Logická syntéza s nativní podporou XOR hradel

Ivo Háleček
1. ročník, prezenční studium
Školitel: Petr Fišer, Specialista: Jan Schmidt

Fakulta informačních technologií, ČVUT
Thákurova 9, 160 00 Praha
ivo.halecek@fit.cvut.cz

***Abstrakt*—V článku je představena možnost využití nové reprezentace logických obvodů pomocí Xor-And-Invertor grafů (XAIG) v syntézních algoritmech. XAIG jsou založeny na And-Invertor grafech, orientovaných acyklických grafech, kde uzly představují dvou-vstupá hradla AND či XOR a hrany mohou být negované. Předpokládá se, že tato reprezentace umožní algoritmům s XORy nativně pracovat a odhalit komplexnější závislosti. Pro experimentální ověření pravdivosti bude reimplementován algoritmus rewrite a bude porovnán s originálním rewritingem.**

***Klíčová slova*—Logická syntéza, XOR, AIG, XAIG, ABC, rewriting**

I. Úvod

Zlepšování logické syntézy je stále aktuální téma. Přesto, že proces logické syntézy a optimalizace se zdál být již v minulých dekádách efektivně vyřešený problém, v nedávné době se objevilo několik nových přístupů, většinou založených na nových datových strukturách používaných pro reprezentaci funkcí a sítí [22].

Původní nástroje logické syntézy vyjadřovaly funkce pomocí sum-of-products (SOP) [1], [2]. Nad touto reprezentací pracovaly dvouúrovňové, později víceúrovňové SOP minimalizační nástroje. Jako jednodušší alternativa k reprezentaci v SOP byla v mnoha algoritmech využívána reprezentace pomocí NOR hradel [3], která však přinesla pouze jednodušší implementaci algoritmů.

Velkým průlomem v reprezentaci funkcí bylo představení binárních rozhodovacích diagramů (Binary Decision Diagrams - BDD) [4], [5]. Syntézní a optimalizační algoritmy byly upraveny pro tyto struktury, což zlepšilo jejich výkon [6], [7], [8], [9].

Tyto reprezentace, ačkoliv hojně využívané, však trpěly špatnou škálovatelností. Z tohoto důvodu vznikla velmi efektivní reprezentace logiky – And-Inverter-Grafy (AIG) [10], [11], [12]. V AIG je logická síť reprezentována pomocí orientovaného grafu, kde uzly jsou 2-vstupá AND hradla a hrany mohou být negované. Mnoho algoritmů, založených na reprezentaci v AIG bylo implementováno do moderního akademického nástroje pro logickou syntézu a verifikaci, ABC [13]. Reprezentace v AIG pravděpodobně je, či brzo bude integrována i do komerčních nástrojů [14].

V tomto článku představujeme rozšíření konceptu AIG o nativní podporu hradel XOR, Xor-And-Inver grafy (XAIG). XAIG představují ortogonální přístup k Majority-Invertor-Graphs (MIGs), což je reprezentace založená na AIG, kde uzly jsou nahrazeny majoritní funkcí 3 proměnných [23].

Reprezentace pomocí XAIG je přístup ortogonální k MIG. XOR není monotónní a použití XAIG může vést k jiným oblastem efektivity implementace než MIG.

II. Popis problému

Přesto, že se několik let považovalo AIG za univerzální reprezentaci pro logické obvody, algoritmy postavené na AIG vykazují u některých obvodů hluboce neoptimální výkon, produkující obvody s mnohem větší plochou, než jaké je možné docílit. Jedna ze společných charakteristik této problémové množiny obvodů je vysoká intenzita hradel typu XOR. Důvodem neschopnosti algoritmů správně využít tato hradla může být v tom, že XOR má jiné vlastnosti než AND nebo NOR. Pokud tedy algoritmus správně neidentifikuje XOR, a pracuje s ním jako se soustavou ANDů a invertorů, může vyprodukovat řádově horší výsledky, než pokud by byl schopný nativně pracovat s XORem [15], [16].

Pro podpoření tohoto tvrzení jsme vygenerovali ze sady benchmarků [17], [18], [19], [20] verzi zkolapsovanou do dvouúrovňového popisu (SOP). Tento proces zamaskuje XORy, což, v případě nedostatečné identifikace a využití XORů optimalizačními algoritmy, může způsobit výrazné zhoršení výsledků syntézy.

V nástroji ABC provedli experimentální porovnání sady benchmarků v originální a collapsed verzi pomocí 20 iterací sekvence příkazů *dch; if; mfs*. Tato sekvence obvod optimalizuje, namapuje do LUTů a optimalizuje neurčené hodnoty. Jak je vidět v Tabulce 1, collapse u některých obvodů způsobil, že syntéza je nedokázala zoptimalizovat.

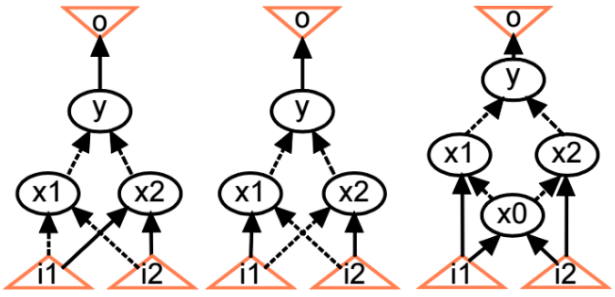
TABULKA 1: POROVNÁNÍ OPTIMALIZACE ORIGINÁLNÍCH A COLLAPSED BENCHMARKŮ

name	original LUTs	collapsed LUTs	LUT ratio
s4863	425	138111	324,97
mm9b	121	6153	50,85
c880	116	5627	48,51
c5315	475	19873	41,84
mm9a	98	2593	26,46
Altera_oc_minirisc	770	11512	14,95
c432	71	827	11,65
i10	665	6285	9,45
ITC_b05	174	1367	7,86
mm4a	40	304	7,60
Altera_oc_des_des3perf	17817	69983	3,93
Altera_oc	635	2431	3,83

III. NAVRHOVANÉ ŘEŠENÍ

Jedním ze způsobů, jak se pokusit zlepšit výsledky syntézy nad obvody s vysokou intenzitou hradel XOR je přepsat současné syntézní algoritmy tak, aby pracovaly s obecnější reprezentací obvodů a uměly XORy efektivně využít. Takovou reprezentací by mohlo být XAIG, rozšířené AIG tak, že uzly mohou mimo funkce AND představovat funkci XOR.

Jak lze vidět na Obrázku 1, XOR je v AIG reprezentován dvěma vstupy (i1,i2), dvěma či třemi vnitřními ANDy (x0, x1, x2) a výstupním ANDem (y), který může být invertovaný.



Obrázek 1: reprezentace hradla XOR v AIG

Náš nedávný výzkum ukázal, že nástroj ABC sice již obsahuje možnost reprezentovat obvody v podobné struktuře jako XAIG, And-Xor-Mux grafy. Tuto reprezentaci však využívá pouze podmnožina optimalizačních algoritmů z nového balíčku příkazů ABC9. Jelikož tyto nové algoritmy nejsou téměř dokumentované ani publikované, provedli jsme nejdříve výzkum toho, jak jsou tyto nové algoritmy schopny využít XORů v XAIG.

Nejprve jsme zjistili, které algoritmy využívají této nové reprezentace tak, že jsme po úpravě zdrojového kódu u algoritmů z balíčku ABC9 zaznamenávali volání funkce pro identifikaci XORů a jimi nahrazení ekvivalentních struktur v AIG. Zkoumali jsme pouze příkazy, u kterých jsme z nápovědy zjistili, že se týkají optimalizace (X)AIG. Výsledky jsou zobrazeny v Tabulce 2. Tyto algoritmy jsme vybrali pro porovnání jejich účinosti s modifikovanou verzí, která struktury v AIG XORy nenahrazovala. Místo vytvoření nativního XORu se vždy vytvořila funkčně ekvivalentní struktura z ANDů, jako na **Obrázek 1.**

TABULKA 2: OPTIMALIZAČNÍ ALGORITMY Z BALÍČKU ABC9 A VÝSLEDEK ZKOUMÁNÍ VYUŽITÍ XAIG (SLOUPEC XAIG)

příkaz	Oficiální popis	XAIG ?
&b	performs AIG balancing to reduce delay and area	Ano
&blut	performs AIG balancing for the given LUT size	Ano
&dc2	performs heavy rewriting of the AIG	Ne
&dch	computes structural choices using a new approach	Ne
&dsd	performs DSD-based collapsing	Ne
&dsdb	performs DSD balancing	Ne
&fadds	detects full-adder chains and puts them into white boxes	Ne
&flow	integration optimization and mapping flow	Ano
&flow2	integration optimization and mapping flow	Ano
&fraig	performs combinational SAT sweeping	Ne
&pack	performs packing for the LUT mapped network	Ne
&satclp	performs SAT based collapsing	Ano
&satfx	performs SAT based shared logic extraction	Ano
&st	performs structural hashing	Ne
&syn2	performs AIG optimization	Ano
&syn3	performs AIG optimization	Ano
&syn4	performs AIG optimization	Ano
&synch2	computes structural choices using a new approach	Ano

K porovnání jsme výsledky optimalizace namapovali na buňky LUT se 6 vstupy a sledovali jejich počet. V případě, že

by algoritmy uměly reprezentaci s XORy využít, měl by být počet LUTů po namapování menší u originálního algoritmu, proti algoritmu s výše uvedenou modifikací.

Jak je vidět v Tabulce 3, mezi výkonem algoritmu &syn4 nad AIG a XAIG je velmi silná korelace. Tabulka ukazuje pouze podmnožinu zkoumaných benchmarků a jeden vybraný optimalizační algoritmus, výsledky dalších algoritmů a kompletní sady benchmarků jsou velmi podobné; korelace je srovnatelná – dostupné algoritmy tedy s XORy pravděpodobně správně pracovat neumí.

TABULKA 3: SROVNÁNÍ ALGORITMU &SYN4 NAD AIG A XAIG

		LUTs	
name	ANDs	AIG	XAIG
s38417	8166	2247	2284
des	4123	482	561
apex4	2587	686	692
bca	2292	805	811
C6288	2290	893	703
bcb	2066	719	704
bcc	1972	677	676
apex1	1842	605	606
apex3	1561	444	452
bcd	1424	497	482
C5315	1393	250	286
frg2	1164	184	178
Correlation			**0,993**

IV. ZÁVĚR A PRÁCE DO BUDOUCNA

Jelikož výsledky experimentu přesvědčivě ukázaly na to, že optimalizační algoritmy ABC nedokáží obecnou strukturu XAIG využít, rozhodli jsme se reimplementovat algoritmus rewrite [12], který pro novou reprezentaci nebyl vůbec k dispozici.

Algoritmus rewrite (algoritmus popsaný v Zdrojovém kódu 1) identifikuje v obvodu maximální k-feasible řezy (cuts), podgrafy takové, že z jejich kořene vedou všechny cesty k primárním vstupům přes právě *k* listů tohoto podgrafu [21]. Řezy jsou konstruovány rekurzivně pro každý uzel dle algoritmu viz Zdrojový kód 2. Pro každý z nalezených řezů je simulací získáno pravdivostní ohodnocení. Pro k = 4 existuje takových ohodnocení 2^{16}, avšak permutací a negací vstupů lze všechna ohodnocení převést do 224 NPN ekvivalentních tříd. Předpočítané třídy ekvivalence pro k = 4 již jsou v ABC implementovány pro rewrite algoritmus pracující nad AIG, který jsme použili. Pro každou třídu je předpočítána optimální implementace a řez je nahrazen právě tou funkčně odpovídající jeho třídě, s příslušnou permutací a negací listů. Vnitřní uzly řezu s výstupy vedoucími mimo řez jsou před náhradou zduplikovány. Změna je přijata v případě, že se její aplikací počet uzlů grafu zmenší.

```
Rewriting (network AIG, hash table
PrecomputedStructures, bool UseZeroCost)
{
  for each node N in the AIG in topological order
  {
    for each 4-input cut C of node N computed using
cut enumeration {
      F = Boolean function of N in terms of the
leaves of C
      PossibleStructures =
HashTableLookup(PrecomputedStructures, F);
      // find the best logic structure for rewriting
      BestS = NULL; BestGainGain = -1;
      for each structure S in PossibleStructures {
        NodesSaved = DereferenceNode(AIG, N);
        NodesAdded = ReferenceNode(AIG, S);
        Gain = NodesSaved - NodesAdded;
        Dereference(AIG, S); Reference(AIG, N);
        if (Gain > 0 || (Gain = 0 && UseZeroCost)){
          if (BestS = NULL || BestGain < Gain){
            BestS = S; BestGain = Gain;
          }
        }
      }
      if (BestS = NULL){
        continue;
      }
      // use the best logic structure to update the
netlist
      NodesSaved = Dereference(AIG, N);
      NodesAdded = ReferenceNode(AIG, N);
      assert (BestGain = NodesSaved - NodesAdded);
    }
  }
}
```

Zdrojový kód 1: algoritmus rewrite [12]

```
void NetworkKFeasibleCuts (Graph g, int k){
  for each primary output node n of g {
    NodeKFeasibleCuts(n, k);
  }
}

cutset NodeKFeasibleCuts (Node n, int k){
  if (n is primary input) return {{n}};
  if (n is visited) return NodeReadCutSet(n);
  mark n as visited;
  cutset Set1 = NodeKFeasibleCuts(NodeReadChild1(n),
k);
  cutset Set2 = NodeKFeasibleCuts(NodeReadChild2(n),
k);
  cutset Result = MergeCutSets(Set1, Set2, k) U {n};
  NodeWriteCutSet(n, Result);
  return Result;
}

cutset MergeCutSets (cutset Set1, cutset Set2, int
k){
  cutset Result = {};
  for each cut Cut1 in Set1{
    for each cut Cut2 in Set2{
      if (|Cut1 U Cut2| <= k){
        Result = Result U {Cut1 U Cut2};
      }
    }
  }
  result Result;
}
```

Zdrojový kód 2: hledání k-feasible řezů [21]

Optimální reprezentace řezů v XAIG jsme předpočítali načtením pravděpodobnostní tabulky pomocí příkazu *read_truth* do ABC, optimalizovány příkazem *dch* a následně namapovány příkazem *map* do standartních buněk knihovnou obsahující invertory s cenou 1, a hradla AND a XOR s cenou 2. Namapované netlisty byly následně převedeny do XAIG. Totožná cena ANDů a XORů umožní vytvořit každý identifikovaný XOR. Rozhodnutí o tom, jak výsledné XORy implementovat tak bude necháno na mapperu.

Implementaci rewrite nad XAIG bude následovat porovnání jeho výkonu s rewritingem nad AIG. Tento experiment by měl potvrdit to, že nad XAIG lze reimplementovat algoritmy pracující nad AIG a dosáhnout tak lepšího výkonu těchto algoritmů. V opačném případě by měl ukázat, že rewriting nelze zlepšit identifikací a využitím XORů a bude třeba hledat jinou cestu.

PODĚKOVÁNÍ

- GAČR: GA16-05179S Výzkum vztahů a společných vlastností spolehlivých a bezpečných architektur založených na programovatelných obvodech (Fault-Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features) (03/2016 - 12/2018)
- SGS16/121/OHK3/1T/18 (Bezpečné a spolehlivé architektury vhodné pro implementaci v FPGA)
- Výpočetní prostředky byly poskytnuty CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, pod programy "Projects of Large Research, Development, and Innovations Infrastructures"

REFERENCE

[1] E. McCluskey, “Minimization of boolean functions,” The Bell System Technical Journal, vol. 35, no. 6, pp. 1417–1444, Nov 1956.

[2] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, Logic Minimization Algorithms for VLSI Synthesis. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[3] S. Muroga, Y. Kambayashi, C. Lai, Hung, and N. Culliney, Jay, “The transduction method-design of logic networks based on permissible functions,” vol. 38, no. 10, pp. 1404–1424, 10 1989, iEEE Transactions on Computes

[4] S. B. Akers, “Binary decision diagrams,” IEEE Transactions on Computers, vol. 27, no. 6, pp. 509–516, Jun. 1978.

[5] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” IEEE Trans. Comput., vol. 35, no. 8, pp. 677–691, Aug. 1986.

[6] K. Karplus, “Using if-then-else DAGs for multi-level logic minimization,” in Proc. of Advance Research in VLSI, C. Seitz Ed. MIT Press, 1989, pp. 101–118.

[7] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula, “BDD based decomposition of logic functions with application to FPGA synthesis,” in Proceedings of the 30th International Design Automation Conference, ser. DAC’93. New York, NY, USA: ACM, 1993, pp. 642–647.

[8] C. Yang and M. Ciesielski, “BDS: a BDD-based logic optimization system,” vol. 21, no. 7, pp. 866–876, 08 2002, iEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[9] N. Vemuri, P. Kalla, and R. Tessier, “BDD-based logic synthesis for LUT-based FPGAs,” ACM Transactions on Design Automation of Electronic Systems, vol. 7, no. 4, pp. 501–525, 12 2001.

[10] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, “Robust boolean reasoning for equivalence checking and functional property verification,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 12, pp. 1377–1394, 12 2001.

[11] P. Bjesse and A. Borlv, “DAG-aware circuit compression for formal verification,” in IEEE/ACM International Conference on ComputerAided Design, 2004, pp. 42–49.

[12] K. Brayton, Robert, A. Mishchenko, and S. Chatterjee, “DAG-aware AIG rewriting: a fresh look at combinational logic synthesis,” in 43rd ACM/IEEE Design Automation Conference. ACM, 2006, pp. 532–535.

[13] A. Mishchenko et al., “ABC: A system for sequential synthesis and verification,” 2012. [Online]. Available: http://www.eecs.berkeley.edu/˜alanmi/abc

[14] R. Brayton and A. Mishchenko, “ABC: an academic industrial-strength verification tool,” in Proceedings of the 22Nd International Conference on Computer Aided Verification, ser. CAV’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40

[15] J. Cong and K. Minkovich, “Optimality study of logic synthesis for LUT-based FPGAs”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2007, Vol. 26, No. 2, pp. 230–239.

[16] P. Fišer and J. Schmidt, “Small But Nasty Logic Synthesis Examples”, Proceedings of the 8th. Int. Workshop on Boolean Problems, 2008, Freiberg, pp. 183–189.

[17] F. Brglez and H. Fujiwara, “A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran,” in Proceedings of IEEE International Symposium Circuits and Systems (ISCAS 85). IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[18] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in IEEE International Symposium on Circuits and Systems, 1989., May 1989, pp. 1929–1934 vol.3.

[19] S. Yang, “Logic synthesis and optimization benchmarks user guide: Version 3.0,” MCNC Technical Report, Tech. Rep., Jan. 1991.

[20] K. McElvain, “IWLS’93 Benchmark Set: Version 4.0,” Tech. Rep., May 1993.

[21] A. Mishchenko et al. Technology Mapping with Boolean Matching, Supergates and Choices, ERL Technical Report, EECS Dept., UC Berkeley, 2005.

[22] L. Amarù, P.-E. Gaillardon, G. De Micheli, “Biconditional Binary Decision Diagrams: A Novel Canonical Representation Form”, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), Vol. 4, No. 4, 2014, pp. 487-500.

[23] L. Amarù, P.-E. Gaillardon, G. De Micheli, “Boolean Logic Optimization in Majority-Inverter Graphs”, Design Automation Conference (DAC), San Francisco, CA, USA, 2015.