Kahetasemeline minimeerimine ja mitmetasemeline realisatsioon

abc	xy
000	-0
001	11
010	00
011	00
100	10
101	11
110	11
111	0-

mintermid

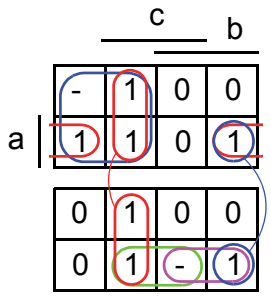
gr.		abc	e
0	o	000	10
1		001	10
		001	01
		100	10
2		101	10
		101	01
		110	10
		110	01
3	o	111	01

1. etapp

gr.	abc	e	
0	00-	10*	
	-00	10*	
1	001	11*	
	-01	10*	
	-01	01*	
	10-	10*	
	1-0	10	A
2	101	11*	
	110	11	B
	1-1	01	C
	11-	01	D

2. etapp

gr.	abc	e	
0	-0-	10	E
1	-01	11	F



lihtimplikantide tabel

	abc	e	A	B	C	D	E	F
~~—~~	~~001~~	~~10~~					+	+
*	001	01						*
	100	10	+				+	
~~—~~	~~101~~	~~10~~					+	+
~~—~~	~~101~~	~~01~~				+		+
	110	10	+	+				
	110	01		+		+		

abc	e	A	B	C	D	E
100	10	+				+
110	10	+	+			
110	01		+		+	

3 varianti

1: F, A, B
2: F, A, D
3: F, B, E

Kahetasemeline minimeerimine ja mitmetasemeline realisatsioon

3 varianti
1: F, A, B
2: F, A, D
3: F, B, E

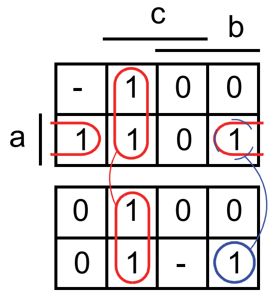
Lisaks üksikult minimeeritud
4: A, E, D, F

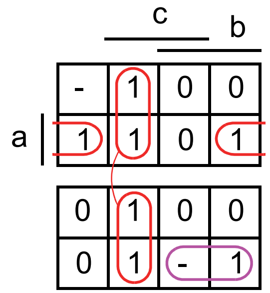
abc	xy
-01	11
1-0	10
110	01

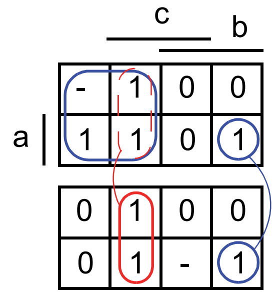
abc	xy
-01	11
1-0	10
11-	01

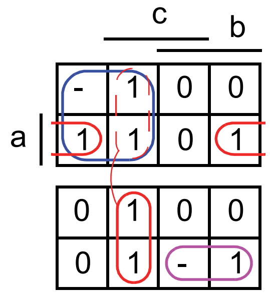
abc	xy
-01	01
110	11
-0-	10

abc	xy
1-0	10
-0-	10
11-	01
-01	01

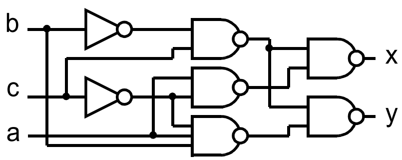








bi=b'; ci=c'; t1=bi·c;
t2=a·ci; t3=a·b·ci;
x=t1+t2; y=t1+t3;



NOT - 2
2-NAND - 4
3-NAND - 1
26 transistori
[13 literaali]

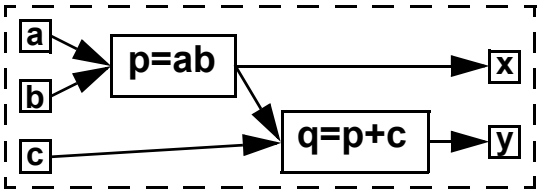
NOT - 2
2-NAND - 5
3-NAND - 0
24 transistori
[12 literaali]

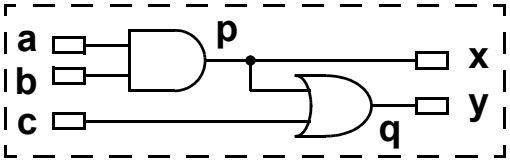
NOT - 2
2-NAND - 3
3-NAND - 1
22 transistori
[11 literaali]

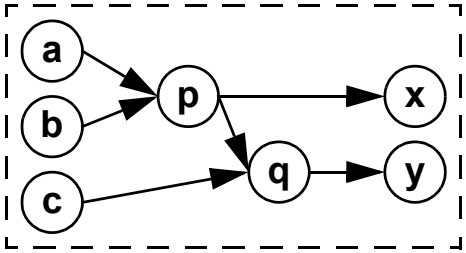
NOT - 2
2-NAND - 5
3-NAND - 0
24 transistori
[12 literaali]

Mitmetasemeline loogikafunktsioonide minimeerimine

- Loogikaelementide teegid – loogikalülid / makrod
- Esitusviis – loogikavõrkgraaf (logic network)
  - omavahel ühendatud loogikafunktsioonid
- Seotud võrkgraaf (bound/mapped network)
  - omavahel ühendatud loogikalülid (struktuurne mudel)
- Pindala (võimsustarbe) ennustuse minimeerimine
  - arvestada tuleb viite piiranguid
- Suurima viite minimeerimine
  - arvestada tuleb pindala (võimsustarbe) piiranguid
- Testitavuse parendamine / võimsustarbe vähendamine
- Ennustus (estimation)
  - literaalide arv, loogikalülide mudelid, olulised signaaliteed, ...
- Mitmetasemeline minimeerimine on *raske* !
- Heuristilised optimeerimis-strateegiad
  - samm-sammuline parendamine – teisendused (transformations) võrkgraafil
  - erinevad meetodid – teisenduste variandid & teisenduste rakendamise järjekorrad

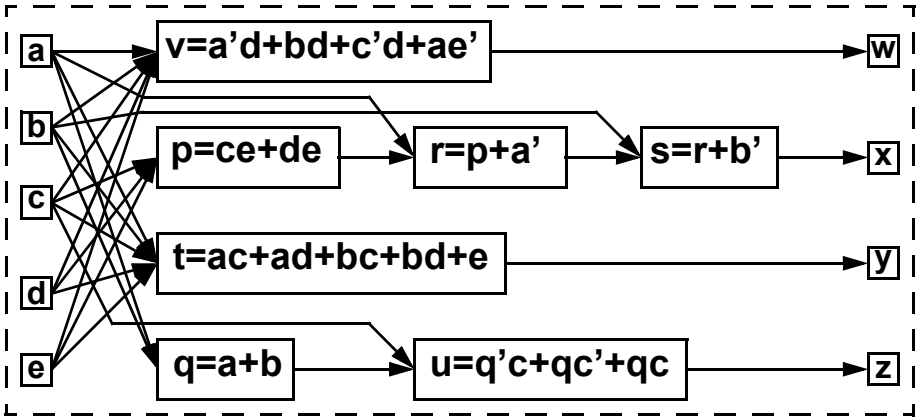


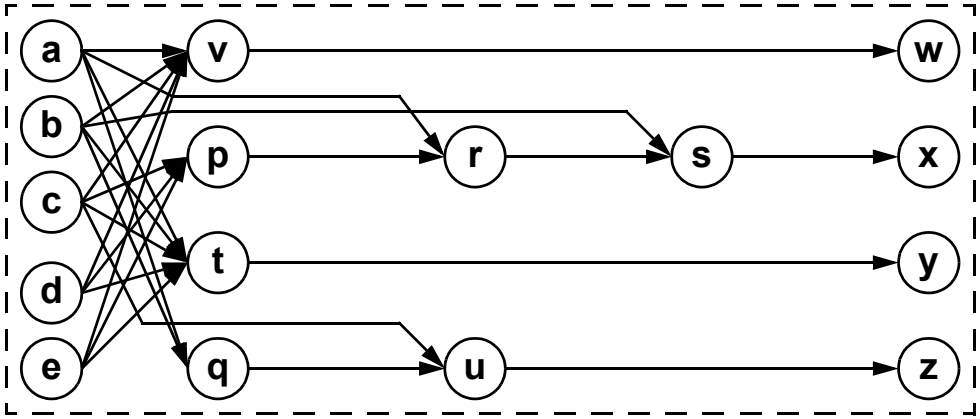




Näitevõrkgraaf

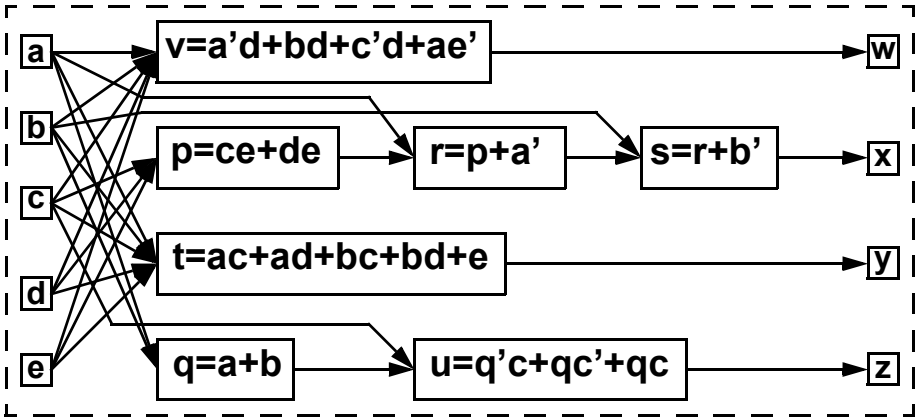
```
p=ce+de
q=a+b
r=p+a'
s=r+b'
t=ac+ad+bc+bd+e
u=q'c+qc'+qc
v=a'd+bd+c'd+ae'
w=v
x=s
y=t
z=u

   w= a'd+bd+c'd+ae'
f: x= a'+b'+ce+de
   y= ac+ad+bc+bd+e
   z= a+b+c
```

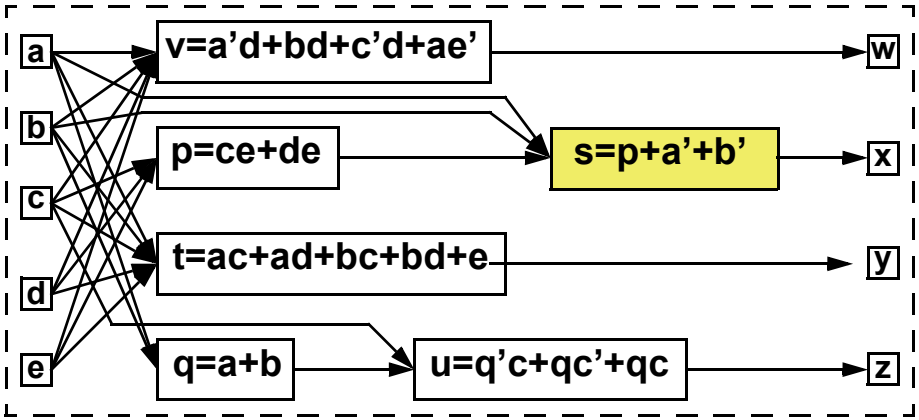




Eemaldamine – Elimination

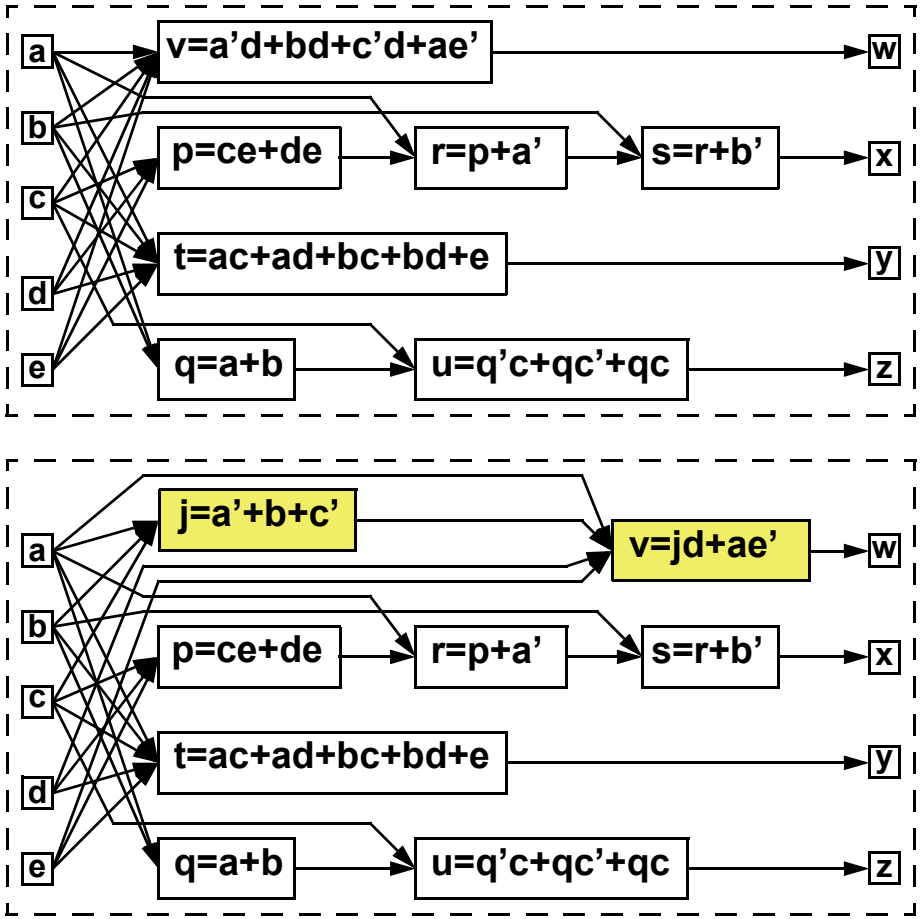
- *Eemaldatakse* üks funktsioon
- Asendatakse vastavad muutujad
- Näide
  - eemaldatakse *r*
    `s=r+b';  r=p+a';`
  - asendus *s*-s
    `s=p+a'+b';`





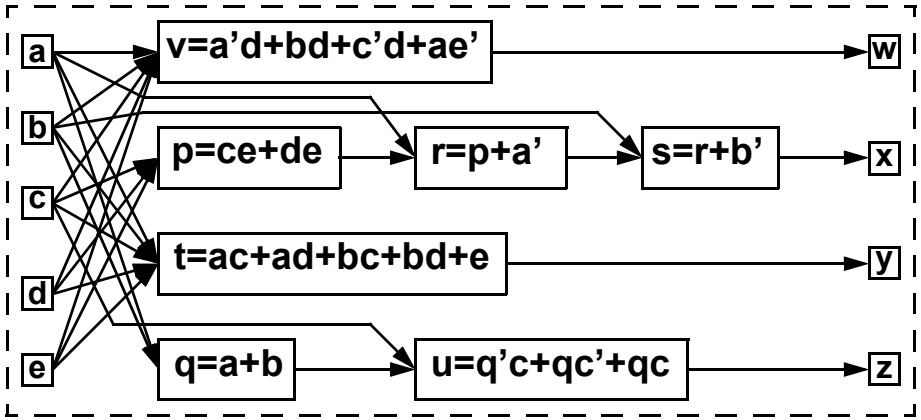
Dekompositsioon – Decomposition

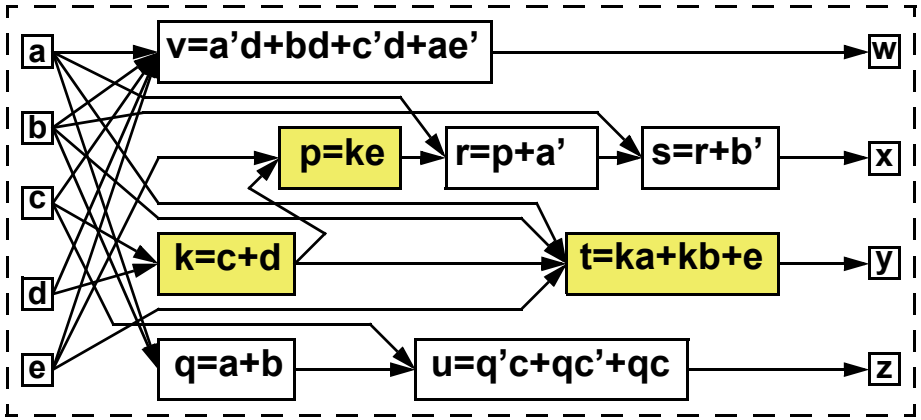
- Üks funktsioon *jagatakse väiksemateks* funktsioonideks
- Luuakse uus sõlm (uued sõlmed)
- Näide
  - *v* jagatakse kaheks
    `v=a'd+bd+c'd+ae';`
  - luuakse *j*
    `j=a'+b+c';  v=jd+ae';`



Eraldamine – Extraction

- Leitakse *ühine alam-avaldis* (common sub-expression) kahele (või enamale) funktsioonile
- Alam-avaldis eraldatakse kui uus funktsioon
- Luuakse uus sõlm
- Näide
  - *p* ja *t* jagavad alam-avaldist
    `p=ce+de;  t=ac+ad+bc+bd+e;`
    `p=(c+d)e;  t=(c+d)(a+b)+e;`
  - luuakse *k*
    `k=c+d;  p=ke;  t=ka+kb+e;`

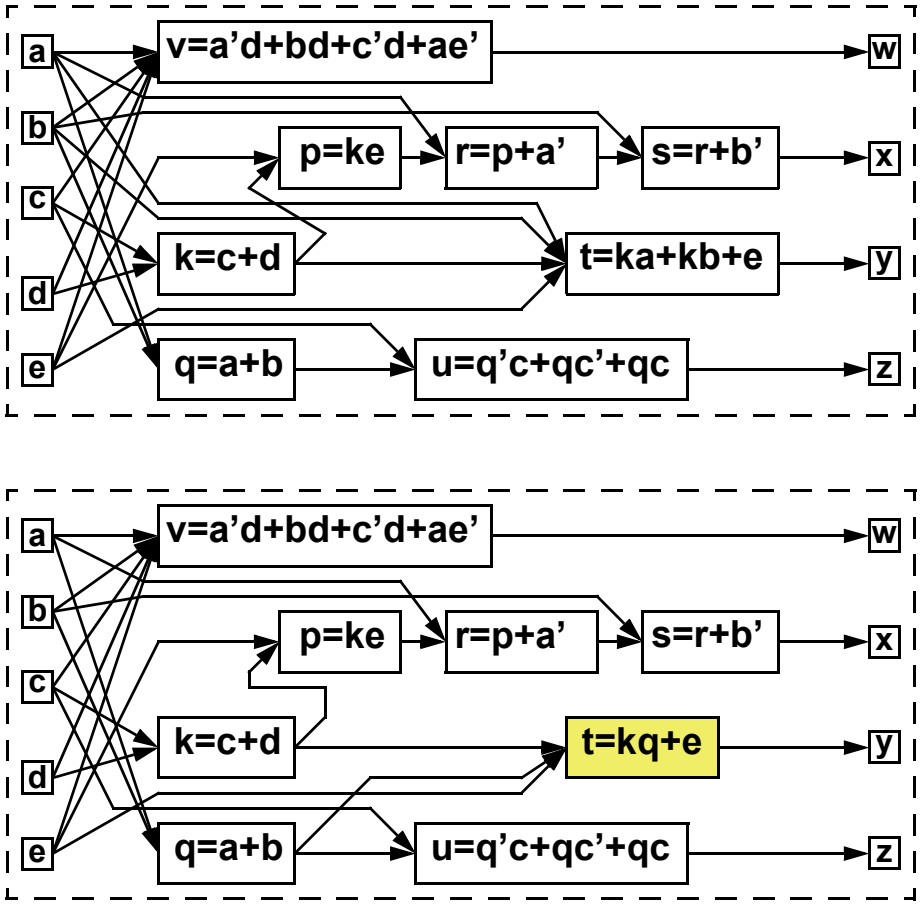




Asendamine – Substitution

- **Lihtsustatakse osa-funktsiooni kasutades *lisa-sisendit*, mida varem selles funktsioonis ei esinenud**

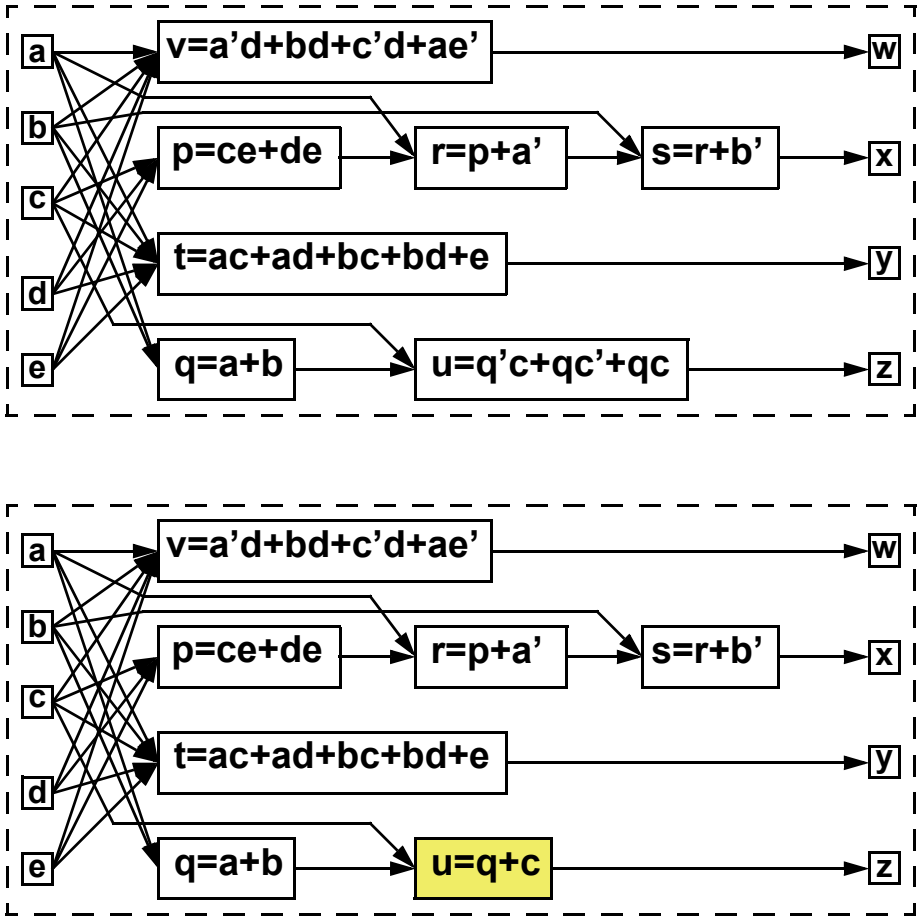
- **Näide**
  - ***t* sisaldab *q* kui alam-avaldist**
    `t=ka+kb+e;`
    `q=a+b;`
  - **uus *t***
    `t=kq+e;`



Lihtsustamine – Simplification

- *Lihtsustatakse* osa-funktsioon

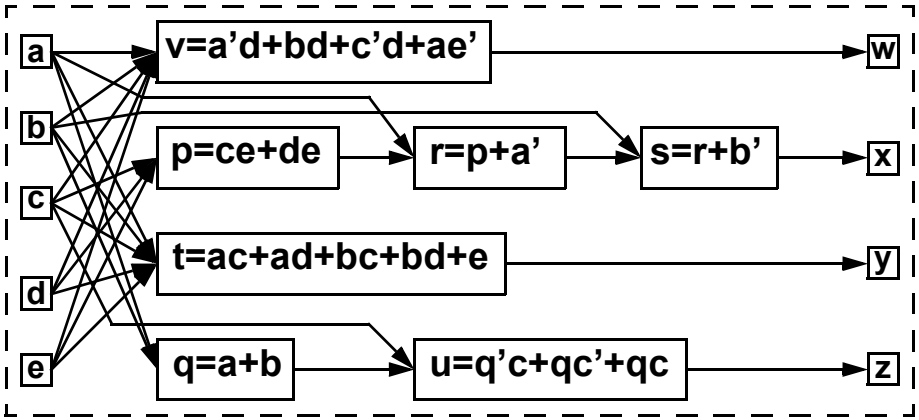
- Näide
  - lihtsustatakse *u*
    u=q′ c+qc′ +qc
  - uus *u*
    u=q+c



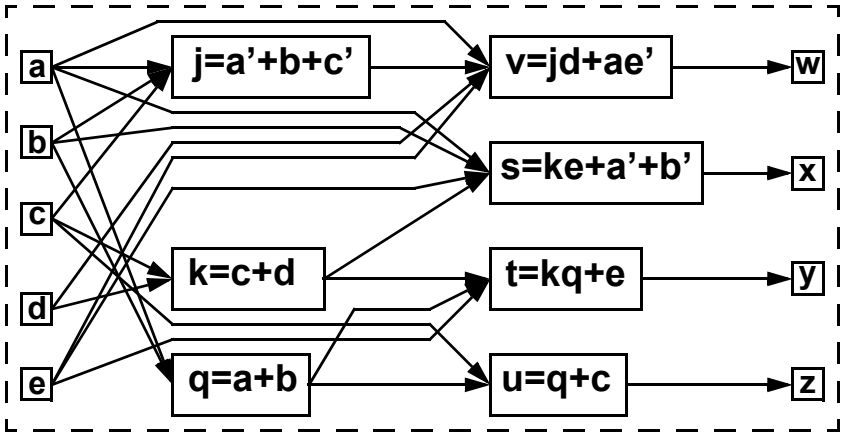
Teisenduste jada – lõplik tulemus

- **Lõplik funktsioon**

```
j=a'+b+c'
k=c+d
q=a+b
s=ke+a'+b'
t=q+c
u=q+c
v=jd+ae'
```



- **Funktsioone/loogikalülisid – 7 ja 7**
- **Literaale – 33 ja 20**
- **Viide – 3 ja 2 (sõlmi)**
- **Viide – 9 ja 7 (sõlmi+literaale)**



Teisendused – kõik sammud

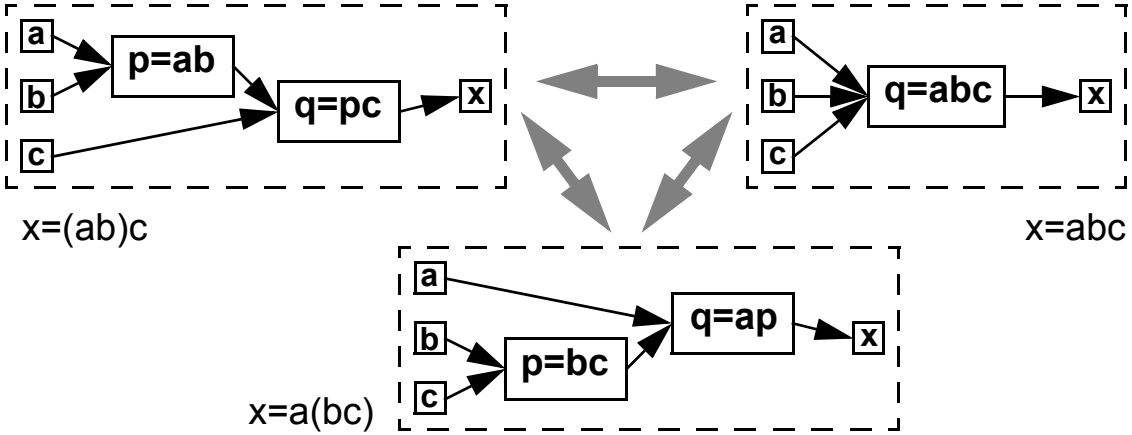
- **Lähteülesanne**
  `p=ce+de;  q=a+b;  r=p+a';  s=r+b';  t=ac+ad+bc+bd+e;`
  `u=q'c+qc'+qc;  v=a'd+bd+c'd+ae';  w=v;  x=s;  y=t;  z=u`
- **Eemaldamine – s, r → s**
  `p=ce+de;  q=a+b;  s=p+a'+b';  t=ac+ad+bc+bd+e;`
  `u=q'c+qc'+qc;  v=a'd+bd+c'd+ae';  w=v;  x=s;  y=t;  z=u`
- **Dekompositsioon – v → j, v**
  `p=ce+de;  q=a+b;  s=p+a'+b';  t=ac+ad+bc+bd+e;`
  `u=q'c+qc'+qc;  j=a'+b+c';  v=jd+ae';  w=v;  x=s;  y=t;  z=u`
- **Eraldamine – p, t → k, p, t**
  `j=a'+b+c';  k=c+d;  p=ke;  q=a+b;  s=p+a'+b';  t=ka+kb+e;`
  `u=q'c+qc'+qc;  v=jd+ae';  w=v;  x=s;  y=t;  z=u`
- **Asendamine – q, t → q, t**
  `j=a'+b+c';  k=c+d;  p=ke;  q=a+b;  s=p+a'+b';  t=kq+e;`
  `u=q'c+qc'+qc;  v=jd+ae';  w=v;  x=s;  y=t;  z=u`
- **Lihtsustamine – u → u**
  `j=a'+b+c';  k=c+d;  p=ke;  q=a+b;  s=p+a'+b';  t=kq+e;`
  `u=q+c;  v=jd+ae';  w=v;  x=s;  y=t;  z=u`
- **Eemaldamine – s, p → s**
  `j=a'+b+c';  k=c+d;  q=a+b;  s=ke+a'+b';  t=kq+e;`
  `u=q+c;  v=jd+ae';  w=v;  x=s;  y=t;  z=u`

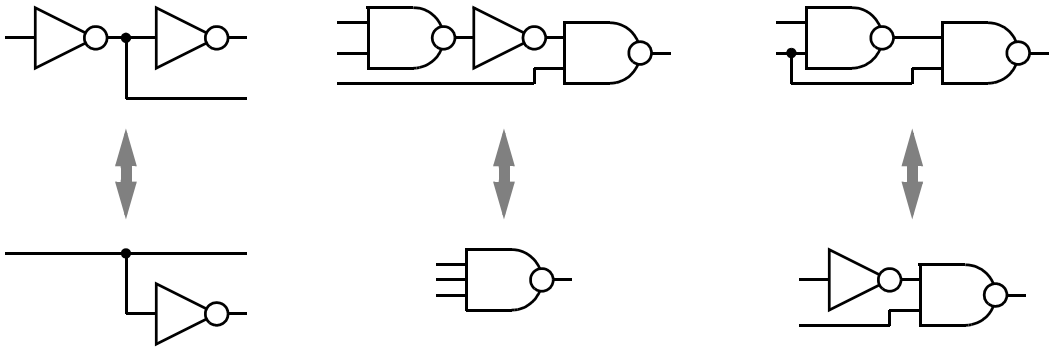
Optimeerimisviisid

- Algoritmiline
  - iga teisenduse tüübi jaoks tuleb määrata algoritm (*operaator*)
    - heuristilised meetodid, nõrk lokaalne optimeerimine
    - ajutine (pinna/viite) ennustuse halvenemin võimalik
  - operaatorite järjekord – skriptipõhine / puhtkogemuslik
  - teisendused – algebralised ja kahendmeetodid
- Reeglitel põhinev
  - andmebaas – mustripaaride hulk
  - mustrite asendused defineeritud reeglite hulgaga
- Tehnoloogiast sõltuv optimeerimine (technology mapping)
  - elementide teegid – elemente realiseerib lihtsamat mõne sisendiga funktsiooni
  - programmeeritav loogika – mõne sisendiga suvalised funktsioonid (CLB)

Reeglitel põhinev optimeerimine

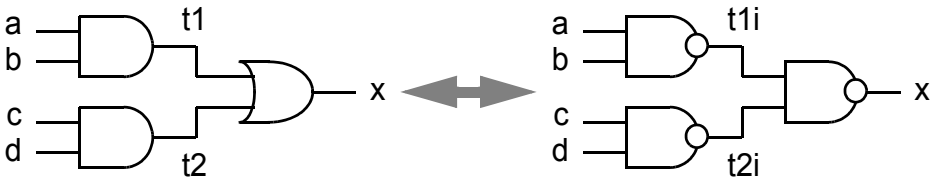
- **Mustri (pattern) otsimine ja asendamine teisega**
  - **vajadus kanoonilise esitusviisi järele**
- **Primitiivsete operatsioonide võrkgraaf**
  - **mustrite keerukus pole piiratud**
- **Seotus kasutatava tehnoloogiaga**
  - **abstrakne tehnoloogia**



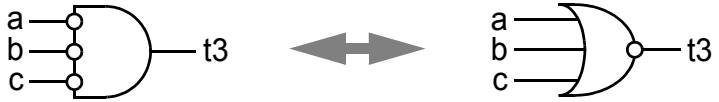


Reeglitel põhinevad teisendused

- **De Morgani seadus**
  - **t1 = a b ; t2 = c d ; x = t1 + t2 ;**
  - **t1' = (a b)' ; t2' = (c d)' ; x = (t1' t2')' ;**
  - **t1i = (a b)' ; t2i = (c d)' ; x = (t1i t2i)' ;**



  - **t3 = a' b' c' ;**
  - **t3 = (a + b + c)' ;**

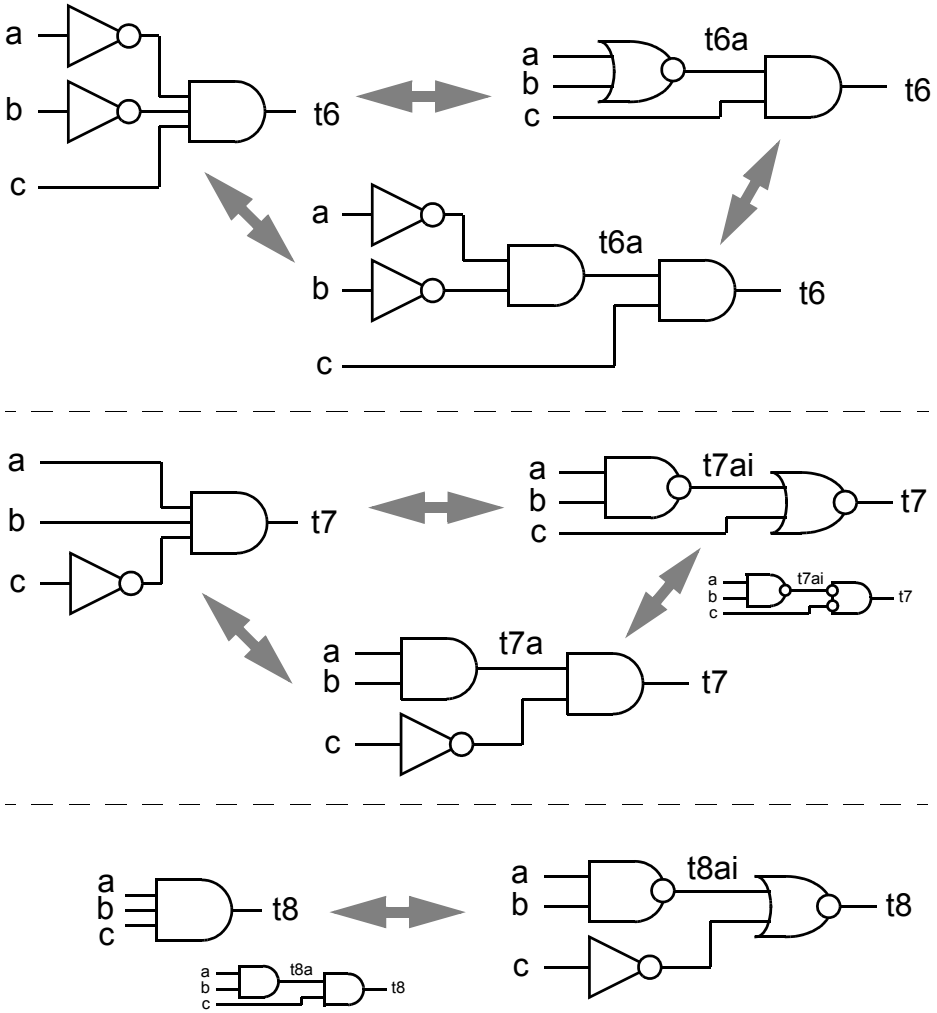


  - **t4 = a b c ;**
  - **t4i = (a b c)' ; t4 = t4i' ;**

- **Topelteituse seadus**
  - **t5i = (a b)' ; t5 = t5' ;**
  - **t5 = a b ;**

Reeglitel põhinevad teisendused

- **De Morgani & topelteituse seadused**



- **t6 = a' b' c ;**
- **t6a = a' b' ; t6 = t6a c ;**
- **t6a = (a + b)' ; t6 = t6a c ;**

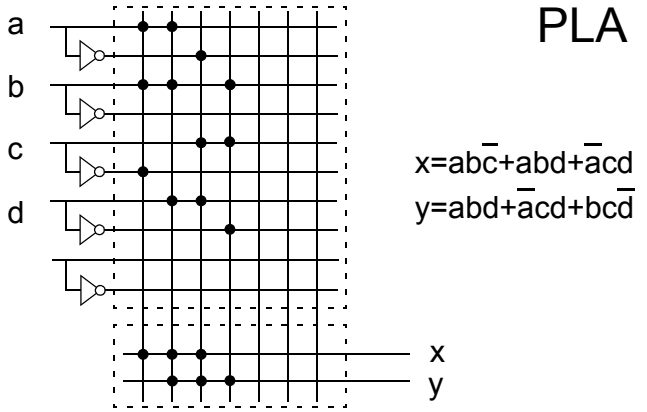
- **t7 = a b c' ;**
- **t7a = a b ; t7 = t7a c' ;**
- **[t7ai = (a b)' ; t7 = t7ai' c' ;]**
- **t7ai = (a b)' ; t7 = (t7ai + c)' ;**

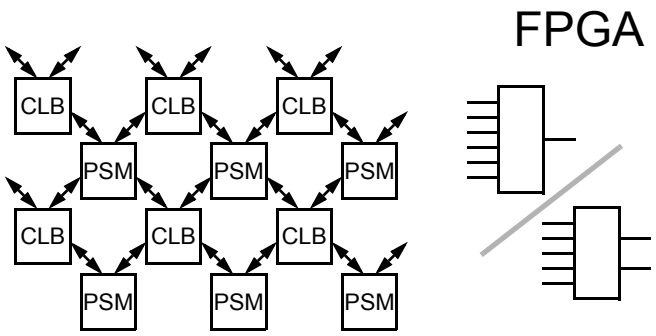
- **t8 = a b c ;**
- **[t8a = a b ; t8 = t8a c ;]**
- **t8ai = (a b)' ; t8 = (t8ai + c')' ;**

Erijuhud – programmeeritav loogika

PLD – Programmable Logic Device

- **PLA – Programmable Logic Array**
  - Programmeeritavad loogikamaatriksid (ja ühendused)
  - Loogikamaatriks
    - mitme sisendiga ja mitme väljundiga loogika-funktsioonide süsteem, implikantide arv piiratud
  - Tükeldamine vastava suurusega funktsioonideks
- **FPGA – Field Programmable Gate Array**
  - väliprogrammeeritav loogika (ka korduvprogr. loogika)
  - Programmeeritavad loogikaplokid (CLB)
    - ... ja programmeeritavad ühendused (PSM)
  - Xilinx – Spartan, Artix, Kintex & Virtex seeriad
    - CLB – neli 6->1 / 5->2 funktsiooni
  - Tükeldamine 5- või 6-sisendiga funktsioonideks





Funktsioonide teisendused – kahendmeetodid

- Loogikafunktsioonide omaduste kasutamine
- Võimalik kasutada osaliselt määratust (don't-care)
- Aegajalt (liigagi) keeruline

- Kahendasendus (boolean substitution)
  - lähtefunktsioonid
    `h=a+bcd+e; q=a+cd`
  - tulemus
    `h=a+bq+e`
  - sest
    `a+bq+e=a+b(a+cd)+e=a+bcd+e` `[a+b(a+cd)+e=a+ab+bcd+e=a+bcd+e]`
  - või hoopis?!
    `h=a+bcd+e; q=cd+e -> h=a+bq+e`
    `a+bq+e=a+b(cd+e)+e=a+bcd+e` `[a+b(cd+e)+e=a+bcd+be+e=a+bcd+e]`

Funktsioonide teisendused – algebralise meetodid

- Funktsioone vaadeldakse kui *polünoome*
- Kasutatakse polünoomide algebra omadusi
- Lihtsam, kiirem, kui nõrgem optimeerimisvõime

- Algebraline asendus (algebraic substitution)
  - lähtefunktsioon
    `t=ka+kb+e`
  - tulemus
    `t=kq+e`
  - sest
    `q=a+b`

Algebralised meetodid

- *Polünoom* (polynomial) – korrutiste summa
- *Monoom* (monomial), üksliige – üksik korrutis e. kuup (e. implikant)
- Ainult distributiivsuse seadus kasutusel
  - a(b+c)=ab+ac, kuid a+bc≠(a+b)(a+c)
- Täiendid pole defineeritud
  - muutuja täiendit vaadeldakse kui lisa muutujat
- Määramatused pole defineeritud
- Operatsioonid ainult avaldistega, mille muutujate hulgad ei kattu
- Liiaste kuupide eemaldamine pole võimalik
  - (a+b) ja (c+d) → (a+b)(c+d)=ac+ad+bc+bd OK!
  - (a+b) ja (a+c) → (a+b)(a+c)=aa+ac+ba+bc≠a+bc ei
  - (a+b) ja ($\bar{a}$+c) → (a+b)($\bar{a}$+c)=a$\bar{a}$+ac+b$\bar{a}$+bc≠ac+b$\bar{a}$ ei

Algebraline jagatis

- Kaks algebralist avaldist
- *jagatav* (dividend), *jagaja* (divisor), *jagatis* (quotient), *jääk* (remainder)
- $f_{jagatis} = f_{jagatav} / f_{jagaja}$, kui
- $f_{jagatav} = f_{jagaja} \cdot f_{jagatis} + f_{jääk}$
- $f_{jagaja} \cdot f_{jagatis} \neq \varnothing$
- ning f_{jagaja} ja $f_{jagatis}$ muutujate hulgad ei kattu ($sup(f_{jagaja}) \cap sup(f_{jagatis}) = \varnothing$)
- $f_{jagatav} = ac + ad + bc + bd + e$ & $f_{jagaja} = a + b$
- $f_{jagatis} = c + d$ & $f_{jääk} = e$
- $(a+b)(c+d)+e=f_{jagatav}$ & $\{a,b\} \cap \{c,d\} = \varnothing$
- Mitte-algebraline jagatis – $f_i = a + bc$ & $f_j = a + b$
  - $(a+b)(a+c)=f_i$ kuid $\{a,b\} \cap \{a,c\} \neq \varnothing$

Jagamisalgoritm

- A = { C_j^A, j=1,2,...,l} – jagatava kuupide hulk
- B = { C_i^B, i=1,2,...,n} – jagaja kuupide hulk
- Jagatis Q ja jääk R on kuupide summad

```
ALGEBRAIC_DIVISION (A,B) {
    for (i=1 to n) {
        D={C_j^A such that C_j^A ⊇ C_i^B};
        if (D==∅) return(∅,A);
        D_i=D with var. in sup(C_i^B) dropped;
        if (i==1)   Q=D_i;   else Q=Q∩D_i;
    }
    R=A-Q×B;
    return(Q,R);
}
```

Jagamine – näide #1

- $f_{jagatav} = ac + ad + bc + bd + e$ & $f_{jagaja} = a + b$
- A = {ac, ad, bc, bd, e} & B = {a, b}
- *i = 1*
  - C_1^B=a, D={ac, ad} & D_1={c,d}
  - Q = {c,d}
- *i = 2*
  - C_2^B=b, D={bc,bd} & D_2={c,d}
  - Q = {c,d}∩{c,d} = {c,d} -- *kuup vastab elemendile!*
- Tulemus
  - Q = {c,d} & R = {e}
  - $f_{jagatis}$=c+d & $f_{jääk}$=e

Jagamine – näide #2

- $f_{jagatav} = axc + axd + bc + bxd + e$ & $f_{jagaja} = ax + b$
- $A = \{axc, axd, bc, bxd, e\}$ & $B = \{ax, b\}$
- *i = 1*
  - $C_1^B = ax$, $D = \{axc, axd\}$ & $D_1 = \{c,d\}$
  - $Q = \{c,d\}$
- *i = 2*
  - $C_2^B = b$, $D = \{bc,bxd\}$ & $D_2 = \{c,xd\}$
  - $Q = \{c,d\} \cap \{c,xd\} = \{c\}$ -- *kuup vastab elemendile!*
- Tulemus
  - $Q = \{c\}$ & $R = \{axd,bxd,e\}$
  - $f_{jagatis} = c$ & $f_{jääk} = axd+bxd+e$

Jagamine – mis siis ikkagi toimub?

- A = ac + ad + bc + bd + e & B = a + b
  - (1) a (c+d) + bc + bd + e
  - (2) a (c+d) + b (c+d) + e → (c+d) "∩" (c+d) "=" (c+d)
  - (R) [ac+ad+bc+bd+e] "-" [(a+b) (c+d)] "="
    "=" [ac+ad+bc+bd+e] "-" [ac+ad+bc+bd] "=" [e]
  - ac+ad+bc+bd+e = a(c+d)+b(c+d)+e = (c+d)(a+b)+e

- A = axc + axd + bc + bxd + e & B = ax + b
  - (1) ax (c+d) + bc + bxd + e
  - (2) ax (c+d) + b (c+xd) + e → (c+d) "∩" (c+xd) "=" (c)
  - (R) [axc+axd+bc+bxd+e] "-" [(ax+b) (c)] "="
    "=" [axc+axd+bc+bxd+e] "-" [axc+bc] "=" [axd+bxd+e]
  - axc+axd+bc+bxd+e= ax(c+d)+b(c+xd)+e = ax(c)+b(c)+axd+bxd+e = c(ax+b)+axd+bxd+e

Jagatise eksisteerimine?

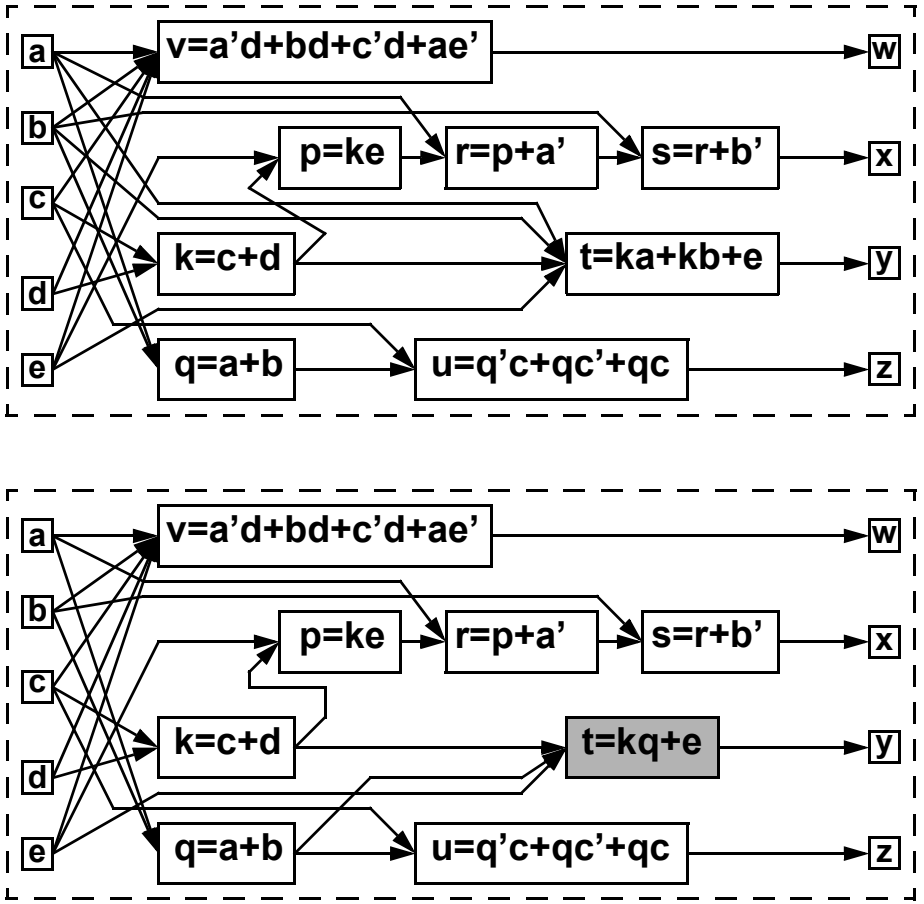
- Teoreem
- Antud kaks algebralist avaldist f_i ja f_j
- f_i / f_j on tühi, kui üks järgnevaist tingimustest on täidetud:
  - f_j sisaldab muutujat, mida pole f_i-s;
  - f_j sisaldab kuupi, mille tugimuutujad ei sisaldu üheski f_i kuubi tugimuutujate hulgas ($\exists$ sup(C^j)$\not\subset$sup(C^i), $\forall C^i \in f_i$);
  - f_j sisaldab rohkem liikemid kui f_i;
  - suvalist muutujat on f_j-s rohkem kui f_i-s.
- Kasutusel kiireks kontrolliks
  - jagatist ei pruugi ikkagi eksisteerida – ac + be / a + b

Jagatise eksisteerimine?

- Antud kaks algebralist avaldist f_i ja f_j
- f_i / f_j on tühi, kui üks järgnevaist tingimustest on täidetud:
  - f_j sisaldab muutujat, mida pole f_i-s;
    ab + cd / a + e — a(b) + e(?) + cd
  - f_j sisaldab kuupi, mille tugimuutujad ei sisaldu üheski f_i kuubi tugimuutujate hulgas ($\exists\ sup(C^j) \not\subset sup(C^i), \forall C^i \in f_i$);
    abc + def / ab + ad — ab(c) + ad(?) + def
  - f_j sisaldab rohkem liikemid kui f_i;
    ab + cd / a + b + c — a(b) + b(?) + c(d)
  - suvalist muutujat on f_j-s rohkem kui f_i-s.
    abc + ade + bcd / ab + ad + ac — ab(c) + ad(e) + ac(?)
  - Osaline kattumine? ab + ac + bc / a + b = ? — ab + ac + bc = ab + (a+b) c

Asendamine

- Vaadeldakse avaldiste paare
- Jagamine suvalises järjekorras
- Kui jagatis pole tühi, siis:
  - ennustatakse pindala/viite muutust
  - $f_{jagatav}$ asemele tuleb $f_{jagaja} \cdot f_{jagatis} + f_{jääk}$
  - $f_t = ka + kb + e$ ($f_{jagatav}$)
  - $f_q = a + b$ (f_{jagaja})
  - $f_{jagatis} = k$ & $f_{jääk} = e$
  - $f_t = kq + e$



Eraldamine

- Ühiste alam-avaldiste otsimine
  - üksik kuup (monoom) / mitmik-kuup (*tuum* e. kernel)
- Sobivate *jagajate* leidmine
- *Kuubivaba* (cube-free) avaldis
  - pole võimalik faktoriseerida kuupi kasutades
    - ehk siis, mitte ühtegi muutujat ei saa sulgude ette tuua
- Avaldise *tuum*
  - avaldise kuubivaba jagatis, kui jagaja on kuup (*kaas-tuum* e. co-kernel))
- Avaldise *tuumade hulk* K(f)
  - kahe (või enama) avaldise ühine mitmik-kuup jagaja – tuumade hulkade ühisosa
- Tuumade leidmine
  - naiivne – üritatakse leida jagatised muutujate kombinatsioonidele
  - rekursioon – tuumade tuumad on samuti tuumad

Tuumad – näide

- f_x = ace + bce + de + g
  - f_x / a → ce → ei ole kuubivaba
  - f_x / b → ce → ei ole kuubivaba
  - f_x / c → ae + be → ei ole kuubivaba
  - f_x / ce → a + b → kuubivaba → *tuum*
  - f_x / d → e → ei ole kuubivaba
  - f_x / e → ac+bc+d → kuubivaba → *tuum*
  - f_x / g → 1 → ei ole kuubivaba
  - f_x / 1 → ace+bce+de+g → kuubivaba → *tuum*

- K(fx) = { (a+b), (ac+bc+d), (ace+bce+de+g) }

Eraldamine – näide

- $f_x = ace + bce + de + g$
- $f_y = ad + bd + cde + ge$
- $f_z = abc$

- $K(f_x) = \{ (a+b), (ac+bc+d), (ace+bce+de+g) \}$
- $K(f_y) = \{ (a+b+ce), (cd+g), (ad+bd+cde+ge) \}$
- $K(f_z) = \{\}$

- $f_w = a+b$
- $f_x = wce + de + g$
- $f_y = wd + cde + ge$
- $f_z = abc$

Dekompositsioon

- f_x = ace + bce + de + g;
- f_t = ac + bc + d; f_x = te + g;
- f_s = a + b; f_t = sc + d; f_x = te + g;

- **Tuumadel põhinev dekompositsioon**
  - **avaldist jagatakse rekursiivselt**

- f_x = ace + bce + de + g
- $K(f_x)$ = { (a+b), (ac+bc+d), (ace+bce+de+g) }
- f_t = ac + bc + d; f_x = te + g;
- $K(f_t)$ = { (a+b), (ac+bc+d) }
- f_s = a + b; f_t = sc + d; f_x = te + g;

Viite mudel & viite minimeerimine

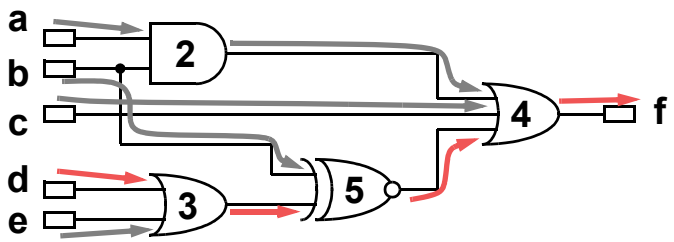
- Sünteesi tulemuse kontroll
  - viide on nõutavast väiksem ja sisend/väljund signaalide ajastus on korrektne
- Viite mudel
  - sõltub loogikalülide (alamavaldiste) viite mudelitest
  - topoloogilised mudelid (funktsionaalsuse arvestamisega või ilma)
- Minimeerimine
  - vähima pindala puhul peab viide jääma etteantud piiridesse
  - viite minimeerimisel peab pindala jääma etteantud piiridesse
  - teisendused elementidel, mis asuvad kriitilisel teel (kriitilistel teedel)
  - pea-aegu kriitilised teed – teed, mille viide on lähedane kriitilisega
- Topoloogiline kriitiline tee (topological critical path)
  - tee maksimaalse viitega / tee minimaalse sobivusega
  - väär kriitiline tee – signaali muutus ei levi mööda vastavat teed

Viite arvutamine

- Loogikaelemendi (alamavaldise) viite mudel
  - virtuaalsed loogikalülid – loogika-avaldised
  - lihtsaim mudel – ühikviide sõlme kohta
  - täpsustatud mudelid – sõltuvad koormatusest (fanout) ja/või avaldise keerukusest
- Andmete valmisoleku-ajad (data-ready time) - t_i
  - sisenditest väljundite suunas arvutamine
    - millal andmed/signaalid kohale jõuavad, nt. hilistumine eelnevates moodulites
  - $t_i = d_i + \max_{j|(j,i)\in E} t_j$
- Nõutud andmete valmisoleku-ajad (required data-ready time) - $\bar{t}_i$
  - väljunditest sisendite suunas arvutamine
    - millal andmed/signaalid peavad valmis olema, nt. taktiperioodi piirang
  - $\bar{t}_i = \min_{j|(i,j)\in E} (\bar{t}_j - d_j)$
- Sobivus (lõtvus, slack) – $s_i = \bar{t}_i - t_i$
  - erinevus nõutud ja tegelike andmete valmisoleku-aegade vahel

Viite arvutamine – andmete valmisoleku-ajad

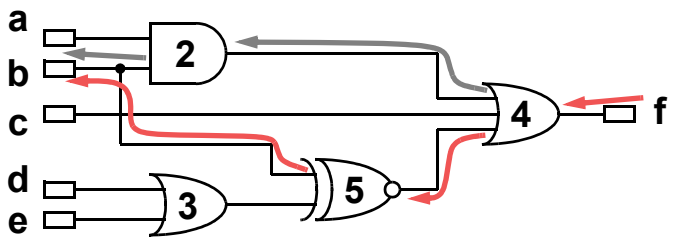
- $t_i = d_i + \max_{j|(j,i)\in E} t_j$
- sisendid → väljundid
  - $t_a=t_b=t_e=0$; $t_c=5$; $t_d=1$;
  - $t_2 = \max(t_a,t_b)+d_2 = \max(0,0) + 2 = 2$
  - $t_3 = \max(t_d,t_e)+d_3 = \max(1,0) + 3 = 4$
  - $t_5 = \max(t_b,t_3)+d_5 = \max(0,4) + 5 = 9$
  - $t_4 = \max(t_2,t_c,t_5)+d_4 = \max(2,5,9) + 4 = 13$
  - $t_f = \max(t_4)+d_f = \max(13) + 0 = 13$
- Suurim viide – 13



Viite arvutamine – andmete nõutavad valmisoleku-ajad

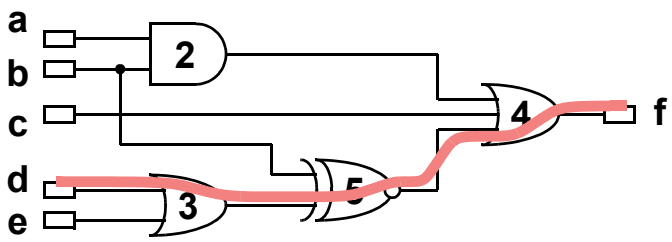
- $\bar{t}_i = \min_{j|(i,j)\in E} (\bar{t}_j - d_j)$

- väljundid → sisendid
  - $t_a=t_b=t_e=0$; $t_c=5$; $t_d=1$; $\bar{t}_f=15$;
  - $\bar{t}_4 = \min(\bar{t}_f-d_f) = \min(15-0) = 15$
  - $\bar{t}_5, \bar{t}_2 = \min(\bar{t}_4-d_4) = \min(15-4) = 11$
  - $\bar{t}_3 = \min(\bar{t}_5-d_5) = \min(11-5) = 6$
  - $\bar{t}_a = \min(\bar{t}_2-d_2) = \min(11-2) = 9$
  - $\bar{t}_b = \min(\bar{t}_2-d_2,\bar{t}_5-d_5) = \min(11-2,11-5) = 6$
  - $\bar{t}_c = \min(\bar{t}_4-d_4) = \min(15-4) = 11$
  - $\bar{t}_d, \bar{t}_e = \min(\bar{t}_3-d_3) = \min(6-3) = 3$



Viite arvutamine – sobivus ja kriitiline tee

- Sobivus (slack) – $s_i = \bar{t}_i - t_i$
  - $s_a = \bar{t}_a - t_a = 9 - 0 = 9$
  - $s_b = \bar{t}_b - t_b = 6 - 0 = 6$
  - $s_c = \bar{t}_c - t_c = 11 - 5 = 6$
  - $s_d = \bar{t}_d - t_d = 3 - 1 = 2$
  - $s_e = \bar{t}_e - t_e = 3 - 0 = 3$



sõlm	a	b	c	d	e	2	3	5	4	f
t_x	0	0	5	1	0	2	4	9	13	13
$\bar{t}_x$	9	6	11	3	3	11	6	11	15	15
s_x	9	6	6	2	3	7	2	2	2	2

- Kriitiline tee – tee väikseima sobivusega

Viite arvutamine – näited

`ai = a'`	t_{ai} = max(0)+1.5 = 1.5
`bi = b'`	t_{bi} = max(0)+1.5 = 1.5
`ci = c'`	t_{ci} = max(0)+1.5 = 1.5
`di = d'`	t_{di} = max(0)+1.5 = 1.5
`t1 = ai bi d`	t_{t1} = max(1.5,1.5,0)+2.5 = 4.0
`t2 = a b c`	t_{t2} = max(0,0,0)+2.5 = 2.5
`t3 = bi ci`	t_{t3} = max(1.5,1.5)+2.0 = 3.5
`t4 = b d`	t_{t4} = max(0,0)+2.0 = 2.0
`t5 = a ci d`	t_{t5} = max(0,1.5,0)+2.5 = 4.0
`t6 = b ci`	t_{t6} = max(0,1.5)+2.0 = 3.5
`t7 = ai c di`	t_{t7} = max(1.5,0,1.5)+2.5 = 4.0
`k1 = t1+t2`	t_{k1} = max(4.0,2.5)+2.0 = 6.0
`k = k1+t3+t4`	t_k = max(6.0,3.5,2.0)+2.5 = 8.5
`l = k1+t5+t6`	t_l = max(6.0,4.0,3.5)+2.5 = 8.5
`m1 = t7+t6`	t_{m1} = max(4.0,3.5)+2.0 = 6.0
`m = k1+m1`	t_m = max(6.0,6.0)+2.0 = 8.0
`n1 = t5+t4+t2`	t_{n1} = max(4.0,2.0,2.5)+2.5 = 6.5
`n = m1+n1`	t_n = max(6.0,6.5)+2.0 = 8.5

4 NOT, 3*2-AND, 4*3-AND, 4*2-OR, 3*3-OR
literaale: 4+18+17=39
eq.gates: 4*1.5+3*2.0+4*2.5+4*2.0+3*2.5=37.5
maks.viide: 8.5, kr.tee nt. *a-ai-t1-k1-k* või *c-ci-t5-n1-n*

`ai = a'`	t_{ai} = max(0)+1.5 = 1.5
`bi = b'`	t_{bi} = max(0)+1.5 = 1.5
`ci = c'`	t_{ci} = max(0)+1.5 = 1.5
`di = d'`	t_{di} = max(0)+1.5 = 1.5
`t1 = a+ci`	t_{t1} = max(0,1.5)+2.0 = 3.5
`t2 = bi ci`	t_{t2} = max(1.5,1.5)+2.0 = 3.5
`t3 = b d`	t_{t3} = max(0,0)+2.0 = 2.0
`t4 = ai bi d`	t_{t4} = max(1.5,1.5,0)+2.5 = 4.0
`t5 = a b c`	t_{t5} = max(0,0,0)+2.5 = 2.5
`t6 = t1 b`	t_{t6} = max(3.5,0)+2.0 = 5.5
`t7 = a ci d`	t_{t7} = max(0,1.5,0)+2.5 = 4.0
`t8 = ai c di`	t_{t8} = max(1.5,0,1.5)+2.5 = 4.0
`k1 = t2+t3`	t_{k1} = max(3.5,2.0)+2.0 = 5.5
`k = k1+t4+t5`	t_k = max(5.5,4.0,2.5)+2.5 = 8.0
`l = t6+t7+t4`	t_l = max(5.5,4.0,4.0)+2.5 = 8.0
`m = t6+t8+t4`	t_m = max(5.5,4.0,4.0)+2.5 = 8.0
`n1 = t6+t3`	t_{n1} = max(5.5,2.0)+2.0 = 7.5
`n = n1+t8+t7`	t_n = max(7.5,4.0,4.0)+2.5 = 10.0

4*NOT, 3*2-AND, 4*3-AND, 3*2-OR, 4*3-OR
literaale: 4+18+18=40
eq.gates: 4*1.5+3*2.0+4*2.5+3*2.0+4*2.5=38.0
maks.viide: 10.0, kr.tee *c-ci-t1-t6-n1-n*

Viite arvutamine – näited

`ai = a'`	t_{ai} = max(0)+1.5 = 1.5
`bi = b'`	t_{bi} = max(0)+1.5 = 1.5
`ci = c'`	t_{ci} = max(0)+1.5 = 1.5
`di = d'`	t_{di} = max(0)+1.5 = 1.5
`t1 = b ci`	t_{t1} = max(0,1.5)+2.0 = 3.5
`t2 = ai di`	t_{t2} = max(1.5,1.5)+2.0 = 3.5
`t3 = a bi c`	t_{t3} = max(0,1.5,0)+2.5 = 4.0
`t4 = bi ci di`	t_{t4} = max(1.5,1.5,1.5)+2.5 = 4.0
`t5 = ai b d`	t_{t5} = max(1.5,0,0)+2.5 = 4.0
`t6 = ai ci`	t_{t6} = max(1.5,1.5)+2.0 = 3.5
`ki = t1+t2+t3`	t_{ki} = max(3.5,3.5,4.0)+2.5 = 6.5
`li = t2+t4+t3`	t_{li} = max(3.5,4.0,4.0)+2.5 = 6.5
`mi = t5+t4+t3`	t_{mi} = max(4.0,4.0,4.0)+2.5 = 6.5
`ni = t6+t4+t3`	t_{ni} = max(3.5,4.0,4.0)+2.5 = 6.5
`k = ki'`	t_k = max(6.5)+1.5 = 8.0
`l = li'`	t_l = max(6.5)+1.5 = 8.0
`m = mi'`	t_m = max(6.5)+1.5 = 8.0
`n = ni'`	t_n = max(6.5)+1.5 = 8.0

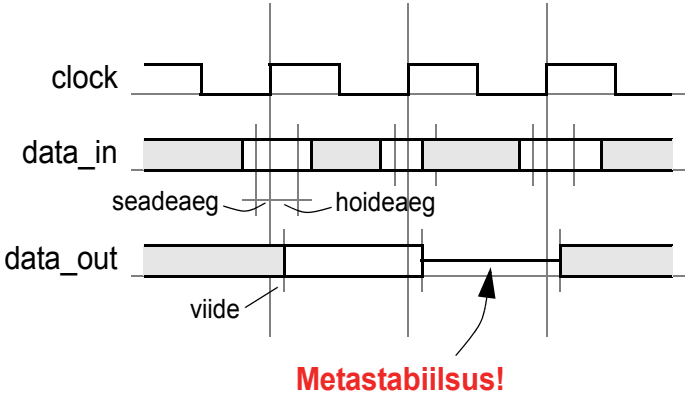
8 NOT, 3*2-AND, 3*3-AND, 4*3-OR
literaale: 8+15+12=35
eq.gates: 8*1.5+3*2.0+3*2.5+4*2.5=35.5
maks.viide: 8.0, kr.tee nt. *a-ai-t6-ki-k* või *c-ci-t4-ni-n*

`ai = a'`	t_{ai} = max(0)+1.5 = 1.5
`bi = b'`	t_{bi} = max(0)+1.5 = 1.5
`ci = c'`	t_{ci} = max(0)+1.5 = 1.5
`di = d'`	t_{di} = max(0)+1.5 = 1.5
`t1 = (bi+c)'`	t_{t1} = max(1.5,0)+1.5 = 3.0
`t2 = (a+d)'`	t_{t2} = max(0,0)+1.5 = 1.5
`t3 = (ai+b+ci)'`	t_{t3} = max(1.5,0,1.5)+2.0 = 3.5
`t4 = (b+c+d)'`	t_{t4} = max(0,0,0)+2.0 = 2.0
`t5 = (a+bi+di)'`	t_{t5} = max(0,1.5,1.5)+2.0 = 3.5
`t6 = (a+c)'`	t_{t6} = max(0,0)+1.5 = 1.5
`k = (t1+t2+t3)'`	t_k = max(3.0,1.5,3.5)+2.0 = 5.5
`l = (t2+t4+t3)'`	t_l = max(1.5,2.0,3.5)+2.0 = 5.5
`m = (t5+t4+t3)'`	t_m = max(3.5,2.0,3.5)+2.0 = 5.5
`n = (t6+t4+t3)'`	t_n = max(1.5,2.0,3.5)+2.0 = 5.5

4*NOT, 3*2-NOR 7*3-NOR
literaale: 4+27=31
eq.gates: 4*1.5+3*1.5+7*2.0=24.5
maks.viide: 5.5, kr.tee nt. *a-ai-t3-k* või *d-di-t5-m*

Kriitiline tee ja mäluelemendid (registrid)

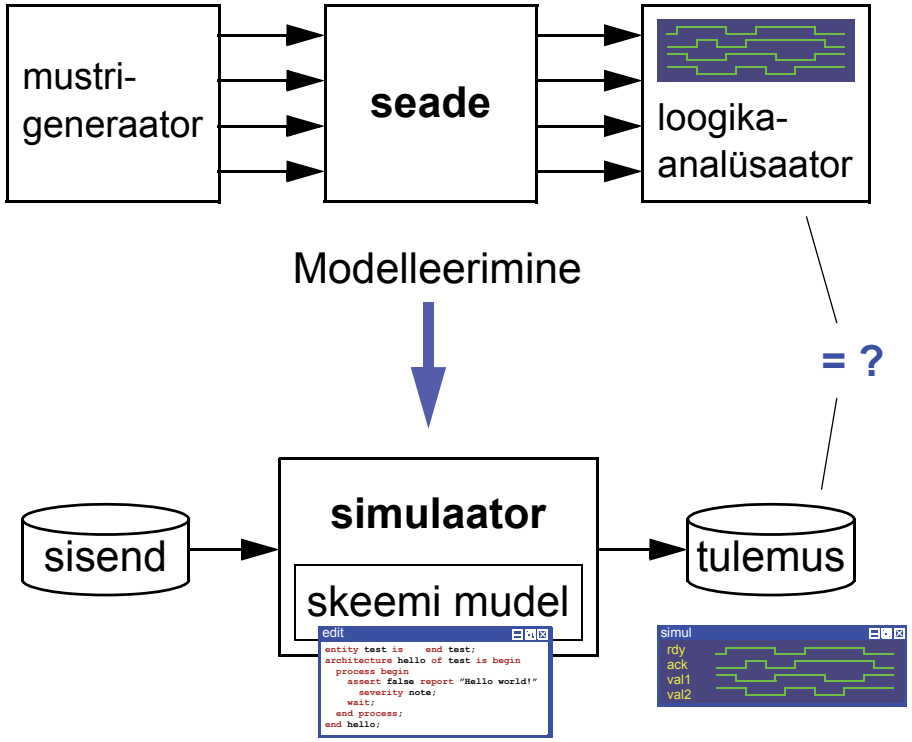
- Seade- & hoideajad mäluelemtides (trigerites)
  - seadeaeg – sisendandmed peavad olema stabiilsed mingi aeg enne taktsignaali aktiivset fronti
  - hoideaeg – sisendandmed peavad olema stabiilsed mingi aeg pärast taktsignaali aktiivset fronti
  - Põhjuseks mäluelementide siseste signaalide erinevad levimisajad
  - Tagajärjeks võib olla metastabiilsus – väljund on ‘0’ ja ‘1’ vahel
  - seadeaeg – setup time
  - hoideaeg – hold time



- Sobivuse arvutamisel peab arvestama ka mäluelementide ajalisi parameetreid:
  - andmete valmisoleku aeg: $t_x + d_{ff}$
  - nõutav andmete valmisoleku aeg: $\bar{t}_x - t_{ff,setup}$
  - sobivus (slack): $s_x - d_{ff} - t_{ff,setup}$

Digitaalsüsteemide modelleerimine

- Tulemuse kontroll?
  - Modelleerimine
    - mudeli korrektsus & simuleeritavus?
- Kasutatav keel?
  - C++?
  - Paralleelsus!
    - C++ lõimed?
    - ADA?
    - Spetsiaalkeeled?
    - Veel üks keel! :-(
- Tarkvara vs riistvara
  - üksik kontrollvoog / mitu kontrollvoogu ?
    - ühel protsessoril / mitmel protsessoril ?
    - hajutatud ülesanded – protsessorite vaheline kaugus
    - infovahetus (muutujad), sünkroniseerimine (semaforid) jne.

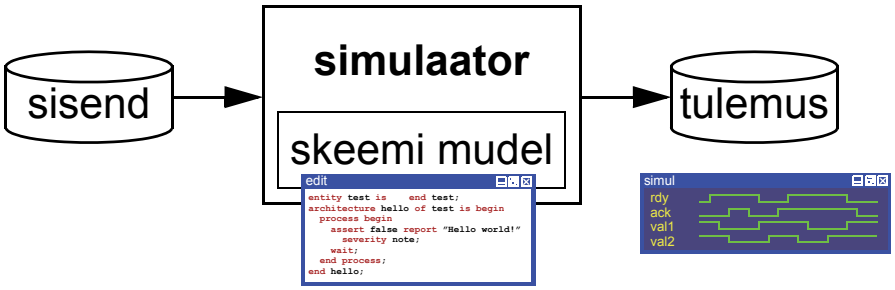


Riistvara modelleerimine

- Paralleelselt töötavad moodulid
  - infovahetus (signaalid, kanalid) / sünkroniseerimine (erisignaalid, sündmused)
- Mudel üksikul protsessoril, tulemus peaks olema korratav
- Lihtne põhimõte – signaalil uus ja vana väärtus (simulatsioonitsükkel)
  - 1 - arvutatakse kõikide signaalide uued väärtused
  - 2 - signaalidele omistatakse uued väärtused
- Simulatsioonimudelid
  - tsükkelpõhised (cycle-based) – igal simulatsioonitsüklil uus arvutus
  - sündmuspõhised (event-based) – uus arvutus ainult muutunud sisendite korral
  - viite modelleerimine
    - ühikviide (unit-delay)
    - nullviide (zero-delay)
    - deltaviide (delta-delay)

Modelleerimine – VHDL & kodutöö #1

- Teisenduste korrektsuse kontroll
  - Tõeväärtustabel
  - Minimeeritud funktsioonid
  - Optimeeritud funktsioonid
    - pluss vahe-etapid
  - Sisendsignaalide generaator




```
-- Sisendsignaalid (sammuga 10 ns)
a <= '0' after 0 ns, '1' after 80 ns, '0' after 160 ns;
b <= '0' after 0 ns, '1' after 40 ns, '0' after 80 ns, '1' after 120 ns;
c <= '0' after 0 ns, '1' after 20 ns, '0' after 40 ns, '1' after 60 ns,
     '0' after 80 ns, '1' after 100 ns, '0' after 120 ns, '1' after 140 ns;
d <= '0' after 0 ns, '1' after 10 ns, '0' after 20 ns, '1' after 30 ns,
     '0' after 40 ns, '1' after 50 ns, '0' after 60 ns, '1' after 70 ns,
     '0' after 80 ns, '1' after 90 ns, '0' after 100 ns, '1' after 110 ns,
     '0' after 120 ns, '1' after 130 ns, '0' after 140 ns, '1' after 150 ns;
```

  - Tulemuste analüüs / võrdlus
    - mitu realisatsiooni korraga?
    - VHDL – entity / architecture paarid

Modelleerimine – VHDL & kodutöö #1

- **Mitme versiooni korraga simuleerimine**
  - **Iga vahe-etapp eraldi komponendina**

```
library IEEE; use IEEE.std_logic_1164.all;
entity f_system is
  port ( a, b, c, d: in std_logic;
         k, l, m, n: out std_logic );
end entity f_system;
architecture tabel of f_system is begin
  process (a, b, c, d)
    variable in_word, out_word:
      std_logic_vector (3 downto 0);
  begin
    in_word := a & b & c & d;
    case  in_word  is
      when "0000" => out_word := "1-00";
      when "0001" => out_word := "01-0";
      when "0010" => out_word := "11-1";
      when "0011" => out_word := "0-01";
      when "0100" => out_word := "1110";
      ...
      when "1110" => out_word := "-000";
      when "1111" => out_word := "1011";
      when others => out_word := "----";
    end case;
    k <= out_word(3);    l <= out_word(2);
    m <= out_word(1);    n <= out_word(0);
  end process;
end architecture tabel;
```

```
architecture espresso of f_system is begin
  k <= ((not a) and (not d)) or (a and c and d) o
       ((not a) and b and (not c)) or
       (b and (not c) and (not d));
  ...
  n <= ((not a) and c and (not d)) or
       (a and c and d) or
       (a and (not b) and (not d)) or
       ((not b) and c);
end architecture espresso;
```

```
architecture opti of f_system is
  signal ai, bi, ci, di: std_logic;
  signal t0i, t2i, ... t7i, t9i: std_logic;
  signal t51, t52, ... t3t6i, t5t6i: std_logic;
begin
  ai <= not (a and a);    bi <= not (b and b);
  ci <= not (c and c);    di <= not (d and d);
  t0i <= a or d;          t2i <= not (ai and c);
  ...
  t1t8i <= not (di and t54);
  t3t6i <= not (t52 and ci);
  t5t6i <= not (t51 and b and ci);
  k <= not (t0i and t4i and t5t6i);
  l <= not (t2i and t3t6i);
  m <= not (t1t8i and t6i and t7i);
  n <= not (t1t8i and t4i and t9i);
end architecture opti;
```

Modelleerimine – VHDL & kodutöö #1

- **Mitme versiooni korraga simuleerimine**

```
library IEEE; use IEEE.std_logic_1164.all;
entity test2 is    end entity test2;
library IEEE; use IEEE.std_logic_1164.all;
architecture bench of test2 is
  signal a, b, c, d, k, l, m, n: std_logic;
  signal k2, k3, l2, l3, m2, m3, n2, n3: std_logic;
  component f_system
    port ( a, b, c, d: in std_logic;
           k, l, m, n: out std_logic );
  end component;
  for U1: f_system use entity work.f_system(tabel);
  for U2: f_system use entity work.f_system(espresso);
  for U3: f_system use entity work.f_system(opti);
begin
  -- Input signaals (after every 10 ns)
  a <= '0' after 0 ns, '1' after 80 ns, '0' after 160 ns;
  b <= '0' after 0 ns, '1' after 40 ns, ...
  c <= '0' after 0 ns, '1' after 20 ns, ...
  d <= '0' after 0 ns, '1' after 10 ns, ...
  -- System of Boolean functions
  U1: f_system port map (a, b, c, d, k, l, m, n);
  U2: f_system port map (a, b, c, d, k2, l2, m2, n2);
  U3: f_system port map (a, b, c, d, k3, l3, m3, n3);
end architecture bench;
```