Nová a efektivní metoda pro zajištení platnosti dat ve vestavných pamětech FPGA se zaměřením na kompresi IP packetů v reálném čase

Matěj Bartík
2. ročník prezenčního studia
Školitel: Sven Ubik, školitel specialista: Pavel Kubalík

České vysoké učení technické v Praze, Fakulta informačních technologií
Thákurova 9, Praha 6 - Dejvice, 160 00, Česká republika
bartimat@fit.cvut.cz

***Abstrakt*—Tento článek se zabývá novým a efektivním způsobem zajištěná platnosti dat ve vestavných pamětech uvnitř FPGA, který je vhodný pro realizaci slovníků v bezztrátových kompresních algoritmech realizovaných v hardwaru (FPGA). Klíčem k této inovaci je chytré využití vlastností LUT (Look-Up Table), které umožňuje dosáhnout menšího počtu využitých zdrojů a vyšší frekvence celého systému oproti běžně používaným způsobům realizace. Tato metoda je navržena pro vysokou propustnost a nízkou latenci, což ji činí vhodnou pro kompresi jumbo IP packetů obsahující multimediální data v reálném čase. Použitou metodu je možné aplikovat na další datové struktury, které jsou mapovány do vestavných bloků RAM v FPGA.**

***Klíčová slova*—FPGA, LUT, slovník, bezztrátová, komprese, LZ4, LZ77, efektivní, jumbo, IP, packety**

I. Úvod a motivace

Na základě motivace shrnuté v [1] jsem se rozhodl za cíl dizertace zvolit „Tvorba nových optimalizací těžících z FPGA architektury“ i když některé tyto optimalizace mohou být inspirovány druhým tématem „Metody pro přenos procesorových optimalizací na FPGA“. V následujících kapitolách popíši především nově zjištěné poznatky vůči předchozímu (prvnímu) ročníku [1] doktorandského studia.

II. Analýza

Moderní implementace bezztrátových kompresních algoritmů, které jsou implementovány v hardwaru (FPGA), mají některé společné vlastnosti [1]. Jednou z nich je zvětšení šířky zpracovávaného slova z původně běžné hodnoty 8 bitů na 32, 64 nebo 128 bitů [2]–[4]. Původní 8-bitový přístup vychází z triviální implementace kompresních algoritmů, kdy takový přístup značně zjednodušuje výsledný hardware, protože se obejde bez pokročilé práce s pamětí (zarovnání paměti) [5].

A. Problémy HW implementací

Typickým přístupem k paralelizaci 8-bitových implementací byla instanciace více stejných kompresních bloků do FPGA, mezi které je rozdělena zátěž. V současné době se kromě použití více stejných bloků dostává do popředí i paralelizace uvnitř každého kompresního bloku právě zvětšením šířky zpracovávaného slova. S tím souvisí nutnost paralelizace celého mechanismu vyhledávání shody.

Prvním typickým příkladem řešeného problému je paralelizovatelnost přístupu ke slovníku, ve kterém je vyhledávána shoda [3]. Druhým typickým příkladem je použití větších slovníků pro zvýšení kompresního poměru, kdy je potřeba tento slovník před každou sadou vstupních dat (re)inicializovat [1] [5]. Právě tento problém bych rád podrobněji rozebral v tomto článku.

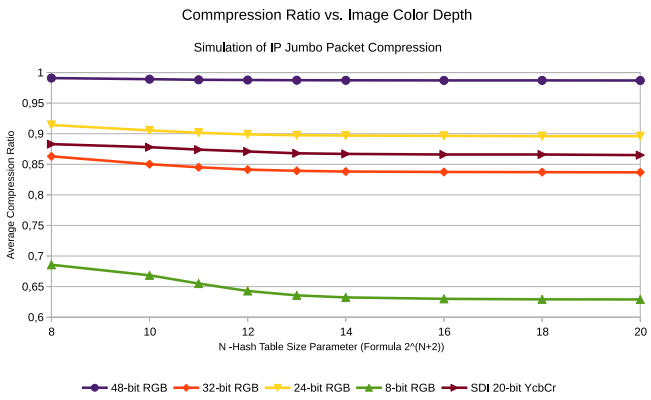
III. Analýza

A. Kompresní algoritmus LZ4

Bezztrátový kompresní algoritmus LZ4 [6] je derivátem LZ77 [7] a příkladem moderního „rychlého“ („fast“) kompresního algoritmu. Tato třída bezztrátových kompresních algoritmů se vyznačuje využitím velkého množství procesorových optimalizací (typicky je velikost slovníku rovna nebo menší než L1 cache) [8]. Tím se dosahuje velké rychlosti komprese, často ale na úkor kompresního poměru. LZ4 jsem zvolil jako výchozí algoritmus pro HW implementaci, jehož optimalizace se pokusím portovat do hardwaru. Pokud identifikuji některé neoptimální části LZ4, můžu se pokusit navrhnout další optimalizace speciálně navržené pro FPGA. Mezi klíčovou optimalizaci LZ4 patří mechanismus vyhledávání shody.

1) Mechanismus vyhledávání shody LZ4: Vyhledání shody je u algoritmu LZ4 založeno na hashovací tabulce [4], která reprezentuje slovník. Každý záznam v hashovací tabulce obsahuje odkaz do paměti vstupních dat (pointer) s potencionálním kandidátem na shodu. Adresu záznamu je vypočtena ze 32-bitových vstupních dat pomocí multiplikativního hashování s prvočíslem 2654435761. Toto prvočíslo je nejblíže hodnotě $\frac{2^{32}}{\phi} = 2654435769$, kde ϕ je hodnota Zlatého řezu. V případě FPGA může být implementace provedena pomocí DSP bloků.

Hashovací tabulka musí být před každým během algoritmu LZ4 (re)inicializována. V případě softwarové varianty LZ4 (referenční zdrojový kód je v jazyce C) se každá inicializace



Obrázek 1. Vliv bitové hloubky, kódovacího schématu a velikost hashovací tabulky na kompresní poměr. [1]

provádí alokováním paměti (malloc) a následně je lineárním průchodem každá buňka pole (záznam tabulky) inicializován na výchozí hodnotu 0 (null pointer). V případě hardwarové implementace je nutné všechny buňky hashovací tabulky inicializovat do výchozí hodnoty nebo udržovat informaci, zda je záznam v dané buňce platný či neplatný.

B. Experimentální měření

V současné době je prováděn výzkum [9], zda lze „rychlé“ bezztrátové kompresní algoritmy použít pro „lehkou“ kompresi multimediálních dat, přestože jsou k tomu určeny jiné algoritmy pro specifické typy dat, převážně ztrátové. Tyto algoritmy jsou však mnohem složitější (je potřeba více logických hradel). Dále komprese trvá delší dobu, tedy nepříznivě ovlivňují zpoždění. Prvním krokem je tedy analýza vhodnosti kompresního algoritmu LZ4 pro přenos IP packetů obsahující multimediální data. Na základě experimentálních měření jsem se rozhodl právě pro algoritmus LZ4, který dosahoval průměrné úspory 23%.

Tyto experimenty však neuvádějí, jak velký byl použit slovník a LZ4 byla implementována ve formě softwaru. Pro realističtější odhad jsem simuloval posílání multimediálních dat v „Jumbo“ packetech s datovou částí o velikosti 9000 bytů a pro velikosti slovníků odpovídající možnostem použitého FPGA. V našem specifickém případě (SDI 20-bit) jde o průměrnou úsporu 11%–13% podle velikosti slovníku. Nejlepší výsledky v poměru velikost slovníku vůči kompresnímu poměru (na obrázku 1) vykazují slovníky o velikosti 1024–4096 záznamů [1].

C. Řešený problém a současná řešení

Jako řešený problém jsem si vybral navržení nového způsobu (re)inicializace slovníku tak, aby se zkrátila latence reinicializace a/nebo se zmenšila spotřeba zdrojů FPGA. Na základě výše uvedených informací jsem si stanovil tyto předpoklady:

- velikost slovníku minimálně 1024 záznamů,
- velikost datové části IP packetu 9000 bytů,
- MVTP používá 64-bitovou datovou cestu na 156,25 MHz (požadavky 10G Ethernetu).

Z předpokladů je patrné, že packet o maximální velikosti 9000 bytů je spodní mez pro kompresi 1125 hodinových taktů. Pro první běh kompresního algoritmu může být hashovací tabulka inicializována pomocí výchozích hodnot v bitstreamu. Další běhy algoritmu je nutné prosvést (re)inicializaci pomocí logiky. Za předpokladu, že (re)inicializace slovníku se dá provádět po dobu kopírování vstupních dat do paměti a výstupních dat z paměti zpět do MVTP, máme k dispozici čas v řádu stovek hodinových taktů.

V současnosti se používají tyto přístupy [10]:

- Nahrání výchozí hodnoty do každé buňky paměti lineárním průchodem paměti (za pomocí čítače) [11]
- Použitím příznakového registru z klopných obvodů, který obsahuje příznak platný/neplatný pro každou buňku paměti a lze všechny klopné obvody resetovat paralelně.

Linární průchod paměti se zápisem výchozí hodnoty je možné považovat za triviální řešení a při HW implementaci se použije čítač, který generuje adresy pro zápis. Toto řešení optimální z hlediska logických hradel, ale dlouhý carry-chain sčítačky (pro velké šířky adresy do paměti) výrazně ovlivňuje dosažitelnou frekvenci. Největší nevýhodou je dlouhá doba potřebná pro (re)inicializaci (mocnina 2 na počet adresních bitů), v našem případě minimálně 1024 taktů nebo více.

Příznakový registr je optimální z pohledu doby potřebné pro (re)inicializaci, která trvá pouze jeden hodinový takt. Jeho nevýhodou jsou exponenciální nároky na logická hradla a při větší šířce adresy nad 8 bitů je takto realizovaný systém příznaků větší, než samotný design LZ4 na FPGA [1].

IV. Návrh možného alternativního řešení

Obě metody (lineární průchod a registr příznaků) mají problémy s optimalitou z hlediska času nebo logických hradel. Pro náš případ by bylo mnohem vhodnější použít řešení, které by sice nemuselo být optimální, ale mohlo být mnohem vhodnější než obě dnes používané metody, případně i rychlejší. Typické moderní FPGA má obvykle tyto možnosti uchování informace (logické hodnoty):

- Klopné obvody,
- Vestavná paměť (BlockRAM),
- Distribuovaná paměť.

Z těchto možností jsem se rozhodl detailněji prozkoumat možnosti distribuované paměti na FPGA. Distribuovaná paměť je jedním ze 3 režimů LUTu, kdy LUT místo realizace logické kombinační funkce pracuje jako malá RAM, v případě Virtex-6 o velikosti $2^6 = 64$ bitů (Virtex-6 používá šestivstupé LUTy). Vzhledem k paralelní povaze FPGA je možné současně pracovat s více bloky distribuované paměti.

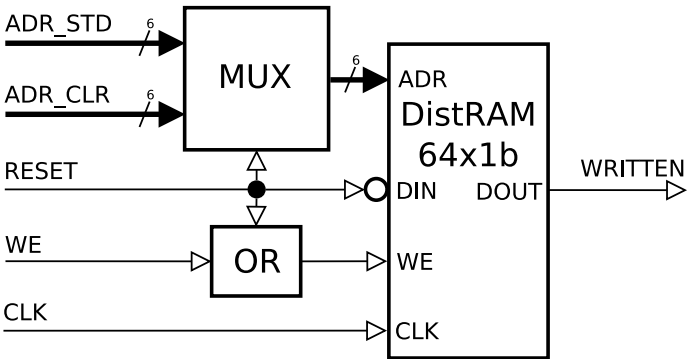
Možným řešením je použít distribuovanou paměť pro realizaci systému příznaků na místo jednotlivých klopných obvodů. Jednotlivý blok distribuované paměti je nutné inicializovat stejným způsobem jako samotnou hashovací tabulku (lineárním průchodem). Díky možnosti paralelního přístupu k jednotlivým blokům distribuované paměti je možné inicializovat všechny bloky naráz a tím je zároveň omezena doba (re)inicializace celého systému příznaků na 64 hodinových taktů (počet bitů bloku distribuované paměti).

V. Příklad možné implementace

Systém příznaků realizovaný pomocí distribuované paměti lze rozdělit na několik hlavních částí (Obr. 3).

A. Elementární blok

Elementární blok (EB) obsahující jednu buňku distribuované paměti, adresní multiplexor (Obr. 2), který přepíná mezi adresami standardního režimu a režimu (re)inicializace. Signál resetu zároveň slouží jako datový vstup. V našem specifickém případě se při přístupu ve standarním režimu provádí zápis a čtení příznaku současně (dochází automaticky k zápisu logické jedničky), zatímco v režimu (re)inicializace je automaticky zapisována logická nula.



Obrázek 2. Vnitřní architektura elementárního bloku.

B. Adresní dekodér

Adresní dekodér se sestává z bloků SPLIT a EB SEL. SPLIT rozděluje adresu na vyšší (která jde do EB SEL) a nižší část o velikosti 6 bitů, kterou se přímo adresují jednotlivé bity uvnitř elementárního bloku ve standardním režimu. EB SEL na základě vyšší části adresy a signálu write-enable (WE) generuje signály chip-enable (CE), kterým vybírá EB s požadovanou hodnotou. Zbylé signály CE vymaskují výstupní hodnoty z nevybraných EB.

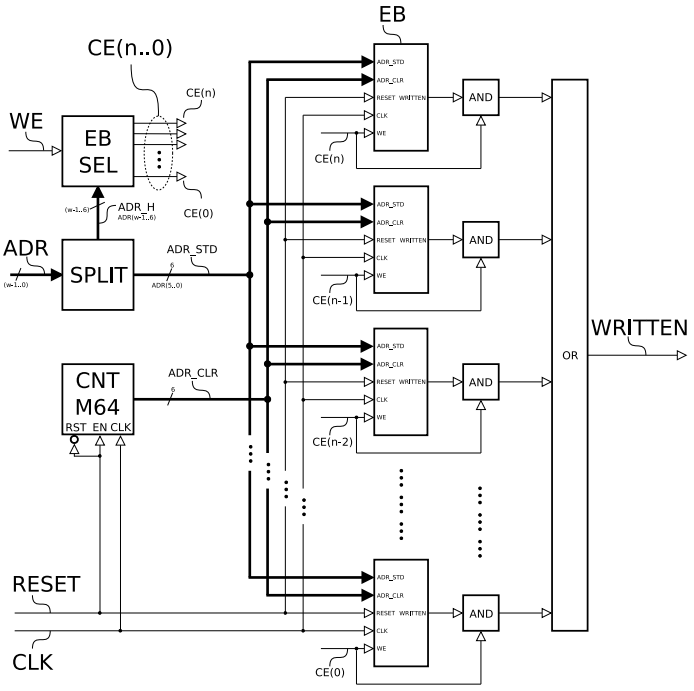
C. Čítač modulo 64

Čítač M64 slouží jako generátor adres po dobu (re)inicializace. 6-bitová sčítačka nebude s největší pravděpodobností omezujícím faktorem pro maximální dosažitelnou frekvenci.

VI. Experimentální měření a výsledky

Pro snažší měření jsem sjednotil rozhraní (Program 1) pro všechny typy přístupů (lineární průchod, příznakový registr pomocí klopných obvodů, pomocí distribuované paměti), kde velikost hashovací tabulky je nastavena parametrem „W“. Velikost datového vstupu a výstupu je 36 bitů, abych simuloval případný 32-bitový pointer, tak jak je to v SW implementaci LZ4 (36 bitů je jedna z nativních šířek BlockRAM bloků v FPGA).

Pro měření jsem použil FPGA Virtex-7 690T (XC7V690T-2FFG1158) a vývojové prostředí Xilinx ISE 14.7. Měření



Obrázek 3. Architektura celého systému příznaků.

Program 1 VHDL rozhraní pro všechny typy implementací

```
entity top is port (
  clk, reset, we : in std_logic;
  adri : in std_logic_vector(W-1 downto 0);
  din : in std_logic_vector(35 downto 0);
  dout : out std_logic_vector(35 downto 0);
  writen : out std_logic);
end top;
```

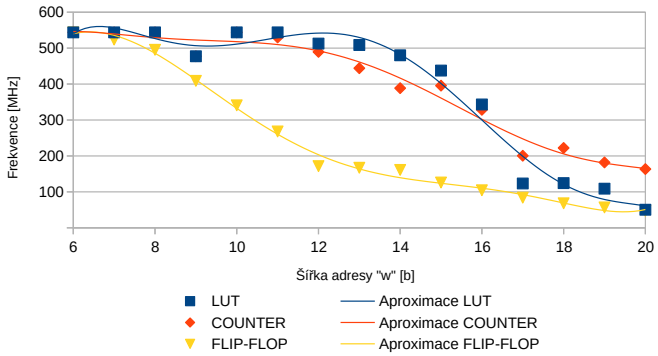
probíhala na počítači s 6-ti jádrovým procesorem Intel Xeon E5-1650 v3 (15M Cache, 3.50 GHz) a 32 GB DDR4 RAM. Nastavená syntézní a implementační strategie preferovala maximální dosažitelnou frekvenci. Následně jsem měnil požadované časování celého designu, než jsem nalezl jeho maximum. Naměřené výsledky jsou v Tabulce I a vizualizovány jako Obr. 4. Z výsledků je patrné, že nová metoda předčí oba dnes používané přístupy z hlediska dosažitelné frekvence, tak má i výborné výsledky co se týče nároků na logická hradla.

Pro hodnoty 17–20 parametru „W“ nastává výrazné zhoršení, které však přičítáme neschopnosti ISE efektivně pracovat s velkými návrhy (jak je patrné z výrazného úbytku klopných obvodů – patrně přestává fungovat duplikace čítače M64 a tím se poruší prostorová lokalita). Dále se nepodařilo provést syntézu (hodnoty 17–20 parametru „W“) pro příznakový registr využívající klopné obvody z důvody nestability Xilinx ISE 14.7 s největší pravděpodobností ze stejného důvodu, přestože by mělo být využito méně než 100% zdrojů použitého FPGA (XC7V690T-2FFG1158).

Tabulka I
VLASTNOSTI (LOGICKÁ HRADLA, FREKVENCE) JEDNOTLIVÝCH IMPLEMENTACÍ V ZÁVISLOSTI NA PARAMETRU „W“ PRO ČIP XC7V690T-2FFG1158.

COUNTER (Lineární průchod)															
Šířka adresy „W“	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slice	96	85	93	100	111	110	112	118	122	249	287	393	194	631	347
LUTy	179	185	191	196	197	203	206	211	215	386	457	533	299	805	568
Klopné obvody	50	52	54	56	59	61	63	65	67	75	76	95	79	90	91
Frekvence [MHz]	543,8	543,8	543,8	477,3	543,8	529,7	489,5	444,0	388,7	395,6	328,3	200,5	222,3	182,0	163,5
FLIP-FLOP (Příznakový registr pomocí klopných obvodů)															
Šířka adresy „W“	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slice	89	137	155	268	453	1157	2205	3776	10108	15774	*30335*	*55482*	*118295*	*236468*	*463594*
LUTy	221	309	478	837	1542	3417	6562	13850	31750	57582	*153323*	*242202*	*525419*	*1119292*	*2349075*
Klopné obvody	111	176	303	594	1148	2223	4157	8270	16607	32928	*75852*	*128087*	*258144*	*512953*	*1023707*
Frekvence [MHz]	543,8	523,8	495,3	408,8	340,8	268,2	171,5	167,1	161,1	126,0	*104,4*	*83,8*	*68,1*	*56,7*	*47,5*
LUT (Příznakový registr pomocí distribuované paměti)															
Šířka adresy „W“	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Slice	73	79	77	75	98	123	206	337	488	984	2364	3686	7033	13104	24987
LUTy	146	151	159	171	209	235	377	471	912	1889	3657	6737	13341	25227	55607
Klopné obvody	54	57	63	73	91	129	195	239	619	1118	2176	71	72	190	74
Frekvence [MHz]	543,8	543,8	543,8	477,3	543,8	543,8	512,6	508,6	480,1	437,4	343,2	123,4	124,3	108,9	50,2

*šedé výsledky jsou extrapolovány, protože syntézní nástroj nezvádl tak velký návrh.



Obrázek 4. Závislost frekvence na parametru „W“ pro různé přístupy.

VII. POKRAČOVÁNÍ PRÁCE A CÍL DIZERTAČNÍ PRÁCE

Na základě rešerší a současných výsledků:

- Detailní analýza vlastností kompresního algoritmu LZ4,
- Vyhodnocení (ne)vhodnosti LZ4 pro přenos multimediálních dat v softwaru (hardwaru),
- Návrh nové rychlé a na zdroje nenáročné metody pro realizaci systémů příznaků,

jsem se rozhodl pokračovat v současné práci a za cíl dizertační práce jsem si stanovil tvorbu nových optimalizací pro hardwarové implementace bezztrátových kompresních algoritmů založených na algoritmu LZ77 a které jsou určené pro vysokou propustnost nebo nízkou latenci.

VIII. ZÁVĚR

Navrhl a implementoval jsem novou metodu, kterou lze (re)inicializovat hashovací tabulku nebo jinou RAM orientovanou datovou strukturu na FPGA, která vykazuje lepší výsledky než dosud používané metody lineárního průchodu paměti a příznakového registru z klopných obvodů. Výsledná metoda založená na použití distribuované paměti dosahuje řádově konstatního času nutného pro (re)inicializaci při velmi nízkých nárocích na logická hradla.

PODĚKOVÁNÍ

Tento výzkum byl částečně podpořen z grantu SGS16/121/OHK3/1T/18.

REFERENCE

[1] M. Bartík, S. Ubik, and P. Kubalik, “Rychlé bezztrátové kompresní algoritmy,” in *Sborník příspěvků PAD 2015*, Zlín, CZ, 2015, pp. 31–36.

[2] J. L. Nunez and S. Jones, “Gbit/s lossless data compression hardware,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 499–510, June 2003.

[3] B. Sukhwani, B. Abali, B. Brezzo, and S. Asaad, “High-throughput, lossless data compresion on fpgas,” in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, May 2011, pp. 113–116.

[4] J. Fowers, J. Y. Kim, D. Burger, and S. Hauck, “A scalable high-bandwidth architecture for lossless compression on fpgas,” in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 52–59.

[5] M. Bartík, S. Ubik, and P. Kubalik, “Lz4 compression algorithm on fpga,” in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, Dec 2015, pp. 179–182.

[6] Y. Collet, “Realtime data compression: Development blog on compression algorithms.” tinyurl.com/qc9yve4, [Online; accessed 15-May-2016].

[7] J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[8] J. Kane and Q. Yang, “Compression speed enhancements to lzo for multi-core systems,” in *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, Oct 2012, pp. 108–115.

[9] R. D. Gomes, Y. G. G. d. Costa, L. L. Aquino Júnior, M. G. d. Silva Neto, A. N. Duarte, and G. L. d. Souza Filho, “A solution for transmitting and displaying uhd 3d raw videos using lossless compression,” in *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*, ser. WebMedia ’13. New York, NY, USA: ACM, 2013, pp. 173–176.

[10] M. Stohanzl and Z. Fedra, “The fpga implementation of dictionary; hw consumption versus latency,” in *Telecommunications and Signal Processing (TSP), 2013 36th International Conference on*, July 2013, pp. 82–85.

[11] S. Rigler, W. Bishop, and A. Kennings, “Fpga-based lossless data compression using huffman and lz77 algorithms,” in *2007 Canadian Conference on Electrical and Computer Engineering*, April 2007, pp. 1235–1238.