicky násobení s akumulací) sloučit paralelně, možností paralelního slučování je však výrazně nižší než u filtrů s konečnou impulsní charakteristikou. Důvodem je větší závislost jednotlivých kroků na předchozích.

Koeficienty všech sekcí jsou uloženy v paměti za sebou v pořadí, jak jsou používány ve stavových rovnicích (viz obr. 4.13), a to včetně znaménka a dělení dvěma. Adresa prvního koeficientu je uložena do registru **r3**. Dále stavové proměnné $v_1[n]$ všech sekcí $1, 2, \dots, K$ jsou ukládány za sebou a to včetně dělení 2. Ušetří se tím bitové posuny v rámci algoritmu bez vlivu na přesnost, neboť $2^{\frac{1}{2}} = 1$. Adresa stavové proměnné $\frac{v_{11}[n]}{2}$ první sekce je uložena do registru **r0**. Podobně jsou uloženy stavové proměnné $v_2[n]$ všech sekcí a adresa stavové proměnné $\frac{v_{21}[n]}{2}$ první sekce je uložena do registru **r1**. Vstupní vzorek je uložen na adrese **INPUT**, výstupní vzorek má být uložen na adresu **OUTPUT**.



Obr. 4.13: Ilustrační rozložení paměti při implementaci 1. kanonické formy v signálovém procesoru s harvardskou architekturou.

Ve výpisu 4.8 je nejprve vstupní vzorek z adresy **INPUT** načten do registru **y1**. V následujícím cyklu jsou implementovány stavové rovnice 1. kanonické formy pro každou sekci filtru. Provádění cyklu je podporována hardwarově instrukcí **do**. Ta je podobná instrukci **rep** pro opakování následující instrukce použité při implementaci systému s konečnou impulsní charakteristikou, ale instrukce **do** umožňuje opakování několika instrukcí. První parametr je počet opakování, tj. počet sekcí. Druhý parametr udává adresu první instrukce, která již není součástí cyklu, tedy instrukce za cyklem.

Výpis 4.8: Příklad implementace 1. kanonické formy v signálovém procesoru s harvardskou architekturou.

```

104  _FILT
105  ; nacteni vstupniho vzorku
106  move.w    X:(INPUT),y1
107  ; y1 <-- x[n]
108
109  do        #SECT, _ENDFILT

```

Jako první je implementována výstupní rovnice. Do střadače **a** je z paměti přečtena stavová proměnná $v_{1k}/2$. K hodnotě je instrukcí **macr** připočten součin $c_0/2x[n]$ a výsledek je zaokrouhlen na 16 bitů. Paralelním přesunem je do registru **x0** uložen následující koeficient $c_{1k}/2$ a **r3** zvýšen o 1. Bitovým posunem vlevo o 1 bit **asl** je vypočten vzorek výstupního signálu $y_k[n]$. V paralelním přenosu je do střadače **b** vložena polovina stavové proměnné $v_{2k}[n]/2$. Protože paralelní přenos umožňuje pouze přenos s modifikací adresovacího registru, zatímco nyní se adresa měnit nemá, je přičtena hodnota offsetového registru, který je ovšem na začátku programu nastavený na $n = 0$. Protože instrukce **move.w a,y0** nebo instrukce **tfr a,y0** pro přenos registru **a** do **y0** nepodporují paralelní přenosy, je hodnota **a** přenesena do **y0** pomocí instrukce **sat** pro saturaci. V paralelním přenosu je do registru **c** vložena další koeficient $-d_{1k}/2$.

Výpis 4.8: Příklad implementace 1. kanonické formy v signálovém procesoru s harvardskou architekturou (pokračování).

```

111  move.w    X:(r0),a
112  ; a <-- 1/2 v1i[n]
113  macr     y1,x0,a    X:(r3)+,x0
114  ; a <-- 1/2 v1i[n] + k1 c0i/2 xi[n] = 1/2 yi[n]
115  ; x0 <-- k1 c1i / 2
116  asl     a          X:(r1)+n,b
117  ; a <-- yi[n]
118  ; b <-- v2i[n]/2
119  sat     a,y0      X:(r3)+,c
120  ; y0 <-- yi[n]
121  ; c <-- -d1i / 2

```

V následující instrukci je ke stavové proměnné $v_{2k}[n]/2$ připočten součin $c_{1k}[n]x[n]$. Paralelně je provedeno čtení dalšího koeficientu c_{2k} do registru **x0**. Přičtením dalšího součinu $-d_{1k}y[n]$ je dokončen výpočet nové hodnoty stavové proměnné $v_{1k}[n+1]$, proto je po akumulaci provedeno zaokrouhlení na 16bitů instrukcí **macr**. Paralelně je přenesen další koeficient $-d_{2k}$ do registru **c**.

Zbývající stavová rovnice je provedena vynásobením $c_{2k}x[n]$. Paralelně s násobením je v paměti **r0** aktualizována hodnota stavové proměnné $v_{1k}[n]$. Adresa **r0** je přitom zvýšena o 1, takže obsahuje adresu stavové proměnné $v_{1(k+1)}$ následující sekce. Následuje přičtení součinu $-d_{2k}[n]$ a zaokrouhlení na 16 bitů instrukcí **macr**. Tím je vypočtena nová hodnota stavové proměnné $v_{2k}[n+1]$. Paralelně s tím je přenesena do registru

Výpis 4.8: Příklad implementace 1. kanonické formy v signálovém procesoru s harvardskou architekturou (pokračování).

```

123     mac      y1,x0,b      X:(r3)+,x0
124         ; b    <--      v2i[n] / 2 + k1 b1i / 2 xi[n]
125         ; x0   <--      k1 c2i / 2
126     macr     c1,y0,b      X:(r3)+,c
127         ; b    <--      v2i[n]/2+k1 c1i/2 xi[n] - d1i/2 yi[n]
128         ; c    <--      -d2i / 2

```

$x0$ hodnota koeficientu $c_{0(k+1)}$ následující sekce. Předposlední instrukce v paměti $r1$ aktualizuje hodnotu stavové proměnné $v_{2k}[n]$ a k adrese $r1$ připočte 1, takže obsahuje adresu stavové proměnné $v_{2(k+1)}[n]$ následující sekce. Poslední instrukce `move.w` přesune hodnotu výstupního vzorku $y[n]$ z registru $y0$ do registru $y1$, ve které má být umístěn vstupní vzorek. Vše je tak připraveno pro opakování cyklu pro další sekci systému. Návěští `_ENDFILT` označuje konec bloku instrukce do.

Výpis 4.8: Příklad implementace 1. kanonické formy v signálovém procesoru s harvardskou architekturou (pokračování).

```

130     mpy      y1,x0,a      b,X:(r0)+
131         ; a    <--      k1 b0i / 2 * xi[n]
132         ; X:11 <--      b
133     macr     c1,y0,a      X:(r3)+,x0
134         ; a    <--      k1c2i/2 xi[n] - d2i/2 yi[n] = v2i[n+1]/2
135         ; x0   <--      k2 c0(i+1) / 2
136     move.w   a,X:(r1)+
137         ; X:v21 <--      a
138     move.w   y0,y1
139     _ENDFILT

```

Jakmile je celý cyklus do opakován pro všech K sekcí, je v registru $y1$ hodnota vzorku výstupního signálu, která je zapsána na adresu `OUTPUT`. Tím je implementace ukončena.

Výpis 4.8: Příklad implementace 1. kanonické formy v signálovém procesoru s harvardskou architekturou (pokračování).

```

141     move.w   y1,X:(OUTPUT)
142
143     moveu.w  #COEFF,r3
144     move.w   X:(r3)+,x0
145
146     jmp     _FILT

```

Je zřejmé, že všechny možnosti architektury nejsou využity tak, jako v případě implementace systémů s konečnou impulsní charakteristikou. Např. dva paralelní přesuny se v kódu nevyskytují, naopak v závěru cyklu (řádek 136 a 138) jsou dva samostatné

přesuny, které lze jen obtížně sloučit paralelně s jinými částmi. Uložení nové hodnoty stavové proměnné $v_{2k}[n+1]$ je možné teprve v okamžiku, kdy je vypočtena, tj. provedena instrukce `macr` na řádku 133. Důvodem malého stupně paralelního zpracování je sekvenční struktura algoritmu, kdy jednotlivé části algoritmu na sebe navazují, závisí na sobě.

4.2.2 Signálové procesory s velmi dlouhým instrukčním slovem

Velká míra závislostí mezi jednotlivými kroky algoritmu se projeví i při implementaci v signálových procesorech s velmi dlouhým instrukčním slovem. Struktura DALU jádra StarCore je podobná struktuře DALU procesorů DSC56F8367 s harvardskou architekturou. Nicméně větší počet jednotek MAC umožní provádět násobení vzorku $x[n]$ a $y[n]$ paralelně.

Sekvenční implementace

Oproti tomu u jádra VelociTI je možné využít zřetěžené zpracování cyklů (*software pipelining*) podobně jako při implementaci systému typu FIR ve výpisu 4.7. Základem je opět sekvenční implementace bez využití paralelismu. Rozložení koeficientů a stavových proměnných v paměti je podobné jako na obr. 4.13, ale adresa koeficientů je uložena v registru A4 a B4, adresa stavových proměnných $v_1[n]$ je uložena v registru B6 a adresa stavových proměnných $v_2[n]$ pro čtení je v registru A4, zatímco adresa pro zápis je v registru A3. Použití dvou registrů je výhodnější při sdružování instrukcí paralelně. Vstupní vzorek je uložen v registru A7, počet sekcí je uložen do registru B0.

Přestože ve výpisu 4.9 je uvedena sekvenční implementace, instrukce čtení z paměti LDH, LDW a instrukce skoku BDEC jsou předsunuty tak, aby nebylo nutné vkládat instrukce NOP. Pro větší přehlednost jsou jednotlivé pakety očíslovány na konci řádku v komentáři za znakem středník ;. Dokončení zpožděných instrukcí je pak naznačeno na konci řádku znaky (\sim p), kde p je číslo dokončeného instrukčního paketu.

Výpis 4.9: Příklad sekvenční implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW.

63	<code>_loop</code>			
64	<code>LDW</code>	<code>.D2T2</code>	<code>*B5++, B8</code>	<code>; 0</code>
65	<code>LDH</code>	<code>.D1T1</code>	<code>***A4(10), A9</code>	<code>; 1</code>
66	<code> </code>	<code>LDW</code>	<code>.D2T2</code>	<code>*B6, B7 ; 1</code>
67		<code>LDH</code>	<code>.D2T2</code>	<code>***B4(10), B17 ; 2</code>
68		<code>LDH</code>	<code>.D1T1</code>	<code>**A4(6), A8 ; 3</code>
69		<code>LDH</code>	<code>.D1T1</code>	<code>**A4(8), A5 ; 4</code>
70	<code>; B8</code>	<code><--</code>	<code>v2n / 2</code>	<code>(^0)</code>
71		<code>LDH</code>	<code>.D1T2</code>	<code>**A4(4), B7 ; 5</code>
72	<code>; A9</code>	<code><--</code>	<code>c0 / 2</code>	<code>(^1)</code>
73	<code>; B7</code>	<code><--</code>	<code>v1n / 2</code>	<code>(^1)</code>

Opět je jako první vypočten vzorek výstupního signálu sekce $y[n]$. Násobení $\frac{c_{0k}}{2}x[n]$ v instrukčním paketu číslo 6 má zpoždění 1 hodinový cyklus. Součin je tak možné přičíst

k hodnotě stavové proměnné $\frac{v_{1k}}{2}$ až v instrukčním paketu číslo 8. V instrukčním paketu číslo 7 je tak vypočten součin $\frac{c_{1k}}{2}x[n]$ v předstihu. Podobně v instrukčním paketu číslo 8 je kromě uvedeného součtu provedeno násobení $\frac{c_{2k}}{2}x[n]$, opět v předstihu. Přičemž hodnota součinu bude použita až v instrukčním paketu č. 12. Když je bitovým posunem vlevo o 1 bit (instrukce **SSHL** v instrukčním paketu č. 9) vypočten výstupní vzorek $y[n]$ sekce, jsou tak již připraveny dílčí činitele $\frac{c_1}{2}x[n]$ a $\frac{c_2}{2}x[n]$ pro aktualizaci stavových proměnných.

Výpis 4.9: Příklad sekvenční implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW (pokračování).

73		SMPY	.M1	A9, A7, A9	; 6	^
74		; B17	<-- c1 / 2		(^2)	
75		SMPY	.M2X	B17, A7, B9	; 7	
76		; A8	<-- c2		(^3)	
77		; A9	<-- c0 / 2 xn		(^6)	
78		SADD	.L1X	B7, A9, A18	; 8	^
79		SMPY	.M1	A8, A7, A8	; 8	
80		; A18	<-- c0 / 2 xn + v1n / 2 = yn / 2			
81		; A5	<-- -d2 / 2		(^4)	
82		; B9	<-- c1 / 2 xn		(^7)	
83		SSHL	.S1	A18, 1, A9	; 9	^
84		SADD	.L2	B8, B9, B16	; 9	
85		; A9	<-- yn			
86		; B16	<-- c1 / 2 xn + v2n / 2			
87		; B7	<-- -d1 / 2		(^5)	
88		; A8	<-- c2 / 2 xn		(^8)	

Oproti harvardské architektuře, kdy vlivem malého počtu registrů DALU nebylo možné uchovávat mezivýsledky delší dobu v DALU, ale naopak byla snaha mezivýsledky ihned použít a přepsat novým, u architektury typu VLIW (Very Long Instruction Word) je patrná snaha počítat mezivýsledky kdykoliv (i paralelně), kdy jsou k dispozici požadované vstupní hodnoty. To je podstatná výhoda oproti harvardské architektuře, která se ještě výrazněji projeví při paralelizaci iterací cyklu.

Délka vypočteného výstupního vzorku $y[n]$ v registru A9 je 32 bitů, zatímco předpokládaná délka vzorků vstupního a výstupního signálu je 16 bitů. Před uložením do registru A7, kde následující sekce očekává vstupní vzorek je registr A9 posunut o 16 bitů vpravo. Díky instrukcím **SMPYHL**, která násobí horní 16bitovou polovinu prvního operandu krát dolní 16bitovou polovinu druhého operandu však není nutné čekat na dokončení posunu, ale násobení $-\frac{d_1}{2}y[n]$ provést ve stejném instrukčním paketu č. 10. Podobně násobení $-\frac{d_2}{2}y[n]$ podobnou instrukcí **SMPYLH** v instrukčním paketu č. 11.

Přestože dílčí kroky různých rovnic bylo možné provádět paralelně, pořadí některých operací nelze zaměnit. Řádky označené ve výpisu 4.9 na konci znakem stříška '^' jsou limitující z hlediska závislostí. Jejich pořadí nelze zaměnit a je nutné dodržet počet hodinových cyklů mezi nimi, aby algoritmus počítal správně. Např. instrukční paket s číslem 6 počítá součin $x[n]c_0$. Zpoždění instrukce **SMPY** je 1 hodinový cyklus, proto součet $x[n]c_0 + v_{1k}[n]/2 = y[n]/2$ může být proveden až v instrukčním paketu označeném

Výpis 4.9: Příklad sekvenční implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW (pokračování).

```

89          SHR      .S1      A9,16,A6          ; 10  ^
90 ||          SMPYLH .M1      A5,A9,A5          ; 10
91 || [ B0] BDEC    .S2      _loop,B0          ; 10
92          ; A6 <-- yn      (twin register)
93          SMPYHL .M2X      A9,B7,B9          ; 11
94 ||          MV      .L1      A6,A7          ; 11  ^
95          ; A7 <-- yn      (xn = yn)
96          ; A5 <-- -d2 / 2 yn          (^10)
97          SADD    .L1      A8,A5,A5          ; 12
98          ; A5 <-- c2 / 2xn - d2 / 2 yn
99          ; B9 <-- -d1 / 2 yn          (^11)
100         SADD    .L2      B16,B9,B7          ; 13
101         ; B7 <-- c1 / 2 xn + v2n / 2 - d1 / 2 yn
102         STW     .D1T1     A5,*A3++          ; 14
103         STW     .D2T2     B7,*B6++          ; 15

```

číslem 8. Instrukce SADD má nulové zpoždění, proto je výsledek uložen do registru A18 uložen okamžitě. Pro získání hodnoty je však nutné ho vynásobit dvěma. V instrukčním paketu číslo 9 je to provedeno bitovým posunem vlevo o 1 bit instrukcí SSSL. Teprve poté je možné násobit výstupní vzorek koeficientem $-d_{1k}$ nebo $-d_{2k}$ pro výpočet nové hodnoty stavových proměnných $v_{1k}[n+1]$ a $v_{2k}[n+1]$. To je provedeno v instrukčním paketu s číslem 10, resp. 11. V instrukčním paketu č. 11 je také hodnota výstupního vzorku $y[n]$ překopírována do registru A7, ve kterém další iterace cyklu, tj. další sekce, očekává vzorek vstupního signálu.

Paralelní implementace

Podobně jako v části 4.1.3 je možné iterace cyklu podle výpisu 4.9 provádět paralelně. Skládáním jednotlivých iterací podobně jako na obr. 4.8 lze získat prolog, jádro i epilog paralelní implementace. Minimální doba jádra je limitována minimální délkou trvání jedné iterace, tj. dobou mezi prvním a posledním instrukčním paketem sekvenční posloupnosti uvedené výše. V tomto případě to je $11 - 6 + 1 = 6$ hodinových cyklů neboli 6 instrukčních paketů.

Příklad jádra je ve výpisu 4.10. Na konci řádku je opět uvedeno číslo iterace a číslo paketu, ve kterém se instrukce vyskytuje ve výpisu 4.9, <iterace,paket>. Instrukční pakety limitující délku jádra jsou označeny znakem stříška ^ na konci řádku.

Využití SIMD

Pokud procesor podporuje princip SIMD (Single Instruction Multiple Data), nabízí se provádět násobení $c_{1k}x[n]$ a $c_{2k}x[n]$ paralelně s využitím instrukce SMPY2. Tato instrukce patří mezi instrukce rozšiřující základní instrukční sadu jádra VelociTI a paralelně násobí

Výpis 4.10: Příklad jádra paralelní implementace 1. kanonické formy v signálovém procesoru s architekturou typu VLIW.

```

74  _loop
75      SMPYLH  .M1    A5 , A9 , A5          ; <0 , 10>
76  || [ B0]  BDEC  .S2    _loop , B0      ; <0 , 10>
77  ||      SHR   .S1    A9 , 16 , A6      ; <0 , 10>  ^
78  ||      LDH   .D1T1  **A4 ( 8 ) , A5   ; <1 , 4>
79
80      MV     .L1    A6 , A7              ; <0 , 11>  ^
81  ||      SMPYHL .M2X  A9 , B7 , B9      ; <0 , 11>
82  ||      LDH   .D1T2  **A4 ( 4 ) , B7   ; <1 , 5>
83
84      SADD   .L1    A8 , A5 , A5        ; <0 , 12>
85  ||      SMPY   .M1    A9 , A7 , A9      ; <1 , 6>  ^
86  ||      LDW   .D2T2  *B5++ , B8       ; <2 , 0>
87
88      SADD   .L2    B16 , B9 , B7       ; <0 , 13>
89  ||      SMPY   .M2X  B17 , A7 , B9     ; <1 , 7>
90  ||      LDH   .D1T1  ***A4 ( 10 ) , A9 ; <2 , 1>
91  ||      LDW   .D2T2  **B6 ( 8 ) , B7   ; <2 , 1>
92
93      STW    .D1T1  A5 , *A3++          ; <0 , 14>
94  ||      SMPY   .M1    A8 , A7 , A8      ; <1 , 8>
95  ||      SADD   .L1X  B7 , A9 , A8      ; <1 , 8>  ^
96  ||      LDH   .D2T2  ***B4 ( 10 ) , B17 ; <2 , 2>
97
98      STW    .D2T2  B7 , *B6++          ; <0 , 15>
99  ||      SADD   .L2    B8 , B9 , B16     ; <1 , 9>
100 ||      SSSL  .S1    A8 , 1 , A9       ; <1 , 9>  ^
101 ||      LDH   .D1T1  **A4 ( 6 ) , A8   ; <2 , 3>

```

spodní poloviny i horní poloviny dvou registrů a výsledek uloží do registrového páru. Zároveň však provedení instrukce trvá 4 hodinové cykly, což je dvojnásobek doby provedení jednoduchého násobení instrukcí SMPY. Přesto by mohlo být výhodou, že ihned v dalším cyklu může jednotka M zahájit další dvojici násobení. Zrychlení algoritmu tak závisí především na tom, zda je možné algoritmus upravit pro paralelní provádění násobení.

Především je nutné v paměti změnit uspořádání koeficientů podle obr. 4.14. Koeficienty c_{1k}, c_{2k} musí být v paměti uloženy za sebou, takže po přesunutí instrukcí LDDW *A6++(12), A5:A4 bude v horní 16bitové polovině registru A5 uložen 16bitový koeficient c_{2k} a ve spodní 16bitové polovině 16bitový koeficient c_{1k} . Podobně v registru A4 bude uložena dvojice koeficientů $-d_{2k}, -d_{1k}$. Nulové hodnoty jsou vloženy proto, aby adresa čtveřice koeficientů byla vždy násobkem 8 bajtů, což vyžaduje použitá instrukce přenosu LDDW. Aby bylo možné instrukcí SMPY2 vynásobit obě poloviny hodnotou $x[n]$, resp. $y[n]$, je nutné podobně připravit do horní i spodní poloviny dalšího registru hodnotu $x[n]$, resp. $y[n]$. K tomu je použita instrukce PACKH2, která pracuje se třemi operandy. Instrukce vloží horní 16bitovou polovinu prvního operandu do horní 16bitové poloviny posledního operandu. Současně do spodní 16bitové poloviny posledního operandu vloží horní 16bitovou polovinu druhého operandu. Pokud v registru A5 je 32bitový vzorek výstupního signálu $y[n]$, pak instrukce PACKH2 A5, A5, A7 vloží do horní i spodní poloviny registru A7 horních 16 bitů registru A5, tj. obě poloviny obsahují nejvýznamnější bity vzorku výstupního signálu $y[n]$.

0
0
$-\frac{d_{21}}{2}$
$-\frac{d_{11}}{2}$
$\frac{c_{21}}{2}$
$\frac{c_{11}}{2}$
0
$\frac{c_{01}}{2}$

Obr. 4.14: Ilustrační zobrazení uložení koeficientů pro implementaci 1. kanonické formy s využitím principu SIMD.

Ukázka sekvenčně prováděného kódu s využitím principu SIMD je na následujícím výpisu. Instrukční pakety označené '^' jsou opět limitující pro délku jádra. Pakety jsou v podstatě tytéž jako v případě implementace bez SIMD. V případě sériového zapojení sekcí je totiž omezující doba pro výpočet vzorku výstupního signálu, kterou SIMD princip neovlivní. Teprve poté je možné zahájit implementaci další sekce.

Jednotlivé iterace je možné zpracovávat paralelně, přičemž minimální doba jedné iterace je počet hodinových cyklů mezi prvním posledním instrukčním paketem v uvedené sekvenční posloupnosti označené stříškou '^', tj. $10 - 5 + 1 = 6$ hodinových cyklů. Jádro cyklu prováděného paralelně tedy bude obsahovat minimálně 6 instrukčních paketů zobrazených ve výpisu 4.12.

Výpis 4.11: Příklad sekvenční implementace 1. kanonické formy s využitím principu SIMD.

```

1      ; A8 <-- xn : xn
2  _loop:
3      LDDW    .D1T1    *A6++(16), A5:A4    ; 0
4      NOP                                ; 1
5      LDW     .D1T1    *A3, A8              ; 2
6      LDW     .D2T2    *B5, B7              ; 3
7      NOP                                ; 4
8      ; A5 <-- c2 : c1                    (^0)
9      ; A4 <-- 0 : c0                      (^0)
10     SMPY    .M1      A4, A8, A5           ; 5 ^
11 || LDW     .D2T2    *B6++(16), B9        ; 5
12
13     SMPY2   .M1      A5, A8, A17:A16     ; 6
14     ; A8 <-- v1k / 2                      (^2)
15     ; A5 <-- c0 xn                        (^5)
16     SADD    .L1      A8, A5, A5         ; 7 ^
17     ; A5 <-- c0 xn + v1k / 2 = yn / 2
18     ; B7 <-- v2k / 2                      (^3)
19     SSHL   .S1      A5, 1, A5          ; 8 ^
20     ; A5 <-- yn
21     PACKH2 .L1      A5, A5, A7         ; 9 ^
22     ; A7 <-- yn:yn
23     ; B9 <-- -d2 : -d1                    (^5)
24     ; A17:A16 <-- c2 xn : c1 xn          (^6)
25     MV     .S1      A7, A8             ; 10 ^
26     ; A8 <-- xn = yn
27     SMPY2   .M2X    B9, A7, B9:B8      ; 11
28 || SADD    .L1X    B7, A16, A4        ; 11
29     ; A4 <-- c1k xn + v2k / 2
30 [ B0] BDEC  .S2     _loop, B0          ; 12
31     NOP                                ; 13 14
32     ; B9:B8 <-- -d2 yn: -d1 yn          (^11)
33     SADD    .L2X    A17, B9, B4        ; 15
34     ; B4 <-- c2 xn - d2 yn = v2k / 2
35     SADD    .L1X    A4, B8, A9         ; 16
36 || STW     .D2T2    B4, *B5++         ; 16
37     ; A9 <-- c1 xn + v2k - d1 yn = v1k / 2
38     STW     .D1T1    A9, *A3++         ; 17

```

Výpis 4.12: Příklad jádra paralelní implementace 1. kanonické formy s využitím principu SIMD.

```

1  _loop:
2      [ B0]  BDEC    .S2    _loop,B0        ; <0,12>
3  ||      SMPY2   .M1    A5,A8,A17:A16    ; <1,6>
4  ||      LDDW    .D1T1   *A6++(16),A5:A4    ; <2,0>
5
6          SADD    .L1    A8,A5,A5          ; <1,7>  ^
7
8          SSHL   .S1    A5,1,A5          ; <1,8>  ^
9  ||      LDW     .D1T1   **A3(8),A8        ; <2,2>
10
11         SADD    .L2X   A17,B9,B4        ; <0,15>
12  ||     PACKH2  .L1    A5,A5,A7         ; <1,9>  ^
13  ||     LDW     .D2T2   **B5(8),B7       ; <2,3>
14
15         STW     .D2T2   B4,*B5++        ; <0,16>
16  ||     SADD    .L1X   A4,B8,A9         ; <0,16>
17  ||     MV      .S1    A7,A8           ; <1,10> ^
18
19         STW     .D1T1   A9,*A3++        ; <0,17>
20  ||     SADD    .L1X   B7,A16,A4        ; <1,11>
21  ||     SMPY2   .M2X   B9,A7,B9:B8      ; <1,11>
22  ||     LDW     .D2T2   *B6++(16),B9    ; <2,5>
23  ||     SMPY    .M1    A4,A8,A5         ; <2,5>  ^

```

Jinou možností zavedení operací typu SIMD téměř libovolné délky je rozšíření stavových proměnných. V případě 1. kanonické formy jsou základem stavové rovnice

$$\begin{bmatrix} v_1[n+1] \\ v_2[n+1] \end{bmatrix} = \begin{bmatrix} 0 & -d_2 \\ 1 & -d_1 \end{bmatrix} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \begin{bmatrix} c_2 - d_2c_0 \\ c_1 - d_1c_0 \end{bmatrix} x[n], \quad (4.13)$$

$$y[n] = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \begin{bmatrix} c_0 \end{bmatrix} x[n]. \quad (4.14)$$

Rekurzivním dosazením rovnice (4.13) lze odvodit vztahy pro výpočet stavových proměnných pro následující vstupní vzorek

$$\begin{aligned} \begin{bmatrix} v_1[n+2] \\ v_2[n+2] \end{bmatrix} &= \begin{bmatrix} 0 & -d_2 \\ 1 & -d_1 \end{bmatrix} \begin{bmatrix} v_1[n+1] \\ v_2[n+1] \end{bmatrix} + \begin{bmatrix} c_2 - d_2c_0 \\ c_1 - d_1c_0 \end{bmatrix} x[n+1] \\ &= \mathbf{A} \left(\mathbf{A} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \mathbf{B}x[n] \right) + \mathbf{B}x[n+1], \\ \begin{bmatrix} v_1[n+2] \\ v_2[n+2] \end{bmatrix} &= \mathbf{A}^2 \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \begin{bmatrix} \mathbf{AB} & \mathbf{B} \end{bmatrix} \begin{bmatrix} x[n] \\ x[n+1] \end{bmatrix}. \end{aligned} \quad (4.15)$$

Zobecněním lze získat stavové rovnice pro libovolné posunutí stavových proměnných a tím i délku vektoru vstupního signálu

$$\begin{bmatrix} v_1[n+M] \\ v_2[n+M] \end{bmatrix} = \underbrace{\mathbf{A}^M}_{\mathbf{A}'} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{A}^{M-1}\mathbf{B} & \dots & \mathbf{AB} & \mathbf{B} \end{bmatrix}}_{\mathbf{B}'} \begin{bmatrix} x[n] \\ x[n+1] \\ \vdots \\ x[n+M-1] \end{bmatrix}. \quad (4.16)$$

Pokud např. procesor podporuje operace typu SIMD pro registry složené ze čtyřech dílčích slov, je možné zvolit $M = 4$ a zpracovávat vstupní signál po čtyřech vzorcích. Druhý člen v rovnici (4.16) je pak realizován pouhými dvěma operacemi SIMD násobení. Matici $\mathbf{B}' = [\mathbf{A}^{M-1}\mathbf{B}, \mathbf{A}^{M-2}\mathbf{B}, \dots, \mathbf{AB}, \mathbf{B}]$ o velikosti $2 \times M$ je možné předpočítat před zahájením výpočtu. Podobně lze předpočítat matice \mathbf{A}^M , která zůstává velikosti 2×2 . Vzhledem k tvaru matice \mathbf{A} jsou však všechny prvky nenulové, např.

$$\begin{bmatrix} 0 & -d_2 \\ 1 & -d_1 \end{bmatrix} \begin{bmatrix} 0 & -d_2 \\ 1 & -d_1 \end{bmatrix} = \begin{bmatrix} -d_2 & d_1d_2 \\ -d_1 & d_1^2 - d_2 \end{bmatrix}. \quad (4.17)$$

Z tohoto pohledu je výhodné zaměnit pořadí stavových proměnných, čímž se zamění pořadí řádků matice \mathbf{A} a její mocniny pak mají jeden prvek vždy nulový

$$\begin{bmatrix} 1 & -d_1 \\ 0 & -d_2 \end{bmatrix} \begin{bmatrix} 1 & -d_1 \\ 0 & -d_2 \end{bmatrix} = \begin{bmatrix} 1 & d_1d_2 - d_1 \\ 0 & d_2^2 \end{bmatrix}. \quad (4.18)$$

Tím lze ušetřit jedno násobení. V praxi se však ukazuje, že hodnoty prvků matice klesají rychleji než v původním případě a výpočty v pevné řádové čáře jsou pak méně přesné.

Podobně lze rozšířit rovnici výstupu (4.14)

$$\begin{aligned}
 y[n+1] &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} v_1[n+1] \\ v_2[n+1] \end{bmatrix} + \begin{bmatrix} c_0 \end{bmatrix} x[n+1] \\
 &= \mathbf{C} \left(\mathbf{A} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \mathbf{B}x[n] \right) + \mathbf{D}x[n+1] \\
 &= \mathbf{CA} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \begin{bmatrix} \mathbf{CB} & \mathbf{D} \end{bmatrix} \begin{bmatrix} x[n] \\ x[n+1] \end{bmatrix}.
 \end{aligned} \tag{4.19}$$

Dvojice výstupních vzorků je potom rovna

$$\begin{bmatrix} y[n] \\ y[n+1] \end{bmatrix} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \end{bmatrix} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \begin{bmatrix} \mathbf{D} & 0 \\ \mathbf{CB} & \mathbf{D} \end{bmatrix} \begin{bmatrix} x[n] \\ x[n+1] \end{bmatrix}. \tag{4.20}$$

Rovnici lze zobecnit do tvaru

$$\begin{bmatrix} y[n] \\ y[n+1] \\ \vdots \\ y[n+M] \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^M \end{bmatrix}}_{\mathbf{C}'} \begin{bmatrix} v_1[n] \\ v_2[n] \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{D} & 0 & \dots & 0 \\ \mathbf{CB} & \mathbf{D} & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{CA}^{M-1}\mathbf{B} & \dots & \mathbf{CB} & \mathbf{D} \end{bmatrix}}_{\mathbf{D}'} \begin{bmatrix} x[n] \\ x[n+1] \\ \vdots \\ x[n+M] \end{bmatrix}. \tag{4.21}$$

Nová matice \mathbf{C}' má velikost $M \times 2$ a nová matice \mathbf{D}' velikost $M \times M$. Obě matice lze předpočítat před zahájením zpracování. Velikost M je možné přizpůsobit velikosti vektorů, které je možné v daném systému zpracovávat.

Paralelní 4. kanonická struktura

Trochu jiná situace nastane, pokud bude uvažováno paralelní spojení sekcí 2. řádu. Sekce jsou řazeny paralelně, takže je možné provádět jejich stavové rovnice rovněž paralelně. Výpočet další sekce je možné zahájit ještě před výpočtem výstupního signálu předchozí sekce. Další výhodou je, že při rozkladu do paralelního zapojení mají přenosové funkce dílčích sekcí 2. řádu řád polynomu čitatele roven 1 ($c_0 = 0$). Stavové rovnice tedy přejdou do tvaru

$$\begin{aligned}
 y[n] &= 2v_1[n], \\
 v_1[n] &= c_1x[n] - d_1y[n] + v_2[n], \\
 v_2[n] &= c_2x[n] - d_2y[n].
 \end{aligned} \tag{4.22}$$

Vzorek výstupního signálu je tak připraven okamžitě bez nutnosti čekání na násobení vzorku vstupního signálu.

Ukázka sekvenčně prováděného kódu 4. paralelní struktury je ve výpisu 4.13. Instrukční pakety označené stříškou '^' jsou opět limitující pro délku jádra. Tentokrát to ovšem nejsou pakety pro výpočet vzorku výstupního signálu, ale pakety pro výpočet nových hodnot stavových proměnných.

Minimální doba jedné iterace je počet hodinových cyklů mezi prvním a posledním instrukčním paketem označeným stříškou v uvedené sekvenční posloupnosti, tj. $11 - 7 +$

Výpis 4.13: Příklad sériové implementace 4. kanonické formy v procesoru s architekturou typu VLIW.

```

1      ; A9  <-- xn
2      ; A8  <-- yn = 0
3  _loop
4      LDW    .D1T1    *A6 , A3          ; 0
5      NOP    1          ; 1
6      LDW    .D1T1    *A7++(8) , A16    ; 2
7  ||     LDW    .D2T2    *B5++(8) , B9    ; 2
8      NOP    2          ; 3 4
9      ; A3  <-- v1k          (^0)
10     SSSL   .S1      A3 , 1 , A3        ; 5
11     ; A3  <-- 2 v1k = yn
12     PACKH2 .S1      A3 , A3 , A4      ; 6
13  ||     LDW    .D2T2    *B8 , B4        ; 6
14     ; A4  <-- yn:yn
15     ; A16 <-- c2k:c1k          (^2)
16     ; B9  <-- -d2k:-d1k        (^2)
17     SADD   .L1      A8 , A3 , A8      ; 7
18  ||     SMPY2  .M1      A16 , A9 , A5:A4 ; 7 ^
19     ; A8   <-- sum yk
20     SMPY2  .M2X      B9 , A4 , B7:B6   ; 8
21  [ B0] BDEC  .S2      _loop , B0      ; 9
22     NOP    1          ; 10
23     ; B4  <-- v2k/2          (^6)
24     ; A5:A4 <-- c2 xn:c1 xn        (^7)
25     SADD   .L1X      B4 , A4 , A10     ; 11 ^
26     ; A10 <-- c1xn + v2k/2
27     ; B7:B6 <-- -d2yn:-d1yn        (^8)
28     SADD   .L2X      A5 , B7 , B4     ; 12
29     ; B4  <-- c2xn - d2yn = v2k/2
30     SADD   .L1X      A10 , B6 , A3     ; 13
31  ||     STW    .D2T2    B4 , *B8++    ; 13
32     ; A3  <-- c1xn - d1yn + v2k/2 = v1k/2
33     STW    .D1T1      A3 , *A6++      ; 14

```


1 = 5 hodinových cyklů. Jádro cyklu prováděného paralelně tedy bude obsahovat pouze 5 instrukčních paketů, což je o jeden méně než v předchozích případech jak je patrné z výpisu 4.14.

Výpis 4.14: Příklad paralelní implementace 4. kanonické formy v procesoru s architekturou typu VLIW.

1	_loop				
2		SSHL	.S1	A3,1,A3	; <1,5>
3		LDW	.D1T1	*+A6(8),A3	; <2,0>
4					
5		SADD	.L1X	B4,A4,A10	; <0,11> ^
6		LDW	.D2T2	*+B8(4),B4	; <1,6>
7		PACKH2	.S1	A3,A3,A4	; <1,6>
8					
9		SADD	.L2X	A5,B7,B4	; <0,12>
10		SADD	.L1	A8,A3,A8	; <1,7>
11		SMPY2	.M1	A16,A9,A5:A4	; <1,7> ^
12		LDW	.D2T2	*B5++(8),B9	; <2,2>
13		LDW	.D1T1	*A7++(8),A16	; <2,2>
14					
15		STW	.D2T2	B4,*B8++	; <0,13>
16		SADD	.L1X	A10,B6,A3	; <0,13>
17		SMPY2	.M2X	B9,A4,B7:B6	; <1,8>
18					
19		STW	.D1T1	A3,*A6++	; <0,14>
20	[B0]	BDEC	.S2	_loop,B0	; <1,9>

I v případě paralelní 4. kanonické formy je možné stavové rovnice upravit pro využití principu SIMD. Protože stavové proměnné a výstupy jednotlivých sekcí jsou nezávislé, je výsledek o něco vhodnější pro paralelizaci. Stavové rovnice jedné sekce mají tvar

$$\begin{bmatrix} v_{1k}[n+1] \\ v_{2k}[n+1] \end{bmatrix} = \begin{bmatrix} 0 & -d_{2k} \\ 1 & -d_{1k} \end{bmatrix} \begin{bmatrix} v_{1k}[n] \\ v_{2k}[n] \end{bmatrix} + \begin{bmatrix} c_{2k} - d_{2k}c_{0k} \\ c_{1k} - d_{1k}c_{0k} \end{bmatrix} x[n], \quad (4.23)$$

$$y_k[n] = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} v_{1k}[n] \\ v_{2k}[n] \end{bmatrix} + \begin{bmatrix} c_{0k} \end{bmatrix} x[n]. \quad (4.24)$$

Všechny sekce mají společný vstupní signál $x[n]$ a celkový výstupní signál je dán součtem výstupních signálů dílčích sekcí $y[n] = \sum_{k=1}^K y_k[n]$. Proto rovnici výstupu lze vyjádřit

$$y[n] = \begin{bmatrix} y_1[n] \\ y_2[n] \\ \vdots \\ y_K[n] \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \left(\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} v_{11}[n] & v_{21}[n] \\ v_{12}[n] & v_{22}[n] \\ \vdots & \vdots \\ v_{1K}[n] & v_{2K}[n] \end{bmatrix}^T + \begin{bmatrix} c_{01} \\ c_{02} \\ \vdots \\ c_{0K} \end{bmatrix}^T x[n] \right) \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}^T \begin{bmatrix} v_{11}[n] & v_{21}[n] \\ v_{12}[n] & v_{22}[n] \\ \vdots & \vdots \\ v_{1K}[n] & v_{2K}[n] \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}^T x[n] \begin{bmatrix} c_{01} \\ c_{02} \\ \vdots \\ c_{0K} \end{bmatrix}. \quad (4.25)$$

První člen rovnice přejde do součtu stavových proměnných $\sum_{k=1}^K v_{2k}[n]$ a druhý do součinu vstupního vzorku $x[n]$ a součtu koeficientů $\sum_{k=1}^K c_{0k}$, který je možné předpočítat před zahájením algoritmu. Obě operace lze snadno implementovat s využitím SIMD principu.

Stavové rovnice je vhodné rozdělit na dvě samostatné pro stavovou proměnnou $v_{1k}[n]$ a $v_{2k}[n]$. Pro stavovou proměnnou $v_{1k}[n]$ lze odvodit

$$\begin{bmatrix} v_{11}[n+1] \\ v_{12}[n+1] \\ \vdots \\ v_{1K}[n+1] \end{bmatrix} = \text{diag} \left(\begin{bmatrix} -d_{21} \\ -d_{22} \\ \vdots \\ -d_{2K} \end{bmatrix} \begin{bmatrix} v_{21}[n] \\ v_{22}[n] \\ \vdots \\ v_{2K}[n] \end{bmatrix}^T \right) + \begin{bmatrix} c_{21} - d_{21}c_{01} \\ c_{22} - d_{22}c_{02} \\ \vdots \\ c_{2K} - d_{2K}c_{0K} \end{bmatrix} x[n]. \quad (4.26)$$

První člen tvoří součin prvků obou vektorů, druhý člen je pak násobení vektoru skalární hodnotou vzorku $x[n]$. Obě operace jsou jednoduše realizovatelné s využitím principu SIMD.

Podobný vztah lze odvodit i pro druhou stavovou proměnnou

$$\begin{bmatrix} v_{21}[n+1] \\ v_{22}[n+1] \\ \vdots \\ v_{2K}[n+1] \end{bmatrix} = \begin{bmatrix} v_{11}[n] \\ v_{12}[n] \\ \vdots \\ v_{1K}[n] \end{bmatrix} + \text{diag} \left(\begin{bmatrix} -d_{11} \\ -d_{12} \\ \vdots \\ -d_{1K} \end{bmatrix} \begin{bmatrix} v_{21}[n] \\ v_{22}[n] \\ \vdots \\ v_{2K}[n] \end{bmatrix}^T \right) + \begin{bmatrix} c_{11} - d_{11}c_{01} \\ c_{12} - d_{12}c_{02} \\ \vdots \\ c_{1K} - d_{1K}c_{0K} \end{bmatrix} x[n]. \quad (4.27)$$

Vztah opět obsahuje pouze vektorové operace snadno implementovatelné s využitím principu SIMD.

Je zřejmé, že úprava stavových rovnic 4. kanonické formy pro vektorové operace je mnohem jednodušší než v případě kaskádní 3. kanonické formy. Výhodou také je, že koeficienty v upravených stavových rovnicích jsou v podstatě totožné s původními koeficienty a tak nedochází k jejich snižování jako v případě 3. kanonické formy. To je důležité zejména při implementaci v pevné řádové čarce, kdy by kvantování snížených hodnot koeficientů mohlo způsobit vážné problémy se přesností.

Porovnání výpočetní náročnosti

Při implementaci podle výpisu 4.9 je na jednu sekci potřeba 16 hodinových cyklů. Celková výpočetní náročnost tak je přibližně $16K$, kde K je počet sekcí 2. řádu zapojených sériově.

Pokud bude využito zřetěžené zpracování cyklů (software pipelining), zkrátí se doba jádra na pouhých 6 hodinových cyklů, jak plyne z výpisu 4.10. Jádro však předpokládá, že počet sekcí je 3 a více, pro nižší počet sekcí nelze implementaci použít. Zároveň režijní činnosti spojené s přípravou před zahájením cyklu (*prolog*) nebo ukončením cyklu

(*epilog*) spotřebují 10 hodinových cyklů. Celková výpočetní náročnost je odhadována jako $10 + 6 * K$ hodinových cyklů. Pro 3 sekce je tedy snížení ze $16 \cdot 3 = 48$ na $10 + 6 \cdot 3 = 28$ hodinových cyklů.

Výpis 4.12 využívá navíc ještě princip SIMD. Jádro opět předpokládá, že počet sekcí je 3 a více. Výpočetní náročnost je odhadována jako $12 + 6 * K$ hodinových cyklů. Délka jádra je tedy stejná jako v případě bez použití SIMD zpracování, ale sekce prolog a epilog jsou o dva hodinové cykly delší. Pro 3 sekce je celková výpočetní náročnost přibližně $12 + 6 \cdot 3 = 30$ hodinových cyklů. Využití principu SIMD pro implementaci 1. kanonické formy tedy nepřináší u procesorů VelociTI žádnou výhodu. Je to způsobeno zejména tím, že instrukce pro násobení datových paketů SMPY2 sice provádí dvě násobení paralelně, což by teoreticky mělo přinést snížení výpočetní náročnosti na polovinu, ale naopak na dokončení spotřebuje 4 hodinové cykly namísto 2 pro jednoduchou instrukci SMPY. Dvojnásobný počet násobení je tak vyrovnán dvojnásobnou délkou trvání instrukce. U procesorů, kde by obě instrukce trvaly přibližně stejnou dobu, by se pokles výpočetní náročnosti díky SIMD instrukcím projevil.

Naopak ke snížení výpočetní náročnosti dojde, pokud bude systém implementován ve 4. kanonické formě paralelního zapojení sekcí 2. řádu. Podle výpisu 4.14 spotřebuje jádro paralelní implementace 5 hodinových cyklů. Výpočetní náročnost je odhadována jako $10 + 5 * K$ hodinových cyklů, kde K je počet sekcí, 5 je délka jádra, a 10 je počet hodinových cyklů pro sekce algoritmu prolog a epilog. Opět to platí za předpokladu, že počet sekcí je 3 a více. Pro 3 sekce je celková výpočetní náročnost $10 + 5 \cdot 3 = 25$ hodinových cyklů. Tzn. že pro každý 1 vzorek vstupního signálu bude výpočet přibližně o 3 hodinové cykly rychlejší než v případě sériové implementace ve 3. kanonické formě.

4.3 Shrnutí postupů optimalizace

Z uvedeného vyplývá, že postupy optimalizace struktur výpočetních algoritmů pro moderní architekturu signálových procesorů s velmi dlouhým slovem se zcela odlišují od postupů optimalizace struktur pro signálové procesory s harvardskou architekturou. Důvodem je větší míra paralelního zpracování ať už v podobě více současně pracujících jednotek v jádře procesoru nebo ve využití paralelního zpracování dat typu SIMD i větší počet datových registrů DALU. S větší mírou paralelního zpracování přiměřeně roste i teoretický výpočetní výkon. Zvýšení výpočetního výkonu v praktických aplikacích však závisí na struktuře výpočetního algoritmu. Největší přínos lze očekávat u aplikací zpracovávajících paralelně několik nezávislých kanálů stejným algoritmem. V takovém případě je evidentně možné algoritmus pro jednotlivé kanály zpracovávat současně s využitím maximálního počtu paralelních jednotek jádra.

Příkladem může být implementace řečových kodeků v signálových procesorech telefonních ústředen. Každý kanál je nezávislý na ostatních a proto je možné kódování všech kanálů provádět paralelně. Postupy optimalizace uvedené v této kapitole byly použity např. při optimalizaci referenčních zdrojových kódů organizace ITU (International Telecommunication Union) kodeků GSM Half Rate, GSM Full Rate a G.723.1 pro firmu STROM Telecom [98, 93, 31]. Firma v telefonních ústřednách využívá signálové procesory TMS320C6416 s velmi dlouhým instrukčním slovem VLIW. Výpočetní náročnost

v počtu hodinových cyklů potřebných pro zpracování jednoho rámce je uvedena v tab. 4.1. Je zřejmé, že díky optimalizaci výpočetní náročnost klesla i více než 10-krát. Díky tomu jeden procesor s hodinovým kmitočtem 1 GHz může zpracovávat až 30 současně probíhajících telefonních kanálů, tzn. může zpracovávat celé telefonní rámce E1.

Tab. 4.1: Porovnání výpočetní náročnosti procesu kódování a dekódování v signálovém procesoru TMS320C6416 pro vybrané řečové kodeky před a po optimalizaci.

proces kodek	počet hodinových cyklů pro zpracování 1 rámce			
	kódování		dekódování	
	původní	optimalizovaný	původní	optimalizovaný
GSM Half Rate	8 899 171	361 161	974 413	46 349
GSM Full Rate	25 759 990	299 324	1 505 949	38 122
G.723.1 6,3 kbit/s	14 980 051	802 545	3 386 378	112 045
G.723.1 5,3 kbit/s	13 093 041	546 053	2 127 424	105 939

Uvedené postupy optimalizace se neuplatní pouze u výpočetně vysoce náročných aplikací, jako byl výše uvedený případ řečových kodeků. Další aplikací, při které byly využity, byl návrh a implementace filtrů pro měřič kvality napětí rozvodné sítě podle normy [46] pro firmu MEgA. Měřič byl implementován v signálovém procesoru firmy Texas Instruments TMS320C5510 s harvardskou architekturou [70]. Výsledek implementace je využíván v přenosném PQ monitoru MEg37.

5 VLIV OMEZENÉ DÉLKY SLOVA NA ČÍSLICOVÉ ZPRACOVÁNÍ

Při optimalizaci struktur výpočetních algoritmů často dochází k rozdělení algoritmu na dílčí části, k záměně pořadí provádění dílčích částí, apod. V případě lineárního časově invariantního systému tyto operace nemají mít vliv na výsledný signál. Ovšem při implementaci v pevné řádové čarce však u každé z aritmetických operací dochází ke vzniku kvantovacího šumu. Původně lineární operace se stanou nelineárními a v takovém případě záleží na pořadí jejich provádění. Proto se v následující kapitole habilitační práce věnuje problémům implementace číslicových systémů způsobených omezenou délkou slova.

5.1 Kvantování koeficientů číslicového systému

5.1.1 Kvantování koeficientů systémů typu FIR

V případě implementace v procesorech s pevnou nebo pohyblivou řádovou čárkou, jsou v použitém formátu jsou vyjádřeny nejen vzorky signálu, ale také koeficienty přenosové funkce. V případě nerekurzivních systémů (bez zpětných vazeb nebo též s konečnou impulsní charakteristikou FIR (Finite Impulse Response)) je vliv kvantování koeficientů relativně malý. Přenosová funkce má totiž póly v počátku

$$H(z) = \frac{\sum_{k=0}^{N-1} b_k z^k}{a_{N-1} z^{N-1}} = \sum_{k=0}^{N-1} \frac{b_k}{a_{N-1}} z^{-k-N+1} \equiv \sum_{n=0}^{\infty} h[n] z^{-n} \quad (5.1)$$

a koeficienty jsou přímo rovny vzorkům impulsní charakteristiky $h[n]$. Vzorky impulsní charakteristiky jsou většinou $|h[n]| \leq 1$ a tak při použití zlomkových čísel ve dvojkovém doplňku nehrozí přetečení. Pokud by některý z koeficientů $\frac{b_k}{a_{N-1}}$ byl mimo rozsah dvojkového doplňku, je možné provést vynásobení všech koeficientů hodnotou $\beta < 1$ tak, aby byl rozsah dodržen. Vynásobení všech koeficientů odpovídá vynásobení celé přenosové funkce a výstupní signál tedy bude k krát zeslaben. To lze kompenzovat vynásobením výstupního signálu hodnotou $\frac{1}{\beta}$.

I když budou koeficienty v rozsahu zvoleného formátu, dojde k jejich změně vlivem kvantování. Chyby způsobené kvantováním jsou však mnohem menší než v případě chyb z přetečení. Dále s ohledem na skutečnost, že koeficienty přímo udávají hodnoty vzorků impulsní charakteristiky a ta přímo definuje přenosovou funkci, budou malé i odchylky kmitočtové charakteristiky.

5.1.2 Kvantování koeficientů systémů typu IIR

V případě rekurzivních systému (IIR (Infinite Impulse Response)) je impulsní charakteristika nekonečná a její průběh ovlivňují koeficienty přenosové funkce nepřímo. Průběh impulsní charakteristiky je ovlivněn polohou pólů přenosové funkce (kořenů jmenovatele přenosové funkce). Uvažujme čistě rekurzivní číslicový systém s přenosovou funkcí 2. řádu

ve tvaru

$$H(z) = \frac{b_0}{z^2 + a_1 z + a_0} \quad (5.2)$$

s reálnými koeficienty. Vzhledem k reálným koeficientům bude mít jmenovatel dva komplexně sdružené kořeny $z_{x1,2} = re \pm jim$. Potom jmenovatel lze vyjádřit jako

$$\begin{aligned} & (z - (re + jim)) \cdot (z - (re - jim)) = \\ = & z^2 - re \cdot z - jim \cdot z - re \cdot z + re^2 - jre \cdot im - jim \cdot z + jre \cdot im - j^2 im^2 = \\ & = z^2 - 2re \cdot z + (re^2 + im^2). \end{aligned} \quad (5.3)$$

Ze srovnání jmenovatele v rovnici (5.2) a vztahu (5.3) plyne, že koeficient $a_1 = -2re$ je roven záporně vzatému dvojnásobku reálné složky pólu a koeficient $a_0 = re^2 + im^2$ je roven kvadrátu absolutní hodnoty pólu. Kvantování koeficientu a_1 tedy ovlivní reálnou část pólu a kvantování koeficienty a_0 ovlivní jeho modul. Pokud jsou oba koeficienty vyjádřeny ve zvoleném číselném formátu, musejí oba být celočíselným násobkem kvantovacího kroku

$$a_1 = -2re \equiv m \cdot q, m \in \mathbb{Z}, \quad (5.4)$$

$$a_0 = re^2 + im^2 \equiv n \cdot q, n \in \mathbb{Z}. \quad (5.5)$$

Přenosová funkce s reálnými koeficienty může mít také dvojici reálných pólů. V takovém případě lze jmenovatel vyjádřit následovně:

$$\begin{aligned} & (z - re_1) \cdot (z - re_2) = \\ = & z^2 - re_1 \cdot z - re_2 \cdot z + re_1 \cdot re_2 = \\ & = z^2 - (re_1 + re_2) \cdot z + re_1 \cdot re_2. \end{aligned} \quad (5.6)$$

Ze srovnání jmenovatele v rovnici (5.2) a vztahu (5.6) plyne, že koeficient $a_1 = -(re_1 + re_2)$ je roven záporně vzatému součtu reálných složek obou pólů a koeficient $a_0 = re_1 \cdot re_2$ je součinu reálných složek obou pólů. Kvantování koeficientu a_1 i a_2 tedy ovlivní polohu obou pólů, jejich součet resp. součin. Lze odvodit, že dvojice reálných kořenů musí splňovat podmínku

$$r_1, r_2 = \frac{kq \pm \sqrt{n^2 q^2 + 4mq}}{2}; m, n \in \mathbb{Z}.$$

V případě systémů vyšších řádů je situace ještě složitější, neboť jednotlivé póly se vzájemně ovlivňují. Předpokládejme polynom 4. řádu s reálnými koeficienty, který lze vyjádřit jako součin kořenových činitelů

$$\begin{aligned} & (z - (re_1 + jim_1)) \cdot (z - (re_1 - jim_1)) \cdot (z - (re_1 + jim_1)) \cdot (z - (re_1 - jim_1)) \\ & = (z^2 - 2re_1 \cdot z + (re_1^2 + im_1^2)) \cdot (z^2 - 2re_1 \cdot z + (re_1^2 + im_1^2)) = \\ & = z^4 - 2(re_1 + re_2) z^3 + (re_1^2 + re_2^2 + 4re_1 re_2 + im_1^2 + im_2^2) z^2 - \\ & - 2(re_1(re_2^2 + im_2^2) + re_2(re_1^2 + im_1^2)) z + (re_1^2 + im_1^2)(re_2^2 + im_2^2). \end{aligned}$$

Kvantování jednotlivých koeficientů ovlivní vzájemnou polohu pólů, protože musí být splněno:

$$\begin{aligned}
-2(re_1 + re_2) &\equiv m \cdot q, m \in \mathbb{N}, \\
(re_1^2 + re_2^2 + 4re_1re_2 + im_1^2 + im_2^2) &\equiv n \cdot q, n \in \mathbb{N}, \\
-2(re_1(re_2^2 + im_2^2) + re_2(re_1^2 + im_1^2)) &\equiv k \cdot q, k \in \mathbb{N}, \\
(re_1^2 + im_1^2)(re_2^2 + im_2^2) &\equiv o \cdot q, o \in \mathbb{N}.
\end{aligned}$$

Je zřejmé, že splnit všechny tyto podmínky bude mnohem složitější než u polynomu 2. řádu a vliv kvantování na polohu kořenů polynomu roste s řádem polynomu. Proto se číslicové systémy vyšších řádů rozdělují do sekcí druhého řádu spojených sériově nebo paralelně, viz část 5.2. Neplatí to ale v případě křížové struktury, kde rozlišení polohy kořenů s vyšším řádem neklesá.

Podobně lze postupovat i v případě polynomu čitatele přenosové funkce a uvedené závěry musí platit i pro případné nulové body.

Kvantování koeficientů v první a druhé kanonické struktuře

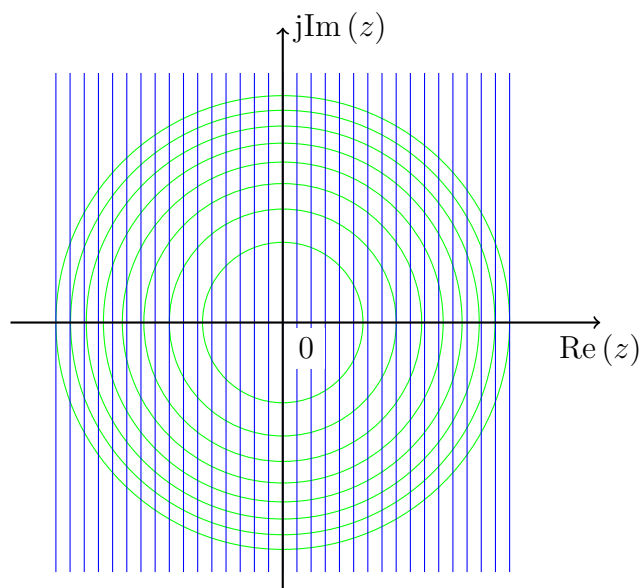
V případě první a druhé kanonické struktury probíhá kvantování přímo koeficientů a_1, a_0 . Rovnice (5.4) omezuje polohu kořene tak, že reálná část musí být celočíselným násobkem poloviny kvantovacího kroku. V grafické reprezentaci musí ležet na svislých přímkách vzájemně vzdálených o polovinu kvantovacího kroku $re \equiv m \frac{q}{2}, m \in \mathbb{Z}$. Rovnice (5.5) omezuje polohu pólu tak, že kvadrát modulu pólu musí být celočíselným násobkem kvantovacího kroku. V grafické podobě musí ležet na soustředných kružnicích, jejichž poloměr je odmocninou celočíselného násobku kvantovacího kroku $|z_x| \equiv \sqrt{mq}, m \in \mathbb{N}$. Po kvantování tak mohou kořeny ležet pouze na průsečících svislých přímek a soustředných kružnic podle obr. 5.1.

Pokud kořeny přenosové funkce podmínky (5.4) a (5.5) nesplňují, dojde po kvantování k jejich posunutí do místa blízkého průsečíku. Z obr. 5.1 je patrné, že v okolí bodu $\langle 1; 0 \rangle$ a $\langle -1; 0 \rangle$ je mnohem menší hustota průsečíků než v okolí bodů $\langle 0; j \rangle$ a $\langle 0; -j \rangle$. Kvantování tak mnohem více ovlivní přenosové funkce s propustným pásmem v okolí nízkých kmitočtů $f \rightarrow 0$ (póly v okolí bodu $\langle 1; 0 \rangle$) nebo nejvyšších kmitočtů $f \rightarrow \frac{f_{vz}}{2}$ (póly v okolí bodu $\langle -1; 0 \rangle$) než přenosové funkce s propustným pásmem v okolí středních kmitočtů $f \rightarrow \frac{f_{vz}}{4}$ (póly v okolí bodů $\langle 0; j \rangle$ a $\langle 0; -j \rangle$).

Protože součástí mřížky je i jednotková kružnice, mohou se kořeny ležící blízko jednotkové kružnice přemístit na jednotkovou kružnici. V případě pólu se tak stabilní systém stane systémem na mezi stability.

Kvantování koeficientů ve vazební struktuře

V případě vazební struktury je situace odlišná. V této struktuře koeficienty I a R vyjadřují přímo reálnou a imaginární složku pólu (viz část 1.2.3). V takovém případě předchozí analýza vlivu kvantování neplatí a kvantování koeficientů znamená přímo kvantování



Obr. 5.1: Zobrazení možných poloh kořenů polynomu 2. řádu při kvantování na 4 bity pro první a druhou kanonickou strukturu. Kořeny mohou ležet pouze na průsečících svislých čar a soustředných kružnic.

reálné a imaginární složky pólů

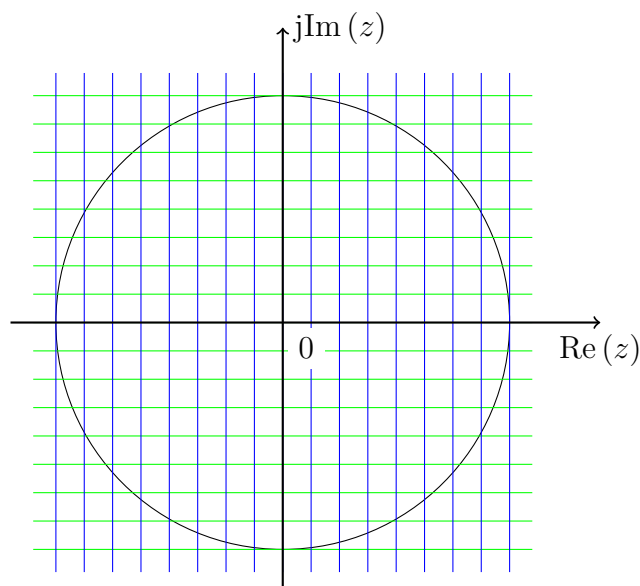
$$R = re \equiv k \cdot q, k \in \mathbb{Z},$$

$$I = im \equiv m \cdot q, m \in \mathbb{Z}.$$

V grafické podobě musí kořen ležet na některé ze svislých čar vzdálených o kvantovací krok a zároveň na některé z vodorovných čar vzdálených opět o kvantovací krok. Kořeny tedy mohou ležet na průsečících pravidelné sítě podle obr. 5.2.

Z obr. 5.2 je patrné, že změna polohy pólů je v celé ploše přibližně rovnoměrná. Oproti první nebo druhé kanonické struktuře tak kvantování ovlivní přenosové funkce bez rozdílu na tvar jejich kmitočtové charakteristiky (polohu nulových bodů a pólů). V případě realizace dolní nebo horní propusti se z tohoto pohledu jeví vazební struktura jako vhodnější, ale její maximální přenos je roven imaginární složce pólu I . V případě dolní nebo horní propusti, kdy je pól umístěn v blízkosti bodu $\langle 1; 0 \rangle$, resp. $\langle -1; 0 \rangle$, je imaginární složka pólu malá a proto je malý i přenos. Proto se vazební struktura využívá častěji pro realizaci rezonátorů než pro realizaci filtrů.

Oproti první nebo druhé kanonické struktuře není jednotková kružnice součástí mřížky (až na několik hodnot). Proto kořeny ležící v blízkosti jednotkové kružnice mohou po kvantování přejít mimo jednotkovou kružnici a stabilní systém se stane nestabilním. V případě rezonátoru pak systém na mezi stability buď póly přejdou vně jednotkové kružnice nebo dovnitř jednotkové kružnice. V prvním případě systém přejde do nestabilního stavu a amplituda generovaných kmitů rezonátoru bude narůstat. V druhém případě se pak systém stane stabilním a amplituda generovaných kmitů bude naopak klesat až kmitání zcela ustane.



Obr. 5.2: Zobrazení možných poloh kořenů polynomu 2. řádu při kvantování na 4 bity u vazební struktury. Kořeny mohou ležet pouze na průsečících pravidelné sítě.

Kvantování koeficientů v křížové struktuře

Nejsložitější situace nastane u křížové struktury. Koeficienty v křížové struktuře nejsou ani koeficienty polynomu, ani reálná nebo imaginární složka kořene. Koeficienty jsou koeficienty odrazu (viz část 1.2.4). Koeficienty polynomu ${}^M A(z) = \sum_{k=0}^M a_k z^k$ řádu M získáme z koeficientů odrazu k_m rekurzivním postupem:

1. Začneme polynomem řádu $m = 1$,
2. ${}^m a_m = 1$
3. ${}^m a_0 = k_m$,
4. ${}^m a_k = {}^{m-1} a_{k-1} + {}^{m-1} a_{m-k-1} k_m, k \in \langle 1; m-1 \rangle$,
5. a opakuje se bod 2 pro $m+1$ až po M .

V případě systému 1. řádu má polynom tvar:

$${}^1 A(z) = z + k_1.$$

Kořen polynomu tedy musí být násobkem kvantovacího kroku formátu použitého pro koeficient odrazu. Pro 16bitový dvojkový doplněk by kořen byl

$$z_x = -k_1 \equiv k 2^{-15}, k \in \mathbb{Z}$$

Po zvýšení řádu má polynom tvar

$${}^2 A(z) = z^2 + (k_1 + k_1 k_2)z + k_2.$$

Koeficient ${}^2 a_0 = k_2$ má stejný počet bitů jako koeficient odrazu. Díky zdvojnásobení počtu bitů při násobení však má koeficient ${}^2 a_1 = k_1 + k_1 k_2$ dvojnásobný počet bitů. Při srovnání s rovnicí (5.3) dojdeme k podobným závěrům, ale kvantovací krok pro koeficient

2a_1 bude mít dvojnásobný počet bitů:

$$a_1 = -2re \equiv m \cdot q + m \cdot n \cdot q^2, m \in \mathbb{Z} \quad (5.7)$$

$$a_0 = re^2 + im^2 \equiv n \cdot q, n \in \mathbb{Z} \quad (5.8)$$

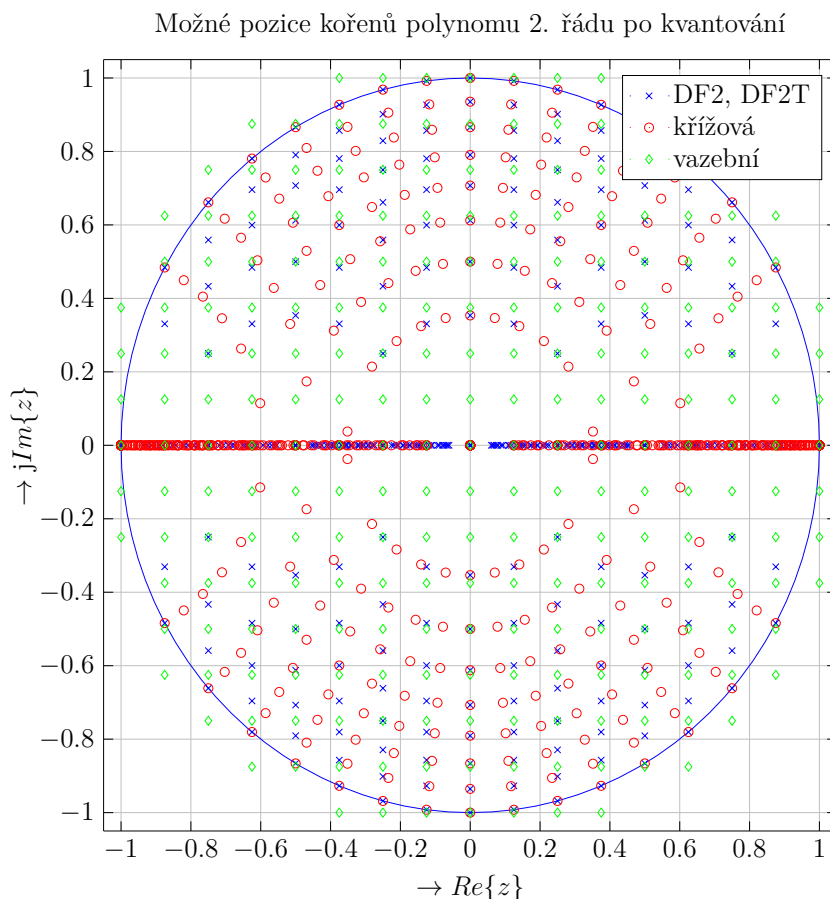
Kořeny tak musí ležet na průsečících soustředných kružnic a svislých čar podobně jako v případě první nebo druhé kanonické struktury (viz obr. 5.1), ale svislé čáry jsou v tomto případě mnohem hustěji rozmístěné.

V případě polynomu 3. řádu je poslední koeficient ${}^3a_0 = k_3$ opět kvantován na stejný počet bitů jako koeficient odrazu. Ostatní koeficienty budou mít tvar:

$${}^3a_1 = {}^2a_0 + {}^2a_1k_3 = k_2 + (k_1 + k_1k_2)k_3 = k_2 + k_1k_3 + k_1k_2k_3,$$

$${}^3a_2 = {}^2a_1 + {}^2a_0k_3 = k_1 + k_1k_2 + k_2k_3.$$

Vidíme, že počet bitů koeficientu 3a_1 vzroste na trojnásobek, v případě 3a_2 na dvojnásobek, původního počtu bitů. Obecně lze tedy říci, že vyšší řád polynomu zvyšuje několika násobně rozlišení a přesnost polohy kořenů realizovaného polynomu. Z tohoto důvodu je křížová struktura málo citlivá na kvantování koeficientů.



Obr. 5.3: Porovnání možného umístění kořenů čitatele nebo jmenovatele přenosové funkce 2. řádu po kvantování koeficientů různých struktur na čtyři bity ve formátu Q0.3.

5.2 Rozdělení na sekce 2. řádu

Jak je ukázáno v části 5.1 se vzrůstajícím řádem přenosové funkce roste citlivost na kvantování koeficientů. Proto jsou přenosové funkce vyššího řádu rozkládány na sekce 2. řádu. Přenosové funkce 1. řádu by mohly mít komplexní koeficienty, čemuž se snažíme vyhnout. Podle (5.3) je 2. řád nejnižší, kdy jsou zaručeny reálné koeficienty přenosové funkce. Je několik možností, jak rozdělit celkovou přenosovou funkci na sekce 2. řádu, z nichž nejběžnější jsou: *sériové spojení* a *paralelní spojení*.

5.2.1 Rozdělení do sériového spojení sekcí 2. řádu

Rozdělení celkové přenosové funkce do sériového spojení sekcí 2. řádu se někdy označuje jako 3. kanonická struktura (viz obr. 5.4). Celková přenosová funkce je potom rovna součinu dílčích přenosových funkcí

$$H(z) = \prod_{k=0}^{K-1} H_k(z).$$

Jako racionálně lomenou funkci ji lze rozložit na součin dílčích racionálně lomených funkcí

$$H(z) = \frac{\sum_{r=0}^R b_r z^r}{\sum_{s=0}^S a_s z^s} = \frac{b_R z^R + \sum_{r=0}^{R-1} \frac{b_r}{b_R} z^r}{a_S z^S + \sum_{s=0}^{S-1} \frac{a_s}{a_S} z^s} = \beta \frac{\prod_{r=0}^{\frac{R}{2}} z^2 + b_{1r} z + b_{0r}}{\prod_{s=0}^{\frac{S}{2}} z^2 + a_{1s} z + a_{0s}}, \quad (5.9)$$

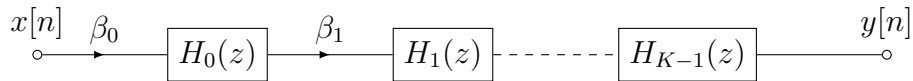
kde

$$H_k = \beta_k \frac{z^2 + b_{1r} z + b_{0r}}{z^2 + a_{1s} z + a_0} \quad (5.10)$$

je přenosová funkce dílčí sekce složená z dvojice kořenů čitatele a z dvojice kořenů jmenovatele celkové přenosové funkce a

$$\beta = \prod_{k=0}^{K-1} \beta_k \quad (5.11)$$

je měřítkový koeficient, který zabraňuje přetečení výstupního signálu.



Obr. 5.4: Sériové zapojení sekcí druhého řádu pro realizaci celkové přenosové funkce.

Aby měla přenosová funkce dílčí sekce 2. řádu reálné koeficienty, musí vzniknout sloučením komplexně sdružených kořenů nebo čistě reálných kořenů čitatele a jmenovatele celkové přenosové funkce.

Uvažujme přenosovou funkci typu dolní propust 6. řádu

$$H(z) = \frac{10^{-3} (0,7388z^6 - 0,1286z^5 + 1,2485z^4 + 0,1494z^3 + 1,2485z^2 - 0,1286z + 0,7388)}{z^6 - 4,84441z^5 + 10,30688z^4 - 12,24803z^3 + 8,54811z^2 - 3,31795z + 0,55974}. \quad (5.12)$$

Přenosová funkce má 6 nulových bodů a pólů vždy po dvojicích komplexně sdružených

k	z_{0k}	z_{xk}
0	$-0,62010 \pm 0,78452j$	$0,83522 \pm 0,16218j$
1	$0,23508 \pm 0,97198j$	$0,80070 \pm 0,42484j$
2	$0,47210 \pm 0,88154j$	$0,78629 \pm 0,56823j$

V případě implementace v pohyblivé řádové čárce nezávisí na pořadí, v jakém budou nulové body přiřazovány pólům. V případě implementace v pevné řádové čárce však s ohledem na nelineární operace kvantování záleží na pořadí operací a tím i pořadí sekcí nebo přiřazování nulových bodů a pólů.

V případě systému 2. řádu je jediná možnost přiřazení nulových bodů a pólů. Pokud řád systému vzroste na 4, pak existují čtyři možnosti pořadí a přiřazení

variace v	1. sekce	2. sekce
0	z_{00}, z_{x0}	z_{01}, z_{x1}
1	z_{00}, z_{x1}	z_{01}, z_{x0}
2	z_{01}, z_{x0}	z_{00}, z_{x1}
3	z_{01}, z_{x1}	z_{00}, z_{x0}

Počet variací rychle roste. Počet možných pořadí sekcí odpovídá permutaci bez opakování

$$P(K) = K!,$$

přičemž zvolené pořadí lze aplikovat na nulové body i póly. Celkový počet variací tak odpovídá variaci 2. třídy z počtu možných pořadí

$$V_2(P(K)) = P(K)^2 = (K!)^2.$$

V případě 6. řádu, tj. tří sekcí $K = 3$, je počet variací sestavení sekcí druhého řádu roven už

$$V_2(P(3)) = (3!)^2 = 36.$$

Každá z těchto variací (možné variace pro systém 6. řádu jsou v tab. 5.1) implementuje stejnou celkovou kmitočtovou charakteristiku jako původní přenosová funkce. Jednotlivé variace se však budou lišit v kmitočtových charakteristikách dílčích sekcí a proto se budou lišit i v hodnotách váhovacích koeficientů dílčích sekcí. V případě uvedené přenosové funkce se minimum váhovacího koeficientu pro libovolnou sekci pohybuje mezi

$$\begin{aligned} \sup_v \left(\sup_k \beta_k \right) &= 0,012052, \text{ pro } v = 4, \\ \inf_v \left(\sup_k \beta_k \right) &= 0,069336, \text{ pro } v = 33. \end{aligned}$$

Liší se tedy více jak 5násobně.

Na první pohled by se jako vhodné zdálo vybrat variaci, která toto minimum váhovacího koeficientu přes všechny sekce a variace maximalizuje

$$\sup_v \left(\sup_k \beta_k \right).$$

Tab. 5.1: Možné variace uspořádání nulových bodů a pólů systému 6. řádu do sekcí 2. řádu.

variace v	1. sekce	2. sekce	3. sekce	variace v	1. sekce	2. sekce	3. sekce
0	z_{00}, z_{x0}	z_{01}, z_{x1}	z_{02}, z_{x2}	18	z_{01}, z_{x0}	z_{02}, z_{x1}	z_{00}, z_{x2}
1	z_{00}, z_{x0}	z_{01}, z_{x2}	z_{02}, z_{x1}	19	z_{01}, z_{x0}	z_{02}, z_{x2}	z_{00}, z_{x1}
2	z_{00}, z_{x1}	z_{01}, z_{x0}	z_{02}, z_{x2}	20	z_{01}, z_{x1}	z_{02}, z_{x0}	z_{00}, z_{x2}
3	z_{00}, z_{x1}	z_{01}, z_{x2}	z_{02}, z_{x0}	21	z_{01}, z_{x1}	z_{02}, z_{x2}	z_{00}, z_{x0}
4	z_{00}, z_{x2}	z_{01}, z_{x0}	z_{02}, z_{x1}	22	z_{01}, z_{x2}	z_{02}, z_{x0}	z_{00}, z_{x1}
5	z_{00}, z_{x2}	z_{01}, z_{x1}	z_{02}, z_{x0}	23	z_{01}, z_{x2}	z_{02}, z_{x1}	z_{00}, z_{x0}
6	z_{00}, z_{x0}	z_{02}, z_{x1}	z_{01}, z_{x2}	24	z_{02}, z_{x0}	z_{00}, z_{x1}	z_{01}, z_{x2}
7	z_{00}, z_{x0}	z_{02}, z_{x2}	z_{01}, z_{x1}	25	z_{02}, z_{x0}	z_{00}, z_{x2}	z_{01}, z_{x1}
8	z_{00}, z_{x1}	z_{02}, z_{x0}	z_{01}, z_{x2}	26	z_{02}, z_{x1}	z_{00}, z_{x0}	z_{01}, z_{x2}
9	z_{00}, z_{x1}	z_{02}, z_{x2}	z_{01}, z_{x0}	27	z_{02}, z_{x1}	z_{00}, z_{x2}	z_{01}, z_{x0}
10	z_{00}, z_{x2}	z_{02}, z_{x0}	z_{01}, z_{x1}	28	z_{02}, z_{x2}	z_{00}, z_{x0}	z_{01}, z_{x1}
11	z_{00}, z_{x2}	z_{02}, z_{x1}	z_{01}, z_{x0}	29	z_{02}, z_{x2}	z_{00}, z_{x1}	z_{01}, z_{x0}
12	z_{01}, z_{x0}	z_{00}, z_{x1}	z_{02}, z_{x2}	30	z_{02}, z_{x0}	z_{01}, z_{x1}	z_{00}, z_{x2}
13	z_{01}, z_{x0}	z_{00}, z_{x2}	z_{02}, z_{x1}	31	z_{02}, z_{x0}	z_{01}, z_{x2}	z_{00}, z_{x1}
14	z_{01}, z_{x1}	z_{00}, z_{x0}	z_{02}, z_{x2}	32	z_{02}, z_{x1}	z_{01}, z_{x0}	z_{00}, z_{x2}
15	z_{01}, z_{x1}	z_{00}, z_{x2}	z_{02}, z_{x0}	33	z_{02}, z_{x1}	z_{01}, z_{x2}	z_{00}, z_{x0}
16	z_{01}, z_{x2}	z_{00}, z_{x0}	z_{02}, z_{x1}	34	z_{02}, z_{x2}	z_{01}, z_{x0}	z_{00}, z_{x1}
17	z_{01}, z_{x2}	z_{00}, z_{x1}	z_{02}, z_{x0}	35	z_{02}, z_{x2}	z_{01}, z_{x1}	z_{00}, z_{x0}

Potom koeficienty přenosové funkce budou ze všech variací nejméně zeslabeny, nejméně zeslaben bude i vstupní signál a při implementaci v pevné řádové čárce by mělo docházet nejmenším zaokrouhlovacím chybám. Situace však není tak jednoduchá, neboť vzhledem k (5.11) je celkový váhovací koeficient pro všechny variace konstantní. Navíc kvantovací šum první sekce prochází a je ovlivňován následujícími sekcemi.

Pro uvedený filtr byly sestaveny všechny možné variace sdružování nulových bodů a pólů. Dílčí sekce byly implementovány v první a druhé kanonické formě a přitom byl měřen poměr signálu ku kvantovacímu šumu. Kvantovací šum na výstupu filtru byl stanoven jako

$$e_q[n] = \tilde{y}[n] - y[n], \quad (5.13)$$

kde $\tilde{y}[n]$ je výstupní signál implementace v pevné řádové čárce ve formátu Q0.15 a $y[n]$ je výstupní signál implementace stejné struktury filtru, ale v pohyblivé řádové čárce. Jako vstupní signál byl použit náhodný proces s Gaussovým rozdělením, aby kvantovací šum v jednotlivých částech struktury byl nezávislý.

Při implementaci v pohyblivé řádové čárce se rozdíly mezi výstupním signálem různých struktur pohybovaly řádově v okolí 10^{-12} a volba struktury tak téměř nemá význam z hlediska přesnosti. Při implementaci v pevné řádové čárce však byly rozdíly dobře patrné a lišily se i mezi variacemi v rámci jedné struktury. Při implementaci v první

kanonické formě se hodnota poměru signálu ku kvantovacímu šumu pohyboval mezi

$$\begin{aligned}\sup_v SQNR &= 55,62 \text{ dB, pro } v = 12, \\ \inf_v SQNR &= 68,35 \text{ dB, pro } v = 30.\end{aligned}$$

Rozdíl je tak téměř 13 dB což přibližně odpovídá změně délky slova o 2 bity. Střední hodnota a rozptyl pak byly

$$E(SQNR) = 62,67 \text{ dB} \quad (5.14)$$

$$D(SQNR) = 16,63 \text{ dB} \quad (5.15)$$

V případě druhé kanonické formy je střední hodnota $SQNR$ nižší přibližně o 13 dB

$$E(SQNR) = 50,50 \text{ dB} \quad (5.16)$$

$$D(SQNR) = 0,16 \text{ dB} \quad (5.17)$$

To je patrně způsobeno vlivem zeslabení vstupního signálu měřítkovým koeficientem $\alpha = \frac{1}{32}$, aby se zabránilo přetékání stavových proměnných druhé kanonické formy. I když je výstupní signál vynásoben obrácenou hodnotou $\frac{1}{\alpha} = 32$, takže výstupní signál má stejnou velikost, dojde ke ztrátě nejméně významných bitů vzorků vstupního signálu a k nevratné ztrátě přesnosti. Naproti hodnota $SQNR$ se příliš nemění v závislosti na variaci, čemuž odpovídá mnohem nižší rozptyl.

Také se potvrzuje, že vstupní signál v podobě bílého šumu je pro úroveň výstupního kvantovacího šumu nejlepší možný případ. Pokud byl na vstup struktur přiveden deterministický signál, např. v podobě dvousložkového harmonického signálu s kmitočty 800 a 3000 Hz, pak v případě první kanonické poměr signálu ku kvantovacímu šumu poklesl na

$$\begin{aligned}\sup_v SQNR &= 53,34 \text{ dB, pro } v = 27, \\ \inf_v SQNR &= 61,49 \text{ dB, pro } v = 22.\end{aligned}$$

V případě druhé kanonické pak rozmezí bylo

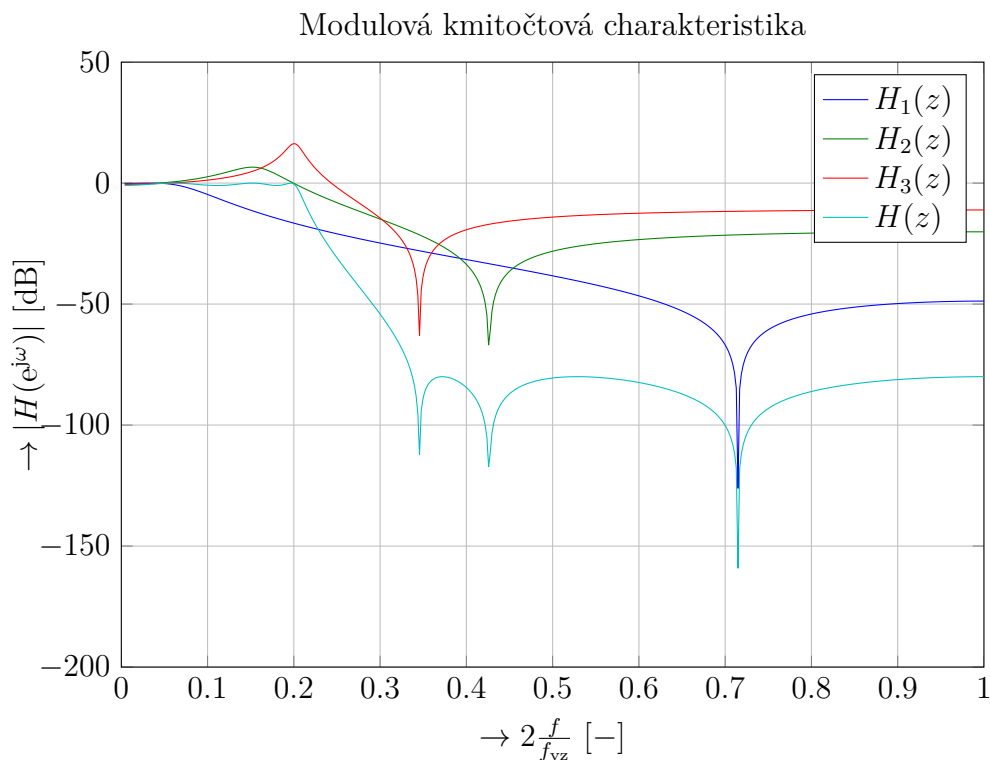
$$\begin{aligned}\sup_v SQNR &= 49,32 \text{ dB, pro } v = 27, \\ \inf_v SQNR &= 53,01 \text{ dB, pro } v = 30.\end{aligned}$$

Je to způsobeno pravděpodobně porušením nezávislosti mezi kvantovacími šумы vznikajícími v různých místech struktury. Na výstupu se pak kvantovací šумы sčítají a to zapříčiní pokles poměru signálu od kvantovacího šumu.

Ukazuje se tedy, že na poměr výstupního signálu od kvantovacího šumu má vliv nejen použitá struktura, ale také pořadí sdružování nulových bodů a pólů i charakter vstupního signálu. Nalézt strukturu filtru a pořadí slučování nulových bodů a pólů optimální pro jakýkoliv vstupní signál je nemožné. Doporučovaným pravidlem je, že sdružování probíhá

od dvojice nevdálenějších nulových bodů a pólů (1. sekce) až po dvojici nejbližších nulových bodů a pólů (poslední sekce), eventuálně v opačném pořadí. Sdružení nejbližších nulových bodů a pólů vede na filtr s největší kvalitou, tj. i s největším maximem modulové kmitočtové charakteristiky.

Pro uvedený filtr doporučené variace nulových bodů a pólů odpovídá variace č. 1. Modulové kmitočtové charakteristiky dílčích sekcí jsou pak zobrazeny na obr. 5.5.



Obr. 5.5: Modulové kmitočtové charakteristiky dílčích sekcí a celkové modulové kmitočtové charakteristiky při sériovém spojení sekcí.

5.2.2 Rozdělení do paralelního zapojení sekcí 2. řádu

Podobně je možné celkovou přenosovou funkci rozdělit do paralelního zapojení sekcí druhého řádu podle obr. 5.6. V takovém případě je celková přenosová funkce dána součtem dílčích přenosových funkcí

$$H(z) = \sum_{k=0}^{K-1} H_k(z). \quad (5.18)$$

Rozdělení odpovídá vlastně rozkladu racionálně lomené funkce na parciální zlomky

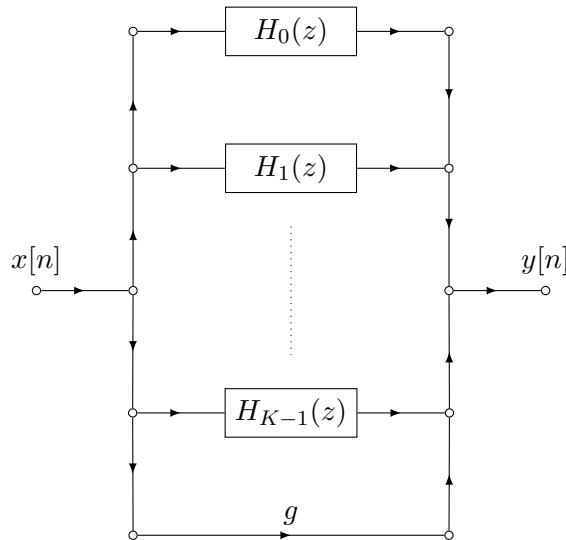
$$H(z) = g + \sum_{s=0}^{S-1} \frac{c_s}{z - z_{xs}}$$

V případech, kdy je čitatel a jmenovatel přenosové funkce soudělný, je k součtu parciálních zlomků ještě přičten nenulový reálný koeficient g . Tento koeficient udává přímý přenos vstupního signálu na výstup.

Kořeny jmenovatele přenosové funkce z_{xs} jsou opět reálné nebo komplexně sdružené páry a proto lze odpovídající páry zlomků sloučit do jednoho zlomku s reálnými koeficienty

$$H(z) = g + \sum_{k=0}^{K-1} \frac{b_1 z + b_0}{z^2 + a_1 z + a_0}.$$

Póly se často vyskytují jako nenásobné, proto je určení koeficientů c_s i následné spojování komplexně sdružených párů jednodušší. Čítec přenosové funkce dílčích sekcí je pak i o 1 nižšího řádu (pouze 1. řádu). To znamená, že při implementaci je možné ušetřit jedno násobení.



Obr. 5.6: Paralelní zapojení sekcí druhého řádu pro realizaci celkové přenosové funkce.

V případě paralelní struktury je rozklad jednoznačný na dílčí sekce jednoznačný a nelze najít různé variace rozdělení pólů a nulových bodů. Teoreticky by mohlo záležet na pořadí sčítání výstupů dílčích sekcí, kdy při nevhodném pořadí dojde k přetečení. V nejhorším případě, kdy výstupy všech sekcí budou rovny maximální hodnotě, bude výstup omezen hodnotou $K + g$. Vzhledem k malému počtu sekcí filtrů v praxi je možné předpokládat, že přetečení nepřekročí rozsah rozšiřujících bitů střadačů signálového procesoru.

Podobně jako při sériové realizaci byl vyhodnocen poměr signálu ku kvantovacímu šumu. Použitý byl stejný filtr typu dolní propust jako v případě sériové realizace. Nejprve byl jako vstupní signál použit bílý šum s Gaussovým rozdělením a jednotlivé sekce byly implementovány v první kanonické formě. Vypočtená hodnota poměru signálu ku kvantovacímu šumu

$$SQNR = 63,78 \text{ dB.}$$

je asi o 6 dB nižší než v případě nejlepší variace sériového zapojení. V případě realizace dílčích sekcí v druhé kanonické formě hodnota poměru signálu ku kvantovacímu šumu klesla na

$$SQNR = 43,90 \text{ dB}$$

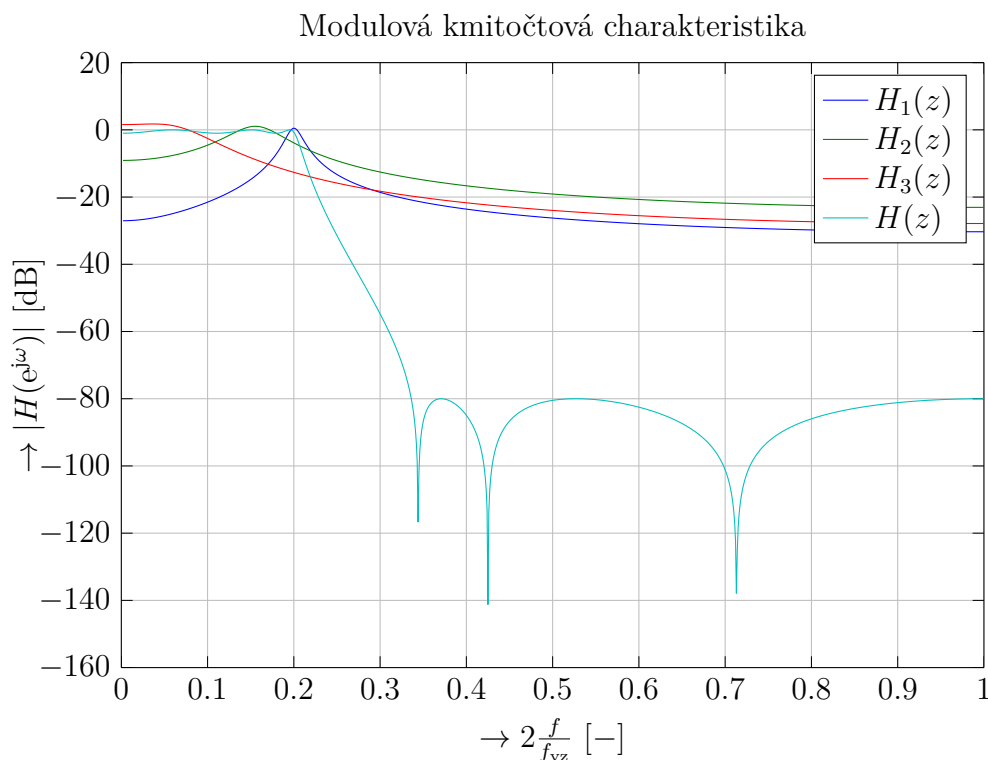
podobně jako u sériového zapojení.

Poté byl jako vstupní signál použit deterministický signál složený ze dvou harmonických signálů s kmitočty 800 a 3000 Hz. Podobně jako u sériového zapojení se projevila korelovanost kvantovacího šumu vzniklého v různých místech struktury implementace a poměr signálu ku kvantovacímu šumu dále klesl. V případě první kanonické formy na hodnotu

$$SQNR = 56,98 \text{ dB}$$

a v případě druhé kanonické formy na hodnotu

$$SQNR = 46,35 \text{ dB.}$$



Obr. 5.7: Modulové kmitočtové charakteristiky dílčích sekcí 2. řádu při paralelních zapojení.

5.3 Shrnutí vlivu omezené délky slova

Analýza vlivu omezené délky slova při rozdělení přenosové funkce na sériové spojení sekcí 2. řádu potvrdila předpoklad, že omezená délka slova a s tím související kvantovací jevy učiní obvod nelineární. Důsledkem je, že záleží na pořadí sekcí a dokonce i na sdružování dvojic nulových bodů a pólů. Pro vzorový systém 6. řádu byl analyzován poměr výstupního signálu ku kvantovacímu šumu pro všech 36 možných kombinací nulových bodů a pólů. Kromě pořadí sekcí závisí poměr signálu ku kvantovacímu šumu i na charakteru vstupního signálu. V případě implementace sekcí v první kanonické formě a vstupního

signálu v podobě náhodného procesu s normálním rozdělením se poměr pohyboval mezi 55 dB a 68 dB. Rozdíl je téměř 13 dB což lze připodobnit zkrácení délky slova o 2 bity. V závislosti na aplikaci může mít pořadí sekcí velký vliv na výsledný signál. V případě zpracování harmonického signálu hodnota poměru klesne na 53 dB až 61 dB. Tím se potvrzuje, že analýza uvažující nezávislost vzorků kvantovacího šumu, což platí pouze pokud je vstupní signál náhodný proces, je nejlepší případ a pro reálné signály mají kvantovací jevy větší vliv. Při implementaci dílčích sekcí v druhé kanonické formě byly výsledné hodnoty poměru signálu od kvantovacího šumu ještě nižší.

Při rozkladu do paralelního spojení sekcí 2. řádu je nevýhodou nutnost provést rozklad přenosové funkce do parciálních zlomků. To je matematicky složitější operace než pouhý výpočet nulových bodů a pólů. Paralelní spojení je však jednoznačně dáno tvarem parciálních zlomků a není nutné hledat nejvhodnější pořadí sekcí 2. řádu jako při sériovém spojení. Při implementaci dílčích sekcí v první kanonické formě a vstupním signálu v podobě náhodného signálu s normálním rozdělením byl poměr výstupního signálu ku kvantovacímu šumu přibližně 64 dB. Hodnota je přibližně v polovině rozsahu poměru signálu ku kvantovacímu šumu při sériovém spojení. Navíc polynom čitatele přenosové funkce sekcí 2. řádu je často pouze 1. řádu, oproti 2. řádu v případě sériového spojení. To může vést k nižší výpočetní náročnosti při implementaci v signálovém procesoru. I když se v drtivé většině aplikací používá sériové spojení dílčích sekcí 2. řádu, provedená analýza ukazuje, že paralelní spojení je srovnatelné a v některých aplikacích může být i výhodnější.

6 ZÁVĚR

V habilitační práci jsou podrobně analyzovány různé typy architektury signálových procesorů. Byly nalezeny společné vlastnosti procesorů s harvardskou architekturou a procesorů s architekturou s velmi dlouhým slovem. Pro obě architektury pak byly optimalizovány výpočetní struktury pro implementaci číslicových systémů s konečnou i nekonečnou impulsní charakteristikou.

Přitom se potvrdil předpoklad, že sériové spojení dílčích sekcí 2. řádu (tzv. třetí kanonická forma), která se nejčastěji používá pro realizaci systémů s nekonečnou impulsní charakteristikou, není pro moderní architekturu s velmi dlouhým slovem příliš vhodná. Třetí kanonická forma totiž vyžaduje sekvenční provádění stavových rovnic dílčích sekcí, neboť výstupní signál dílčí sekce je vstupním signálem následující dílčí sekce. Nelze tak provádět více sekcí paralelně, jak by to odpovídalo myšlence paralelního zpracování využívaném u architektury typu VLIW (Very Long Instruction Word). Naproti tomu realizace paralelním spojením dílčích sekcí 2. řádu (tzv. čtvrtá kanonická forma) přirozeně nabízí paralelní provádění stavových rovnic dílčích sekcí. Dílčí sekce jsou totiž nezávislé, vstupní signál je pro všechny sekce stejný, výstup je pak dán součtem výstupních signálů všech sekcí. Navíc číselná přenosová funkce dílčích sekcí je u čtvrté kanonické formy často o 1 nižšího řádu než u třetí kanonické formy. To vede k dalšímu snížení výpočetní náročnosti, jak je demonstrováno v části 4.2.2.

Na druhou stranu změna struktury algoritmu ovlivní pořadí provádění operací, což s ohledem na nelinearitu aritmetiky v pevné řádové čarce (zaokrouhlení, přetečení) může významně změnit citlivost algoritmu na kvantování. Proto byly podrobně analyzovány místa vzniku kvantovacího šumu u jednotlivých struktur a struktury porovnány. Během analýzy se potvrdil předpoklad, že 2. kanonická forma, též označovaná jako DF2 (Direct Form II), je citlivější na přetékání stavových proměnných než 1. kanonická forma (DF2T (Direct Form II Transposed)). Nižší byl i poměr výstupního signálu ku kvantovacímu šumu. V případě třetí kanonické formy se také prokázalo, že poměr výstupního signálu ku kvantovacímu šumu výrazně závisí na pořadí sekcí a vzájemném skládání nulových bodů a pólů (viz část 5.2.1). Rozdíl může být až 13 dB, což odpovídá zkrácení slova o přibližně 2 bity. Naopak u paralelní 4. kanonické formy je rozdělení na sekce pevně dáno rozkladem na parciální zlomky a je neměnné. Nepotvrdil se však předpoklad, že kvantovací šum na výstupu jednotlivých sekcí je nezávislý a po součtu výstupů všech dílčích sekcí dojde k jeho zeslabení a ke zvýšení poměru výstupního signálu ku kvantovacímu šumu. Naopak poměr výstupního signálu ku kvantovacímu šumu byl v provedeném experimentu nižší přibližně o 6 dB (viz část 5.2.2). To přibližně odpovídá zkrácení délky slova o 1 bit. Záleží tedy na konkrétní aplikaci, zda snížení výpočetní náročnosti při použití paralelní 4. kanonické formy vyváží snížení poměru výstupního signálu ku kvantovacímu šumu.

Postupy optimalizace uvedené v práci byly využity při řešení mnoha výzkumných projektů GA ČR, MPO ČR, AV ČR a smluvního výzkumu. Nejvýznamnější byla např. optimalizace zdrojových kódů řečových kodeků GSM HR, GSM FR a G.723.1 pro firmu STROM Telecom [98, 93, 31]. Optimalizace filtrů použitých v kodecích podle postupů v kapitole 4 byla prováděna pro procesor TMS320C6416 s velmi dlouhým instrukčním slovem. Výpočetní náročnost se podařilo snížit více než 15-krát, takže při hodinovém

kmitočtu 1 GHz je celý telefonní rámec E1 zpracovat jediný procesor. Dále byly závěry použity při návrhu a implementaci měřiče kvality napětí v rozvodných soustavách podle normy [46] pro firmu MEgA. V tomto případě implementace probíhala v procesoru TMS320C5510 s harvardskou architekturou a výsledná knihovna je využívána v komerčním produktu MEg37 [70]. Částečně byly závěry použity i při implementaci ovladače obvodu Si3220, který implementuje účastnické rozhraní analogové telefonní linky, pro pobočkovou ústřednu vyvíjenou společností Vestcom [73]. A dále při implementaci ovladače kamerového čipu MT9M034 pro realizaci duální kamery pro firmu Herman Elektronika.

V pedagogické oblasti byly výsledky habilitační práce použity při aktualizaci skript a přednáškových materiálů předmětů zaměřených na číslicové zpracování signálů. Porovnání signálových procesorů různých výrobců a zobecnění vlastností architektury bylo zapracováno do přednáškových materiálů předmětu *Signálové procesory a Číslicové filtry*. Dále byly využity při realizaci vícedenních odborných školení, např. pro firmy MESIT (2005), MEgA (2012), Honeywell (2015). Během řešení vznikla i odborná monografie *Signálové procesory* věnovaná analýze architektur signálových procesorů a optimalizace struktur výpočetních algoritmů [64].

Poděkování

Na tomto místě bych rád poděkoval zaměstnancům Ústavu telekomunikací, kteří mi poskytli nezbytné zázemí pro vznik této práce, i spolupracovníkům z dalších firem, kteří mi umožnili téma habilitační práce rozvíjet při řešení jejich projektů.

LITERATURA

- [1] Analog Devices: *ADSP-BF-561 Data Sheet*. 2009.
- [2] Analog Devices: *ADSP-BF-609 Data Sheet*. 2014.
- [3] Analog Devices Inc.: *ADSP-2100 Family DSP Microcomputers*. 1996.
- [4] Analog Devices Inc.: *ADSP-219x/2191 DSP Instruction Set Reference*. 2000.
- [5] Analog Devices Inc.: *ADSP-219x/2191 DSP Hardware Reference*. 2001.
- [6] Analog Devices Inc.: *ADSP-2191M DSP Microcomputers*. 2002.
- [7] Analog Devices, Inc: *ADSP-SC582/SC583/SC584/SC587/SC587/SC589/ADSP-21583/21584/21587 SHARC+ Dual Core DSP with ARM Cortex-A5*. 2016.
- [8] J. ČERMÁK: *Optimalizace metod číslicového zpracování signálu pro signálové procesory typu VLIW*. Diplomová práce, Vysoké učení technické v Brně, Brno, 2003, vedoucím diplomové práce byl Zdeněk Smékal.
- [9] ČERMÁK, J.; SMÉKAL, Z.; VRÁNA, J.: Optimalizace IIR filtru na signálovém procesoru typu VLIW Motorola MSC8101. *Elektrorevue - Internetový časopis*, ročník 2004, č. 32, 2004: s. 1–10, ISSN 1213-1539.
URL <http://www.elektrorevue.cz>
- [10] CHASSAING, R.: *DSP Applications Using C and The TMS320C6x DSK*. Wiley, 2002, ISBN 0-471-20754-3.
- [11] DAVÍDEK, V.: *Implementace algoritmů číslicového zpracování signálů v reálném čase*. Praha: ČVUT v Praze, první vydání, 2004, ISBN 80-01-03114-4.
- [12] DVOŘÁK, V.; DRÁBEK, V.: *Architektura procesorů*. Brno: VUTIUM, první vydání, 1999, ISBN 80-214-1458-8.
- [13] EL-SHARKAWY, M.: *Real Time Digital Signal Processing Applications with Motorola's DSP56000 Family*. New Jersey: Prentice Hall, 1990.
- [14] EMBREE, P.; KIMBLE, B. C.: *C Language Algorithms for Digital Signal Processing*. New Jersey: Prentice Hall, 1991, ISBN 0-13-133406-9.
- [15] EYRE, J.: The Digital Signal Processor Derby. *IEEE SPECTRUM*, June 2001: s. 62–68.
- [16] FLYNN, M. J.: Very High-Speed Computing Systems. In *Proceedings of the IEEE*, ročník 54, December 1966, s. 1901–1909.
- [17] FREESCALE: The StarCorre SC140 DSP Core. Technická zpráva, Freescale, 1999.
- [18] Freescale Inc.: *SC140 DSP Core Reference Manual*. 2001, mNSC140CORE/D.
- [19] Freescale Inc.: *MSC8101 Reference Manual*. 2002, mSC8101RM/D.

- [20] Freescale Inc.: *MSC8151 Data Sheet*. 2013, mSC8151.
- [21] Freescale Inc.: *MSC8251 Data Sheet*. 2013.
- [22] GRIFFITHS, L. J.: An Adaptive Lattice Structure for Noise-Cancelling Applications. In *Proceedings of ICASSP*, Tulsa, 1978, s. 87–90.
- [23] IEEE, New Jersey: *IEEE Standard for Binary Floating-Point Arithmetic*. 1985, aN-SI/IEEE Std. 754-1985.
- [24] JOHN R. DELLER, J.; HANSEN, J. H. L.; PROAKIS, J. G.: *Discrete-Time Processing of Speech Signals*. New York: John Wiley, reprint vydání, 2000, ISBN 0-7803-5386-2, 936 s.
- [25] KRAJSA, O.; SYSEL, P.: Optimization of FIR filter implementation for FMT on VLIW DSP. In *Proceedings of the 4th International Conference on Circuits, Systems and Signals (CSS'10)*, Corfu: WSEAS Press, 2010, ISBN 978-960-474-208-0, s. 169–173.
- [26] KRAJSA, O.; ŠILHAVÝ, P.; SYSEL, P.: Filterbank modulation techniques and its utilization in ADSL systems. In *Proceedings of 34th International Conference on Telecommunications and Signal Processing*, 2011, ISBN 978-1-4577-1409-2, s. 1–5.
- [27] KUO, S. M.; GAN, W.-S.: *Digital Signal Processors – Architectures, Implementations, and Applications*. Prentice Hall, 2005, ISBN 0-13-035214-4.
- [28] KUO, S. M.; LEE, B. H.; TIAN, W.: *Real-Time Digital Signal Processing - Implementations and Applications*. Wiley, 2006, ISBN 978-0-470-01495-0.
- [29] LAPSLEY, P.; BIER, J.; SHOHAM, A.; AJ.: *DSP Processor Fundamentals - Architectures and Features*. New York: Wiley IEEE Press, 1996, ISBN 0-7803-3405-1.
- [30] LIU, D.: *Embedded DSP Processor Design – Application Specific Instruction Set Processors*. Burlington: Elsevier, první vydání, June 2008, ISBN 978-0-12-374123-3, 808 s.
- [31] MALENOVSKÝ, V.; SYSEL, P.: Konverze programu pro kodek GSM Half Rate do objektové podoby a rozšíření funkčnosti pro více kanálů. Výzkumná zpráva, STROM telecom, Prosinec 2002.
- [32] MITRA, S. K.: *Digital Signal Processing - A Computer-Base Approach*. New York: McGraw-Hill, 2001, ISBN 0-07-118175-X.
- [33] Motorola Inc.: *DSP56F801/803/805/807 User's Manual*. DSP56F801-7UM/D.
- [34] Motorola Inc.: *DSP56305 Users's Manual*. 1996.
- [35] Motorola Inc.: *DSP56800 Family Manual*. 1996, dSP56800FM/AD.
- [36] Motorola Inc.: *DSP56300 Family Manual*. 1997, dSP56300FM/AD.

- [37] Motorola Inc.: *DSP56307 User's Manual*. 1998, dSP56307UM/D.
- [38] Motorola Inc.: *DSP56824 User's Manual*. 1999, dSP56824UM/D.
- [39] Motorola Inc.: *DSP56303 User's Manual*. První vydání, 2001.
- [40] Motorola Inc.: *DSP56800E 16-bit Digital Signal Processor Core*. 2001, dSP56800ERM/D.
- [41] Motorola Inc.: *Introduction to the StarCore SC140 Tools: An Approach in Nine Exercises*. 9 2001, aN2009/D.
- [42] Motorola Inc.: *56F801 16-bit Hybrid Controller*. 2003, dSP56F801/D.
- [43] Motorola Inc.: *DSP56L307 Technical Data*. 2003, dSP56L307/D.
- [44] Motorola Inc., Agere Systems Inc.: *SC100 C/C++ Compiler User's Manual*. 11 2001, mSC100CCUM/D.
- [45] VON NEUMANN, J.: First Draft of a Report on the EDVAC. Technická zpráva, University of Pennsylvania, Philadelphia, June 1945.
- [46] ČSN: Měřič blikání – Specifikace funkce a dimenzování. Technická zpráva, 2003, ČSN EN61000-4-15, 2003. Elektromagnetická kompatibilita (EMC), část 4 Zkušební a měřicí technika, oddíl 15.
- [47] PROAKIS, J. G.; MANOLAKIS, D. G.: *Digital Signal Processing Principles, Algorithms, and Applications*. New Jersey: Prentice Hall, třetí vydání, 1996, ISBN 0-13-373762-4, 1016 s.
- [48] SESHAN, N.: High Velocity Processing. *IEEE Signal Processing Magazine*, ročník 15, č. 2, March 1998: s. 86–101.
- [49] ŠILHAVÝ, P.; KRAJSA, O.; SYSEL, P.; AJ.: Overlapped filtered multitone modulation and its optimization on VLIW DSP. *Przegląd Elektrotechniczny*, ročník 86, č. 12, 2011: s. 91–95, ISSN 0033-2097.
- [50] SMITH, S. W.: *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, druhé vydání, 1998.
URL DSPguide.com
- [51] SMÉKAL, Z.: Optimum Digital Filter Structure Design Based on Response to Initial Conditions. *Internet Journal Electronicsletters*, ročník 2, č. 12, 2000, ISSN 1213-161X.
- [52] SMÉKAL, Z.: Aktuální trendy architektury signálových procesorů I. *Sdělovací Technika*, , č. 4, 2002: s. 3–6, ISSN 0036-9942.
- [53] SMÉKAL, Z.: Aktuální trendy architektury signálových procesorů II. *Sdělovací Technika*, , č. 5, 2002: s. 16–18, ISSN 0036-9942.

- [54] SMÉKAL, Z.; ET.AL.: Implementace algoritmu separace řeči na digitální signálový procesor TMS320C6711. Výzkumná zpráva k projektu reg. č. fd-k/125, MPO ČR, Zář 2003.
- [55] SMÉKAL, Z.; SYSEL, P.: Influence of Signal Processor Architecture on Generating Optimum Algorithm of Digital Signal Processing Methods. In *Advances in Systems Science: Measurement, Circuits and Control*, Electrical and Computer Engineering Series, Piraeus, GREECE: WSES Press, první vydání, July 2001, ISBN 960-8052-39-4, s. 445–448.
- [56] SMÉKAL, Z.; SYSEL, P.: Implementace algoritmů na signálových procesorech typu VLIW. *Elektrorevue*, ročník 2002, č. 41, Říjen 2002: s. 1–8, ISSN 1213-1539. URL <http://www.elektrorevue.cz/clanky/02041/>
- [57] SMÉKAL, Z.; SYSEL, P.: Souvislost mezi různými popisy diskrétních systémů pomocí diferenčních rovnic. In *Moderní směry výuky elektrotechniky a elektroniky*, Brno: VUT v Brně, October 2002, ISBN 80-214-2190-8, s. 62–65.
- [58] SMÉKAL, Z.; SYSEL, P.: Architecture-Dependent Algorithm Optimization for VLIW Digital Signal Processor. In *Proceedings of The 48th International Scientific Colloquium IWK'2003*, Ilmenau, July 2003, ISBN 1619-4098, s. 143–144.
- [59] SMÉKAL, Z.; SYSEL, P.: Signálové procesory VLIW firmy Texas Instruments I. *Sdělovací technika*, ročník 2003, č. 1, Leden 2003: s. 18–20, ISSN 0036-9942.
- [60] SMÉKAL, Z.; SYSEL, P.: Signálové procesory VLIW firmy Texas Instruments II. *Sdělovací technika*, ročník 2003, č. 2, Únor 2003: s. 15–17, ISSN 0036-9942.
- [61] SMÉKAL, Z.; SYSEL, P.: Signálové procesory VLIW firmy Texas Instruments III. *Sdělovací technika*, ročník 2003, č. 3, Březen 2003: s. 16–17, ISSN 0036-9942.
- [62] SMÉKAL, Z.; SYSEL, P.: Optimalizace algoritmů v signálovém procesoru typu VLIW při využití intrinsic funkcí. In *Sborník X. mezinárodní konference Moderní elektronické součástky 2004*, Brno: Sdělovací technika, November 2004, s. 1–28.
- [63] SMÉKAL, Z.; SYSEL, P.: Digital Signal Processing Algorithm Optimization for VLIW Digital Signal Processors. In *Proceedings of the Second International Symposium on Communications, Control and Signal Processing*, Marrakech, Morocco: SuviSoft, Oy Ltd., Finland, March 2006, ISBN 2-908849-17-8, s. 1–4.
- [64] SMÉKAL, Z.; SYSEL, P.: *Signálové procesory*. Praha: Sdělovací technika, první vydání, Leden 2006, ISBN 80-86645-08-8, 283 s.
- [65] SMÉKAL, Z.; SYSEL, P.: Výkonné jádro pro embedded systémy. *Sdělovací technika*, ročník 2006, č. 10,11, Říjen 2006: s. 8–13, ISSN 0036-9942.
- [66] SMÉKAL, Z.; SYSEL, P.; ČERMÁK, J.: Optimization of Parallel Processing in VLIW Digital Signal Processors. In *Proceedings of the 10th International Workshop on Systems, Signals and Image Processing*, Praha, October 2003, ISBN 80-86645-05-3, s. 280–283.

- [67] SMÉKAL, Z.; VONDRA, M.; VÍCH, R.: State-Space Representation of Cepstral Vocal Tract Model for DSP Implementation. *Internet Journal Electronicsletters*, ročník 3, č. 9, 2002, ISSN 1213-161X.
URL <http://www.electronics-letters.com>
- [68] SMÉKAL, Z.; VÍCH, R.: *Zpracování signálů pomocí signálových procesorů*. Praha: Radix, spol. s r. o., 1998, ISBN 80-86031-18-7.
- [69] SYSEL, P.: Optimization of FIR Filter on VLIW Digital Signal Processors. In *Proceedings of Research in Telecommunication Technology 2004*, Prague: CTU in Prague, Faculty of Electrical Engineering, October 2004, ISBN 80-01-03063-6, s. 1–5.
- [70] SYSEL, P.: Implementace filtrační části měřiče blikání. Technická zpráva, MEgA a.s., 2010.
- [71] SYSEL, P.: *Signálové procesory - laboratorní cvičení*. Brno: Vysoké učení technické v Brně, 2014, ISBN 978-80-214-5204-6, skriptum.
- [72] SYSEL, P.: *Signálové procesory*. Brno: Vysoké učení technické v Brně, 2015, ISBN 978-80-214-5187, skriptum.
- [73] SYSEL, P.; HANÁK, P.: Embedded Asterisk PBX with Analogue Telephone Support. In *Proceedings of the 31st International Conference on Telecommunications and Signal Processing (TSP'08)*, Parádfürdö, Hungary: Budapest University of Technology and Economics, 2008, ISBN 978-963-06-5487-6, s. 132–134.
- [74] SYSEL, P.; ROZSÍVAL, L.: Vzdálené řízení a kontrola průmyslových zařízení GSM modemy. *Elektrorevue*, ročník 2002, č. 43, Listopad 2002: s. 1–10, ISSN 1213-1539.
URL <http://www.elektrorevue.cz/clanky/02046/>
- [75] SYSEL, P.; SMÉKAL, Z.: Architecture-Dependent Optimization of TI VLIW Digital Signal Processors. In *Proceedings of the 25th International Conference on Telecommunications and Signal Processing (TSP 2002)*, ročník 1, Brno: Brno University of Technology, October 2002, ISBN 80-214-2172-X, s. 168–171.
- [76] SYSEL, P.; SMÉKAL, Z.: *Číslicové filtry*. Brno: Vysoké učení technické v Brně, 2012, ISBN 98-80-214-4454-6, skripta.
- [77] SYSEL, P.; VRÁNA, J.: Signálové procesory typu VLIW. In *Proceedings of Audio Technologies and Processing ATP2003*, Brno: Nakladatelství VUT, Květen 2003, ISBN 80-214-2391-9, s. 76–82.
- [78] Texas Instruments: *TMS320C6000 DSP/BIOS 5.x Application Programming Interface (API): Reference Guide*. August 2012.
URL <http://www.ti.com/lit/ug/spru403s/spru403s.pdf>
- [79] Texas Instruments Inc.: *TMS320C54x DSP Reference Set. 3* 2001, spru131g.

- [80] Texas Instruments Inc.: *TMS320C55x DSP CPU Programmer's Reference*. 2002, spru652g.
- [81] Texas Instruments Inc., Dallas: *TMS320C6000 Assembly Language Tools User's Guide*. 2002, spru186k.
- [82] Texas Instruments Inc., Dallas: *TMS320C6000 Programmer's Guide*. 2002, spru198g.
- [83] Texas Instruments Inc.: *TMS320C55x DSP CPU Reference Guide*. 2004, spru371f.
- [84] Texas Instruments Inc.: *TMS320C6201 Digital Signal Processor*. March 2004, sprs051h.
- [85] Texas Instruments Inc.: *TMS320C6701 Floating-Point Digital Signal Processor*. 2004, tms320c6701.pdf.
- [86] Texas Instruments Inc.: *TMS320VC5510/5510A Fixed-Point Digital Signal Processors*. 2007, sprs076o.
- [87] Texas Instruments Inc.: *TMS320C6727, TMS320C6726B, TMS320C6722B, TMS320C6720 Floating-Point Digital Signal Processors*. 2008, tms320c6720.pdf.
- [88] Texas Instruments Inc.: *TMS320C6412 Fixed-Point Digital Signal Processors*. 2010, sprs219j.
- [89] Texas Instruments, Inc.: *TMS320C6000 Optimizing Compiler v 7.3: User's Guide*. July 2011.
 URL http://processors.wiki.ti.com/images/d/de/TMS320C6000_Optmizing_Compiler_v7.3_User's_Guide.pdf?keyMatch=spru187t&tisearch=Search-EN-Everything
- [90] Texas Instruments Inc.: *TMS320C6414, TMS320C6415, TMS320C6416 Fixed-Point Digital Signal Processors*. 2015, tms320c6416.pdf.
- [91] Texas Instruments Inc.: *TMS320C6000 CPU and Instruction Set Reference Guide*. October 2000, spru189f.
- [92] VONDRA, M.; VÍCH, R.: Comb Filtering in Speech Enhancement. In *Proceedings of 16th Czech-German Workshop SPEECH PROCESSING*, Praha: IREE AS CR, 2006, ISBN 80-86269-15-9, s. 73–77.
- [93] VRBA, K.; SMÉKAL, Z.; SYSEL, P.: Optimalizace ANSI-C kódu GSM Half Rate kodeku pro signálový procesor TMS320C64xx. Výzkumná zpráva, STROM telecom, Srpen 2002.
- [94] VÍCH, R.: *Transformace Z a některá její použití*. Praha: SNTL, druhé vydání, 1983.
- [95] VÍCH, R.: *Z Transform Theory and Application*. Praha: SNTL, druhé vydání, 1987, ISBN 90-277-1917-9.

- [96] VÍCH, R.: Verallgemeinerte und kanonische Synthese cepstraler Sprachmodelle. In *Proceedings of the 43rd International Scientific Colloquium*, Ilmenau, Germany, september 1998, ISSN 0943-7207, s. 428–432.
- [97] VÍCH, R.; SMÉKAL, Z.: *Číslicové filtry*. Praha: Academia, 2000, ISBN 80-200-0761-X.
- [98] ZDENĚK, S.; VRBA, K.; SYSEL, P.: Optimalizace ANSI-C kódu realizujícího GSM kodér pro signálový procesor TMS320C6000. Výzkumná zpráva, STROM telecom, Listopad 2001.
- [99] ZDENĚK, S.; VRBA, K.; SYSEL, P.: Software GSM kodeku pro telefonní ústředny. Výzkumná zpráva, STROM telecom, Listopad 2001.