Multi-Level Logic Optimization

Kiyoung Choi

School of Electrical Engineering

Seoul National University

Introduction

- **Two-Level vs Multi-Level**
  - **Two-level**
    - **Sum of products or product of sums**
    - **Implementation: PLA, ROM**
  - **Multiple-level**
    - **Factored form**
    - **Implementation: standard cells**

– **Example**

X = AB + AC + ADE

Y = FB + FC + FAD

--> 6 product terms, 14 literals, 2 levels

K = B + C (common term)

X = AK + ADE

Y = FK + FAD

--> 6 product terms, 12 literals, 3 levels

K = B + C

X = A(K + DE) (factor)

Y = F(K + AD) (factor)

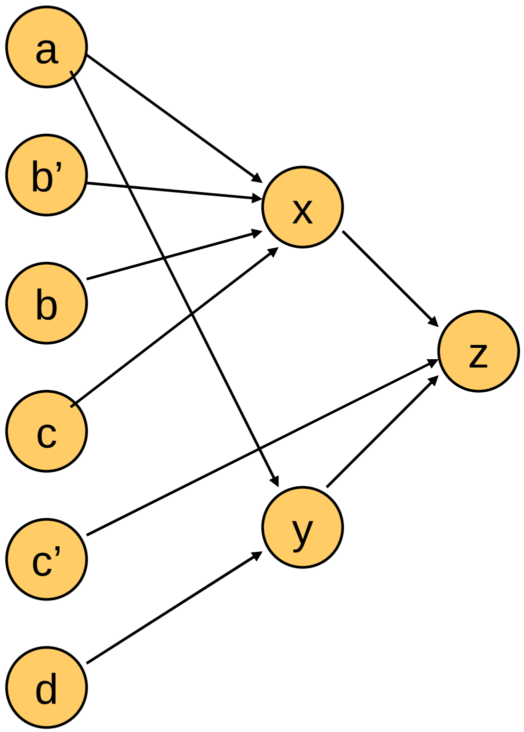
--> 10 literals, 3 levels

Representation

- **Boolean network**
  - **Set of variables X={x}**
    - **inputs, outputs, intermediates**
  - **Set of functions F={f}**
    - **Specified for each intermediate and output variable**
    - **Each function depends on some other variables**
  - **Directed acyclic graph representation**
    - **vertex: variable**
    - **edge: dependency**
  - **Implementation**
    - **vertex: I/O, gates**
    - **edge: nets**
  - **ex)** $x = ab' + bc$

    $y = ad$

    $z = x + c'y$

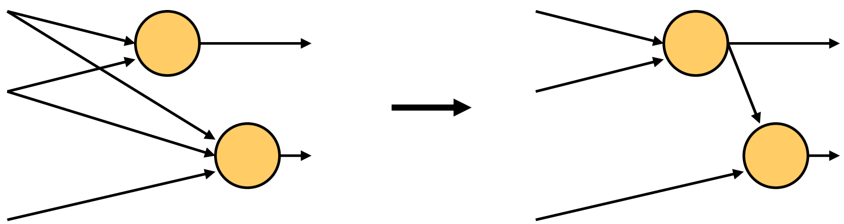


Multiple-Level Logic Synthesis

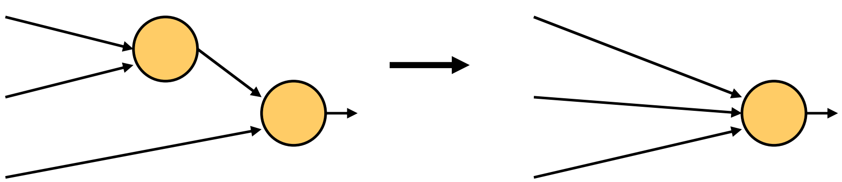
- **Problem**
  - **Minimize area under delay constraints**
  - **Minimize maximum delay under area constraints**
  - **Open book: arbitrary functions**
  - **Closed book: all factored forms must conform to a library**
  - **Technology mapping: transformation of an open book representation into a closed book one**
- **Area minimization**
  - **Minimize total number of literals**
  - **Strategy:**
    - **Modify the network incrementally**
    - **Preserve I/O equivalence**
    - **Optimize open book model first, then map to closed book**
- **Methods**
  - **Algorithmic approach (MIS, BOLD)**
  - **Rule-based approach (LSS)**
  - **Combination (SOCRATES --> Synopsys)**

- **Reference**
  - **R. K. Brayton, R. Rudell, A. Sangiovanni-Vinceltelli, and A. R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Trans. on CAD*, Nov. 1987**
- **Network transformation**
  - **Global transformation**
    - **Exploit the dependencies among functions**
    - **ex: find commonalities**
  - **Local transformation**
    - **Operate on each individual function**
    - **ex: find best factorization**

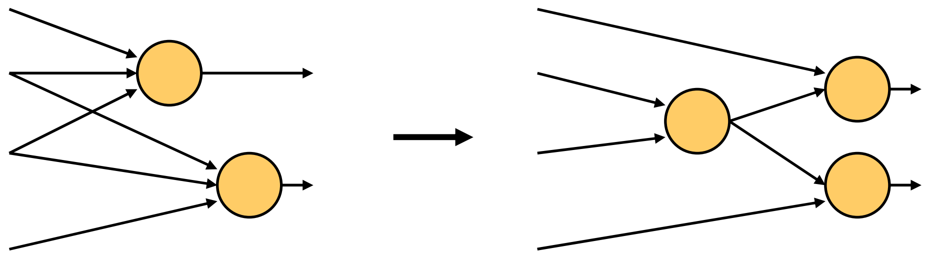
- **Global transformation**
  - **Resubstitution**
    - **Simplify an expression by using another input**



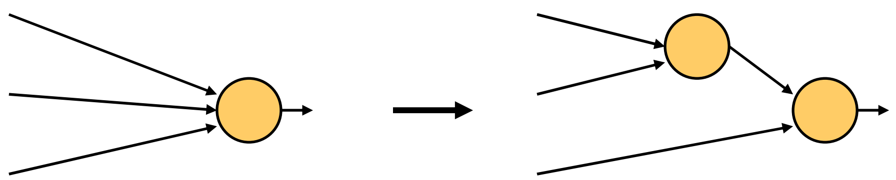
  - **Elimination**
    - **Merge two expressions**



– **Extraction**
  - **Add an extra expression for common subexpressions**



– **Decomposition**
  - **Split an expression into two or more simpler expressions**



Division

- **Boolean division**
  - **Boolean divisor:**
    - **g is a Boolean divisor of f if**

      f = gh+r, gh != 0
    - **g is a Boolean factor of f if**

      f = gh
  - **Example**

    f = a+bc+e

    g = a+c

    --> h = a+b,   r = e

    --> f = gh+r = (a+c)(a+b)+e = a+bc+e
  - **Hard to compute --> algebraic approximation**

- **Algebraic division**
  - **Algebraic expression**
    - **An expression which is minimal w.r.t. single cube containment**
    - **ex) a+ab is not an algebraic expression**
  - **Algebraic product**
    - **fg is an algebraic product if f and g are algebraic expressions and have no input variables in common.**
    - **ex) (a+b)(c+d)=ac+ad+bc+bd: algebraic product**

      (a+b)(b'+c)=ab'+ac+bc:Boolean product
  - **Algebraic division**
    - **View a Boolean expression as polynomials**
    - **g is an algebraic divisor of f if**

      $$f = gh+r, \quad gh \neq 0,$$

      and gh is an algebraic product (no non-algebraic operation for computing gh).

- **All algebraic divisors are Boolean divisors**
- **Quotient $h = f/g$:**
  Largest algebraic divisor satisfying $f = gh+r$
  --> g is not an algebraic divisor of r
- **Algorithm**

```
ALG_DIV (f, g) {
    U = set {u_j} of cubes in f with literals not in g deleted
    V = set {v_j} of cubes in f with literals in g deleted
    /* u_j v_j is the j-th term of f */
    V^i = {v_j ∈ V: u_j = g_i}
    h = ∩V^i
    r = f - gh
}
```

ex)

$f = ac + ad + bc + bd + e$

$g = a + b$

$U = a + a + b + b + 1$

$V = c + d + c + d + e$

$V^1 = c + d$

$V^2 = c + d$

$h = c + d$

$r = e$

Resubstitution

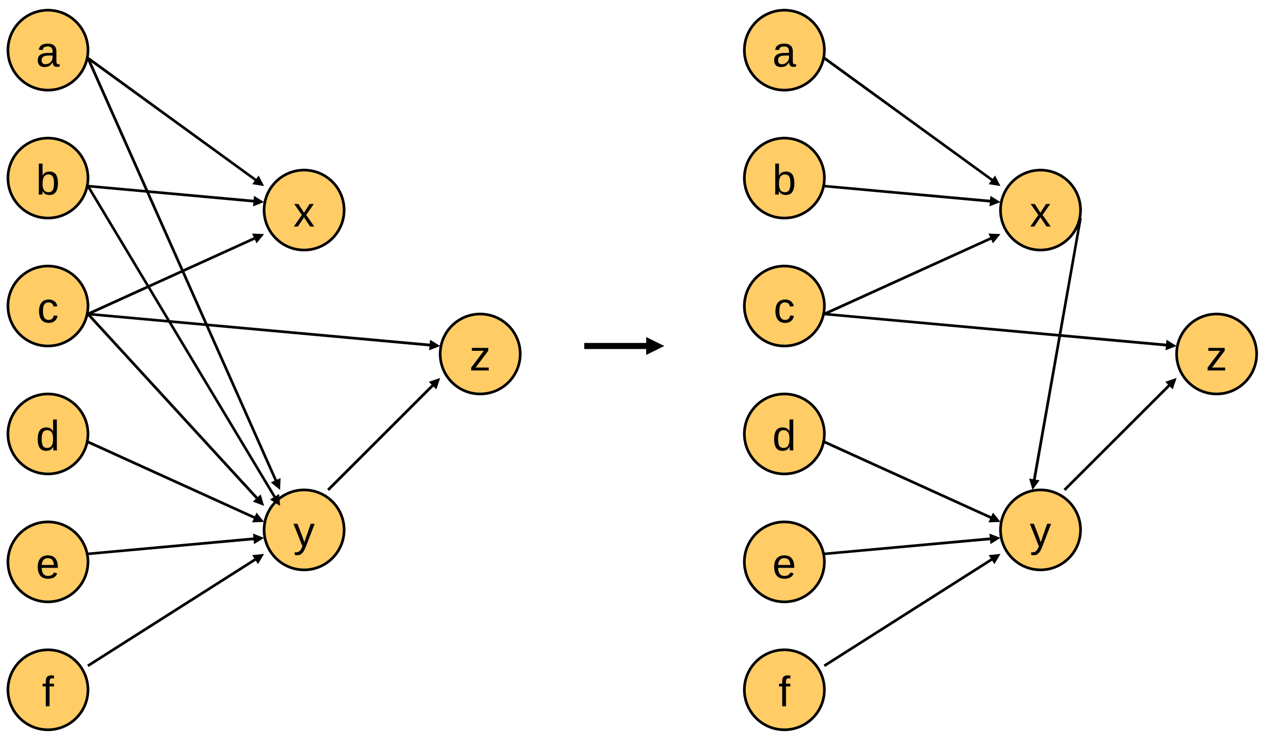
- **Example**

  x = a+bc

  y = ad+ae+bcd+bce+f = (a+bc)(d+e)+f

  z = cy

  --> y = x(d+e)+f --> need division



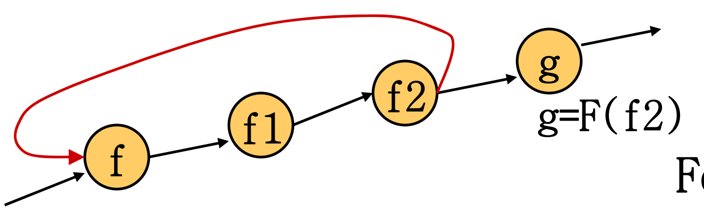
- **For all pairs (f, g), compute f/g and g/f and do resubstitution**

  --> $O(n^2)$ algebraic division --> use filters
- **Filtering: Function g is not an algebraic divisor of f if**
  1. **g contains a literal not in f**

     f = ab + c, g = ad
  2. **g has more terms than f**

     f = ab + c, g = a + b + c
  3. **For any literal, the count in g exceeds that in f**

     f = ab + c, g = ab + ac
  4. **f is in the transitive fan-in of g**



For g=F(f2) to be an algebraic divisor of f,
f must also be a function of f2.
But the network is acyclic.

Kernel

- **Definition**
  - Kernel k of f is

    k = f/c

    where c is a cube and k is cube free (no cube is an algebraic factor of k)
  - K(f): set of all kernels of f
  - c: co-kernel of k

  ex)

  f = abc + abde

  f/a = bc + bde (not cube free)

  f/ab = c + de (cube free --> kernel)
- **No single cube is cube free**

  --> A kernel has at least two cubes
- **If f is cube free, f/1 = f is a kernel**

- **Co-kernel of a kernel is not unique.**

ex) f = adh+aeh+bdh+beh+cdh+ceh+g

= (a+b+c)(d+e)h+g

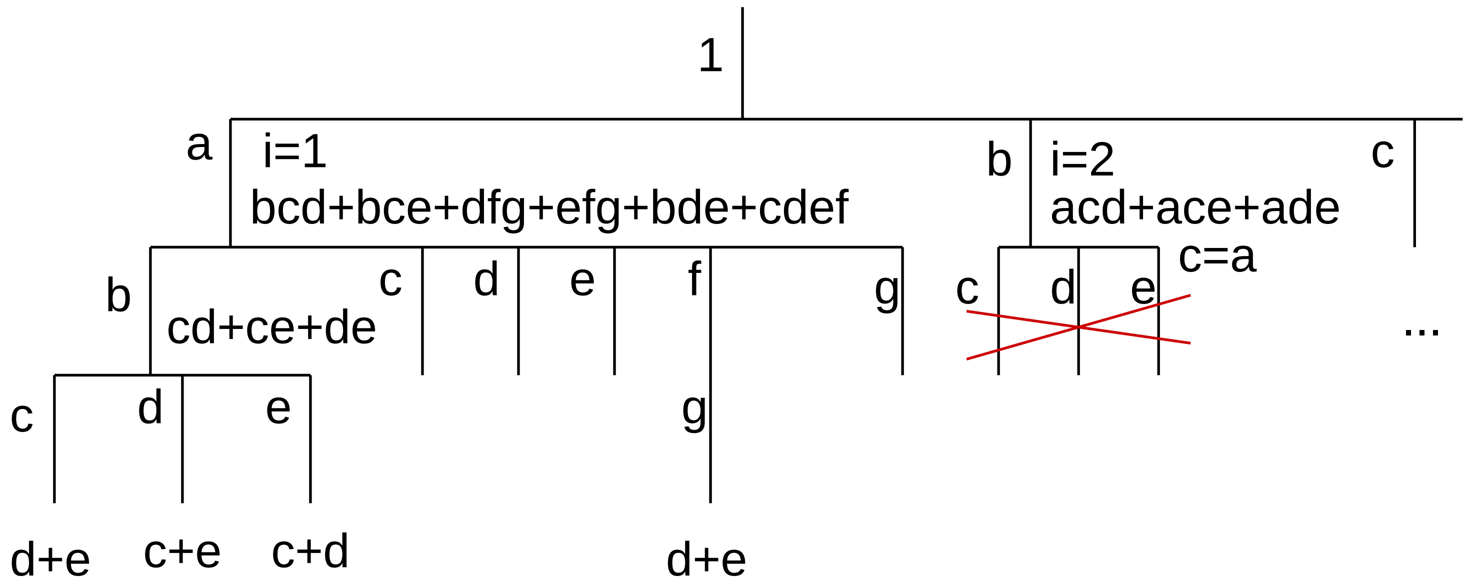
kernel	co-kernel
a+b+c	dh, eh
d+e	ah, bh, ch
(a+b+c)(d+e)	h
(a+b+c)(d+e)h+g	1

- **Algorithm for computing all kernels**

```
KERNELS(f) {
    c=largest cube factor of f
    K=KERNEL1(0, f/c)
    return K
}
KERNEL1(j, g) {
    R={g}
    for (i=j+1; i<=n; i++) {
        if (l_i appears in more than one cube) {        -- multi-cube
            c=largest cube factor of (g/l_i)
            if (l_k not in c for all k<=i) {  -- not yet computed
                R=R ∪ KERNEL1(i, g/(l_i∩c))
            }
        }
    }
    return R
}
```

- **Example**

abcd+abce+adfg+aefg+abde+acdef+cg



kernel	co-kernel
a(bc+fg)(d+e)+ade(b+cf)+cg	1
(bc+fg)(d+e)+de(b+cf)	a
c(d+e)+de	ab
d+e	abc, afg
c+e	abd
... ...	

Extraction

- Multiple cube common sub-expression and single cube common sub-expression
- Increase node cardinality
- Multiple cube common sub-expression
  - Extraction
    - Compute all kernels
    - Compute all kernel intersections
  - Example

    K = {k1, k2, k3}

    k1 = abc + de + fg

    k2 = abc + de + fh

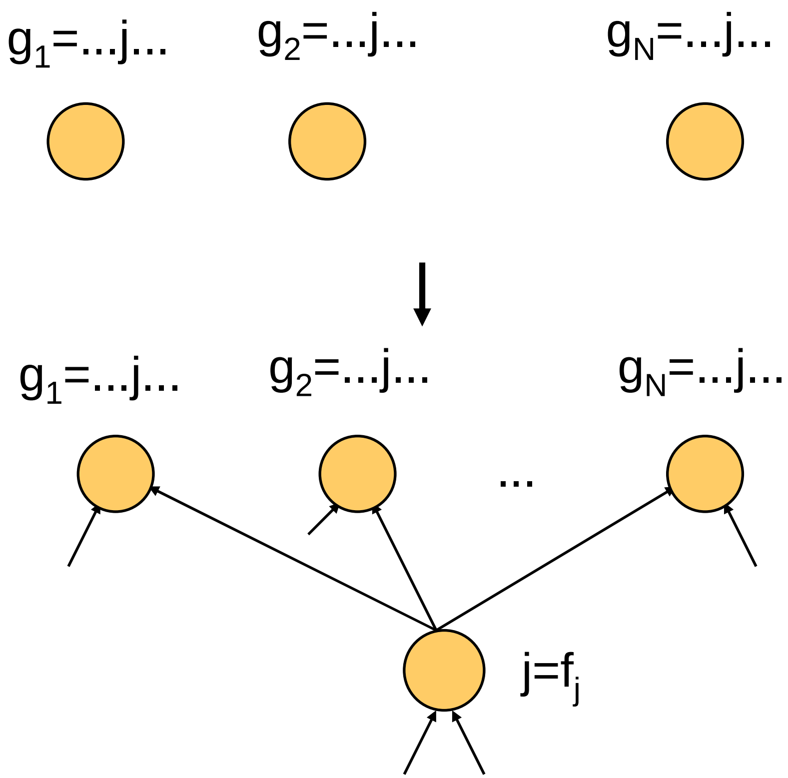
    K3 = abc + fh + gh

    I(K) = {abc, abc+de, abc+fh}

- **Area value of a node (area increase due to xformation)**
  - **Number of literals increased by introducing a kernel intersection as a new intermediate node j**
  - **area_value(j) = (N + L) - NL**

    where N: # of times literal j appears in the network

    L: # of literals in f_j



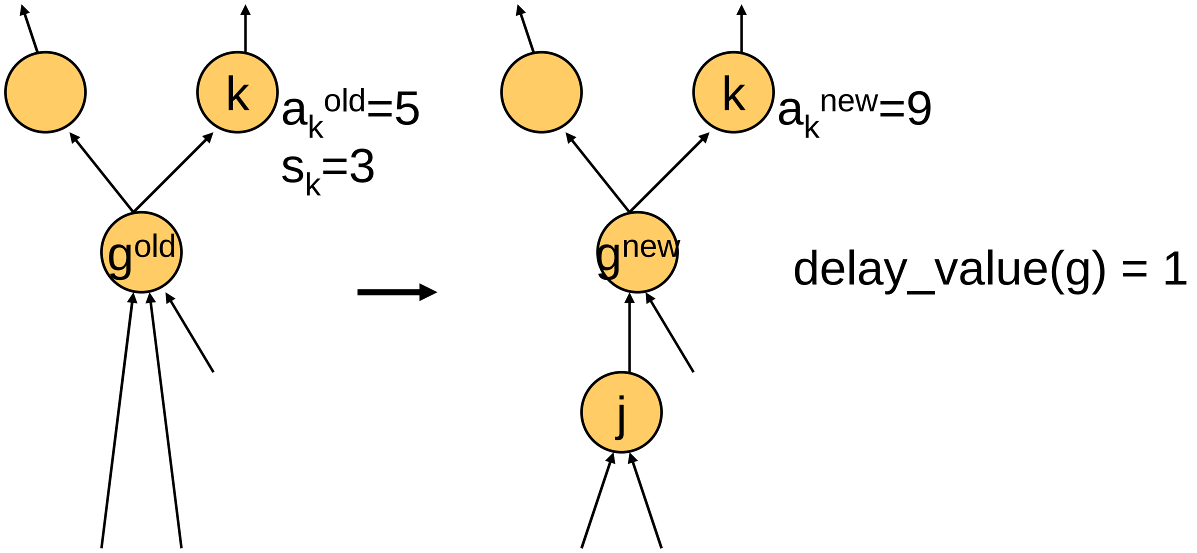
– **Delay value of a node (delay increase due to xformation)**

- **delay_value(g) = max(0, $max_k(a_k^{new} - a_k^{old} - max(0, s_k))$)**

  where a_k^{new}: signal arrival time at fanout node k after xformation

  a_k^{old}: signal arrival time at fanout node k before xformation

  s_k: slack at fanout node k



Factoring

- **Factor an expression by recursive division**

  --> Reduce literal count locally

- **Algorithm**

  GFACTOR(f) {

  If(|f| = 1) return f

  k = CHOOSE_DIVISOR(f) -- choose best literal or kernel

  (h,r) = DIVIDE(f,k) -- algebraic or Boolean division

  return (GFACTOR(k)GFACTOR(h)+GFACTOR(r))

  }

- **ex)**

  f = abde + acde + abh + ach + eh

  = (b + c)(ade + ah) + eh

  = (b + c)a(de + h) + eh

Decomposition

- **Implement one or more divisors by additional nodes**
- **Break large expressions (slow gates)**
- **Ease resubstitution**
- **Increase node cardinality**
- **ex)**

  f = (b + c)a(de + h) + eh

  -->

  g = de + h

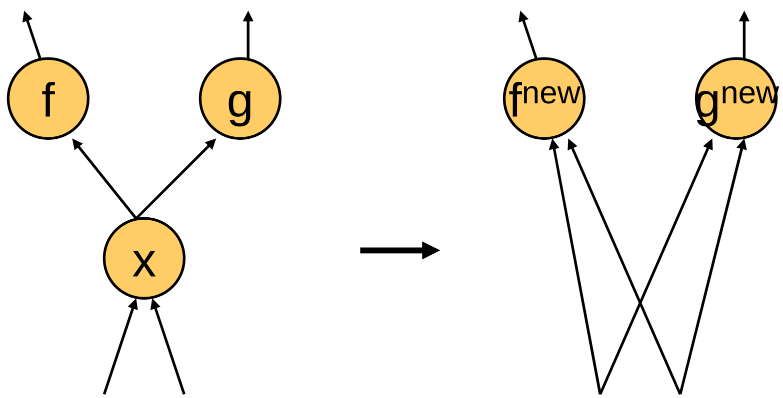
  f = (b + c)ag + eh

Elimination

- **Decrease node cardinality**
- **Exraction, resubstitution and decomposition may increase some path lengths (timing).**
- **Eliminate nodes with area_value below threshold**
- **Example**

x=a+b, f=ex+cde, g=(d+e)x+bf

area_value(x)=NL-(N+L)=4-(2+2)=0



- **Repeat until no change**