# Лекция 6.
# Оптимизация задержки комбинационных логических сетей

Математические модели и методы логического синтеза СБИС

Осень 2015

# Оптимизация глубины

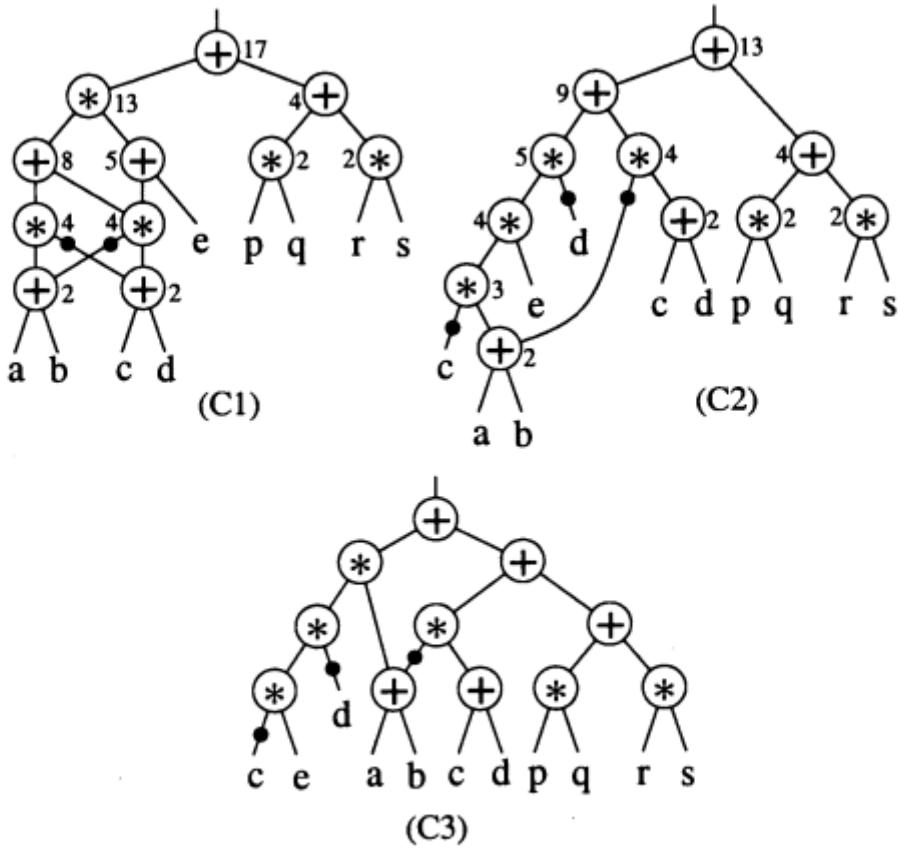


Fig. 1. Three different circuits implementing the same Boolean function.

# Оптимизация глубины

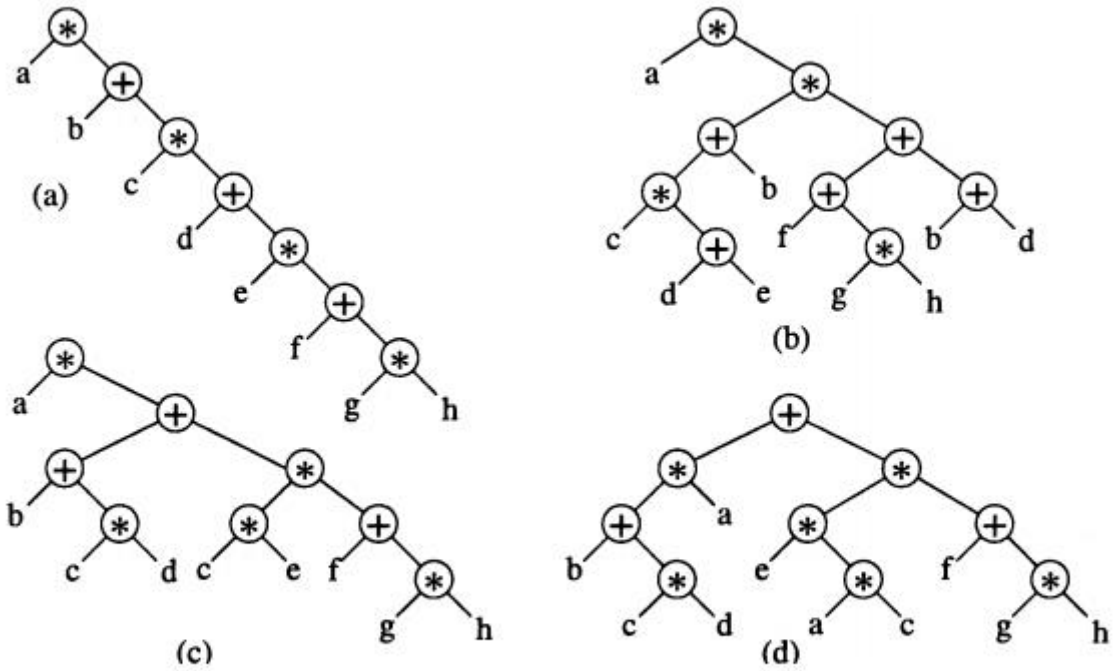


Fig. 2. Equivalent factored forms.

# Оптимизация глубины

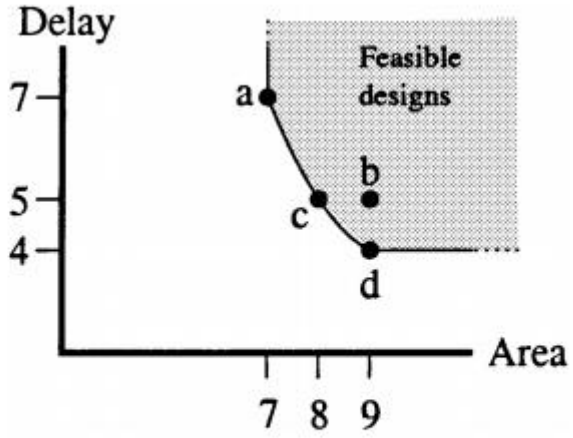


Fig. 3. Area/delay tradeoff for the trees.

# Оптимизация глубины

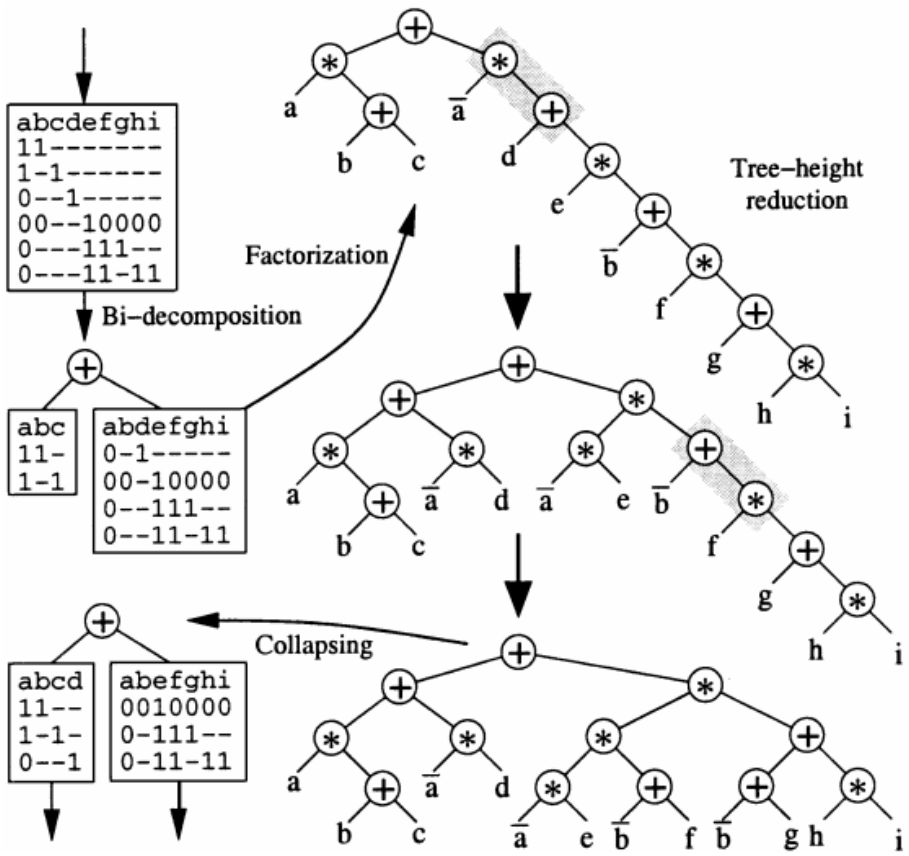


Fig. 4. Example of timing-driven bi-decomposition.

# Оптимизация глубины

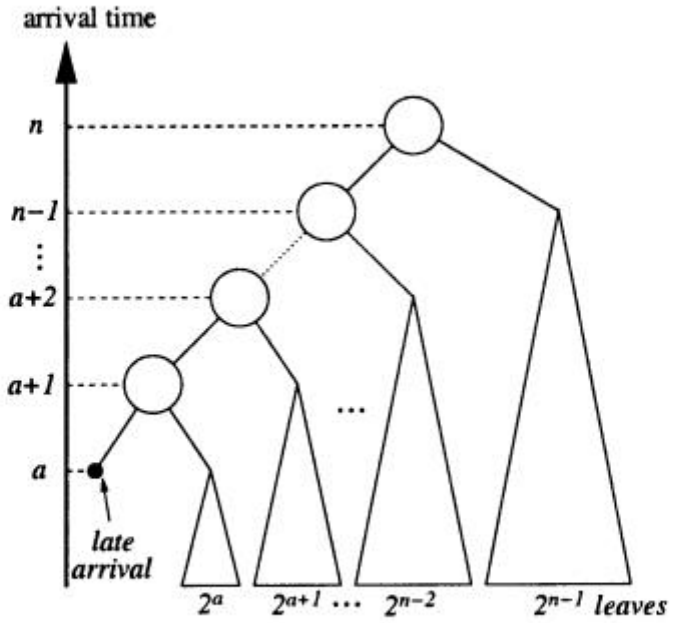


Fig. 5. Illustration of the proof of theorem 3.

# Оптимизация глубины

```
CLUSTER (G)
    { Pre-cond: G.op ≠ ⊥. Returns a set of sub-BDAGs }
    If G.op = G.left.op then C_L := CLUSTER(G.left);
    else C_L := G.left;
    If G.op = G.right.op then C_R := CLUSTER(G.right);
    else C_R := G.right;
    return C_L ∪ C_R;
```

---

```
MIN_DELAY_CLUSTERS (G)
    { Returns a BDAG with min-delay clusters }
    { Q is a list ordered by depth }
    if G.op = ⊥ then return G;
    C := CLUSTER(G); Q := ∅;
    for each c ∈ C do
        INSERT (Q, MIN_DELAY_CLUSTERS(c));
    while |Q| > 1 do
        X := EXTRACT_MIN_DEPTH(Q);
        Y := EXTRACT_MIN_DEPTH(Q);
        INSERT(Q, (G.op X Y));
    return EXTRACT(Q);
```

# Оптимизация глубины

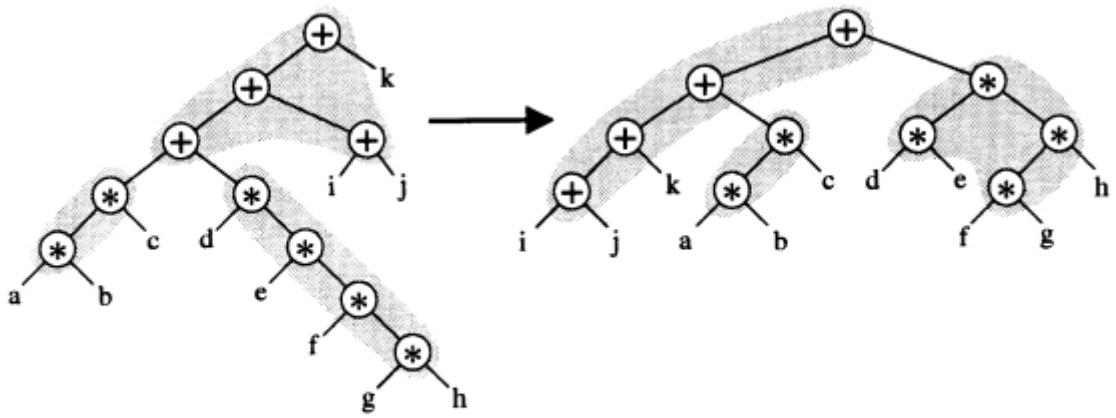


Fig. 7. Application of MIN_DELAY_CLUSTERS.

# Оптимизация глубины

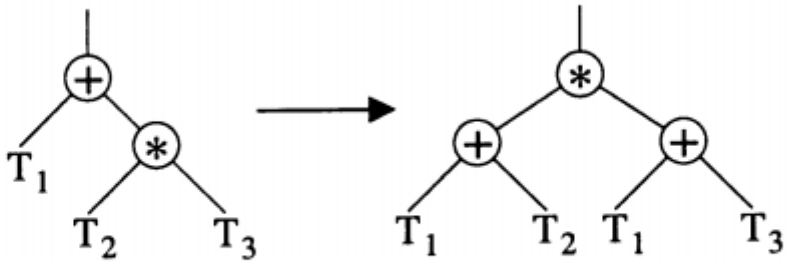


Fig. 8. Distributive law.

# Оптимизация глубины

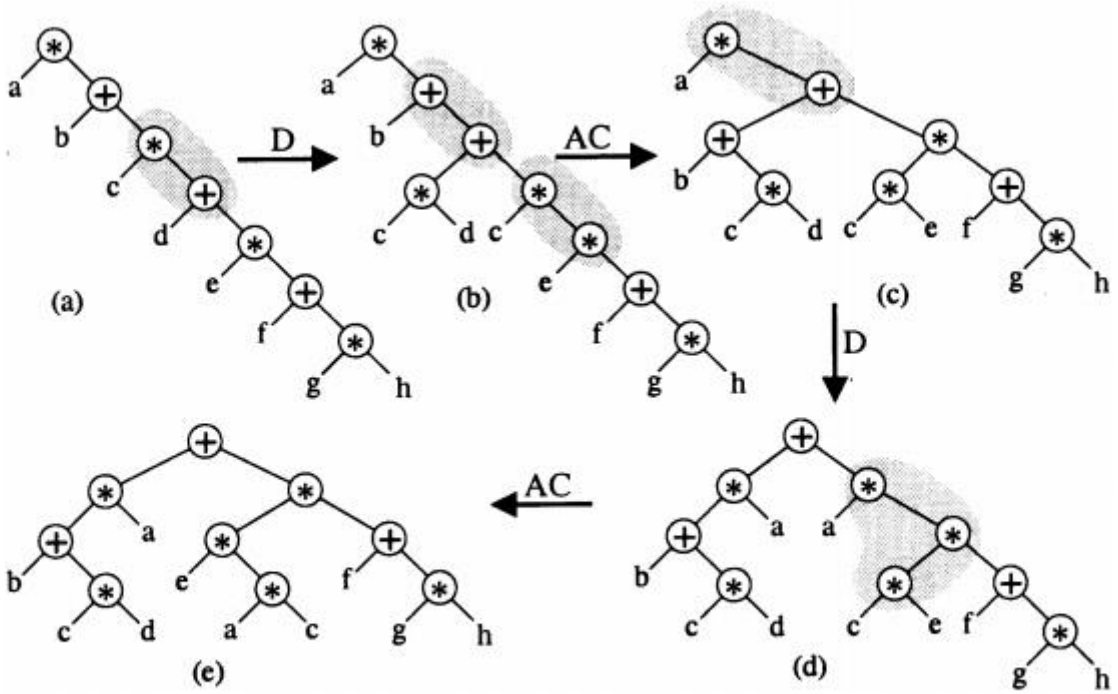


Fig. 9. Application of ACD rules to optimize performance.

# Оптимизация глубины

```
ACD_SPEED (F, ReqTime)
   { F is a BDAG. Returns a BDAG }
   { ReqTime is the required time (in logic levels) }
   Best := MIN_DELAY_CLUSTERS (F);
   MaxTime := MAX(ReqTime, Lower_Bound_Depth(F));
   frontier := {Best}; explored := {Best};
   while depth(Best) > MaxTime  ∧  "improving" do
      new := ∅;
      for each F_r ∈ frontier do
         for each node n ∈ F_r
         such that D-rule is applicable do
            F' := APPLY_DISTRIBUTIVE (F_r, n);
            F'' := MIN_DELAY_CLUSTERS (F');
            if F'' ∉ explored then
               explored := explored ∪ {F''};
               new := new ∪ {F''};
               Best := Best_Delay_Area (Best, F'');
      frontier := Select_Best_k_Circuits (new, k);
   return Best;
```

Fig. 10. Algorithm for speeding-up.

# Оптимизация глубины

```
BI-DECOMP (ON, DC, ReqTime, method)
    { ON and DC are covers. Returns a tree }
    { "method" determines the decomposition strategy }
    F_d := Decompose_2input_gates (ON, DC, method);
    F_s := ACD_Speed (F_d, ReqTime);
    return F_s;
```