Tehnološko mapiranje

Stefan Nikolić

Departman za matematiku i informatiku
Prirodno-matematički fakultet, Novi Sad

06.11.2024.

Šta je tehnološko mapiranje?

Kao rezultat logičke sinteze, dobijamo minimizovan graf u kom svaki čvor predstavlja neku osnovnu Bulovu funkciju (na primer AND2 u slučaju AIG-a)

Međutim, fizičke kapije koje imamo na raspolaganju najčešće mogu da realizuju i druge funkcije (uključujući i sve sa manje od k ulaza, u slučaju lukap tabela), a njihova upotreba može dovesti do bolje implementacije

Zadatak tehnološkog mapiranja (eng. **TECHNOLOGY MAPPING** ili **LIBRARY BINDING**) je da kolo dobijeno logičkom sintezom predstavi kao graf u kom svaki čvor implementira funkciju jedne od raspoloživih ćelija

Prvi pristupi: primena pravila (eng. RULE-BASED MAPPERS)

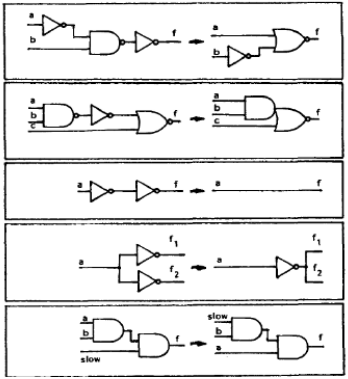
SOCRATES: A SYSTEM FOR AUTOMATICALLY SYNTHESIZING AND OPTIMIZING COMBINATIONAL LOGIC

David Gregory*, Karen Bartlett**, Aart de Geus*, and Gary Hachtel**

*GE Calma Company Research Triangle Park, North Carolina
**Department of Electrical Engineering University of Colorado at Boulder

Circuit Optimization

The *Circuit optimizer* improves measurable circuit characteristics by iteratively replacing and rearranging small portions of a circuit. The program uses a library which describes alternative circuit implementations in a rule (*if* antecedent *then* consequent) form. The antecedent portion of the rule lists conditions that must hold before the alternative is valid, and the consequent lists actions which implement the alternative. New rules can be generated automatically from netlists of two alternatives. Figure 3 shows some example rules.



Strukturalni maperi: patern mečing u grafovima

DAGON: Technology Binding and Local Optimization by DAG Matching

Kurt Keutzer
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

† We begin our treatment of the problem of technology binding with local optimizations by outlining a formalization of it. Using this formalization we will find that the problem is related to those that have been encountered in the field of programming language compilers. Indeed, we claim that compiler techniques are highly relevant to many problems in logic synthesis. This thesis was originally stated in [Jo82] and in [TJB86]. In particular here we claim that technology binding for logic synthesis is a very closely related problem to code generation for programming language compilers. More specifically, matching a graph-like description of a technology independent circuit against a library of patterns in a technology, such as a standard cell library, is similar to matching a graph-like intermediate representation of a computer program against the patterns of an instruction set of a given machine. Thus *twig* [Tj86], a tree manipulator used for constructing code generators for programming language compilers is used to build an optimizing technology binder. The result is a technology binder, *DAGON*, that is capable of optimizing for time, area or a function of both.

Osnovni principi koje je uveo DAGON

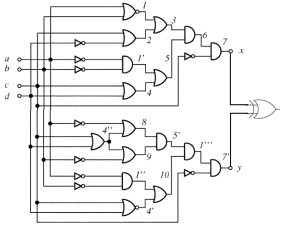
1. Svaku funkciju biblioteke predstavimo kao niz paterna koje tražimo u kolu
2. U svakom čvoru kola, mečujemo sve paterne koje možemo
3. Upotrebom dinamičkog programiranja, vršimo izbor optimalnih mečeva

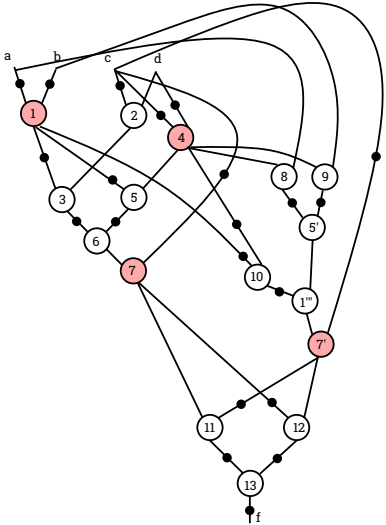
Problemi i rešenja

U opštim DAG-ovima, patern mečing je težak, ali za stabla je lak $\implies$ particionišemo graf kola tako da dobijemo šumu stabala

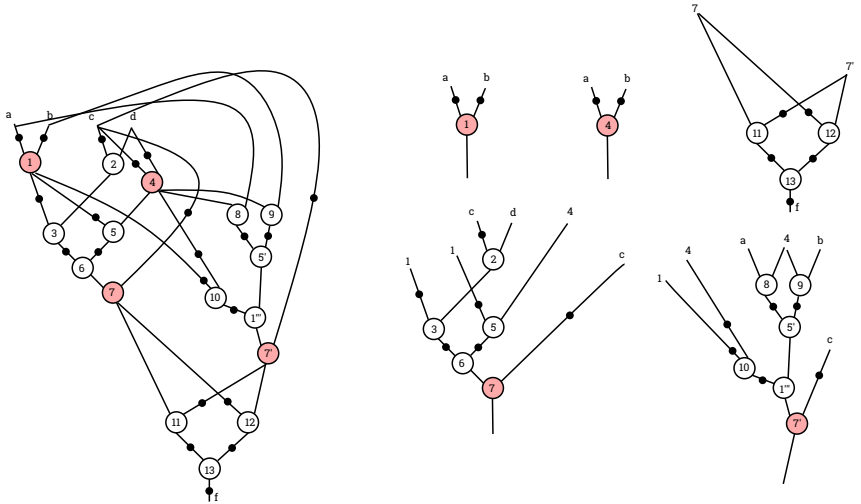
U opštim DAG-ovima, dinamičko programiranje ne garantuje minimalno rešenje, osim za neke specifične funkcije cene $\implies$ u stablima garantuje, tako da prethodni vid particionisanja dovodi do lokalnog optimuma

Primer: svaki čvor sa više od jednog deteta postaje koren jednog stabla u šumi





Primer: svaki čvor sa više od jednog deteta postaje koren jednog stabla u šumi

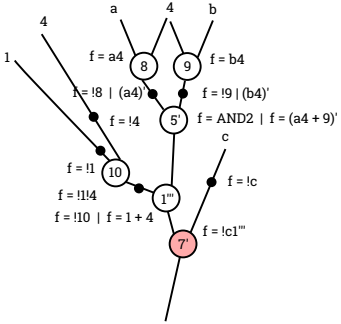


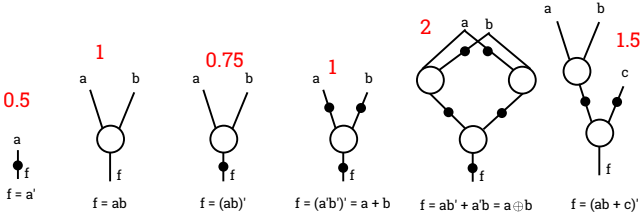
Leaf DAG

Primetimo da ulazi u particiju u nekim slučajevima imaju više od jednog deteta. Takva struktura nije stablo, nego lisni DAG.

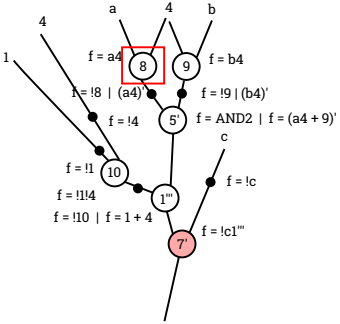
Međutim, pošto je dupliranje ulaza moguće izvršiti prilikom rutiranja, maper takve grafove tretira kao stabla

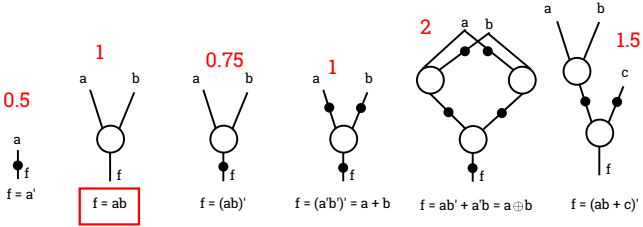
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



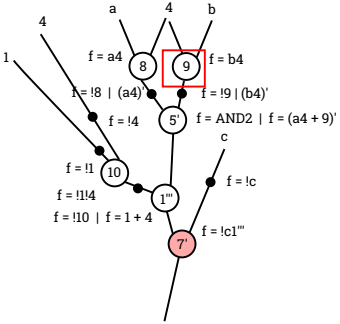


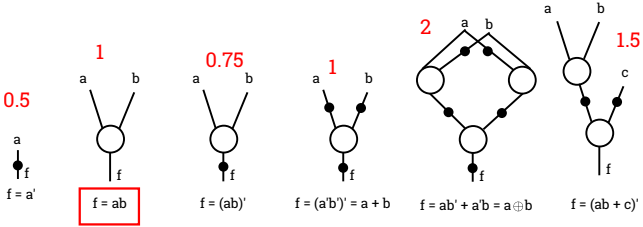
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



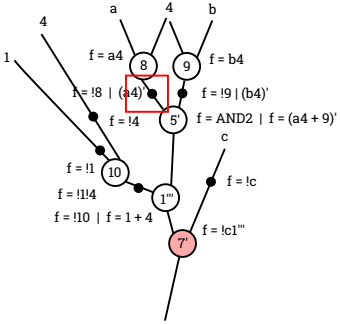


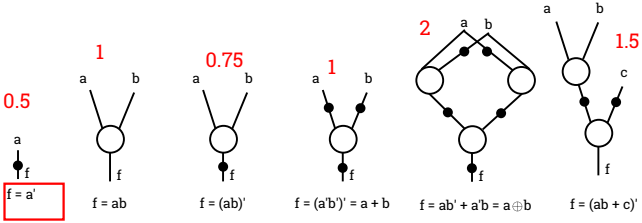
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



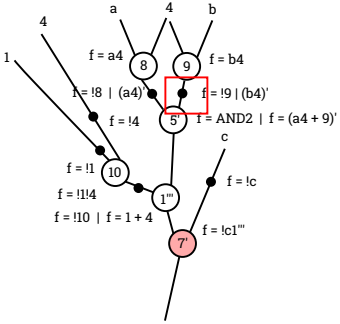


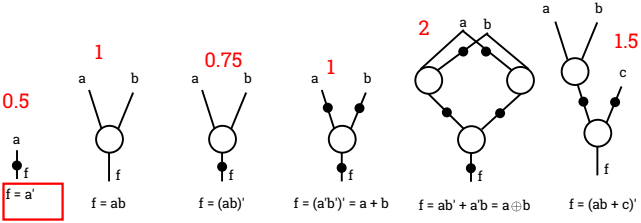
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



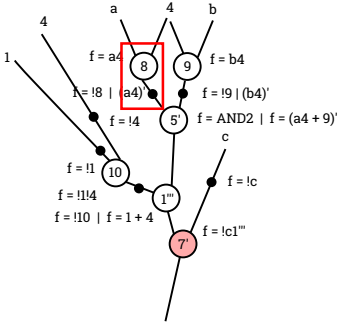


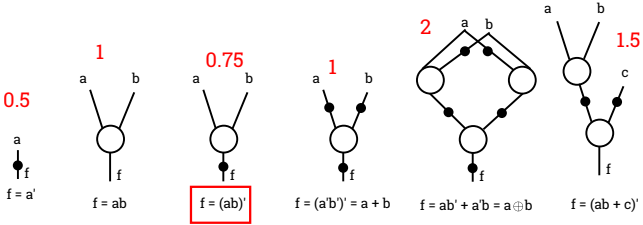
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



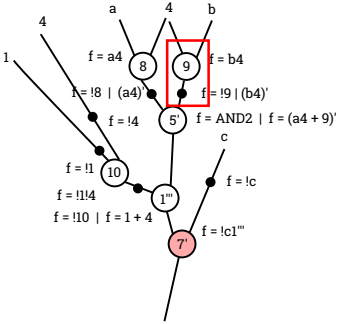


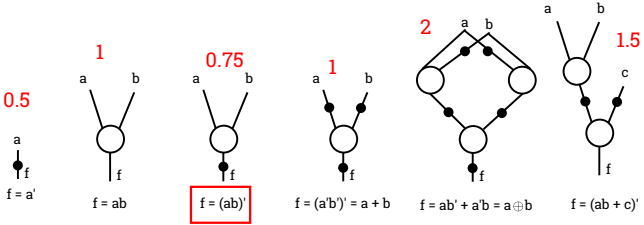
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



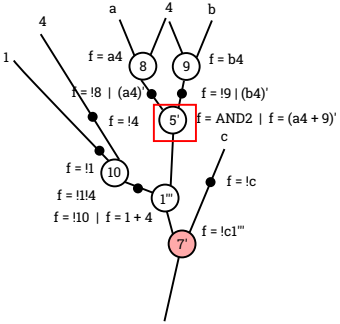


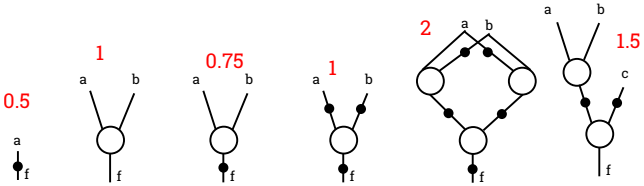
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve



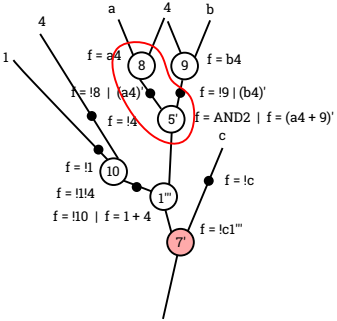


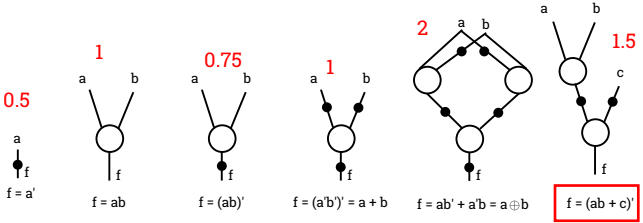
Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve





Primer: u svakom čvoru tražimo sve moguće ukorenjene mečeve

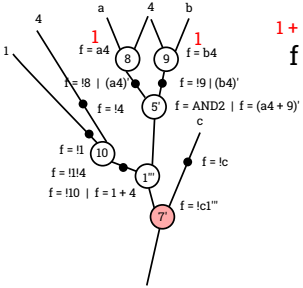




Određivanje najpovoljnijeg meča

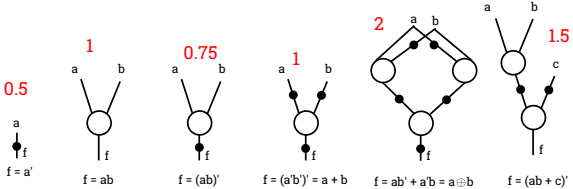
S obzirom na to da je deo kola predstavljen kao stablo, izlaz svakog čvora je korišćen u jedinstvenom nizu sledbenika. Ako prilikom pokrivanja sledbenika pokrijemo i prethodnika, njega možemo ukloniti iz kola. Iz tog razloga cenu pokrivanja svakog čvora računamo kao cenu kapije čiji smo meč identifikovali sabranu sa cenama minimalnih mečeva u čvorovima koji predstavljaju ulaz u dati meč. Obilazeći graf kola u topološkom poretku možemo odrediti najpovoljnije mečeve za svaki čvor u jednom obilasku.

Primer: u topološkom poretku određujemo najpovoljniji meč za svaki čvor



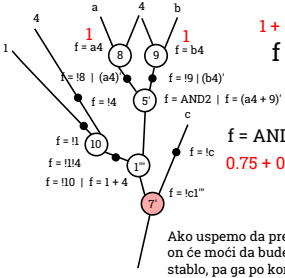
1 + 0.5 = 1.5 | 0.75

f = !9 | (b4)'



Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

Primer: u topološkom poretku određujemo najpovoljniji meč za svaki čvor



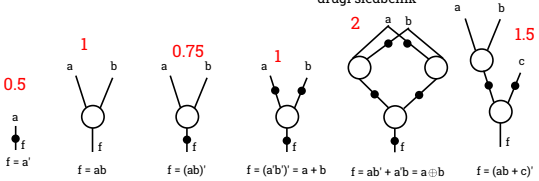
1 + 0.5 = 1.5 | 0.75

f = !9 | (b4)'

f = AND2 | f = (a4 + 9)'

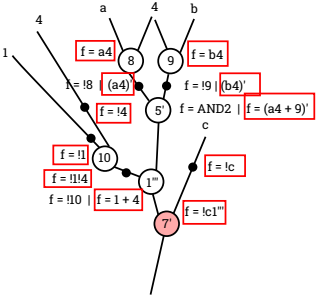
0.75 + 0.75 + 1 = 2.5 | 1 + 1.5 = 2.5

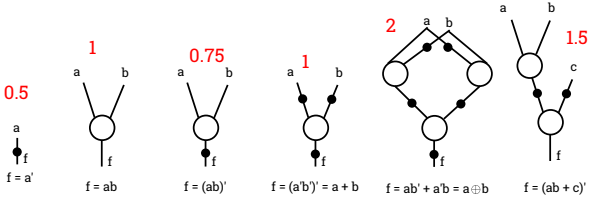
Ako uspemo da prekrijemo nekog prethodnika, on će moći da bude uklonjen iz kola, jer je u pitanju stablo, pa ga po konstrukciji ne zahteva ni jedan drugi sledbenik



Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

Primer: u topološkom poretku određujemo najpovoljniji meč za svaki čvor



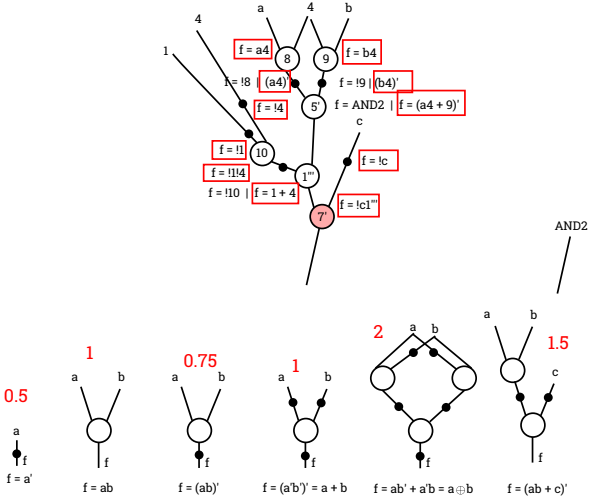


Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

Generisanje mapiranog kola

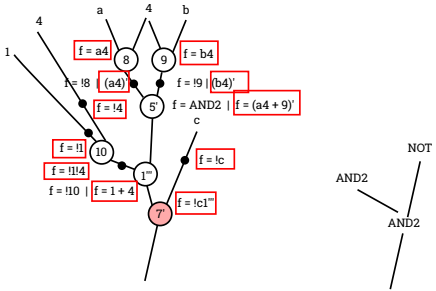
Kada smo odredili najpovoljnije mečeve, u obrnutom topološkom poretku generišemo ćeliju po ćeliju, preskačući one čvorove koji su već pokriveni prilikom pokrivanja nekog sledbenika

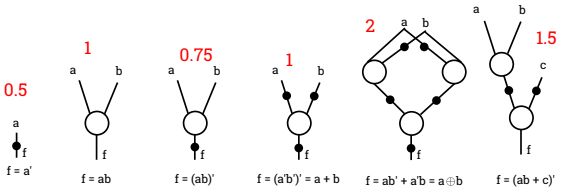
U našem primeru



Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

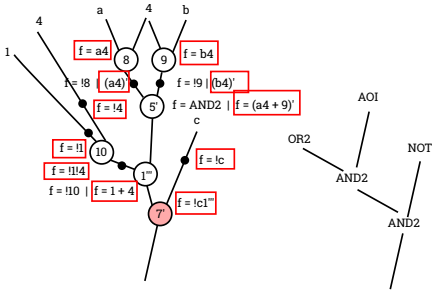
U našem primeru

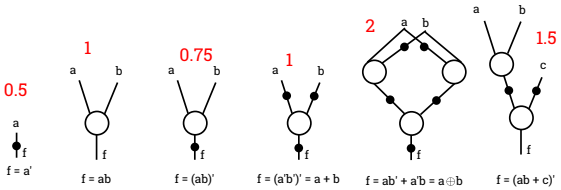




Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

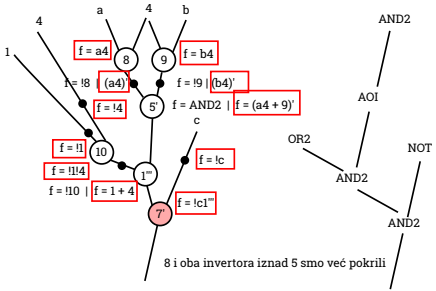
U našem primeru

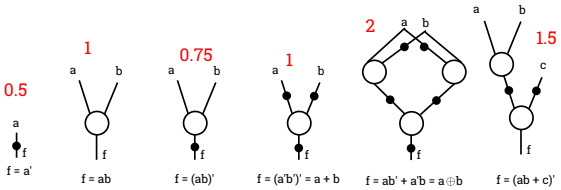




Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

U našem primeru





Biblioteka ćelija: u opštem slučaju, jedna funkcija može imati više različitih patern grafova

Ključne mane ovog algoritma

1. Particionisanje DAG-a u stabla
2. Zavisnost od izabranih paterna za datu funkciju (jer za neke ne možemo da izgenerišemo sve)

Zavisnost od strukture kola, ali i paterna nazivamo **STRUKTURNOM PRISTRASNOŠĆU** (eng. **STRUCTURAL BIAS**)

Međutim

DAGON: Technology Binding and Local Optimization by DAG Matching

Kurt Keutzer
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

3.3 Quality of Tree Matching

Matching against a forest of trees rather than against the original DAG loses the guarantee of a globally optimal solution. But as mentioned above, finding a globally optimal solution to DAG matching or covering is an NP-complete problem. However, the simple observation that optimal tree matching is realizable in linear time and optimal DAG matching is NP-Complete shows that there is a considerable range between these two classes of problems. There is a very general optimization question involved as to whether it is better to optimally solve a subproblem or to suboptimally the whole problem. Greedy approaches to the problem of covering DAG's are considered in [AJU77] and are shown to be very suboptimal. A bottom-up greedy approach is used to match against high-level modules in the silicon compiler described in [Ka86]. In the following section we discuss our own experience with the limitations of tree matching as well as our answers to them.

Zaključak

- Inspiraciju za rešenje problema možemo naći u rešenju srodnog problema koji se pojavljuje u nekoj drugoj oblasti (u ovom slučaju u programskim prevodiocima)
- Ako imamo na raspolaganju optimalan algoritam za neku specifičnu klasu instanci problema, polazni problem možemo rešiti na zadovoljavajući način ako ga podelimo na potprobleme koji pripadaju datoj klasi ili čak i bez particionisanja ako pretpostavimo da je struktura stvarnih instanci dovoljno blizu datoj klasi i ako dati algoritam može da prihvati i instance van klase, ali bez garancija optimalnosti

Tehnološko mapiranje za FPGA

Kako bismo mogli da izvršimo tehnološko mapiranje na LUT-ove?

Naivni pristup

Izgenerišemo paterne za sve funkcije koje LUT može da implementira i koristimo DAGON

Koji je problem sa ovim pristupom?

Koji je problem sa ovim pristupom?

Broj funkcija koje može da implementira K-LUT je 2^{2^K}

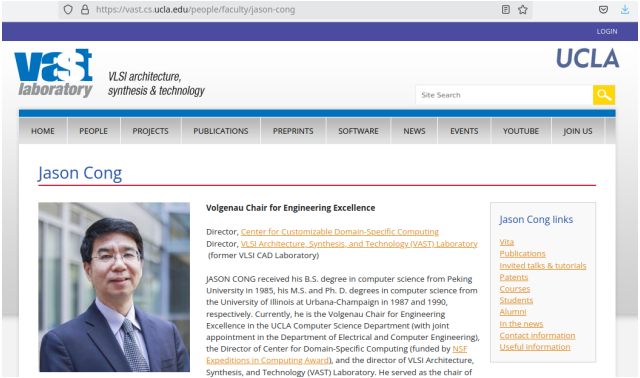
Za $K = 3$ bi ovaj pristup još i radio, jer 256 otprilike odgovara veličini neke biblioteke standardnih ćelija, ali je K u praksi 4–6, te bi tu biblioteka paterna bila daleko prevelika

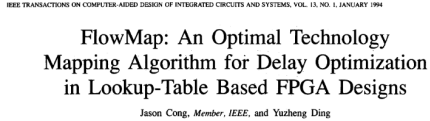
Zadatak

Potrebno je da nađemo mapiranje AIG-a na LUT-ove takvo da je broj LUT-ova na najdužoj putanji od nekog PI do nekog PO minimalan

To nazivamo **DEPTH-OPTIMAL LUT MAPPING** i to je jedan od retkih (verovatno jedini) problem u FPGA CAD-u za koji imamo optimalan polinomijalan algoritam

FTJP istraživanje




IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 13, NO. 1, JANUARY 1994 1

FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs

Jason Cong, *Member, IEEE*, and Yuzheng Ding

Dubina preseka

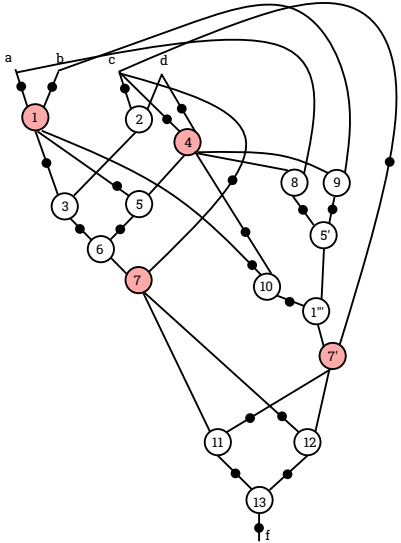
Neka svaki PI ima dubinu 0

Dubina čvora v, $d(v) = \max(d(u)|(u, v) \in E) + t(v)$

Algoritam

1. Generišemo preseke u topološkom poretku
2. Optimalni presek za čvor *u* je onaj sa najmanjom dubinom
3. Kao i kod DAGON-a, kad smo pronašli optimalne preseke za sve čvorove, u obrnutom topološkom poretku generišemo LUT-ove

U našem primeru



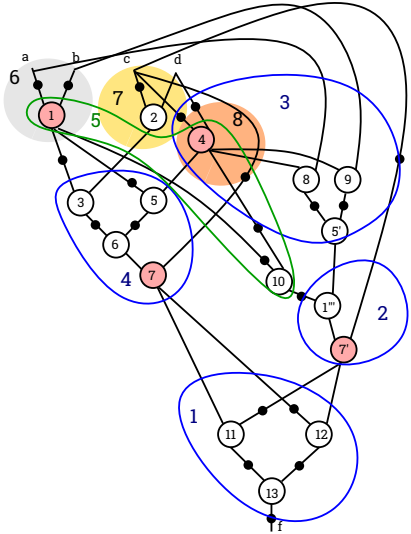
4-LUT

{presek}:dubina

a: {{a}:0}
b: {{b}:0}
c: {{c}:0}
d: {{d}:0}
1: {{1}:1, {a, b}:max({0, 0}) + 1 = 1}
2: {{2}:1, {c, d}:1}
3: {{3}:2, {1, 2}:2, {1, c, d}:2, {2, a, b}:2, {a, b, c, d}:1}
4: {{4}:1, {c, d}:1}
5: {{5}:2, {1, 4}:2, {1, c, d}:2, {4, a, b}:2, {a, b, c, d}:1}
6: {{6}:2, {3, 5}:max({1, 1}) + 1 = 2, {3, 1, 4}:2, {3, 1, c, d}:2, {3, 4, a, b}:2, {5, 1, 2}:2, {5, 1, c, d}:2...}
7: {{7}:2, {6, c}:3, {3, 5, c}:max({1, 1, 0} + 1 = 2, {1, 2, 5, c}:2, {3, 1, 4, c}:2, {1, 2, 4, c}:2}
8: {{8}:1, {4, a}:2, {c, d, a}:1}
9: {{9}:1, {4, b}:2, {c, d, b}:1}
10: {{10}:1, {1, 4}:2, {a, b, 4}:2, {1, c, d}:2, {a, b, c, d}:1}
5': {{5'}:1, {8, 9}:2, {8, 4, b}:2, {8, c, d, b}:2, {9, 4, a}:2, {4, a, b}:2, {a, b, c, d}:1}
1''': {{1'''}:2, {5, 10}:2, {1, 4, 8, 9}:2, {1, 4, 8, b}:2, {1, 4, 9, a}:2, {1, 4, a, b}:2, {a, b, 4, 8}:2, {a, b, 4, 9}:2, {a, b, 4}: 2, ...}
7': {{7'}:2, {1''', c}:3, {5, 10, c}:2}
11': {{11}:3, {7, 7'}:3, {7, 1''', c}:3, {7', 6, c}:3...}
12: {{12}:3, {7, 7'}:3,...}
13: {{13}:, {11, 12}:4, {7, 7'}:3, {6, c, 1'''}:3}...}

Konačna dubina je 3

U našem primeru



4-LUT

{presek}:dubina

a: {{a}:0}
b: {{b}:0}
c: {{c}:0}
d: {{d}:0}
1: {{1}:1, {a, b}:max({0, 0}) + 1 = 1}
2: {{2}:1, {c, d}:1}
3: {{3}:2, {1, 2}:2, {1, c, d}:2, {2, a, b}:2, {a, b, c, d}:1}
4: {{4}:1, {c, d}:1}
5: {{5}:2, {1, 4}:2, {1, c, d}:2, {4, a, b}:2, {a, b, c, d}:1}
6: {{6}:2, {3, 5}:max({1, 1}) + 1 = 2, {3, 1, 4}:2, {3, 1, c, d}:2, {3, 4, a, b}:2, {5, 1, 2}:2, {5, 1, c, d}:2...}
7: {{7}:2, {6, c}:3, {3, 5, c}:max({1, 1, 0} + 1 = 2, {1, 2, 5, c}:2, {3, 1, 4, c}:2, {1, 2, 4, c}:2}
8: {{8}:1, {4, a}:2, {c, d, a}:1}
9: {{9}:1, {4, b}:2, {c, d, b}:1}
10: {{10}:1, {1, 4}:2, {a, b, 4}:2, {1, c, d}:2, {a, b, c, d}:1}
5': {{5'}:1, {8, 9}:2, {8, 4, b}:2, {8, c, d, b}:2, {9, 4, a}:2, {4, a, b}:2, {a, b, c, d}:1}
1''': {{1'''}:2, {5, 10}:2, {1, 4, 8, 9}:2, {1, 4, 8, b}:2, {1, 4, 9, a}:2, {1, 4, a, b}:2, {a, b, 4, 8}:2, {a, b, 4, 9}:2, {a, b, 4}: 2, ...}
7': {{7'}:2, {1''', c}:3, {5, 10, c}:2}
11': {{11}3:, {7, 7'}:3, {7, 1''', c}:3, {7', 6, c}:3...}
12: {{12}:3, {7, 7'}:3,...}
13: {{13}:, {11, 12}:4, {7, 7'}:3, {6, c, 1'''}:3}...}

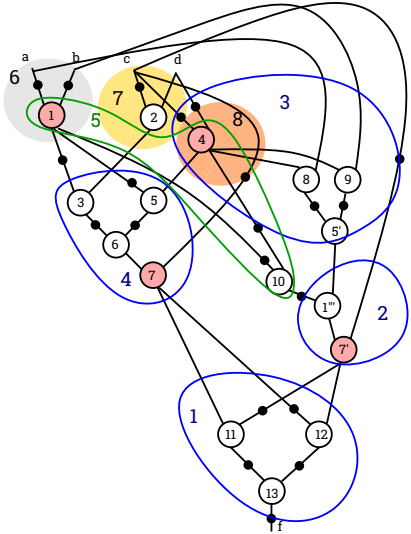
Konačna dubina je 3

Implicitno dupliranje čvorova

Primetimo da graf nismo razlagali na stabla kao i to da su mnogi čvorovi sa više dece u AIG-u ušli u sastav više od jednog LUT-a

Oni su na taj način implicitno duplirani, što je neophodno da bismo ostvarili optimalnu dubinu

Ako ne bismo duplirali čvor 1, kritična putanja bi porasla na 4



4-LUT

{presek}:dubina

a: {{a}:0}
b: {{b}:0}
c: {{c}:0}
d: {{d}:0}
1: {{1}:1, {a, b}:max({0, 0}) + 1 = 1}
2: {{2}:1, {c, d}:1}
3: {{3}:2, {1, 2}:2, {1, c, d}:2, {2, a, b}:2, {a, b, c, d}:1}
4: {{4}:1, {c, d}:1}
5: {{5}:2, {1, 4}:2, {1, c, d}:2, {4, a, b}:2, {a, b, c, d}:1}
6: {{6}:2, {3, 5}:max({1, 1}) + 1 = 2, {3, 1, 4}:2, {3, 1, c, d}:2, {3, 4, a, b}:2, {5, 1, 2}:2, {5, 1, c, d}:2...}
7: {{7}:2, {6, c}:3, {3, 5, c}:max({1, 1, 0} + 1 = 2, {1, 2, 5, c}:2, {3, 1, 4, c}:2, {1, 2, 4, c}:2}
8: {{8}:1, {4, a}:2, {c, d, a}:1}
9: {{9}:1, {4, b}:2, {c, d, b}:1}
10: {{10}:1, {1, 4}:2, {a, b, 4}:2, {1, c, d}:2, {a, b, c, d}:1}
5': {{5'}:1, {8, 9}:2, {8, 4, b}:2, {8, c, d, b}:2, {9, 4, a}:2, {4, a, b}:2, {a, b, c, d}:1}
1''': {{1'''}:2, {5, 10}:2, {1, 4, 8, 9}:2, {1, 4, 8, b}:2, {1, 4, 9, a}:2, {1, 4, a, b}:2, {a, b, 4, 8}:2, {a, b, 4, 9}:2, {a, b, 4}: 2, ...}
7': {{7'}:2, {1''', c}:3, {5, 10, c}:2}
11': {{11}3:, {7, 7'}:3, {7, 1''', c}:3, {7', 6, c}:3...}
12: {{12}:3, {7, 7'}:3,...}
13: {{13}:, {11, 12}:4, {7, 7'}:3, {6, c, 1'''}:3}...}

Konačna dubina je 3

Ideje se prostiru dvosmerno

Delay-Optimal Technology Mapping by DAG Covering*

Yuji Kukimoto Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
{kukimoto,brayton}@eecs.berkeley.edu

Prashant Sawkar
Strategic CAD Laboratories
Intel Corporation, Hillsboro, OR 97124
psawkar@ichips.intel.com

Abstract

We propose an algorithm for minimal-delay technology mapping for library-based designs. We show that subject graphs need not be decomposed into trees for delay minimization; they can be mapped directly as DAGs. Experimental results demonstrate that significant delay improvement is possible by this new approach.

1 Introduction

In 1987 Keutzer [7] proposed an algorithmic approach to the technology mapping problem, in which he observed similarity between this problem and the code optimization problem for programming languages and adapted an existing tree-covering technique for the latter problem to technology mapping. This approach soon dominated rule-based approaches and became the de facto standard.

In Keutzer's formulation a technology-independent circuit and each gate in a given library are decomposed into NAND2-INV circuits. The decomposed circuit is called a *subject graph* while each decomposed gate is called a *pattern graph*. The technology

library-based technology mapping since one needs to generate pattern graphs for all 2^{2^k} k-input functions. Based on this observation many ideas have been proposed for the FPGA mapping problem again under different cost criteria [4]. As for minimum area mapping Levin *et al.* [10] and Farrahi *et al.* [6] proved that the problem is NP-hard for $k = 4$ and $k \geq 5$ respectively. Minimum-delay mapping, on the other hand, was shown for LUT-based FPGAs to be solvable in polynomial time by Cong *et al.* in [1, 2]. Here the given circuit is directly mapped without decomposing its DAG structure to trees unlike conventional library-based mapping.

In this paper we consider the minimum-delay technology mapping problem for library-based designs where a subject graph is a DAG. Careful analysis of [2] shows that the basic dynamic programming approach in [2] is not specific to FPGA mapping and can be easily adapted to library-based mapping. This leads to a linear time algorithm for minimum-delay DAG covering under load-independent delay models. As far as we know, this is the first result that shows that the minimum-delay technology mapping problem for DAGs can be solved optimally in polynomial time.

Smanjenje površine (duplikata)

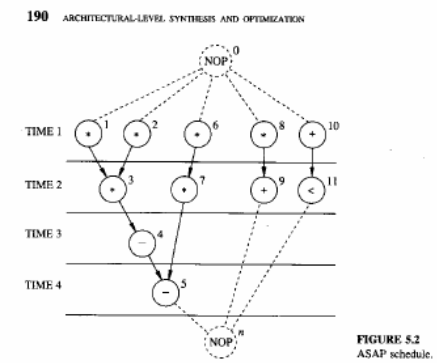


FIGURE 5.2
ASAP schedule.

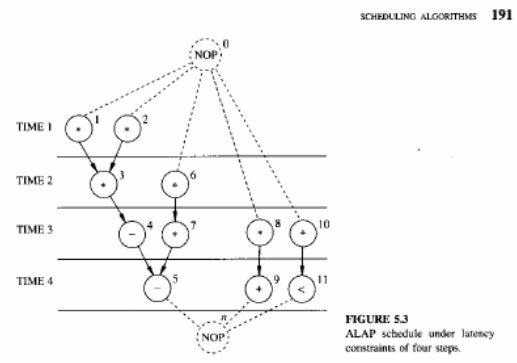


FIGURE 5.3
ALAP schedule under latency constraints of four steps.

Ne moraju svi čvorovi da računaju svoj izlaz u najranijem mogućem trenutku

Opšti pristup

1. Biramo preseke sa najmanjom dubinom iz čega računamo najmanju moguću dubinu
2. Iterativno menjamo optimalne preseke u smislu dubine onima koji smanjuju broj LUT-ova, tako da to ne naruši ukupnu dubinu

Više o ovome u jednom od radova koji ćemo čitati