Kodutöö #1 - näidislahendus

Matriklinumber: 999999

Funktsioonide leidmine:
1. 5 : 2540BBCF : C6AE945 : 0,2,4,5,b,c,f / 6,9,a,e
2. 7 : 1471C269 : 6D09623 : 1,2,4,6,7,9,c / 0,3,d
3. B : 4F556D8D : 1A71CF2F : 4,5,6,8,d,f / 1,2,7,a,c
4. D : 82F386AB : 2BA68239 : 2,3,6,8,a,b,f / 9

Minimeerimine

Lähte-ülesanne	espresso tulemus	espr. v2 (-Dexact)	espr. v3 (#1010)	espr. v4 (#1011)
0000 1-00	-001 0100	-001 0100	10-0 0110	0101 1100
0001 01-0	-100 1100	010- 1000	0101 1100	1-11 0001
0010 11-1	1-11 1001	1-11 1001	01-1 0001	-100 1010
0011 0-01	10-0 0011	10-0 0011	11-0 1001	-1-1 0010
0100 1110	010- 1010	-1-1 0010	1-1- 1100	-01- 0001
0101 1010	-1-1 0010	0-10 0011	0--0 1000	0-10 0011
0110 -111	0-10 0011	0-1- 0100	-1-1 0010	0--0 1000
0111 01-0	0-1- 0100	-100 1110	0-0- 0001	10-0 0111
1000 0011	-01- 0001	-01- 0001	-10- 0001	1-1- 1100
1001 -10-	0--0 1000	0--0 1000	01-- 0010	
1010 -0-1				
1011 1001				
1100 11-0				
1101 0-10				
1110 -000				
1111 1011				

Tulemuste võrdlus

Kõik espresso tulemused on kas 10 või 9 implikandiga. Kuna loogikaelementidel on kuni 3 sisendit, siis on oluline, ka implikandid oleksid 3 või vähema muutujaga ja väljund sõltuks mitte rohkem kui 3-st implikandist. Esimesel ja teisel versioonil on sisendite arv korrar, kuid kolm väljundit vajavad 4 implikanti, neist y3 ja y4 omavad kahte ühist implikanti (10-0 & 0-10), mida saaks eelnevalt grupeerida (2-OR). Väljund y1 vajab siiski 4 implikanti. Kasutades käsku "espresso -Dopoall", on võimalik proovida erinevate väljundite inverteerimist. Tulemustele on lisatud "c" ja "g", mis tähistavavad vastavalt vähimat implikantide ja elementide arvu.

```
phase 0000 -- c=10(0) in=28 out=14 tot=42
phase 0001 -- c=10(0) in=29 out=16 tot=45
phase 0010 -- c=11(0) in=29 out=14 tot=43
phase 0011 -- c=10(0) in=28 out=15 tot=43
phase 0100 -- c=10(0) in=28 out=15 tot=43
phase 0101 -- c=10(0) in=28 out=16 tot=44
phase 0110 -- c=9(0)  in=26 out=17 tot=43 c
phase 0111 -- c=9(0)  in=25 out=17 tot=42 c
phase 1000 -- c=9(0)  in=25 out=15 tot=40 c
phase 1001 -- c=10(0) in=26 out=14 tot=40
phase 1010 -- c=10(0) in=25 out=14 tot=39   g
phase 1011 -- c=9(0)  in=24 out=15 tot=39 c g
phase 1100 -- c=9(0)  in=25 out=17 tot=42 c
phase 1101 -- c=10(0) in=27 out=15 tot=42
phase 1110 -- c=11(0) in=29 out=16 tot=45
phase 1111 -- c=10(0) in=26 out=15 tot=41
```

Kaks vähima elementide arvuga varianti (vt. "g") on siis lahendused 3 ja 4, mis vajavad 4-muutjalist implikanti (0101) ning kas y1 & y4 või y1, y3 & y4 jaoks nelja implikanti. Samas on mõlemal juhul y1 & y2 kahe ühise implikandiga (0101 & 1-1-), kuid 4. variandil on lisaks ka y3 & y4 kahe ühise implikandiga (0-10 & 10-0). Allpool olevas tabelis on implikantide loetelule tähistatud '!'-dega implikandid (AND) ja väljundid (OR), mis vajaksid 4-sisendiga loogikaelemente. Lisaks on implikantide juures kirjas millise väljundi jaoks on nad kasutusel, et lihtsustada implikantide grupeerimist.

espr. v3 (#1010)	espr. v4 (#1011)
10-0 0110	
0101 1100 ! 12	0101 1100 ! 12
01-1 0001	1-11 0001
11-0 1001	-100 1010
1-1- 1100 12	-1-1 0010
0--0 1000	-01- 0001
-1-1 0010	0-10 0011 34
0-0- 0001	0--0 1000
-10- 0001	10-0 0111 34
01-- 0010	1-1- 1100 12
! !	! !!

Arvestades võimalikke grupeerimisi, on aluseks võetud 4. lahendus.

espresso -Dopoall & .phase 1011

y1 = x1'x2 x3'x4 + x2 x3'x4' + x1'x4' + x1 x3

y2' = x1'x2 x3'x4 + x1 x2'x4' + x1 x3

y3 = x2 x3'x4' + x2 x4 + x1'x3 x4' + x1 x2'x4'

y4 = x1 x3 x4 + x2'x3 + x1'x3 x4' + x1 x2'x4'

Valideerimiseks vajalikud VHDL-koodid - tõeväärtustabel ja espresso tulemus. Testpingi ja simuleerimise kirjelduse leiab lehe lõpust.

Reliseerimine loogikaelementidel

Esialgne skeem

Minimeerimise tulemusest välja kirjutatud esialgne skeem ilma sisendite arvu piiranguta.

```
t1 = x1' & x2  & x3' & x4    [ == t4 & x1' & x3' ]
t2 = x1  &       x3  & x4
t3 =       x2  & x3' & x4'
t4 =       x2  &       x4
t5 =       x2' & x3
t6 = x1' &       x3  & x4'   [ == t7 & (x3) ]
t7 = x1' &           x4'
t8 = x1  & x2' &     x4'
t9 = x1  &       x3

y1 =  t1 + t3 + t7 + t9      [ == (t1 + t9) + t3 + t7 ]
y2 = (t1 + t8 + t9)'         [ == ((t1 + t9) + t8)' / (t1 + t8 + t9)' ]
y3 =  t3 + t4 + t6 + t8      [ == (t6 + t8) + t3 + t4 ]
y4 =  t2 + t5 + t6 + t8      [ == (t6 + t8) + t2 + t5 ]
```

Sellele skeemile vastava VHDL-koodi leiab siit. Vrdl. *t1*, *t6*, *y1*, *y3* ja *y4* kirjeldust enne ja pärast sobivateks elementideks teisendamist.

Skeem elementidena **#1** (vt. ka VHDL-koodi). Iga elemendi taga: [pindala/viide] ja andmete valmisoleku aeg (eeldusel, et sisendites on see 0).

```
x1i = x1'                 [1.5/1.5]  1.5
x2i = x2'                 [1.5/1.5]  1.5
x3i = x3'                 [1.5/1.5]  1.5
x4i = x4'                 [1.5/1.5]  1.5

t1  = t4  & x1i & x3i     [2.5/2.5]  4.5
t2  = x1  & x3  & x4      [2.5/2.5]  2.5
t3  = x2  & x3i & x4i     [2.5/2.5]  4.0
t4  = x2  & x4            [2.0/2.0]  2.0
t5  = x2i & x3            [2.0/2.0]  3.5
t6  = x3  & t7            [2.0/2.0]  5.5
t7  = x1i & x4i           [2.0/2.0]  3.5
t8  = x1  & x2i & x4i     [2.5/2.5]  4.0
t9  = x1  & x3            [2.0/2.0]  2.0

t19 =  t1  + t9           [2.0/2.0]  6.5
y1  =  t19 + t3  + t7     [2.5/2.5]  9.0
y2  = (t19 + t8)'         [1.5/1.5]  8.0
      (t1  + t8  + t9)'   [2.0/2.0]  6.5    [area +0.5]
t68 =  t6  + t8           [2.0/2.0]  7.5
y3  =  t68 + t3  + t4     [2.5/2.5]  10.0
y4  =  t68 + t2  + t5     [2.5/2.5]  10.0
```

Elemendid: 4 x NOT, 5 x 2-AND, 4 x 3-AND, 2 x 2-OR, 3 x 3-OR, 1 x 2/3-NOR.

Kokku: 19 elementi, suurus 39 (või 39,5), kriitiline tee 10.

Ühiste alamavaldiste otsimine

Ühised alamavaldised - ühised tuumad = mittetriviaalsed, ühised konjunktsioonid = triviaalsed.
Tuumade leidmisel on näidatud ainult tuumadeni viivad jagajad ja jagatised. Lisaks on võimalusel ka tuuma minimeeritud ja/või hinnatud realiseerimiseks kasutatavaid elemente.

```
y1  = x1'x2 x3'x4 + x2 x3'x4' + x1'x4' + x1 x3
/x1'    -->  x2 x3'x4 + x4'     ==>  x2 x3' + x4'
/x2 x3' -->  x1'x4 + x4'        ==>  x1' + x4' --> 2-NAND / 2-OR
/x4'    -->  x2 x3' + x1'

y2' = x1'x2 x3'x4 + x1 x2'x4' + x1 x3
/x1  -->  x2'x4' + x3
```

```
y3  = x2 x3'x4' + x2 x4 + x1'x3 x4' + x1 x2'x4'
/x2  -->  x3'x4' + x4              ==>  x3' + x4  --> 2-NAND / 2-OR
/x4' -->  x2 x3' + x1'x3 + x1 x2'  ==>  4-OR --> 3-OR + 2-OR

y4  = x1 x3 x4  + x2'x3 + x1'x3 x4' + x1 x2'x4'
/x1  -->  x3 x4  + x2'x4'
/x2' -->  x3 + x1 x4'
/x3  -->  x1 x4  + x2' + x1'x4'  ==>  x1 x4  + x1'x4' == (x1 xor x4)'
/x4' -->  x1'x3 + x1 x2'
```

Järelduste tegemisel on eeldatud, et 4-AND realiseerub 3-AND ja 2-AND elementidena ning 4-OR kui 3-OR+2-OR. Samuti on hinnatud literaalide (sisendite arvu) muutust (nt. '[11->9]'). '#2' näitab, millised variandid on valitud skeemi realiseerimiseks. Esimes väljundi (y1) puhul on võrdlused detailsemalt lahti seletatud.

```
y1:  /x2 x3': 4-AND + 3-AND + 4-OR --> 3-AND + 2-OR + 3-OR     [(3+2)+3+(3+2)=13 -> 3+2+3=8 == -5] #2
    [ x1'x2 x3'x4 + x2 x3'x4' + x1'x4' + x1 x3 == 4-AND + 3-AND + 2*2-AND + 4-OR ==>
      x2 x3'(x1' + x4') + x1'x4' + x1 x3       == 3-AND + 2-OR  + 2*2-AND + 3-OR -- 2*2-AND ei muutu ja pole arvestatud ]

     /x1': 4-AND + 2-AND + 4-OR --> 2*2-AND + 2-OR + 3-OR      [(3+2)+2+(3+2)=12 -> 2*2+2+3=9 == -3 - kehvem]
    [ x1'x2 x3'x4 + x1'x4' + x2 x3'x4' + x1 x3 == 4-AND + 3-AND + 2*2-AND + 4-OR ==>
      x1'(x2 x3' + x4') + x2 x3'x4' + x1 x3    == 3-AND + 2-OR +  3*2-AND + 3-OR -- 3-AND ja 2-AND ei muutu pole arvestatud ]

     /x4': 3-AND + 2-AND + 4-OR --> 2*2-AND + 2-OR + 3-OR      [3+2+(3+2)=10 -> 2+2+2+3=9 == -1 - veel kehvem ja 4-AND on alles!]
    [ x1'x2 x3'x4 + x2 x3'x4' + x1'x4' + x1 x3 == 4-AND + 3-AND + 2*2-AND + 4-OR ==>
      x1'x2 x3'x4 + x4'(x2 x3' + x1') + x1 x3  == 4-AND + 3*2-AND + 2-OR + 3-OR ]

y2': /x1: 3-AND + 2-AND + 3-OR --> 2*2-AND + 2*2-OR            [8->8 ja 4-AND on alles!]

y3:  /x2: 3-AND + 2-AND + 4-OR --> 2-AND + 2-OR/2-NAND + 3-OR  [10->7] #2
     /x4': 3*3-AND + 4-OR --> 4*2-AND + 3-OR + 2-OR            [14->13]

y4:  /x3: 2*3-AND + 4-OR --> 2-XOR + 2-AND + NOT + 3-OR        [11->8] #2
     /x2': 3-AND + 2-AND + 4-OR --> 2*2-AND + 2-OR + 3-OR      [10->9]
     /x1: 2*3-AND + 4-OR --> 3*2-AND + 2-OR + 3-OR             [11->11]
     /x4': 2*3-AND + 4-OR --> 3*2-AND + 2-OR + 3-OR            [11->11]
```

Skeem pärast ühiste alamavaldiste leidmist

Tulemus avaldistena, aluseks on y1, y3 ja y4 tuumade parimad variandid.

```
y1  = (x2 x3') (x1' + x4') + x1'x4' + x1 x3
y2' = (x2 x3') x1'x4 + x1 x2'x4' + x1 x3       [4-AND?! -> 2-AND+3-AND]
y3  = x2 (x3' + x4) + x1'x3 x4' + x1 x2'x4'
y4  = x3 (x1 xor x4)' + x2'x3 + x1 x2'x4'
```

Skeem elementidena **#2** (vt. ka VHDL-koodi).

```
x1i = x1'                 [1.5/1.5]  1.5
x2i = x2'                 [1.5/1.5]  1.5
x3i = x3'                 [1.5/1.5]  1.5
x4i = x4'                 [1.5/1.5]  1.5

t1a = x1i + x4i           [2.0/2.0]  3.5
t1b = x2  & x3i           [2.0/2.0]  3.5
t1c = t1b & t1a           [2.0/2.0]  5.5
t1d = t1b & x1i & x4      [2.5/2.5]  6.0
t2a = x1 xor x4           [2.0/2.0]  2.0
t2b = t2a'                [1.5/1.5]  3.5
t2c = x3  & t2b           [2.0/2.0]  5.5
t3a = x3i + x4            [2.0/2.0]  3.5
t3b = x2  & t3a           [2.0/2.0]  5.5
t5  = x2i & x3            [2.0/2.0]  3.5
t6  = x3  & t7            [2.0/2.0]  5.5
t7  = x1i & x4i           [2.0/2.0]  3.5
t8  = x1  & x2i & x4i     [2.5/2.5]  4.0
t9  = x1  & x3            [2.0/2.0]  2.0

y1  = t1c + t7 + t9       [2.5/2.5]  8.5
y2  = (t1d + t8 + t9)'    [2.0/2.0]  8.0
y3  = t3b + t6 + t8       [2.5/2.5]  8.0
y4  = t2c + t5 + t8       [2.5/2.5]  8.0
```

Elemendid: 5 x NOT, 8 x 2-AND, 2 x 3-AND, 2 x 2-OR, 3 x 3-OR, 1 x 3-NOR, 1 x 2-XOR.

Kokku: 22 elementi, suurus 44, kriitiline tee 8,5.

Elementide arv ja pindala suurenes, viide paranes (kiirem).

Optimeerimine

Esialgu proovin optimeerida varianti **#1**, sest selle suurus oli parem kui tuumadega variandil (#2).

Eesmärgiks on lahti saadi kallitest elementidest - invertorid, AND ja OR elemendid. Ning NAND on parem kui NOR. Teisenduste alusteks on DeMorgani ja topelteituse seadused: **(x' + y') = (x y)'**, **(x' y') = (x + y)'** ja **(x')' = x**.

Üldjoontes toimub teisendus selliselt, et nii AND kui ka OR elemendid muudetakse NAND elementideks - **x y + w z = ((x y)' (w z**

)')'. Sisendmuutujate inverteerimisest lahti saamiseks sobivad järgmised teisendused (otse- ja inverteeritud väärtuste kombinatsioonid):
a) **x y z' = (x y) z' = ((x y)' + (z')')' = ((x y)' + z)'**
b) **x y' z' = x (y' z') = x (y + z)'**
c) **x' y' z' = (x + y + z)'**

Teisendused on teostatud implikantide gruppide kaupa. Paaril korral on esitatud alternatiivid koos võrdlusega.

```
x1i = x1'               ===>  x1i = (x1 & x1)'           [1.0/1.0]  1.0
x2i = x2'               ===>  x2i = (x2 & x2)'           [1.0/1.0]  1.0
x3i = x3'               ===>  x3i = (x3 & x3)'           [1.0/1.0]  1.0
x4i = x4'               ===>  x4i = (x4 & x4)'           [1.0/1.0]  1.0

t2  = x1 & x3 & x4      ===>  t2i = (x1 & x3 & x4)'      [1.5/1.5]  1.5
t3  = x2 & x3i & x4i    =1=>  t3i = (x2 & x3i & x4i)'    [1.5/1.5]  2.5
                        =2=>  t3x = (x3 + x4)'           [1.5/1.5]  1.5
                              t3i = (x1 & t3x)'          [1.0/1.0]  2.5
    [1 vajab x3 & x4 inverteeritult, nende kadumine vähendaks suurust!!]
    [2 on pindalalt suurem (kuid pluss invertorid!), viide sama]
    [x3i & x4i pole mujal vaja --> variant 2 on parem]
t4  = x2 & x4           ===>  t4i = (x2 & x4)'           [1.0/1.0]  1.0
t5  = x2i & x3          ===>  t5i = (x2i & x3)'          [1.0/1.0]  2.0
                              t6  = x3 & t7              [2.0/2.0]  3.5
t7  = x1i & x4i         ===>  t7  = (x1 + x4)'           [1.5/1.5]  1.5
t8  = x1 & x2i & x4i    ===>  t8x = (x2 + x4)'           [1.5/1.5]  1.5
                              t8  = x1 & t8x             [2.0/2.0]  3.5
t68 = t6 + t8           ===>  t68i = (t6 + t8)'          [1.5/1.5]  5.0
t1  = t4 & x1i & x3i    =1=>  t1  = (t4i + x1 + x3)'     [2.0/2.0]  3.0
t9  = x1 & x3           =1=>  t9  = x1 & x3              [2.0/2.0]  2.0
t19 = t1 + t9           =1=>  t19 = t1 + t9              [2.0/2.0]  5.0
                        =2=>  t1i = t4i + x1 + x3        [2.5/2.5]  3.5
                        =2=>  t9i = (x1 & x3)'           [1.0/1.0]  1.0
                        =2=>  t19 = (t1i & t9i)'         [1.0/1.0]  4.5
    [1: 6.0/5.0 &  2: 4.5/4.5 --> variant 2 on parem]

y1  = t19 + t3 + t7     ===>  t197i = (t19 + t7)'        [1.5/1.5]  6.0
                              y1  = (t197i & t3i)'       [1.0/1.0]  7.0
y2  = (t19 + t8)'       =1=>  y2  = (t19 + t8)'          [1.5/1.5]  6.0
                        =2=>  y2  = (t1 + t8 + t9)'      [2.0/2.0]  6.5
     [variant 1 on väiksem ja kiirem, vt. ka t19 variante]
y3  = t68 + t3 + t4     ===>  y3  = (t68i & t3i & t4i)'  [1.5/1.5]  6.5
y4  = t68 + t2 + t5     ===>  y4  = (t68i & t2i & t5i)'  [1.5/1.5]  6.5
```

Teisendusi ei pea sisse viima korraga, vaid saab teha ka ükshaaval. Peab ainult meeles pidama, et kui mõni signaal on kasutusel mitmes kohas, siis selle inverteerimisel peab ka esialgse signaali tekitama, kuni seda enam vaja pole. Näitena on toodud **y4** genereeriva VÕI-elemendi (OR) asendamine JA-EI-ga (NAND), millega kaasneb signaalide *t2*, *t5* ja *t68* inverteerimine (uute nimedega *t2i*, *t5i* ja *t68i*). Samuti on asendatud JA-EI-dega *t2i* ja *t5i* genereerivad JA-elemendid ning *t68i*-d genereerib VÕI asemel VÕI-EI. Kuna *t68* on kasutusel ka *y3*-s, tuli sinna tekitada *t68i* inversioon. Esialgsed read on jäetud sisse kommentaaridena, et vajadusel saaks taastada olukorra enne ühte või teist muudatust.

Tulemus #1

Aluseks vahevariant #1 (ilma tuumadeta, VHDL-kood).

```
x1i = (x1 & x1)'            [1.0/1.0]  1.0
x2i = (x2 & x2)'            [1.0/1.0]  1.0

t1i = t4i + x1 + x3         [2.5/2.5]  3.5
t2i = (x1 & x3 & x4)'       [1.5/1.5]  1.5
t3x = (x3 + x4)'            [1.5/1.5]  1.5
t3i = (x1 & t3x)'           [1.0/1.0]  2.5
t4i = (x2 & x4)'            [1.0/1.0]  1.0
t5i = (x2i & x3)'           [1.0/1.0]  2.0
t6  = x3 & t7               [2.0/2.0]  3.5
t7  = (x1 + x4)'            [1.5/1.5]  1.5
t8x = (x2 + x4)'            [1.5/1.5]  1.5
t8  = x1 & t8x              [2.0/2.0]  3.5
t68i = (t6 + t8)'           [1.5/1.5]  5.0
t9i = (x1 & x3)'            [1.0/1.0]  1.0
t19 = (t1i & t9i)'          [1.0/1.0]  4.5
t197i = (t19 + t7)'         [1.5/1.5]  6.0

y1  = (t197i & t3i)'        [1.0/1.0]  7.0
y2  = (t19 + t8)'           [1.5/1.5]  6.0
y3  = (t68i & t3i & t4i)'   [1.5/1.5]  6.5
y4  = (t68i & t2i & t5i)'   [1.5/1.5]  6.5
```

Elemendid: 2 x 2-AND, 8 x 2-NAND, 3 x 3-NAND, 1 x 3-OR, 6 x 2-NOR.

Kokku: 20 elementi (+5,3% esialgsega võrreldes), suurus 28 (-28,2%), kriitiline tee 7 (-30%).

Kui võrrelda tulemust varasema näidislahendusega, siis loogikaelemente on küll vähem (22->20), kuid pindala on pisut suurem (27.0->28.0) ja kriitiline tee pikem (5.5->7.0). Põhjuseks ilmselt keerulisem osadeks jagamine, mis suurendas koguviidet.

Valideerimine

Üks kontrolli võimalus on simuleerimine VHDL abil kasutades ette antud testpinki. Simulatsiooni tulemusena saadud lainekujudel vrdl. signaale **y1a**, **y1b** ja **y1c**, **y2a**, **y2b** ja **y2c**, **y3a**, **y3b** ja **y3c** ning **y4a**, **y4b** ja **y4c** (y?**a** on ülesanne, y?**b** pärast minimeerimist ja y?**c** pärast optimeerimist). Signaalid y1x, y2x, y3x ja y4x on kasutusel automaatseks võrdlemiseks (vt. võrdlusfunktsiooni 'compare_signals'). Üksikud pulsid pole vead, vaid on põhjustatud signaalide hilistumisest loogikalülides (nn. delta-viide).

Põhjalikum simuleerimine kirjelduse koos simulaatori kasutamise lühijuhendiga leiab siit.

Kui simulatsioonide tulemused ei klapi, tuleks viga üles otsida. Silumisnäidis annab ehk ettekujutuse, kuidas seda teha. See on küll tehtud vanema näidislahenduse jaoks, kui vea otsimise põhimõte jääb siiski samaks.

Illustreerimaks viiteid loogikaelementides, on optimeeritud skeemis omistuskäsud asendatud komponentidega, mis modelleerivad loogikaelemente (vt. skeemi, komponente ja tulemust). Tähelepanu tasuks pöörata sellele, kui kaua on mõnel väljundil tulemus vigane - simulatsiooni kursor on ühel sellisel kohal, kus *y3* ja *y4* on suurima viitega (6.5).

Kasutada võib suvalist VHDL simulaatorit:

- ModelSim PE Student Edition (Mentor Graphics) tudengitele mõeldud HDL simulaator (piiratud suurusega disainid, piisav kursusetööde jaoks, litsentsi saamiseks on vajalik Internetiühendus, ainult Windows).
- ISE WebPACK - Xilinx'i pakett väiksemate disainide simuleerimiseks ja sünteesiks (MS Windows & Linux-d), kasutab ISim simulaatorit (Xilinx).
- zamiaCAD - avatud lähtekoodiga modulaarne, platvormist sõltumatu simulaator ja disainivahend.
- GHDL ja gtkWave - simulaator ja lainekujude näitaja (MS Windows, Linux-d jt. platvormid; GPL - vabavara & avatud lähtekood).

TTÜ arvutiklassides ICT-501/502 (Linux) on kasutada nii Mentor Graphics ModelSim (vt. ka juhendit) kui ka Xilinx ISE (vt. ka juhendit).

Kui on soov kasutada skeemisimulaatori apletit, tuleks näidata kõik sisendite kombinatsioonid (ja neile vastavad väljundid). Üheks võimaluseks on kasutada sama simulaatori vahendeid (vt. näidet, pluss vastav skeem).

Esitatud mahus teostatud analüüs ja teisendused on täiesti piisavad, sest on teostatud kolm otsustust:
1) espresso tulemuste analüüs ja valik;
2) tuumade analüüs ja otsus neid mitte kasutada; ning
3) hulk võrdlusi ja teisendusi skeemi optimeerimisel.
Samuti on teostatud kolme tulemuse võrdlus simuleerimise abil (komponentidega *fs1_opt_sch.vhd* simulatsioon pole vajalik).

Variant #2 (ühiste alamavaldistega) lisandub hiljem.