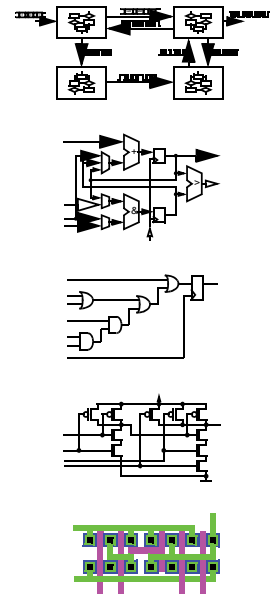


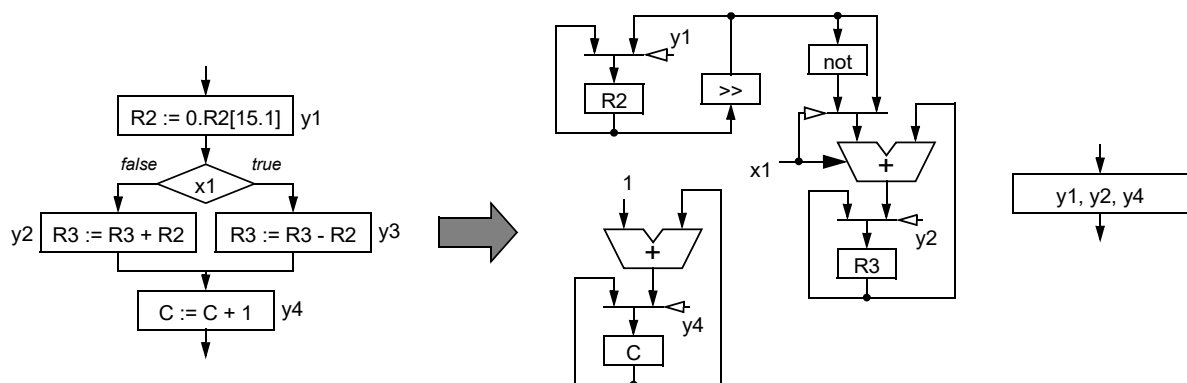
Digitaalsüsteemide struktuuri optimeerimine

- **Süsteemi disain – System design**
a.k.a. **Arhitektuuri süntees – Architectural-level synthesis**
 - Süsteemi taseme süntees
 - tükeldamine & liideste süntees
 - Kõrgtaseme süntees
 - planeerimine, eraldamine & sidumine
- **Loogikadisain – Logic Design**
 - Registersiirete taseme süntees
 - andme- & juhtosa süntees
 - Loogikataseme süntees
 - loogika minimeerimine & optimeerimine
- **Füüsiline disain – Physical design**
a.k.a. **Geomeetria süntees – Geometrical-level synthesis**
 - Füüsikalise taseme süntees
 - teisendus loogikaelementideks
 - paigaldamine & trasseerimine/ruutimine



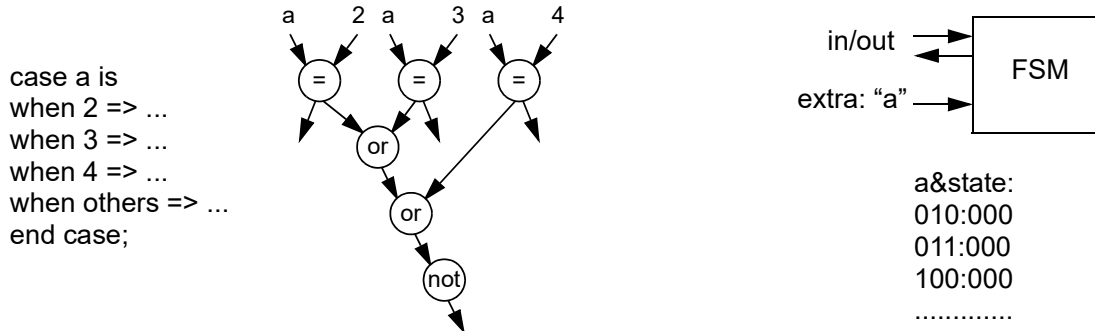
Realisatsiooni optimaalsus?

- **operatsiooni hind riistvaras**
 - nihutamine == traatide teisiti viimine
 - liitmist ja lahutamist teostab sama seade – summaator (+ülekanne)
- **sõltumatud operatsioonid võib täita korraga**
 - tegelike andmesõltuvuste analüüs – nt. R2 nihutamise tulemus



Andmeosa → juhtosa

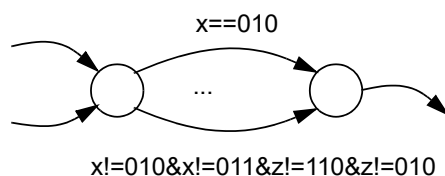
- Osa operatsioonidest andmeosas teisendatakse operatsioonideks juhtosas
 - võrdlused konstantidega
 - siirdetingimuste avaldised (*if-then-else*, *case*, tsüklid)
 - optimeeritakse koos automaadiga – võib toimuda olekute arvu plahvatuslik kasv



Juhtosa → andmeosa

- Osa operatsioonidest juhtosas teisendatakse operatsioonideks andmeosas
 - siirdetingimuste arvutuste grupeerimine
 - lihtsustab automaadi sünteesi
 - parem (osade) loogikafunktsioonide minimeerimine

```
if x="010" then y:="0110";
elsif x="011" and
      z="110" then y:="0101";
else
  if z="010" then y:="1001";
  else
    y:="1100";
  end if;
end if;
```



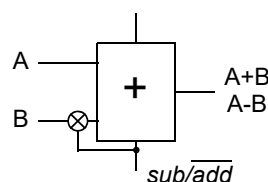
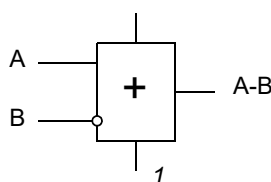
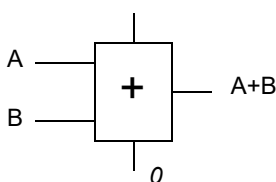
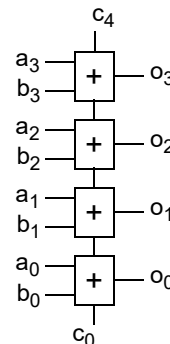
Aritmeetika-operatsioonide realiseerimine

Summator e. liitja

- poolsummator – $(c_i, o_i) = a_i + b_i$
 - $o_i = a_i \oplus b_i$ $c_i = a_i \& b_i$
- täis-summator – $(c_i, o_i) = a_i + b_i + c_{i-1}$
 - $o_i = a_i \oplus b_i \oplus c_i$ $c_i = (a_i \& b_i) \mid (a_i \& c_{i-1}) \mid (b_i \& c_{i-1})$

Lahutaja

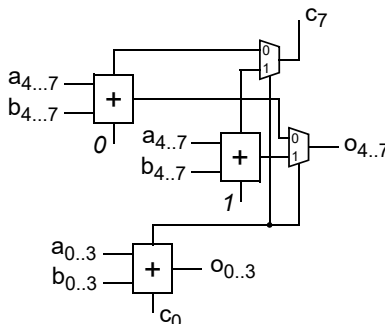
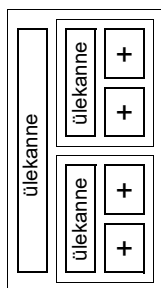
- täiendkood – $O = A - B = A + \bar{B} + 1$
- liitja-lahutaja – $O = n ? (A - B) : (A + B)$



Aritmeetika-operatsioonide realiseerimine

Ülekanne kiirendamine

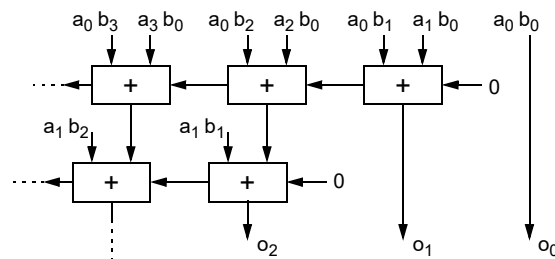
- jagatud gruppideks, grupi sees ülekanne eraldi funktsioonina
- “carry-lookahead”
 - “generate” – $g_i = a_i \& b_i$ “propagate” – $p_i = a_i \mid b_i$ “carry” – $c_i = g_i \mid (p_i \& c_{i-1})$
 - $g_{i+1} = a_{i+1} \& b_{i+1}$; $p_{i+1} = a_{i+1} \mid b_{i+1}$; $c_{i+1} = g_{i+1} \mid (p_{i+1} \& c_i)$
 - $c_{i+1} = g_{i+1} \mid (p_{i+1} \& (g_i \mid (p_i \& c_{i-1}))) \dots$
- “carry-select”
 - spekulatiivne arvutus – vanemates järkudes leitakse summa ja ülekanne edasi nii 0 kui 1 jaoks, tulemus valitakse multipleksoriga, kui ülekanne alt “lõpuks kohale jõuab”



Aritmeetika-operatsioonide realiseerimine

Korrutamine

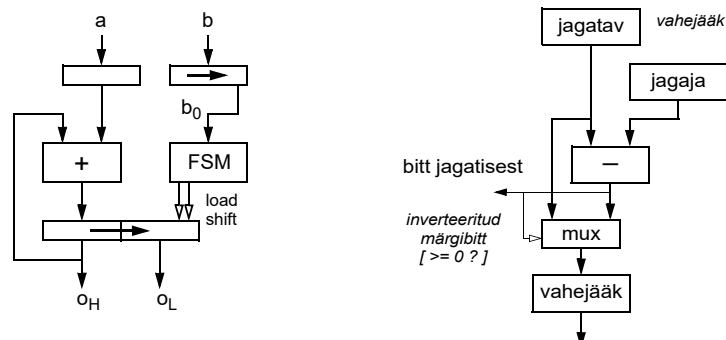
- $A = a_{n-1} \cdot 2^{n-1} + a_{n-2} \cdot 2^{n-2} + \dots + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$
- $B = b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$
- $A \cdot B = a_{n-1} \cdot 2^{n-1} \cdot (b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) + a_{n-2} \cdot 2^{n-2} \cdot (b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) + \dots + a_2 \cdot 2^2 \cdot (b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) + a_1 \cdot 2^1 \cdot (b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) + a_0 \cdot 2^0 \cdot (b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0) = a_{n-1} \cdot b_{n-1} \cdot 2^{2n-2} + a_{n-1} \cdot b_{n-2} \cdot 2^{2n-3} + \dots + a_0 \cdot b_2 \cdot 2^2 + a_0 \cdot b_1 \cdot 2^1 + a_0 \cdot b_0 \cdot 2^0$

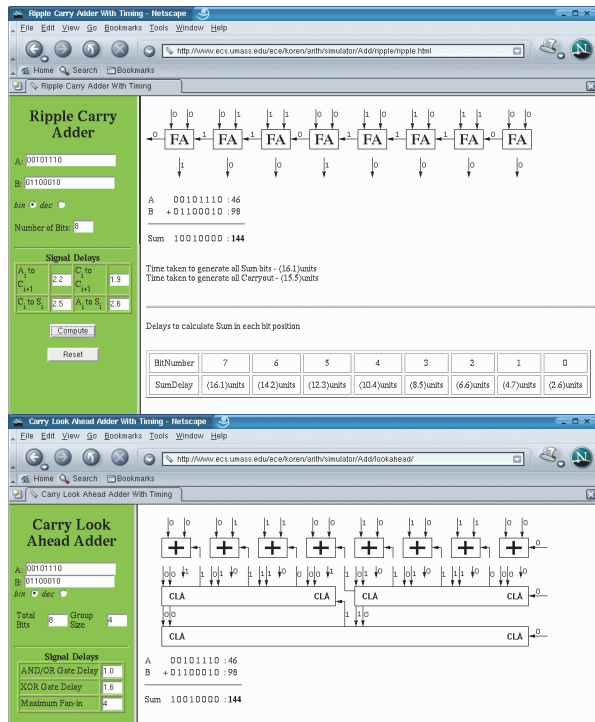


Aritmeetika-operatsioonide realiseerimine

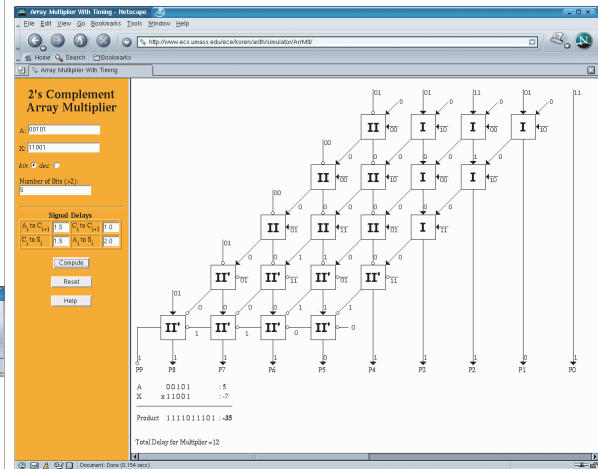
Korrutamine

- paralleelne – kombinatsiooniskeem
 - $m \cdot n$ bitti – m n -bitist summaatorit ja $m \cdot n$ JA-elementi
 - kombinatsiooniskeem võib olla liiga suure viitega – abiregistrid & konveier (pipeline)
- järjestikuline – järk-järgu kaupa tsükliliselt
 - akumulaator ja juhtautomaat
 - mitu järku korraga – juhtloogika lisaks
- Jagamine – tavaliselt tsükliliselt – akumulaator, juhtloogika ja juhtautomaat





<http://www.ecs.umass.edu/ece/koren/arith/simulator/>



Ületäitumine?

• Täiendkood

- 8 bitti: -128...+127
- 125+5=? [130]

$$\begin{array}{r} 125 \quad 01111101 \\ + \quad 5 \quad 00000101 \\ [130] \quad 10000010 == -126 \end{array}$$

• Arvude skaala

$$\begin{array}{r} -1 \quad 11111111 \\ \dots\dots\dots \\ -128 \quad 10000000 \\ +127 \quad 01111111 \\ \dots\dots\dots \\ 0 \quad 00000000 \end{array}$$

• Ekstra märgibitt

- 00 või 11 – OK

$$\begin{array}{r} 75 \quad 001001011 \\ + \quad 15 \quad 000001111 \\ [90] \quad 001011010 \end{array}$$

$$\begin{array}{r} 125 \quad 001111101 \\ + \quad 5 \quad 000000101 \\ [130] \quad \underline{01}0000010 \end{array}$$

$$\begin{array}{r} -75 \quad 110110101 \\ + -15 \quad 111110001 \\ [-90] \quad \underline{11}0100110 \end{array}$$

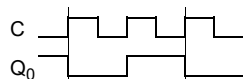
$$\begin{array}{r} -125 \quad 110000011 \\ + \quad -5 \quad 111111011 \\ [-130] \quad \underline{10}1111110 \end{array}$$



Loendur kui juhtautomaat [#1]

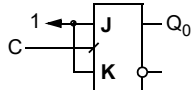
- Loendur – primitiivne generaator – $S \neq \emptyset$, $I = \emptyset$, $O = \emptyset$ ($O \equiv S$), $\delta: S \rightarrow S$, $\lambda \equiv \emptyset$

2-nd loendur
Jada – 0, 1

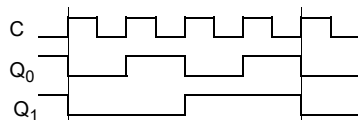


s_t	s_{t+1}	JK_{t+1}
0	1	1-
1	0	-1

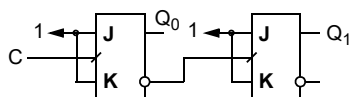
$J=K=1$



4-nd loendur
Jada – 0, 1, 2, 3



Kaks loendurit järjest?



Takti hilistumine!!!

Automaat?

Olekud – 00, 01, 10, 11 [Q1 Q0]

JK-triger

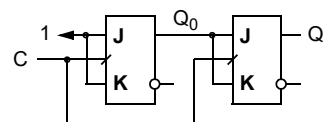


Q_t	Q_{t+1}	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

s_t	s_{t+1}	JK_{t+1}
00	01	0- 1-
01	10	1- -1
10	11	-0 1-
11	00	-1 -1

$J_0=K_0=1$

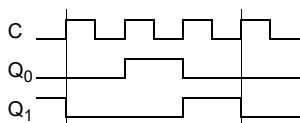
$J_1=K_1=Q_0$



Loendur kui juhtautomaat [#2]

3-nd loendur
Jada – 0, 1, 2

Olekud – 00, 01, 10 [Q1 Q0]

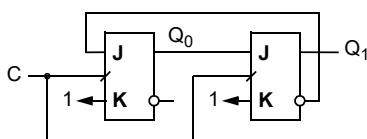


s_t	s_{t+1}	JK_{t+1}
00	01	0- 1-
01	10	1- -1
10	00	-1 0-

$K_0=K_1=1$

$J_0=\bar{Q}_1$

$J_1=Q_0$



JK-triger



Q_t	Q_{t+1}	J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

5-nd loendur
Jada – 0, 1, 2, 3, 4

Olekud – 000, 001, 100, 011, 100 [Q2 Q1 Q0]

s_t	s_{t+1}	JK_{t+1}
000	001	0- 0- 1-
001	010	0- 1- -1
010	011	0- -0 1-
011	100	1- -1 -1
100	000	-1 0- 0-

$(J_0=\bar{Q}_2 \quad K_0=1)$

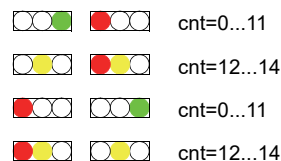
$J_0=K_0=\bar{Q}_2$

$J_1=K_1=Q_0$

$J_2=Q_1Q_0 \quad K_2=1$

Valgusfoor – juhtosa

- roheline 12 sek. punane
- kollane 3 sek. kollane+punane
- punane 12 sek. roheline
- kollane+punane 3 sek. kollane



```

while ( !≥12 )           // S1
    set ( G1, R2 );
while ( ≥12 )             // S2
    set ( Y1, Y2, R2 );
while ( !≥12 )           // S3
    set ( R1, G2 );
while ( ≥12 )             // S4
    set ( Y1, R1, Y2 );
    
```

i^t	s^t	s^{t+1}	o^t
≥ 12	S1	S1	G1, R2
≥ 12		S2	
≥ 12	S2	S3	Y1, Y2, R2
≥ 12		S2	
≥ 12	S3	S3	R1, G2
≥ 12		S4	
≥ 12	S4	S1	Y1, R1, Y2
≥ 12		S4	

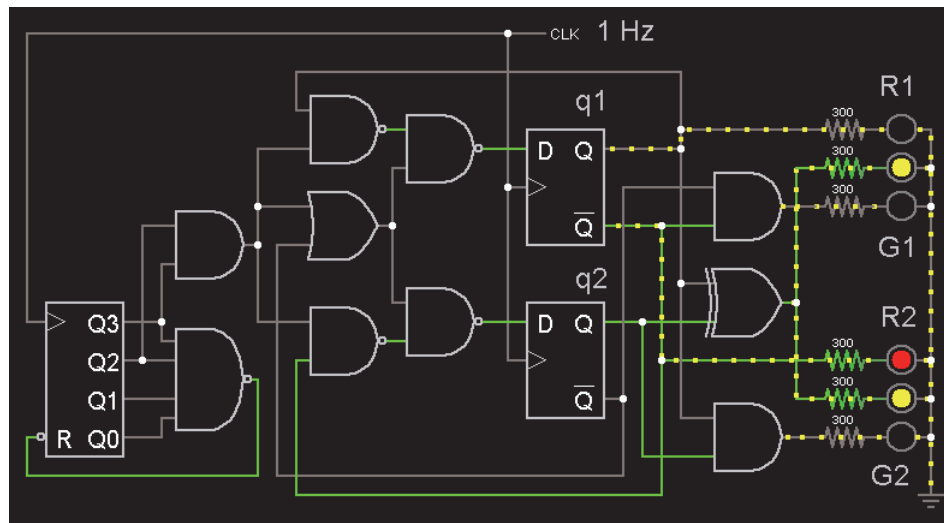
Valgusfoor – juhtosa

i ^t		s ^t		s ^{t+1}		JK		D	o ^t	
!≥12	0	S1	00	S1	00	0- 0-	00	G1,R2	001 100	
≥12	1			S2	01	0- 1-	01			
!≥12	0	S2	01	S3	11	1- -0	11	Y1,Y2,R2	010 110	
≥12	1			S2	01	0- -0	01			
!≥12	0	S3	11	S3	11	-0 -0	11	R1,G2	100 001	
≥12	1			S4	10	-0 -1	10			
!≥12	0	S4	10	S1	00	-1 0-	00	Y1,R1,Y2	110 010	
≥12	1			S4	10	-0 0-	10			

- Väljundid
 - $Y1 = Y2 = q1 \text{ xor } q2$
 - $R1 = q1$
 - $R2 = q1'$
 - $G1 = q1' \text{ } q2'$
 - $G2 = q1 \text{ } q2$

- JK-trigerid
 - $j1 = i' \text{ } q2$
 - $k1 = i' \text{ } q2'$
 - $j2 = i \text{ } q1'$
 - $k2 = i \text{ } q1$
 - D-trigerid
 - $d1 = i \text{ } q1 + i' \text{ } q2$
 - $d2 = i \text{ } q1' + i' \text{ } q2$
- # Optimeerimist...
- $i' \text{ } q2 = (i + q2')'$
- $k = i + q2'$
- $d1 = ((i \text{ } q1)' \text{ } k)'$
- $d2 = ((i \text{ } q1')' \text{ } k)'$

Valgusfoor – tulemus



<https://ati.ttu.ee/~lrv/IAS0150/tlc-applet.txt>

Algoritmi realiseerimine

- Algoritm kui programm
 - operatsioonid – andmeosa
 - järjestus ja tingimuse – juhtosa
- GCD (Greatest Common Divisor) – suurim ühistegur


```
while ( x != y ) {
    if ( x < y ) y = y - x;
    else      x = x - y;
}
```
- operatsioonid – võrdlused ja lahutamine
- konkreetse operatsiooni realiseerimine?
 - $A < B \iff A - B < 0$ / $A \neq B \iff A - B \neq 0$ – kõike saab teha lahutajaga?!
- sama riistvara kõikidele operatsioonidele või korraka arvutamine?
 - kiirus või suurus? [ja kas see ongi probleem?]



Plokkskeemi genereerimine algoritmist

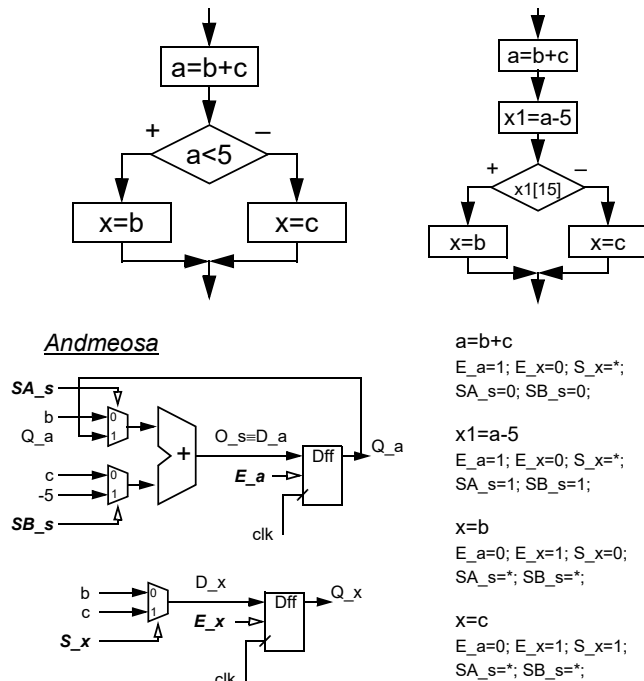
```
...
a = b + c ;
if ( a < 5 )   x = b ;
else          x = c ;
...
```

Andme-osa

- Muutujad e. registrid
 - a (x1), b, c, x
- Operatsioonid e. ALU-d – '+', '<'
- kombinatsiooniskeemid
- antud juhul üks liitja
- $a < 5 \equiv a + (-5) < 0$ [märgibitt!]
- Omistamised e. multiplekserid

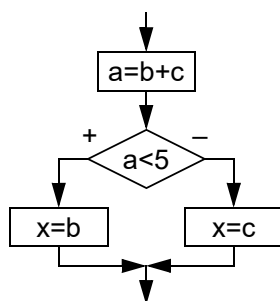
Juht-osa

- Algoritm
 - juhtsignaalid (juht)automaadist
 - kontrollsignaalid (juht)automaati



Plokkskeem ja juhtautomaadid

```
...
a = b + c ;
if ( a < 5 )   x = b ;
else          x = c ;
...
```

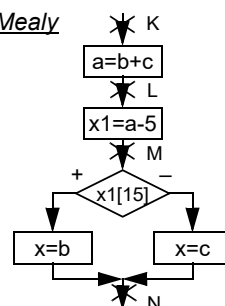


Mealy: $i^t - x1[15] (Q_a[15])$

Moore: $i^t - O_s[15]$

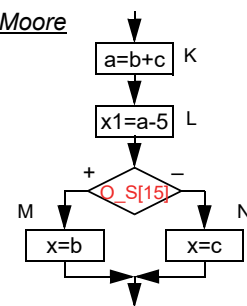
$o^t - E_a, E_x, S_x, SA_s, SB_s$

Mealy



i^t	s^t	s^{t+1}	o^t
...	...	K	
-	K	L	10-00
-	L	M	10-11
0	M	N	011--
1		N	010--
...	N	...	...

Moore



i^t	s^t	s^{t+1}	o^t
...	...	K	
-	K	L	10-00
0	L	N	-0-11
1		M	
-	M	...	010--
-	N	...	011--



Näide #1 – GCD (Greatest Common Divisor)

- Spetsifikatsioon ~ käitumuslik kirjeldus
 - sisendi/väljundi ajastus fikseeritud – juht- ja takt-signaalid

```
process -- gcd-bhv.vhdl
  variable x, y: unsigned(15 downto 0);
begin
  -- Wait for the new input data
  wait on clk until clk='1' and rst='0';
  x := xi;   y := yi;   rdy <= '0';
  wait on clk until clk='1';
  -- Calculate
  while x /= y loop
    if x < y then y := y - x;
    else        x := x - y;   end if;
  end loop;
  -- Ready
  xo <= x;   rdy <= '1';
  wait on clk until clk='1';
end process;
```

Probleemid

- tsüklis puudub takt
- keerukas “wait” käsk
- (- mitu “wait” käsku)

Mida proovida?

- erinevad süntesaatorid
- suuruse minimeerimine
- kiiruse tõstmine

Tehnoloogiad – ASIC, FPGA

VHDL kood & testpingid

<https://ati.ttu.ee/~lrv/gcd/>



GCD – sünteesitav kood?

- Takteeritud käitumuslik stiil

```
process -- gcd-bhvc.vhdl
  variable x, y: unsigned(15 downto 0);
begin
  -- Wait for the new input data
  while rst = '1' loop
    wait on clk until clk='1';
  end loop;
  x := xi;   y := yi;   rdy <= '0';
  wait on clk until clk='1';
  -- Calculate
  while x /= y loop
    if x < y then y := y - x;
    else        x := x - y;   end if;
    wait on clk until clk='1';
  end loop;
  -- Ready
  xo <= x;   rdy <= '1';
  wait on clk until clk='1';
end process;
```

ASIC: sünteesitav

961 e.g. / 20.0 ns
2 lahutajat, 2 võrdlejat

FPGA: mitte-sünteesitav

“wait” käsud tsüklis :(
juhtautomaat vajalik :(

Võimalikud valikud

- funktsioonide jagamine
- universaalsed funktsioonid
- spekulatiivne arvutamine

GCD – käitumuslik automaat (behavioral FSM)

```
process begin -- gcd-bfsm.vhdl
  wait on clk until clk='1';
  case state is
    -- Wait for the new input data
    when S_wait =>
      if rst='0' then
        x<=xi; y<=yi; rdy<='0'; state<=S_start;
      end if;
    -- Calculate
    when S_start =>
      if x /= y then
        if x < y then y <= y - x;
        else x <= x - y; end if;
        state<=S_start;
      else
        xo<=x; rdy<='1'; state<=S_ready;
      end if;
    -- Ready
    when S_ready => state<=S_wait;
  end case;
end process;
```

ASIC: sünteesitav
911 e.g. / 19.4 ns
2 lahutajat, 2 võrdlejat

FPGA: sünteesitav
108 SLC / 9.9 ns
2 lahutajat, 2 võrdlejat

Kas saab teha paremaks?

Jälle need võimalikud valikud

- funktsioonide jagamine
üks operatsioon taktis
- universaalsed funktsioonid
 $A < B \implies A - B < 0$ / $A = B \implies A - B = 0$
- spekulatiivne arvutamine
lahutamine enne otsustamist

GCD – universaalsed funktsioonid?

- $A < B \implies A - B < 0$ / $A = B \implies A - B = 0$

```
-- Three operations:
-- subtraction, and
-- comparisons not-equal &
-- less-than
xo <= xi - yi;

ne <= '1' when xi /= yi else '0';

lt <= '1' when xi < yi else '0';
```

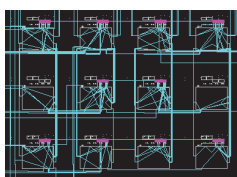
ASIC: 209 e.g. / 19.9 ns

FPGA: 20 SLC / 12.1 ns

kolm liitjat!

```
-- ALU - subtracting and then comparing
x_out <= xi - yi; xo <= x_out;
process (x_out)
  variable or_tmp: unsigned(15 downto 0);
begin
  or_tmp(15) := x_out(15);
  for i in 14 downto 0 loop
    or_tmp(i) := or_tmp(i+1) or x_out(i);
  end loop;
  ne <= to_bit(or_tmp(0));
end process;
lt <= to_bit(x_out(15));
```

ASIC: 148 e.g. / 21.8 ns / FPGA: 12 SLC / 14.6 ns



GCD – disainiruumi uurimine

- Erinevad lahendused – <https://ati.ttu.ee/~lrv/gcd/>
- gcd-bhv.vhdl – puhas käitumuslik kirjeldus, mitte-sünteesitav
- gcd-bhvc.vhdl – takteeritud käitumuslik kirjeldus, osad süntesaatorid OK
- gcd-bfsm.vhdl – nn. käitumuslik automaat (automaat & käitumuslik andmetee), sünteesitav (aga kui efektiivne see on?)
- gcd-rtl1.vhdl – üks ALU, 3 takti iteratsiooni kohta –
1) “pole võrdne?”, 2) “väiksem kui?”, 3) lahutaja

```
#1#    a=x-y;    a!=0 ? #2# : #ready#
#2#    a=x-y;    a<0 ? #3# : #4#
#3#    y=y-x    /    #4#    x=x-y
```

- gcd-rtl2.vhdl – üks ALU, 2 takti iteratsiooni kohta –
1) “pole võrdne?” ja “väiksem kui?”, 2) lahutaja

```
#1#    a=x-y;    a!=0 ? ( a<0 ? #2# : #3# ) : #ready#
#2#    y=y-x    /    #3#    x=x-y
```

GCD – disainiruumi uurimine (2)

- Erinevad lahendused – <https://ati.ttu.ee/~lrv/gcd/>
 - gcd-rtl3.vhdl – võrdleja (väiksem kui) kontrollib lahutajat, 1 takt iteratsiooni kohta – väike kuid aeglane (jadamisi) andmetee
- ```
#1# x<y ? y=y-x : (a=x-y; a!=0 ? x=x-y : #ready#)
```
- gcd-rtl4.vhdl – spekulatiiv arvutamine – mõlemad lahutamised tehakse kõigepealt, siis toimub otsustamine (üks lahutaja võrdleb “väiksem kui”, teine “pole võrdne”)
- ```
#1#    a1=x-y;    a2=y-x;    a2!=0 ? ( a1<0 ? y=a2 : x=a1 ) : #ready#
```
- gcd-rtl5.vhdl – spekulatiiv arvutamine – mõlemad lahutamised tehakse kõigepealt, siis toimub otsustamine (üks lahutaja võrdleb “väiksem kui”, kuid eraldi “pole võrdne”)
- ```
#1# a1=x-y; a2=y-x; x!=y ? (a1<0 ? y=a2 : x=a1) : #ready#
```

## GCD – üksik ALU, 2 takti iteratsiooni kohta

*gcd-rtl2.vhdl*

```
-- Next state function of the FSM
process (state, rst, alu_ne, alu_lt) begin
 ena_x <= '0'; ena_y <= '0'; ena_r <= '0';
 set_rdy <= '0'; xi_yi_sel <= '0'; sub_y_x <= '0';
 next_state <= state;
 case state is
 when S_wait => -- Wait for the new input data
 if rst='0' then
 xi_yi_sel <= '1'; ena_x <= '1'; ena_y <= '1';
 next_state <= S_start;
 end if;
 when S_start => -- Loop: ready?
 if alu_ne='1' then
 if alu_lt='1' then next_state <= S_sub_y_x;
 else next_state <= S_sub_x_y; end if;
 else next_state <= S_ready;
 end if;
 when S_sub_y_x => -- Loop: y-x
 ena_y <= '1'; sub_y_x <= '1';
 next_state <= S_start;
 when S_sub_x_y => -- Loop: x-y
 ena_x <= '1'; sub_y_x <= '0';
 next_state <= S_start;
 when S_ready => -- Ready
 ena_r <= '1'; set_rdy <= '1';
 next_state <= S_wait;
 end case;
 end process;

 -- ALU: subtract / less-than / not-equal
 alu_o <= alu_1 - alu_2;
 alu_lt <= to_bit(alu_o(15));
 process (alu_o)
 variable or_tmp: unsigned(15 downto 0);
 begin
 or_tmp(15) := alu_o(15);
 for i in 14 downto 0 loop
 or_tmp(i) := or_tmp(i+1) or alu_o(i);
 end loop;
 alu_ne <= to_bit(or_tmp(0));
 end process;

 -- Multiplexers
 x_i <= xi when xi_yi_sel='1' else alu_o;
 y_i <= yi when xi_yi_sel='1' else alu_o;
 alu_1 <= y when sub_y_x='1' else x;
 alu_2 <= x when sub_y_x='1' else y;

 -- Registers
 process begin
 wait on clk until clk='1';
 state <= next_state;
 if ena_x='1' then x <= x_i; end if;
 if ena_y='1' then y <= y_i; end if;
 if ena_r='1' then xo <= x; end if;
 rdy <= set_rdy;
 end process;
```

## GCD – komparaator+lahutaja, 1 takt

*gcd-rtl3.vhdl*

```
-- Next state function of the FSM
process (state, rst, alu_ne) begin
 ena_xy <= '0'; ena_r <= '0';
 set_rdy <= '0'; xi_yi_sel <= '0';
 next_state <= state;
 case state is
 when S_wait => -- Wait for the new input data
 if rst='0' then
 xi_yi_sel <= '1'; ena_xy <= '1';
 next_state <= S_start;
 end if;
 when S_start => -- Calculate
 if alu_ne='1' then
 ena_xy <= '1'; next_state <= S_start;
 else
 ena_r <= '1'; set_rdy <= '1';
 next_state <= S_ready;
 end if;
 when S_ready => -- Ready
 ena_r <= '1'; set_rdy <= '1';
 next_state <= S_wait;
 end case;
 end process;

 -- Comparator (less-than)
 sub_y_x <= '1' when x < y else '0';

 -- Subtractor (+not-equal)
 alu_o <= alu_1 - alu_2;
 process (alu_o)
 variable or_tmp: unsigned(15 downto 0);
 begin
 or_tmp(15) := alu_o(15);
 for i in 14 downto 0 loop
 or_tmp(i) := or_tmp(i+1) or alu_o(i);
 end loop;
 alu_ne <= to_bit(or_tmp(0));
 end process;

 -- Multiplexers
 x_i <= xi when xi_yi_sel='1' else alu_o;
 y_i <= yi when xi_yi_sel='1' else alu_o;
 alu_1 <= y when sub_y_x='1' else x;
 alu_2 <= x when sub_y_x='1' else y;
 ena_x <= '1' when (sub_y_x='0' and ena_xy='1') or
 xi_yi_sel='1' else '0';
 ena_y <= '1' when (sub_y_x='1' and ena_xy='1') or
 xi_yi_sel='1' else '0';

 -- Registers
 process begin
 wait on clk until clk='1';
 state <= next_state;
 if ena_x='1' then x <= x_i; end if;
 if ena_y='1' then y <= y_i; end if;
 if ena_r='1' then xo <= x; end if;
 rdy <= set_rdy;
 end process;
```

## GCD – spekuleeriv arvutamine (2 lahutajat)

*gcd-rtl5.vhdl*

```
-- Next state function of the FSM
process (state, rst, alu_ne) begin
 ena_xy <= '0'; ena_r <= '0';
 set_rdy <= '0'; xi_yi_sel <= '0';
 next_state <= state;
 case state is
 -- Wait for the new input data
 when S_wait =>
 if rst='0' then
 xi_yi_sel <= '1'; ena_xy <= '1';
 next_state <= S_start;
 end if;
 -- Calculate
 when S_start =>
 if alu_ne='1' then
 ena_xy <= '1';
 next_state <= S_start;
 else
 ena_r <= '1'; set_rdy <= '1';
 next_state <= S_ready;
 end if;
 -- Ready
 when S_ready =>
 ena_r <= '1'; set_rdy <= '1';
 next_state <= S_wait;
 end case;
 end process;

 -- Subtractor (x-y) / comparator (x<y)
 alu_o1 <= x - y;
 sub_y_x <= '1' when alu_o1(alu_o1'high)='1' else '0';

 -- Subtractor (y-x)
 alu_o2 <= y - x;

 -- Comparator (y/=x)
 alu_ne <= '1' when x /= y else '0';

 -- Multiplexers
 x_i <= xi when xi_yi_sel='1' else alu_o1;
 y_i <= yi when xi_yi_sel='1' else alu_o2;
 ena_x <= '1' when (sub_y_x='0' and ena_xy='1') or
 xi_yi_sel='1' else '0';
 ena_y <= '1' when (sub_y_x='1' and ena_xy='1') or
 xi_yi_sel='1' else '0';

 -- Registers
 process begin
 wait on clk until clk='1';
 state <= next_state;
 if ena_x='1' then x <= x_i; end if;
 if ena_y='1' then y <= y_i; end if;
 if ena_r='1' then xo <= x; end if;
 rdy <= set_rdy;
 end process;
```

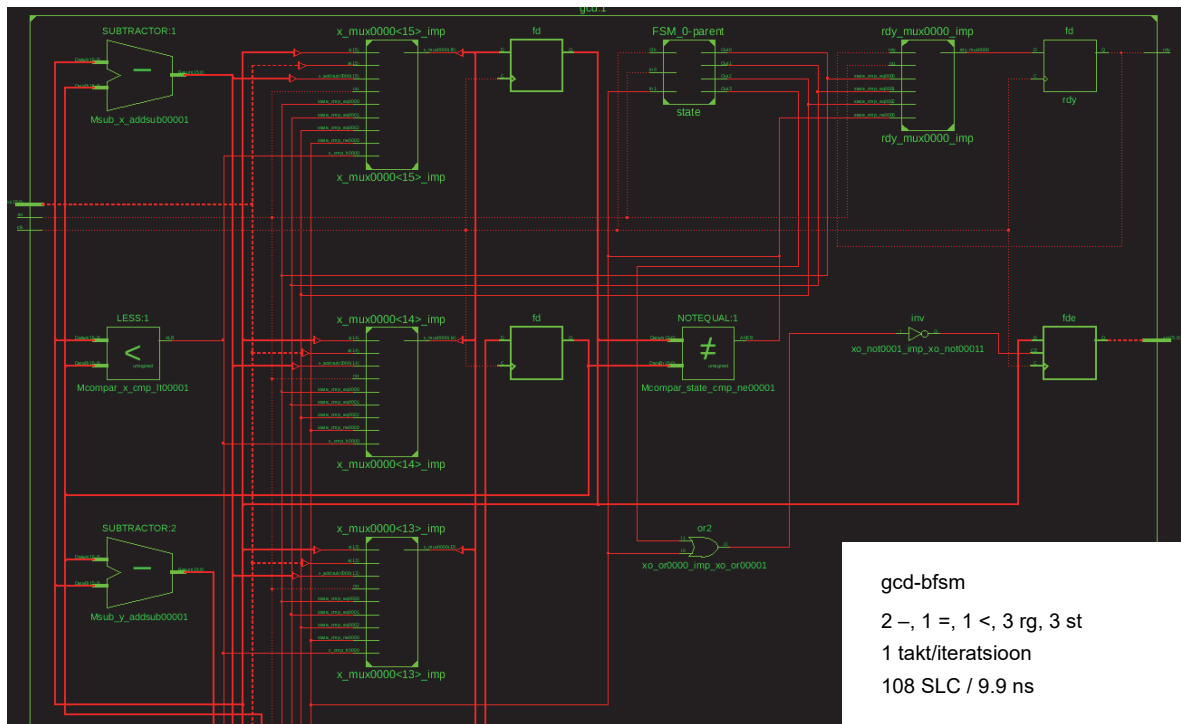
## GCD – sünteeside tulemused

| Tehnoloogia                  |          | FPGA   |      |         |      | ASIC   |      |        |      |
|------------------------------|----------|--------|------|---------|------|--------|------|--------|------|
| Piirangud <sup>1)</sup>      |          | 50 MHz |      | 100 MHz |      | 50 MHz |      | 25 MHz |      |
|                              | clk / FU | [SLC]  | [ns] | [SLC]   | [ns] | [e.g.] | [ns] | [e.g.] | [ns] |
| <i>gcd-bhv</i> <sup>2)</sup> | ? / ?    | 93     | 17.3 | -       | -    | 1141   | 20.0 | -      | -    |
| gcd-bhvc                     | 1 / 2+2  | -      | -    | -       | -    | 961    | 20.0 | 977    | 31.1 |
| gcd-bfsm                     | 1 / 2+2  | 108    | 9.9  | 108     | 9.4  | 911    | 19.4 | 984    | 30.8 |
| gcd-rtl1                     | 3 / 1    | 50     | 10.8 | 50      | 9.7  | 986    | 19.8 | 883    | 32.4 |
| gcd-rtl2                     | 2 / 1    | 48     | 10.8 | 48      | 10.0 | 931    | 19.9 | 882    | 32.3 |
| gcd-rtl3                     | 1 / 1+1  | 58     | 17.0 | 58      | 14.6 | 1134   | 20.0 | 928    | 40.0 |
| gcd-rtl4                     | 1 / 2    | 78     | 12.6 | 78      | 9.0  | 976    | 19.9 | 928    | 29.0 |
| gcd-rtl5                     | 1 / 2+1  | 58     | 8.0  | 58      | 7.6  | 915    | 20.0 | 932    | 26.9 |

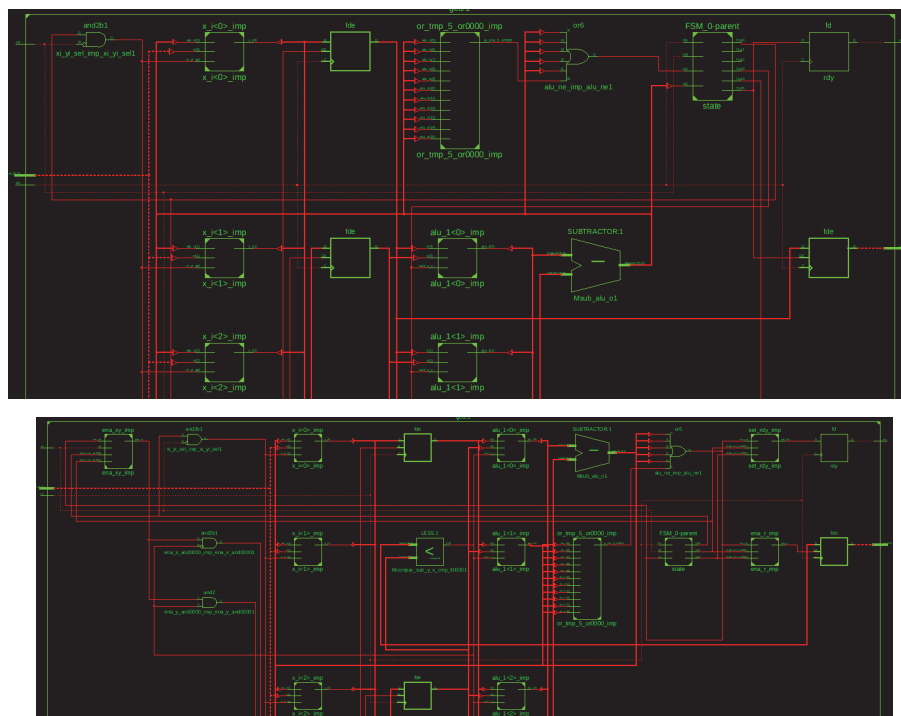
<sup>1)</sup> Taktiperiood oli ainuke piirang

<sup>2)</sup> gcd-bhv sünteesiti prototüüp kõrgtasemesüntesaatoriga xTractor

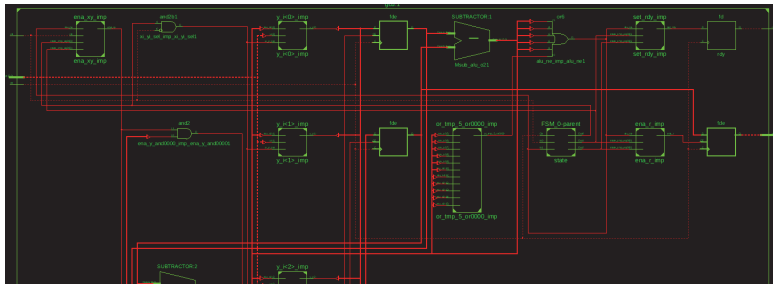
## GCD – kust tulevad need erinevused?



## GCD – kust tulevad need erinevused?



## GCD – kust tulevad need erinevused?

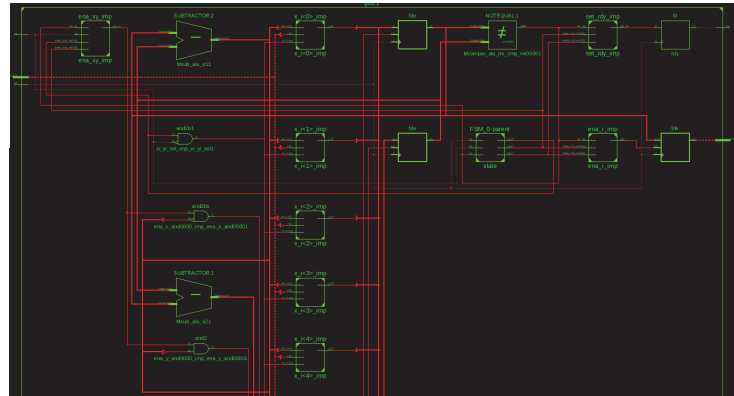


gcd-rtl4

2 –, 3 rg, 3 st

1 takt/iteratsioon

78 SLC / 12.6 ns



gcd-rtl5

2 –, 1 / =, 3 rg, 3 st

1 takt/iteratsioon

58 SLC / 8.0 ns

## Realisatsiooni näide #2 – täisarvude korrutamine

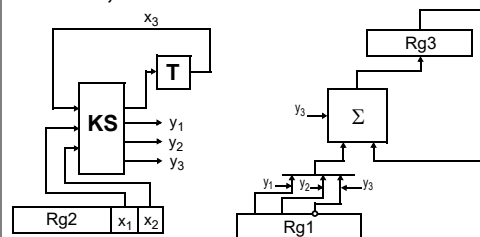
- Märgita täisarvude korrutamine, 2-bitti korruga (radix-4)
- $o = a * b$ 
  - $\dots + [0,1,2,3]*b \rightarrow 3*b = 4*b - b$
  - samm, kui eelmine bitipaar ei olnud '11'
    - $b_{10}=00 ? \rightarrow$  ei midagi
    - $b_{10}=01 ? \rightarrow o+=a$
    - $b_{10}=10 ? \rightarrow o+=2*a$  [ $o+=(a<<1)$ ]
    - $b_{10}=11 ? \rightarrow o=a$  [ja jätkame meelde]
  - samm, kui eelmine bitipaar oli '11'
    - $b_{10}=00 ? \rightarrow o+=a$  [ $4-1==3$ ]
    - $b_{10}=01 ? \rightarrow o+=2*a$  [ $2==1+1$ ]
    - $b_{10}=10 ? \rightarrow o=a$  [ $3==2+1$ , jälle meelde]
    - $b_{10}=11 ? \rightarrow$  ei midagi [ $4==3+1$ , jälle meelde]

### Korrutamine 2 kohaga

00 - blok.  
01 -  $1*Rg1$   
10 -  $2*Rg1$  ( $L1(Rg1)$ )  
11 -  $-Rg1$ ; +1 järgmisesse järku

$11_2 = 100 - 1 = 101$   
N: 0,01101101

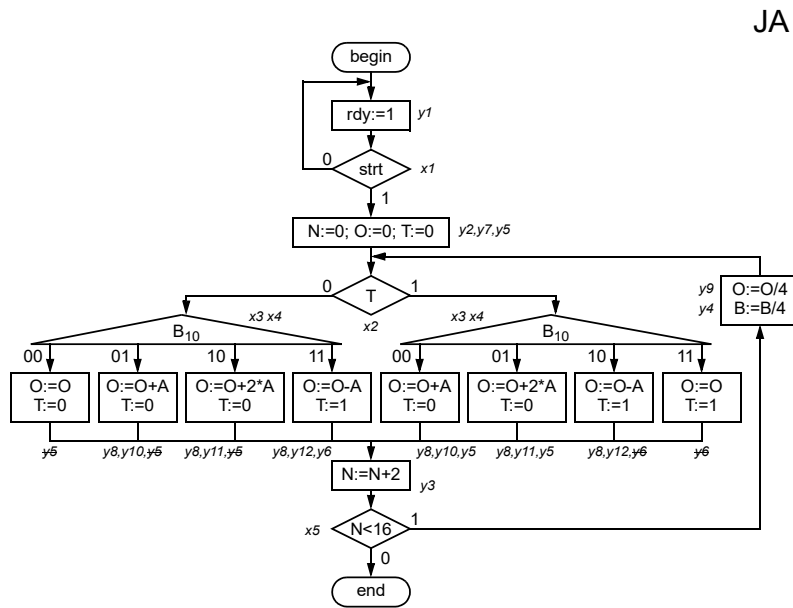
0,10010101



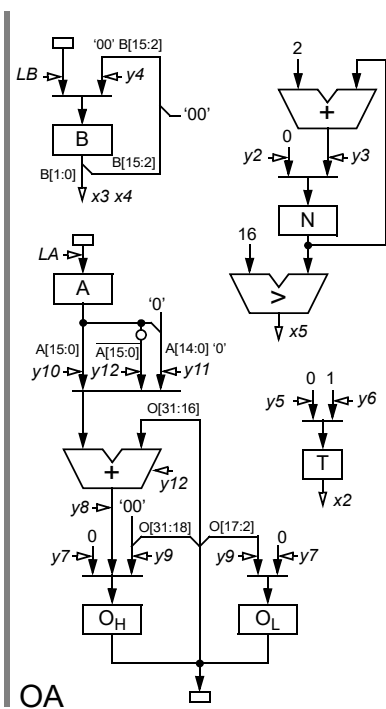
$y1: Rg3 := Rg3 + Rg1$   
 $y2: Rg3 := Rg3 + L1(Rg1)$   
 $y3: Rg3 := Rg3 - Rg1$

Margus Kruus: Korrutusalgoritmid

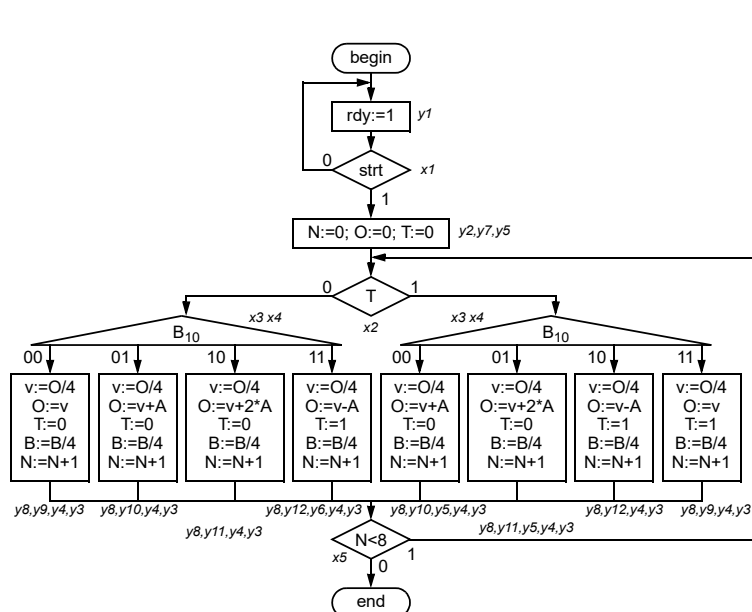
## Realisatsiooni näide (ver. 1)



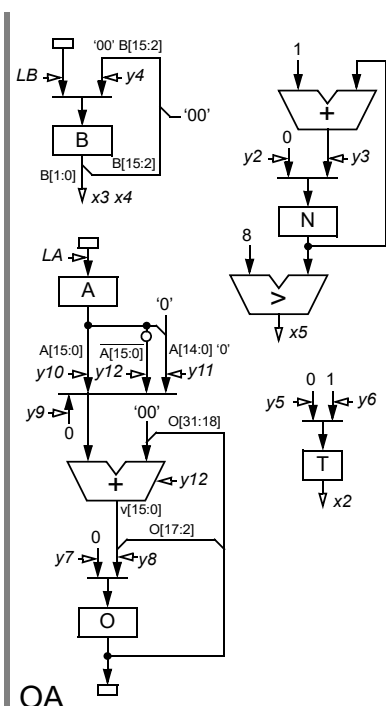
JA Moore: i 5, o 12, s 12, t 28; Mealy: i 5, o 12, s 4, t 13 (tagasiside OK)  
 OA 2 sum., 1 võrdl., 5 (6) mux, 5 (6) reg.  
**3 takti** iteratsiooni kohta, kokku 24 (+1) takti



## Realisatsiooni näide (ver. 2)

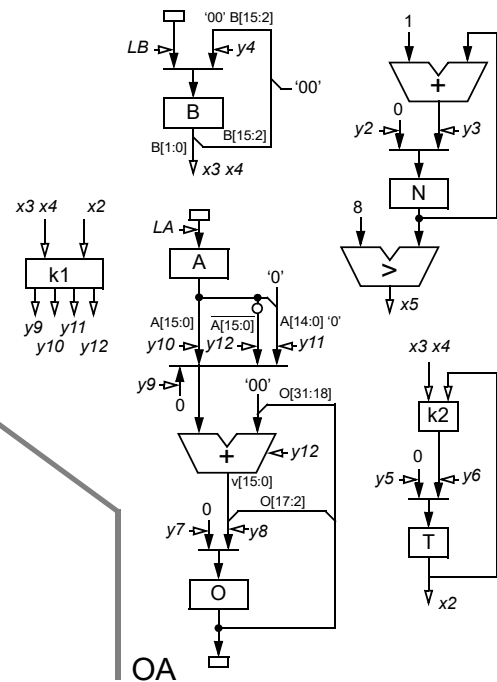
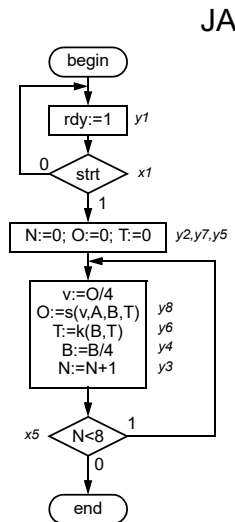


JA Moore: i 5, o 12, s 10, t 82; Mealy: i 5, o 12, s 3, t 17  
 OA 2 sum., 1 võrdl., 5 mux, 5 reg.  
**1 takt** iteratsiooni kohta, kokku 9 (+1) takti



## Realisatsiooni näide (ver. 3)

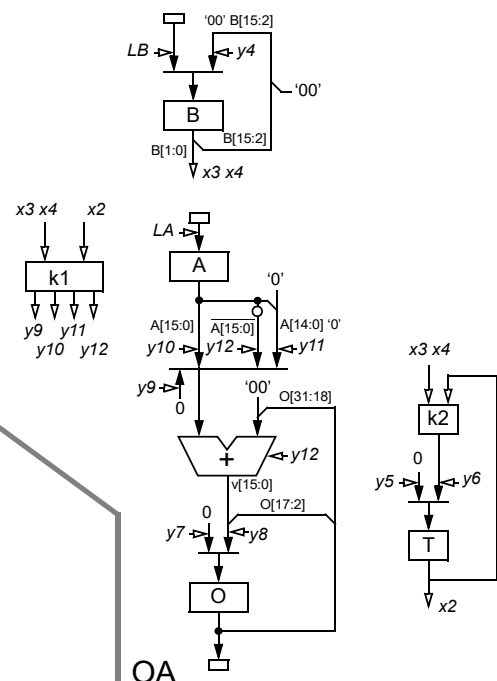
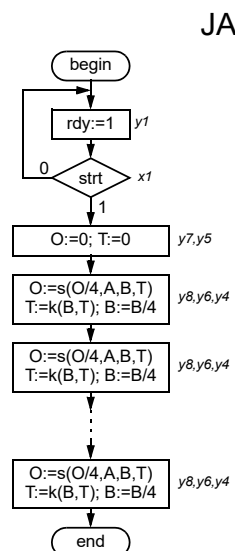
- Keerukus: JA → OA
- siirded ~ kombinatsiooniskeem



JA Moore: i 2, o 8, s 3, t 5; Mealy: i 2, o 8, s 3, t 5  
 OA 2 sum., 1 võrdl., 5 mux, 5 reg., 1 (2) f-n  
 1 takt iteratsiooni kohta, kokku 9 (+1) takti

## Realisatsiooni näide (ver. 3)

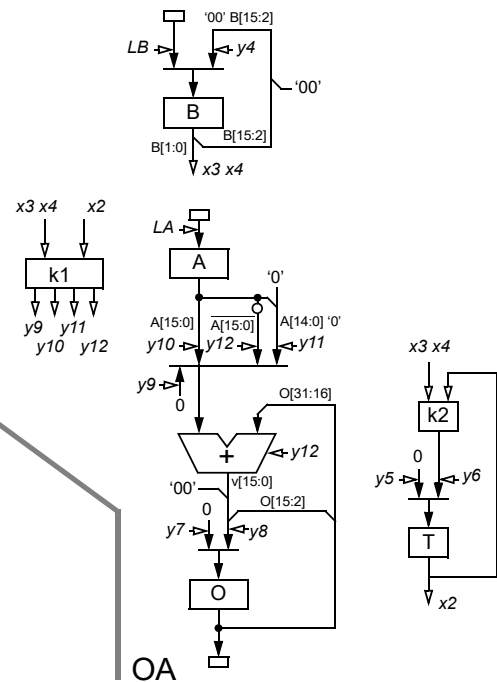
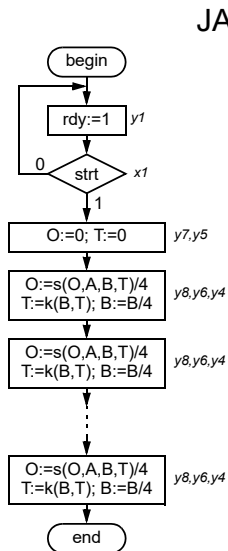
- Keerukus: OA → JA
- loendur ~ automaat
- Veelgi kiirem:
  - ühitada nullimine (y7,y5) ja esimene iteratsioon
  - OA: peaaegu sama
  - JA: 9 olekut
  - kiirus: 8 (+1) takti



JA Moore: i 1, o 6, s 10, t 11; Mealy: i 1, o 6, s 10, t 11  
 OA 1 sum., 4 mux, 4 reg., 1 (2) f-n  
 1 takt iteratsiooni kohta, kokku 9 (+1) takti

## Realisatsiooni näide (ver. 4)

- Eelmised variandid
- 2 noorimat järku kasutamata
- korrektse tulemuse asukoht?
- Nihutamine pärast liitmist/lahutamist
- 2 vanimat järku alati 0-d!!!
- Pisi-erinevused OA-s



JA Moore: i 1, o 6, s 10, t 11; Mealy: i 1, o 6, s 10, t 11  
 OA 1 sum., 4 mux, 4 reg., 1 (2) f-n  
 1 takt iteratsiooni kohta, kokku 9 (+1) takti

## Realisatsioonide võrdlused

- Seitse erinevat korrutaja realisatsiooni
- paralleelne, bitt korruga, 2 bitti korruga (versioonide 3 ja 4 erinevad VHDL koodid)
- VHDL koodide erinevused põhjustatud sünteesi juhtimise vajadusest
  - “what you write is what you get...”
- VHDL failid – <https://ati.ttu.ee/~lrv/IAS0150/multiplier/>

| Tüüp                    | nr. | Synopsys DC         |      | Xilinx ISE     |      |
|-------------------------|-----|---------------------|------|----------------|------|
|                         |     | comb. & reg. [e.g.] | [ns] | [slices]       | [ns] |
| paralleelne             | 0   | 685 + 112 = 797     | 20.0 | 0 (1/24 mult.) | ~10  |
| radix-2 (bhv-rtl)       | 1   | 227 + 508 = 735     | 15.5 | 46 (7680)      | 6.7  |
| radix-4 (v.4, bhv-rtl)  | 2   | 402 + 511 = 913     | 20.0 | 52 (7680)      | 7.8  |
| radix-4 (v.4, ~90% rtl) | 3   | 303 + 511 = 814     | 15.9 | 37 (7680)      | 7.1  |
| radix-4 (v.4, 100% rtl) | 4   | 179 + 511 = 690     | 19.9 | 34 (7680)      | 7.3  |
| radix-4 (v.3, bhv-rtl)  | 5   | 397 + 514 = 911     | 20.0 | 51 (7680)      | 8.2  |
| radix-4 (v.3, 100% rtl) | 7   | 182 + 511 = 693     | 18.7 | 36 (7680)      | 6.4  |