Logická syntéza založená na obecných operátorech

Ivo Háleček
2. ročník, prezenční studium
školitel: Petr Fišer, školitel specialista: Jan Schmidt

Fakulta informačních technologií, ČVUT v Praze
Thákurova 9
halecivo@fit.cvut.cz

***Abstrakt*—V článku je představena nová 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 dvouvstupá hradla AND či XOR a hrany mohou být negované. Nad reprezentací XAIG byl reimplementován syntézní algoritmus rewrite pro nástroj ABC. Reprezentace s více typy uzlů přinesla nová rozhodnutí, která algoritmus musí činit, proto je možné algoritmus ovlivňovat několika parametry, jejichž vliv jsme porovnali na sadě benchmarků. Výsledky dosavadní práce také ukazují, že nový algoritmus je schopen detekovat více XORů než původní algoritmus, a přestože celkově lepší výsledky přináší jen pro podmnožinu testovaných obvodů, možnosti k vylepšení algoritmu jsou známy a shrnuty v závěru článku.**

***Keywords*—AIG, XAIG, rewriting, logická syntéza, ABC**

I. Úvod

Zlepšování logické syntézy bylo v minulých dekádách považováno za dobře řešené téma, v posledních letech však výzkum ukázal, že pro některé struktury není současná logická syntéza schopná produkovat optimální výsledky.

Syntézní nástroje původně reprezentovaly obvody pomocí sumy logických součinů literálů (Sum-of-products - SOP) [1], [2], případně sítí tvořenou z uzlů představujících SOP.

Velký průlom přinesla reprezentace funkcí pomocí binárních rozhodovacích diagramů (Binary decision diagrams - BDD) [3], [4]. Úprava syntézních nástrojů pro tyto struktury vedla k zlepšení jejich výkonu [5], [6], [7], [8].

Binární rozhodovací diagramy trpí špatnou škálovatelností - struktura může růst exponenciálně s počtem vstupů, v závislosti na pořadí vstupních proměnných. Snaha o řešení tohoto problému vedla k fixnímu pořadí proměnných, avšak tím se jen problém posunul k hledání optimálního fixního pořadí vstupních proměnných.

Z těchto důvodů vznikla nová velmi efektivní reprezentace obvodů - And-Inverter-Grafy (AIG) [9], [10], [11]. V AIG je obvod reprezentován pomocí dvouvstupých hradel AND a propojení, která mohou být negovaná. Mnoho algoritmů založených na AIG bylo implementováno do moderního akademického nástroje pro logickou syntézu a verifikaci, ABC [12]. Reprezentace pomocí AIG pravděpodobně je, či brzo bude integrována do komerčních nástrojů [13].

II. Teoretické podklady

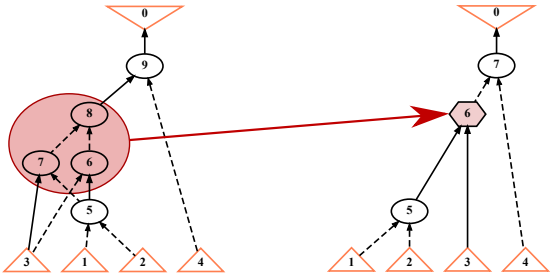
AIG je orientovaný acyklický graf, kde vnitřní uzly představují dvou-vstupá hradla AND a hrany mohou být negované. Kořeny představují výstupy obvodu, listy grafu představují vstupy.

Pro zajištění co nejmenší velikosti logické struktury je důležitá redukce ekvivalentních struktur. Ekvivalence může být buď strukturní nebo funkční. Strukturní ekvivalence znamená, že se v grafu vyskutují dvě stejné struktury reprezentující stejnou funkci primárních vstupů obvodu. Funkční ekvivalence znamená, že se v grafu vyskytují různé struktury, které ale stále reprezentují stejnou funkci primárních vstupů. V ABC je strukturní ekvivalence řešena automaticky při konstrukci AIG pomocí strukturního hashování. Je tedy zajištěno, že v grafu nejsou dvě ekvivalentní struktury. Stále však v grafu mohou být dvě různé struktury reprezentující stejnou funkci.

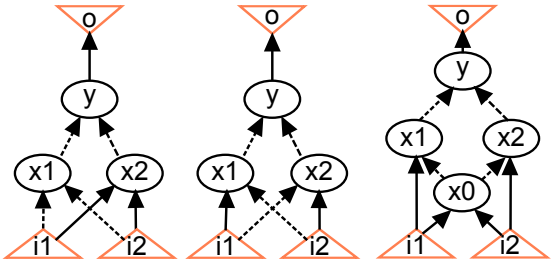
Rewriting [11] je algoritmus, který nahrazuje podgrafy s k listy (tzv. k-fesible CUTs [14]) funkčně ekvivalentními strukturami, s cílem zmenšit celkový počet uzlů grafu. V ABC je rewriting dostupný pro reprezentaci AIG a pracuje s podgrafy o 4 listech. Pro každý uzel jsou enumerovány všechny podgrafy s k listy. Každý takový podgraf je vyjádřen jako funkce jeho listů, reprezentována pravdivostní tabulkou. Tato funkce je následně převedena do kanonického tvaru na funkci, která je NPN-ekvivalentní (převoditelná na původní funkci pomocí permutací a negací vstupů a případné negace výstupu). Pro každou NPN třídu funkcí je k předpočítána jedna nebo více nahrazovacích struktur (replacement structure). Pro $k = 4$ existuje 2^{16} různých funkcí, avšak jen 222 NPN tříd, proto je $k = 4$ vhodný kompromis mezi paměťovou náročností a sílou rewritingu. Aplikováno je nahrazení takovou strukturou, která ušetří co nejvíce uzlů (pokud nějaká vůbec uzly ušetří). Toho se může dosáhnout tak, že je buď nahrazovací struktura přímo menší než nahrazovaná, nebo nový podgraf obsahuje strukturně ekvivalentní části obsažené již ve zbytku grafu a uzly se tak ušetří strukturním hashováním.

III. Popis problému

Přesto, že AIG jsou logickou reprezentací s dobrou škálovatelností, výzkum zaměřený na konstrukci benchmarků pro syntézu FPGA [15] odhalil, že pro některé obvody funguje současná logická syntéza neefektivně, produkuje řádově větší obvody, než optimální.



Obrázek 1: Funkce $Y = \neg(\neg(x_1 \vee x_2) \oplus x3) \wedge \neg x_4$ reprezentovaná v AIG (vlevo) a XAIG (vpravo). Oválné uzly představují funkci AND, šestiúhelníky uzly typu XOR. Čárkované hrany jsou negovány.



Obrázek 2: Funkce XOR reprezentována v AIG.

Pozdější výzkum také ukázal, že exisují dokonce i velmi malé obvody, pro které logická syntéza takto selhává [16], také byl představen způsob, jak vytvořit takové obvody z reálných praktických (průmyslových) obvodů [17], [18].

Jednou ze společných charakteristik těchto obvodů je vysoká intenzita XORů, především stromů XORů, které se staly běžnými s rozvojem aritmetických jednotek a systémů odolných proti poruchám. Hlavním problémem pro současné syntézní nástroje se zdá být, pokud tyto struktury nejsou dostatečně dobře popsané - například pokud jsou rozpuštěny do dvouúrovňové struktury - obvykle SOP.

V nedávné době se objevily nové reprezentace obvodů snažící se neefektivitu algoritmů postavených nad dosavadními reprezentacemi řešit. Jako alternativa k BDD vznikly Bikondicionální binární rozhodovací diagramy (BBDDs) [19], kde uzly jsou multiplexory řízené rovností dvou řídicích signálů. Nad AIG vznikla reprezentace Majority-Inverter-Grafů (MIG) [20], kde uzly představující dvouvstupý AND byly nahrazeny třívstupými majoritami. Uzly typu AND, resp. OR lze jednoduše pomocí uzlu majorita (MAJ) vyjádřit přivedením konstanty *0*, resp. *1* na jeho třetí vstup.

Velmi novou reprezentací jsou také XOR-Majority-Grafy (XMG) [21]. Zde je graf tvořen uzly dvou typů - MAJ a XOR.

IV. Navrhované řešení

Jedním ze způsobů, jak se pokusit zlepšit výsledky syntézy, je přepsat současné syntézní algoritmy tak, aby pracovaly s obecnější reprezentací obvodů. K řešení konkrétně problému s XOR intenzivními strukturami, navrhujeme syntézu postavenou nad reprezentací XAIG.

Xor-And-Inverter-Grafy (XAIG) je námi představená reprezentace, která rozšiřuje původní AIG o dvou-vstupé uzly typu XOR. Ukázka funkce $Y = \neg(\neg(x_1 \vee x_2) \oplus x3) \wedge \neg x_4$ v AIG a XAIG je vidět na Obrázku 1.

V původním AIG bylo nutné funkci XOR reprezentovat pomocí uzlů AND, jak je zobrazeno na Obrázku 2. Takto popsané XORy lze v AIG dobře strukturně identifikovat. V XAIG je XOR reprezentován pomocí speciálního typu uzlu.

Jako základ syntézního procesu nad XAIG jsem reimplementoval rewriting tak, aby pracoval nad XAIG. Algoritmus jsem implementoval stejným způsobem jako originální rewriting s tím rozdílem, že nahrazovací struktury byly reprezentovány pomocí XAIG (obsahovaly tedy nativní XORy) a generoval jsem pouze jednu nahrazovací strukturu pro každou NPN třídu.

Nahrazovací struktury XAIG jsem předpočítal načtením pravděpodobnostní tabulky pomocí příkazu *read_truth* do ABC, následně byly optimalizovány příkazem *dch* a namapovány příkazem *map* do standardních buněk s 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 místo toho, aby v některém případě byl realizován pomocí ANDů.

Různé typy uzlů přinesly algoritmu nová nutná rozhodnutí. Pokud víme, že pro cílovou technologii má pro nás XOR jinou cenu než AND (mapujeme-li do standardních buněk místo do FPGA), může pro nás být výhodnější vytvořit více ANDů místo méně XORů. Proto jsem do XAIG rewritingu implementoval konfigurovatelnou cenu ANDů a XORů a rewriting nerozhoduje o výhodnosti náhrady podgrafu na základě ušetřených uzlů, ale na základě změny celkové ceny uzlů grafu. To umožňuje řídit preferenci XORů nad ANDy.

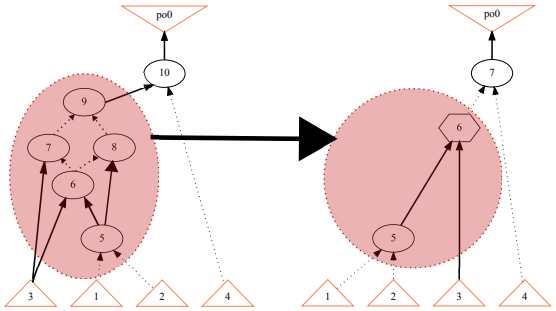
Pokud jsou v grafu ANDy tvořící XOR, ale zároveň jeden z vnitřních ANDů je vstup pro jinou část obvodu, musí se při nahrazení této struktury nativním XORem vnitřní uzel AND zduplikovat, aby nebyl ztracen onen výstup. Takovým XORům říkáme “non fanout-free". Někdy může být proto výhodné nativní XOR nevytvářet. V XAIG rewritingu je toto možné řídit povolením rozpouštění XORů, XOR je nahrazen ANDy, pokud to díky strukturnímu sdílení s ostatní částí grafu vede k menší celkové ceně uzlů.

Příklad jednoho nahrazení v rewritingu nad XAIG je zobrazen na obrázku 3.

V. Předběžné výsledky

Rewriting implementovaný nad XAIG jsem porovnal s původním AIG rewritingem. Zajímal nás především počet identifikovaných XORů a optimalizační výkon algoritmu.

Srovnání algoritmů z hlediska identifikace XORů je zobrazeno v Tabulce I. Sloupce obsahují počet XORů nalezených strukturní identifikací v původním obvodu, po XAIG rewritingu pro různý poměr cen uzlu AND a XOR, a po původním AIG rewritingu následovaným strukturní identifikací. Řádek *celkem* obsahuje sumu dané charakteristiky přes



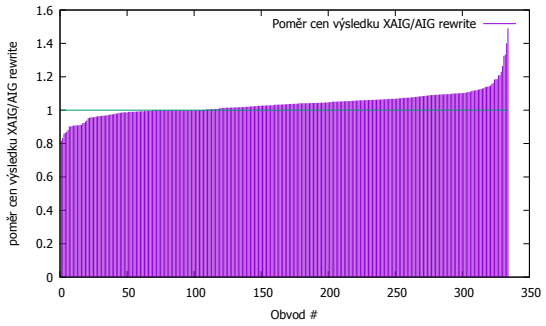
Obrázek 3: Příklad jednoho kroku rewritingu nad XAIG.

Tabulka I: Srovnání identifikace XORů původním AIG rewritingem s naším XAIG rewritingem, a současně se strukturní identifikací v původních obvodech.

obvod	původní	XAIG (cena AND:XOR) 1:1	1:3	2:5	AIG
c6288	0	476	0	433	28
bigkey	5	109	4	4	5
mm30a	60	116	2	30	58
c1355	0	107	82	104	56
prom2	27	67	34	46	32
s635	0	31	0	2	0
g25	50	50	27	27	30
ex1010	43	62	41	51	46
mm9a	18	32	2	10	16
Altera_oc_hdlc	132	149	117	141	134
Mentor_1_11	3	62	50	56	47
Mentor_1_12	3	62	50	56	47
mm9b	19	32	3	11	17
celkem	**5289**	**6632**	**4588**	**5629**	**5345**
více než v AIG		123	25	56	
méně než v AIG		19	146	102	

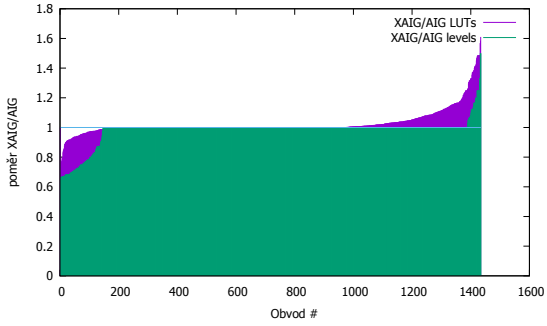
všechny měřené obvody. Poslední dva řádky uvádějí, pro kolik obvodů byla daná charakteristika vyšší resp. nižší po rewritingu nad XAIG proti původnímu rewritingu nad AIG. Obě verze algoritmu rewriting dokáží u většiny obvodů detekovat více XORů, než strukturní identifikace v původním obvodu. Ve většině případů je XAIG rewriting z hlediska identifikace silnější než AIG rewriting, s klesající nastavenou cenou XORů pochopitelně počet nalezených XORů roste. Měření bylo provedeno nad více než 1 400 obvody ze sad benchmarků LGSynth‘91 [22], IWLS‘93 [23], ISCAS‘85 [24], ISCAS‘89 [25], Advanced Synthesis Cookbook [26], IWLS 2005 [27] a dalšími [28], [29], a [30] - dostupnými z [31].

Pro základní srovnání z hlediska redukce velikosti výsledného grafu jsem u obou algoritmů spočítal výslednou cenu uzlů jako součet počtu uzlů typu AND a trojnásobku počtu uzlů typu XOR. Toto ocenění vychází ze skutečnosti, že každý XOR lze vyjádřit pomocí maximálně tří uzlů typu AND. Srovnání je zobrazeno v grafu 4, kde jsou ve vzestupném pořadí vyneseny poměry výsledné ceny grafu pro obě verze algoritmu rewriting. Hodnoty nižší než 1 znamenají nižší cenu pro XAIG rewriting. Je však nutno zmínit, že vyšší cena grafu neznamená automaticky větší plochu po mapování na technologii, mapovací nástroj může lépe zpracovat větší, zato



Obrázek 4: Porovnání rewritingu nad AIG vůči rewritingu nad XAIG z hlediska celkové ceny uzlů obvodu.

vhodněji strukturovaný graf.



Obrázek 5: Porovnání rewritingu nad AIG vůči rewritingu nad XAIG z hlediska zpoždění a počtu LUTů po mapování na FPGA.

Pro srovnání rewritingu nad AIG a XAIG z hlediska optimalizační síly jsem výsledky obou algoritmů namapoval na 6-LUTy a porovnal zpoždění a počet LUTů. Cena uzlů v XAIG rewritingu byla nastavena 1:1, jelikož v FPGA má pro nás XOR stejnou cenu jako AND, poměr mezi výsledkem XAIG rewritingu a AIG rewritingu je pro obě charakteristiky zobrazen v grafu na Obrázku 5. Z výsledků je vidět, že XAIG rewriting je z celkového pohledu lepší v optimalizaci zpoždění, pro některé obvody je lepší i z hlediska minimalizace počtu LUTů.

VI. Závěr a práce do budoucna

Rozšířil jsem původní reprezentaci logických funkcí AIG o XOR uzly a vytvořil tak reprezentaci XAIG. Nad toutou reprezentací jsem reimplementoval jednoduchý syntézní algoritmus rewriting. Jelikož nový typ uzlů přinesl do algoritmu nová rozhodnutí, umožnil jsem kontrolu průběhu algoritmu parametrizovatelnou preferencí XORů a možností XORy rozpouštět do ANDů, pokud je to výhodné. XAIG rewriting jsem srovnal s původním AIG rewritingem z hlediska identifikace XORů a z hlediska výsledného zpoždění a počtu LUTů po mapování na FPGA. XAIG rewriting je silnější v identifikaci XORů, celkově je silnější z hlediska zpoždění po mapování a pro některé obvody je silnější i z hlediska minimalizace počtu LUTů.

XAIG rewriting ale v současné předběžné verzi trpí nahrazovacími strukturami. Odhalili jsme, že ne všechny naším způsobem generované struktury jsou optimální, navíc používáním pouze jedné struktury pro každou z NPN tříd vede k menší redukci uzlů díky strukturnímu sdílení. Proto jsme pro budoucí experimenty připravili struktury generované exaktní syntézou a algoritmus jsem upravil tak, aby pro každou z NPN tříd zkoušel nahrazení každou možnou optimální nahrazovací strukturou. Nová verze algoritmu však byla vytvořena až po odeslání tohoto článku.

Z dlouhodobějšího zaměření je možnost zobecnit syntézní proces na rozdělení do nezávislých vrstev tak, že bude možné kombinovat různé reprezentace (více typů uzlů), různé způsoby generování podgrafů (například vícevýstupové funkce) a různé nahrazovací struktury (předpočítané, případně generované online).

Cílem dizertační práce bude syntézní proces, který bude schopný efektivně pracovat se strukturami, kde je současná syntéza neoptimální. Výsledkem tak nebude jen nová verze algoritmu rewriting, ale zaměřím se také na identifikaci struktur, pro které samotný rewriting není schopný produkovat dostatečně dobrá řešení a hledání dalších způsobů, jak s těmito strukturami efektivně pracovat. Pravděpodobně bude také nutné upravit další syntézní algoritmy. Jak už ale ukázal rewriting postavený nad XAIG, obvykle nestačí pouze nahradit strukturu, nad kterou algoritmus pracuje, ale řešit i nová rozhodnutí, která s novými typy uzlů nastanou.

Poděkování

Práce byla částečně podpořena z grantů:

- GAČR: GA16-05179S Výzkum vztahů a společných vlastností spolehlivých a bezpečných architektur založených na programovatelných obvodech (03/2016 - 12/2018).
- SGS17/213/OHK3/3T/18: Bezpečné a spolehlivé architektury pro programovatelné obvody.

Výpočetní prostředky byly poskytnuty CESNET LM2015042 a 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. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, Jun. 1978.

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[5] 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.

[6] 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 (DAC'93)*. New York, NY, USA: ACM, 1993, pp. 642–647.

[7] C. Yang and M. Ciesielski, "BDS: A BDD-Based Logic Optimization System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 866–876, August 2002.

[8] 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.

[9] 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, 2001.

[10] P. Bjesse and A. Borälv, "DAG-aware circuit compression for formal verification," in *IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 42–49.

[11] 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.

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

[13] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-strength Verification Tool," in *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10)*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 24–40.

[14] A. Mishchenko, S. Chatterjee, K. Brayton, Robert, X. Wang, and T. Kam, "Technology Mapping with Boolean Matching, Supergates and Choices," ERL Technical Report, EECS Dept., UC Berkeley, Tech. Rep., 03 2005.

[15] J. "Cong and K. Minkovich, ""Optimality Study of Logic Synthesis for LUT-Based FPGAs"," in *"14th International ACM Symposium on Field-Programmable Gate Arrays"*, "2006", pp. "33–40".

[16] P. Fiser and J. Schmidt, "Small but Nasty Logic Synthesis Examples," in *8th. Int. Workshop on Boolean Problems (IWSBP)*, 2008, pp. 183–189.

[17] J. Schmidt and P. Fiser, "The Case for a Balanced Decomposition Process," in *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, Aug 2009, pp. 601–604.

[18] P. Fiser and J. Schmidt, "New Ways of Generating Large Realistic Benchmarks for Testing Synthesis Tools," in *9th. Int. Workshop on Boolean Problems (IWSBP)*, 2010, pp. 157–164.

[19] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Biconditional Binary Decision Diagrams: A Novel Canonical Logic Representation Form," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 4, no. 4, pp. 487–500, Dec 2014.

[20] ——, "Boolean logic optimization in Majority-Inverter Graphs," in *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[21] W. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, G. De Micheli, "A Novel Basis for Logic Rewriting," Integrated Systems Laboratory, EPFL, Lausanne, Switzerland, Tech. Rep., 2017.

[22] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0," MCNC Technical Report, Tech. Rep., Jan. 1991.

[23] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0," Tech. Rep., May 1993.

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

[25] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS'89)*, May 1989, pp. 1929–1934 vol.3.

[26] Altera, "Advanced Synthesis Cookbook," Tech. Rep., Jul. 2011.

[27] C. Albrecht, "IWLS 2005 Benchmarks," Tech. Rep., Jun. 2005.

[28] ——, "Altera's Quartus University Interface Program (QUIP)," Tech. Rep., Jun. 2005.

[29] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0," MCNC Technical Report, Tech. Rep., 1989.

[30] "Berkeley PLA Test Set Results," Tech. Rep., Jun. 1986.

[31] P. Fiser and J. Schmidt, "A Comprehensive Set of Logic Synthesis and Optimization Examples," in *12th. Int. Workshop on Boolean Problems (IWSBP)*, 2016, pp. 151–158. [Online]. Available: http://ddd.fit.cvut.cz/prj/Benchmarks/