Hardware pro číslicové zpracování signálů

Příspěvek volně navazuje na článek Návrh aritmetických operátorů na FPGA (Automatizace č. 2/2009, str. 71–74) zabývající se implementací aritmetických obvodů na programovatelných hradlových polích. Tématem nyní budou architektury systémů určených pro číslicové zpracování signálů. Text prezentuje výhody a nevýhody široké škály potenciálních řešení a postupně představuje jednotlivé významné architektonické prvky urychlující běh algoritmu pro číslicové zpracování signálů. Pro čtenářovu lepší představu přínosu jednotlivých obvodových a systémových řešení používáme pro demonstraci výpočet jednostranné konvoluce. S ohledem na složitost celé problematiky jsme přistoupili k několika zjednodušením.

POUŽITÉ ZKRATKY

FIR	*Finite Impulse Response*
LSB	*Least Significant Bit*
LUT	*Look Up Table*
MAC	*Multiply and ACcumulate*
MSB	*Most Significant Bit*
MMX	*MultiMedia eXtension*
RTL	*Register Transfer Level*
SIMD	*Single Instruction stream, Multiple Data stream*
SSE	*Streaming Simd Extension*
SOC	*System On Chip*

1 Úvod

Nejstarší pokusy s číslicovou filtrací datujeme na přelom 60. a 70. let (Bell laboratories, Leland B. Jackson). V uplynulých čtyřech dekádách došlo v oblasti mikroelektroniky ke značnému rozvoji a dnes je k dispozici široká škála řešení pro číslicovou filtraci od jednoduchých mikrokontrolérů zvládajících zpracovat signál se vzorkovací frekvencí v řádu jednotek kHz po výkonné SoC architektury pro softwarové rádio, jež dokáží pracovat se signály vzorkovanými s frekvencí až stovek MHz. Dosažení tak vysokých vzorkovacích frekvencí ovšem není myslitelné bez specializovaných hardwarových prostředků, a právě těmi se bude následující text zabývat.

Vzhledem ke složitosti popisované problematiky jsme se v textu dopustili několika zjednodušení bez újmy na obecnosti. Pro demonstraci vlivu architektury na rychlost výpočtu je použit hypotetický procesor, který v každém hodinovém cyklu provádí jednu či více aritmetických operací. Zanedbáváme zde tedy (pokud není řečeno jinak) režii spojenou s aktualizací ukazatelů při čtení a zápisu vzorku z paměti a do ní, režii spojenou s načítáním instrukcí a další. Dále v textu jen volně operujeme s pojmem časové náročnosti výpočtu. Příspěvek je napsán pro čtenáře s minimální znalostí teorie a pojmů z oblasti algoritmické složitosti.

2 Vlastnosti výpočtů DSP

2.1 Formáty zpracovávaných čísel

Při návrhu a používání hardwaru pro číslicové zpracování signálů často předpokládáme, že zobrazovaná n-bitová čísla představují racionální čísla z rozsahu <–1; 1) s krokem $2–(m–1)$. Pak např. 16 bitové číslo umožňuje vyjádřit 216 hodnot ve dvojkovém doplňku z rozsahu <–1; 1) s krokem 2–15. Takový formát budeme v textu označovat notací 1.15 – jeden bit celočíselné části, 15 bitů zlomkové části, řádová čárka je umístěna hned po prvním bitu zleva (MSB). To je v kontrastu proti zvyklostem běžným při programování, kdy n-bitové číslo v pevné řádové čárce interpretujeme typicky jako celé číslo z rozsahu $–2n–1 \ldots (2n–1–1)$.

2.2 Konvoluce

Jednou z nejčastěji používaných operací při číslicovém zpracování signálů je konvoluce a hardware pro číslicové zpracování signálů je obvykle navržen tak, aby výpočet konvoluce byl maximálně urychlen. Navíc konvoluce je reprezentant širší třídy problémů, protože představuje obecné schéma kombinace dvou datových toků. Algoritmy tohoto typu jsou používány například při hledání vzorů, výpočtu korelace, interpolaci či vyhodnocování hodnot polynomů [1]. Nejjednodušší aplikace konvoluce je číslicová filtrace pomocí filtru s konečnou impulzovou odezvou (filtr FIR), např. [2]. Jednostrannou a konečnou konvoluci vektorů $\boldsymbol{x}$ a $\boldsymbol{h}$ definujeme jako

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

kde N je počet prvků vektoru $\boldsymbol{h}$. Pro výpočet jednoho prvku vektoru $\boldsymbol{y}$ tedy zřejmě potřebujeme provést $2N$ čtení prvků vektoru $\boldsymbol{x}$ a $\boldsymbol{h}$ z paměti, N násobení, N–1 součtů a jeden zpětný zápis výsledku $y[n]$. Dohromady tedy přibližně $4N$ operací.

Počet operací nutných na výpočet nového vzorku v případě filtrace přímo ovlivňuje maximální dosažitelnou vzorkovací frekvenci, se kterou systém může zpracovávat signál. Například pro řád filtru $N = 128$ budeme potřebovat vykonat přibližně 512 operací na zpracování jednoho vzorku. Při výpočtu konvoluce na hypotetickém konvenčním procesoru, který provede jednu operaci za jeden hodinový cyklus s pracovní frekvencí f_{clkb}= 100 MHz, dokážeme dosáhnout vzorkovací frekvence maximálně $f_{smax} = f_{clk}/(4N) =$ 195,3 kHz. Vidíme, že je zde velký nepoměr mezi pracovní frekvencí procesoru a maximální vzorkovací frekvencí, kterou chceme dosáhnout.

3 Možnosti hardwarové realizace výpočtů DSP

3.1 Paměťová architektura

Prvním krokem při výpočtu výrazu (1) je vždy načtení prvků vektorů $\boldsymbol{x}$ a $\boldsymbol{h}$. Nejjednodušší přístup je načítat z paměti vzorky $\boldsymbol{x}$ a $\boldsymbol{h}$ sekvenčně, jeden po druhém. Tímto způsobem je nezbytné konvoluci implementovat, máme-li k dispozici procesor s jednou sadou sběrnic (jedna adresní, datová a řídicí sběrnice mezi pamětí a procesorem) a jednou pamětí společnou jak pro program, tak pro data. O takovém procesoru a počítači pak říkáme, že má von Neumannovu architekturu. Procesor tedy může v jednom hodinovém cyklu načíst jen jedno slovo z paměti a načtení N prvků vektorů $\boldsymbol{x}$ a $\boldsymbol{h}$ bude trvat $2N$ hodinových cyklů. Celkem tedy potřebujeme přibližně $4N$ operací.

Poněkud šikovnějším řešením je načítat hodnoty $x[n – m]$ a $h[m]$ současně. Pak ale potřebujeme nejméně dvě sady sběrnic a fyzicky oddělených pamětí. Jedna z pamětí je potom určena pro data a druhá pro program, případně pro uložení konstant (zde prvky vektoru impulsové odezvy $\boldsymbol{h}$). Tak získáme počítač s tzv. modifikovanou harvardskou architekturou. Zde poznamenejme, že z terminologického hlediska počítač s ortodoxní harvardskou architekturou neumožňuje vůbec číst data z paměti programu, proto hovoříme o modifikované harvardské architektuře, kde to možné je. Pak lze v jednom cyklu načíst současně dvě slova ($x[n – m]$ z datové a $h[m]$ zN prvků vektoru $\boldsymbol{x}$ a $\boldsymbol{h}$ lze provést za N hodinových cyklů. Pro aplikace v oblasti číslicového zpracování signálů je tedy výhodnější modifikovaná harvardská architektura. Její prosté použití umožní snížit počet operací potřebných na výpočet konvoluce na zhruba $3N$. Skutečně, procesory DSP mají běžně harvardskou architekturu.

Při našem rozboru jsme nicméně předpokládali, že režie spojená s vyčítáním vzorků vektorů z paměti je nulová. Ve skutečné aplikaci musíme řešit například aktualizaci ukazatelů do paměti. Proto mají procesory určené pro číslicové zpracování signálů typicky

specializovanou podporu pro adresování paměti – adresní generátory. Ty zajišťují posun ukazatelů do paměti na další prvek načítaných vektorů, příp. poskytují automatickou podporu pro implementaci fronty vzorků paralelně s výpočtem.

3.2 Násobení a sčítání

V textu jsme automaticky předpokládali, že násobení vzorků trvá jeden hodinový cyklus. To zřejmě vyžaduje použití paralelní násobičky. Sériová násobička může být použita tehdy, když vyžadujeme jen malý výpočetní výkon. Pak lze jejím užitím ušetřit značné množství místa na obvodu FPGA (nemá-li integrované násobičky).

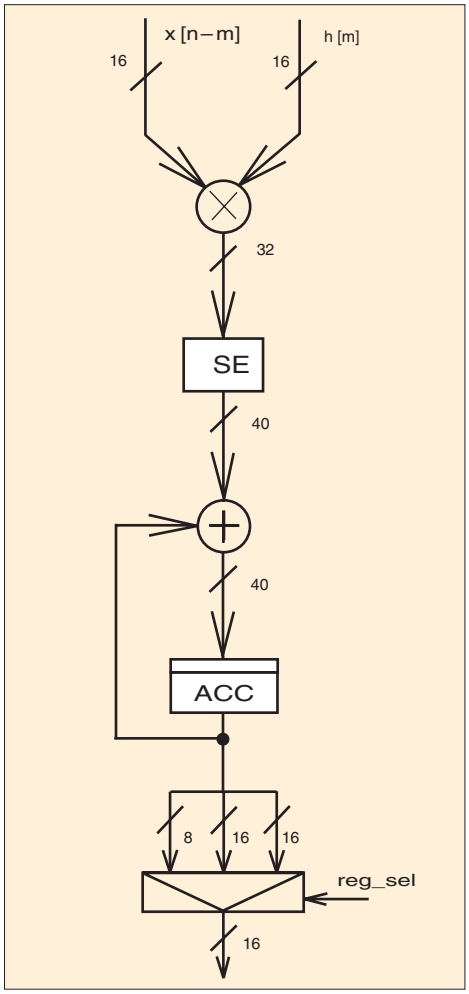
Dále si všimněme, že ve vlastním výpočtu konvoluce (1) hned po sobě figuruje operace násobení a sčítání. Oddělená realizace násobení a sčítání (akumulace v sumě) je při výpočtu konvoluce nevýhodná z důvodů časové náročnosti. Programová implementace výpočtu prvku sumy (1) a akumulace výsledku by při samostatném výpočtu obou operátorů mohla vypadat například takto:

```
NAČTI X[N-M], H[M],
W = X[N-M]×H[M],
ACC[15:00]=W[15:00]+ACC[15:00], ULOŽ
PŘENOS DO CY,
ACC[31:16]=W[31:16]+ACC[31:16]+CY, ULOŽ
PŘENOS DO CY,
ACC[39:33]=ACC[39:33]+CY,
```

kde ACC je registr určený pro akumulaci výsledku. Ve skutečnosti je tedy třeba provést 5, a ne 3 operace na zpracování jednoho prvku součtu. Příčina nárůstu počtu operací tkví v tom, že procesor je schopen nativně pracovat jen s 16 bitů širokým slovem. 32 bitů široký výsledek násobení (formát 2.30) se tak musí k 40 bitů širokému akumulátoru (formát 9.31, horních 8 dodatečných bitů slouží jako ochrana proti přetečení) přičítat třemi instrukcemi po dílčích slovech o šířce 16 bitů. Přitom zkrácení mezivýsledku násobení jen na 16 vyšších bitů w[31:16] obvykle není z důvodů velké ztráty přesnosti výpočtu přípustné.

Uvědomme si zde, že součin dvou čísel ve formátu 1.15 je skutečně číslo ve formátu 2.30. Mají-li nejméně významné řády činitelů váhu 2–15, má nejméně významný řád součinu váhu 2–30. Dva nejvíce významné bity (první 2 bity zleva) mají pak obvyklé váhy v doplňkovém kódu (–2 pro bit 31, +1 pro bit 30) a mohou nabývat hodnot 00 pro kladný výsledek, 11 pro záporný výsledek a 01 ve speciálním případě násobíme-li –1.

Vhodně navržená výpočetní struktura může všechny tyto aritmetické operace sloučit dohromady a provést v jednom cyklu. Použitím obvodu na *obr. 1* získáme možnost provést všechna tři sčítání s původním obsahem akumulátoru spolu s násobením v jednom hodinovém cyklu a tak dále urychlit výpočet na 2*N* cyklů. Výpočet jednoho



Obr. 1 Jednotka MAC. Pomocí signálu reg_sel může řadič procesoru zvolit, kterou část akumulátoru přečte

prvku součtu pak bude proveden následovně:

```
NAČTI X[N-M], H[M],
ACC = ACC + X[N-M]×H[M]
```

Použitím proudového zpracování dále můžeme zkrátit výpočet na jeden jediný cyklus na zpracování jednoho vzorku, ve kterém se provedou následující operace:

```
NAČTI X[N-M], H[M], ACC = ACC +
+ X[N-(M-1)]×H[M-1]
```

Načítáme tedy příští prvky vektorů ***x*** a ***h*** a současně akumulujeme již načtené hodnoty.

Pro svoje výhody je jednotka MAC velmi často používanou strukturou nejen při návrhu procesorů, ale i obvodů pro číslicové zpracování signálů na programovatelných hradlových polích. Již víme, že obvody FPGA obsahují integrované kanály pro urychlení přenosu a násobičky. Novější FPGA mají v programovatelné matici zahrnutou i podporu pro implementaci jednotky MAC [3]. Například obvody Xilinx Virtex IV a novější obsahují tzv. bloky DSP48 s těmito funkcemi:

- integrovanou paralelní násobičkou s šířkou operandů 18 × 18 bitů se znaménkem, součin je o šířce 36 bitů, resp. 17 × 17 bitů bez znaménka a třívstupovou konfigurovatelnou sčítačkou širokou 48 bitů pro akumulaci mezivýsledků;
- zabudovanou podporu pro proudové zpracování a dynamické řízení funkce bloku (za běhu lze ovládat datovou cestu a měnit výpočet);
- s podporou pro zaokrouhlování.

Vybavením hradlového pole specializovaným blokem pro implementaci jednotky MAC se významně snižuje zpoždění na spojích mezi násobičkou a akumulátorem i zpoždění uvnitř obou operátorů, a tak je možné dosáhnout vyšší pracovní frekvence hodin navrhovaného obvodu. Zpoždění na spojích je totiž v použité technologii výroby (90 nm) dominantní a představuje významnou část zpoždění v kombinačních logických cestách v obvodu. Další výhodou užití dedikovaného bloku je i nižší spotřeba elektrické energie a tím menší disipované teplo výslednou strukturou. Použití bloků DSP48 nicméně vyžaduje vhodně napsaný kód VHDL, aby nástroje pro implementaci (syntéza, rozmístění a propojení) poznaly, že ho mají užít.

3.3 Kvantizace a saturace

Reálná implementace na *obr. 1* si vynucuje po dokončení výpočtu (1) provést další dodatečné úpravy výsledku. Jejich cílem je redukce šířky slova ze 40 (formát 9.31) na 16 (formát 1.15) bitů, se kterou nativně pracuje náš hypotetický procesor. To typicky provádíme v následujících krocích:

3.3.1. Úprava měřítka

Při implementaci číslicových filtrů bývá běžné, že posuneme měřítko impulsové odezvy h[*n*] násobením/dělením vhodnou mocninou 2, abychom omezili vliv kvantování koeficientů na výstup číslicového filtru. Stejným faktorem je potom nezbytné vydělit výslednou akumulovanou hodnotu, aby byl zachován původní výkon signálu. Do vztahu (1) proto vkládáme operaci M_s posunutí výsledku o *s* bitů vlevo ($s > 0$) či vpravo ($s < 0$).

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

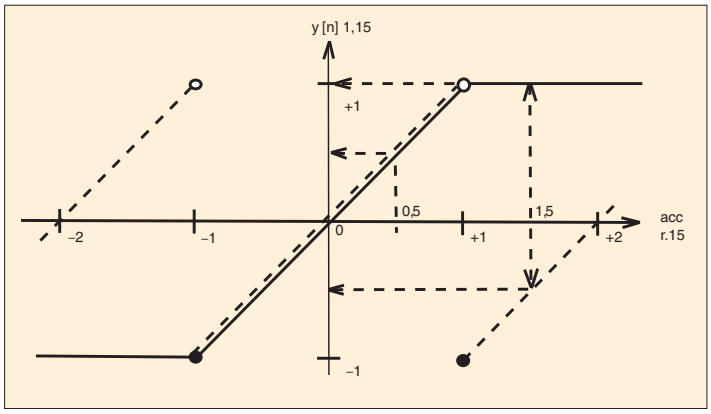
$$u = M_s[v] = v \times 2^s \qquad (2)$$

Posunutí lze implementovat pomocí „barrel shifteru“ [4]. Hodnota y[*n*] bude po provedené operaci úpravy měřítka ve formátu r.40-r.

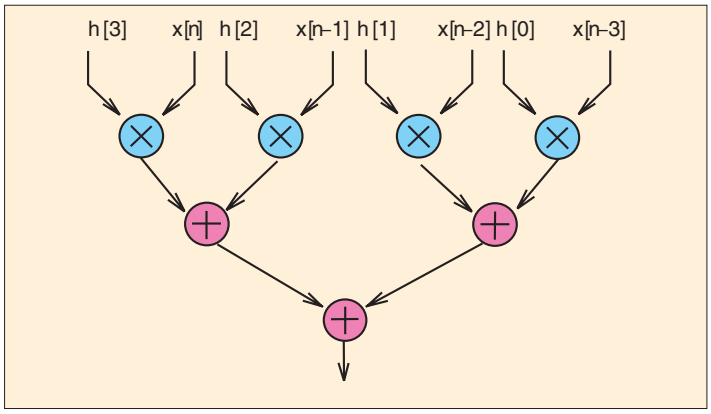
3.3.2. Redukce počtu desetinných míst

V dalším kroku je třeba převést výsledek z formátu r.40-r do romátu r.15 zkrácením zlomkové části akumulované hodnoty na 15 bitů. To si lze představit jako vložení operátoru kvantizace na *w* (zde $w = 15$) bitů Q_w do vztahu (2), získáme tak upravenou podobu konvoluce

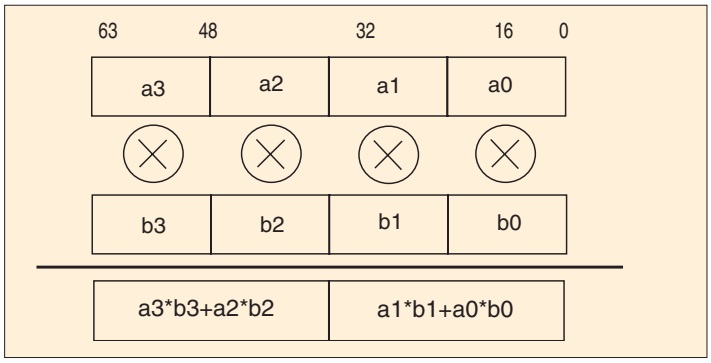
$$y[n] = Q_w\left[M_r\left[\sum_{m=0}^{N-1} x[n-m]h[m]\right]\right] \qquad (3)$$



Obr. 2 Operace saturace. V obrázku jsou zanesené dva ukázkové případy práce systému (pro hodnotu akumulátoru 0.5 a 1.5)



Obr. 4 Schéma paralelního zpracování při výpočtu FIR filtru



Obr. 3 Schematické znázornění SIMD operace MAC – paralelní zpracování čtyř šestnáctibitových slov současně

Operaci kvantizace můžeme modelovat jako

$$u = \mathrm{Q_w}[v] = \frac{\mathrm{f}(v \times 2^{w-1})}{2^{w-1}} \qquad (4)$$

kde číslo v je ve dvojkovém doplňku, proto násobíme $2w$–1 a ne $2w$. Funkce f(x) zde převádí číslo s nenulovou zlomkovou částí na celé číslo, může být realizována více způsoby:

- jako operátor oříznutí (*truncation*). Z hardwarového hlediska se jedná o prosté odříznutí nejnižších bitů výsledku. Ve dvojkovém doplňku má tato operace nicméně mírně odlišné vlastnosti než bychom čekali:

  Q4[01.101112]=
  Q4[+1.71875d]=01.10112=+1.6875d,
  +1.6875d<+1.71875d,
  Q4[10.010012]= Q4[-
  1.71875d]=10.01002=-1.7500d, -
  1.7500d<-1.71875d,

  Vidíme, že oříznutá hodnota je vždy menší než původní (ořezávání se provádí k „mínus nekonečnu“), proto je f pro dvojkový doplněk definována jako

$$\mathrm{f}[x] = \begin{cases} \lfloor x \rfloor, x \geq 0 \\ \lceil x \rceil, x < 0 \end{cases} \qquad (5)$$

- zaokrouhlení (*rounding*) je technicky náročnější. Funkce f(v) zde představuje operátor zaokrouhlení čísla v na jednotky tak, jak jsme zvyklí ze školy.
- jsou možné a používané i další varianty – např. zaokrouhlení s preferencí sudé číslice (*convergent rounding*), příp. náhodné zaokrouhlování, např. [5].

3.3.3. Saturace

Po obou provedených úpravách by nyní hodnota y[n] ve vzorci (3) vypočtená z čísla uloženého v akumulátoru měla představovat desetinné číslo z rozsahu <–1; 1). Výsledek je nicméně ve formátu r.15, může tedy za jistých nevhodných podmínek (například v případě silného signálu na vstupu obvodu) nabývat hodnoty z rozsahu <–2r; 2r). Abychom získali hodnotu ve formátu 1.15 podporovaném procesorem, můžeme nyní teoreticky jen odseknout (ignorovat) nejvyšších r – 1 bitů čísla; tím ovšem můžeme způsobit nezanedbatelnou chybu výsledku (*obr. 2* – čárkovaná křivka). Zásadní chyba tohoto triviálního řešení je ve ztrátě monotonicity operace – velká hodnota (například 1,5) se zde může jednoduše převést na celkem libovolné číslo z rozsahu <–1; 1) (např. –0,5) a nemusí ani dojít k zachování znaménka. Lepším řešením je provedení saturace výsledku, její převodní charakteristika je na *obr. 2* namalována plnou čarou. Převodní charakteristika je zde monotónní a nedochází k nepřípustnému, zdánlivě náhodnému, zkreslení. Saturace v principu napodobuje chování analogových prvků (např. zesilovačů), jejichž výstup zůstane na limitní hodnotě při přebuzení vstupu. Finální podoba implementace vztahu (1) bude po vložení operátoru saturace

$$\mathrm{y}[n] = \mathrm{S}\left[\mathrm{Q_w}\left[\mathrm{M_r}\left[\sum_{m=0}^{N-1} \mathrm{x}[n-m]\mathrm{h}[m]\right]\right]\right] \qquad (6)$$

4 Možnosti paralelizace

4.1 Charakteristika

Použití procesoru s jednotkou MAC s podporou pro změnu měřítka, zaokrouhlení a saturaci spolu s modifikovanou harvardskou architekturou maximálně urychlilo jednotlivé elementární operace a umožnilo zkrátit délku výpočtu konvoluce až na zhruba N operací. Doposud jsme ale hovořili pouze o sekvenčním provádění výpočtu (1). Řád filtru, příp. délka vektoru $\boldsymbol{h}$, je stále rozhodujícím faktorem, který určuje výslednou rychlost výpočtu a maximální dosažitelnou vzorkovací frekvenci zpracovávaného signálu. Samotný výpočet (1) je nicméně dobře paralelizovatelný, neboť jednotlivé dílčí součiny jsou na sobě vzájemně nezávislé. Celý výpočet je možné provést v 1 + $\log_2 N$ cyklech, kde v prvním cyklu provedeme všechna násobení x[n – m] × h[m] a vedujících řádově $\log_2 N$ cyklech sečteme pomocí binární redukce všechny dílčí součiny (*obr. 3*). Takový postup ovšem vyžaduje současnou dostupnost všech prvků vektoru $\boldsymbol{x}$ a $\boldsymbol{h}$.

Další akceleraci výpočtu je zřejmě možné dosáhnout jen rapidním zvýšením propustnosti rozhraní operační paměti a tedy změnou paměťové architektury výpočetního systému. Principiálně máme tyto možnosti:

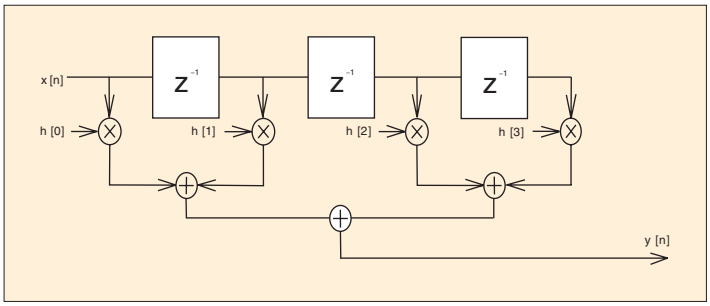
- použít rychlejší paměť, paměť se širším slovem, nebo několik paralelně zapojených pamětí;
- použít architekturu s velkým množstvím vnitřních pamětí (registrů, nebo vyrovnávacích – *cache* – pamětí).

4.2 Procesorová řešení

Budeme-li schopni z operační paměti načítat data po dvojicích slov a budeme-li mít k dispozici dvě paralelně pracující jednotky MAC, můžeme například rovnici (1) implementovat jako

$$\mathrm{y}[n] = \sum_{m=0,m=2k+1}^{N-1} \mathrm{x}[n-m]\mathrm{h}[m] + \sum_{m=0,m=2k}^{N-1} \mathrm{x}[n-m]\mathrm{h}[m], \quad k \in \mathrm{Z} \qquad (7)$$

První sumou tedy sčítáme všechny liché a druhou všechny sudé prvky z původního vzorce (1). Celý výpočet lze pak provést v přibližně N/2 cyklech. Tohoto urychlení lze snadno dosáhnout na procesorech, které



Obr. 5 FIR filtr 3. řádu

obsahují výpočetní jednotku pro zpracování dat v režimu SIMD. Výpočetní jednotka s architekturou SIMD (*Single Instruction stream*, *Multiple Data streams*) umožňuje současně zpracovávat více datových slov s tím omezením, že nad všemi slovy je prováděna stejná operace. Procesor potom například místo manipulace s jedním 64 bitovým slovem v registru s registrem pracuje, jako by byl složen ze čtyř 16 bitů širokých slov (*obr. 4*). Přitom celé 64 bitů široké slovo lze z paměti načíst v jediném kroku. Tím obcházíme problém nedostatečné propustnosti paměťového rozhraní za cenu práce s užším slovem, než jaké je možné z paměti na jednu instrukci vyčíst. Základní informace o celém konceptu lze nalézt např. v [7]. Koncepce operací SIMD byla v moderních procesorech používaných v osobních počítačích dále rozvedena a příslušné architektury jsou označeny jako SSE, SSE2,3,4 či 3DNow! v závislosti na výrobci a verzi procesoru.

Při snaze o další akceleraci výpočtu na programovatelném procesoru už velmi rychle narazíme na limity datové propustnosti rozhraní k paměti – tzv. *von Neumann's bottleneck*. Nemůžeme přidávat další a další výpočetní jednotky a zvyšovat množství paralelně prováděných operací podle libosti, protože nebudeme stačit dostatečně rychle načítat z paměti prvky vektorů $\boldsymbol{x}$ a $\boldsymbol{h}$. Přitom je zřejmé, že teoretického limitu urychlení konvoluce stále zdaleka nebylo dosaženo a výpočet by mělo jít i dále značně urychlit. Další urychlení umožňují až specializované výpočetní architektury.

4.3 Systolické pole

Chceme-li dosáhnout dalšího zvýšení výkonu, musíme sáhnout po diametrálně odlišných architekturách výpočetních systémů. Vodítko k implementaci výkonnějšího systému lze nalézt na *obr. 5*, kde je zakreslené schéma toku dat ve filtru FIR. Je zřejmé, že obvod zapojený podle této struktury (zpoždění Z^{-1} nahradíme registry) bude schopen v každém cyklu vypočíst jeden vzorek signálu $\boldsymbol{y}$. Vysokého výpočetního výkonu je ovšem dosaženo za cenu masivní paralelizace výpočtu a tedy velikosti obvodu i spotřeby elektrické energie a disipovaného tepelného výkonu.

Všimněte si, že problém úzkého hrdla paměťového systému je zde vyřešen opakovaným používáním načítaných vzorků ve struktuře obvodu – každý vzorek strukturou postupně prochází zleva doprava a je násoben jednotlivými prvky vektoru $\boldsymbol{h}$. Vzorky jsou postupně předávány mezi bloky a malé lokální paměti – registry – v blocích dovolují snížit nároky na paměť. Z architektonického pohledu tedy dosahujeme urychlení použitím vnitřních pamětí. Maximální dosažitelná vzorkovací frekvence pak může být rovna až maximální dosažitelné hodinové pracovní frekvenci navrženého obvodu. S moderními obvody FPGA lze snadno dosáhnout řádově stovek MHz.

Nevýhodou prezentované struktury jsou dlouhé kombinační cesty začínající na výstupu zpožďovacího registru, procházející násobičkou a posléze sčítačkami až na výstup obvodu. Délka kombinační cesty snižuje maximální pracovní hodinovou frekvenci obvodu, a tedy i nejvyšší dosažitelnou vzorkovací frekvenci. Nedostatek lze odstranit proudovým zpracováním, případně použitím jiného typu architektury [1].

Popsaná architektura je nazývána systolickým polem; typicky má následující vlastnosti [1]:

- Obvod je složen ze skupiny propojených buněk, každá realizuje relativně jednoduchou operaci (jednoduchost je ovšem relativní, buňky mohou být na různé úrovni abstrakce od jednoduché sčítačky s registrem po procesorový uzel). Ve struktuře je jen několik málo typů buněk. Buňky mohou být řazeny do jedno- či dvourozměrné struktury.
- Mezi buňkami probíhá komunikace podle jednoduchého regulárního schématu (pravidelný přesun dat). Pravidelné předávání dat výpočetními jednotkami evokuje představu krevního oběhu a dalo architekturám název (*systola* je stah srdečních síní).
- Pole má modulární a snadno rozšiřitelnou strukturu.
- Komunikace s vnějším světem probíhá pomocí hraničních buněk.
- Architektura využívá masivního paralelismu.
- Obvykle chybí operace indexování/adresování paměti, plynulý tok dat do systému musí být zajištěn externími prostředky.

5 Závěr

Příspěvek představil hardwarové architektonické prvky běžně používané pro urychlení číslicového zpracování signálů a rozebral jejich dopad jak na časovou náročnost vlastního výpočtu, tak na technické parametry navrhovaného číslicového obvodu (velikost návrhu a spotřeba elektrické energie). Popsány byly pouze základní koncepce, které lze dále rozvíjet a upravovat podle potřeb konkrétní aplikace (například dalším vložením registrů pro proudové zpracování lze zvýšit maximální pracovní hodinovou frekvenci navržených bloků apod.).

Závěrem poznamenejme, že při návrhu příslušné aplikace je třeba brát v potaz nejen architektonické aspekty systému, ale i požadavky na včasné uvedení aplikace na trh, a tedy časové nároky vlastního návrhu aplikace a verifikace číslicového systému. Ty lze velmi účinně regulovat pomocí vhodného rozdělení práce mezi hardwarovou a softwarovou část navrhovaného systému (tzv. *hardware-software partitioning*).

Tato práce byla podpořena výzkumným záměrem MSM6840770012 – Transdisciplinární výzkum v biomedicínském inženýrství 2.

Ing. Jakub Šťastný, Ph.D.
katedra teorie obvodů FEL ČVUT v Praze
ASICentrum, s. r. o.

LITERATURA

[1] KUNG, H. T., Why Systolic Architectures? *Computer*, 15 (1982), no. 1, pp. 37–46.

[2] SOVKA, P., ČMEJLA, R., ŠMEJKAL, L., Když se řekne... Číslicové filtry II. *Automatizace*, 48 (2005), č. 9, s. 560–562.

[3] Xilinx. XtremeDSP for Virtex-IV FPGAs. UG073, May 15, 2008.

[4] ŠŤASTNÝ, J., Návrh aritmetických operátorů na FPGA. *Automatizace*, 52 (2009), č. 2, s. 71–75, 2009.

[5] PLUHÁČEK, A., *Projektování logiky počítačů (Skripta ČVUT)*. Praha : ČVUT, 1996.

[6] RUČKAY, L., *Z05-2. Implementace MAC jednotky na FPGA*. Praha : ČVUT, 2005. Dostupné na *http://amber.feld.cvut.cz/fpga/research_teaching.html* [kontrolováno 21. 12. 2008]

[7] MITTAL, M., PELEG, A., WEISER, U., MMX™ Technology Architecture Overview. Intel Technology Journal Q3' 97, pp. 1–12.

[8] ŠŤASTNÝ, J., High performance cross-correlator, in Preprints, IFAC Workshop On Programmable Devices And Systems, Ostrava, Czech Republic, pp. 184 189, February 11–13, 2003.