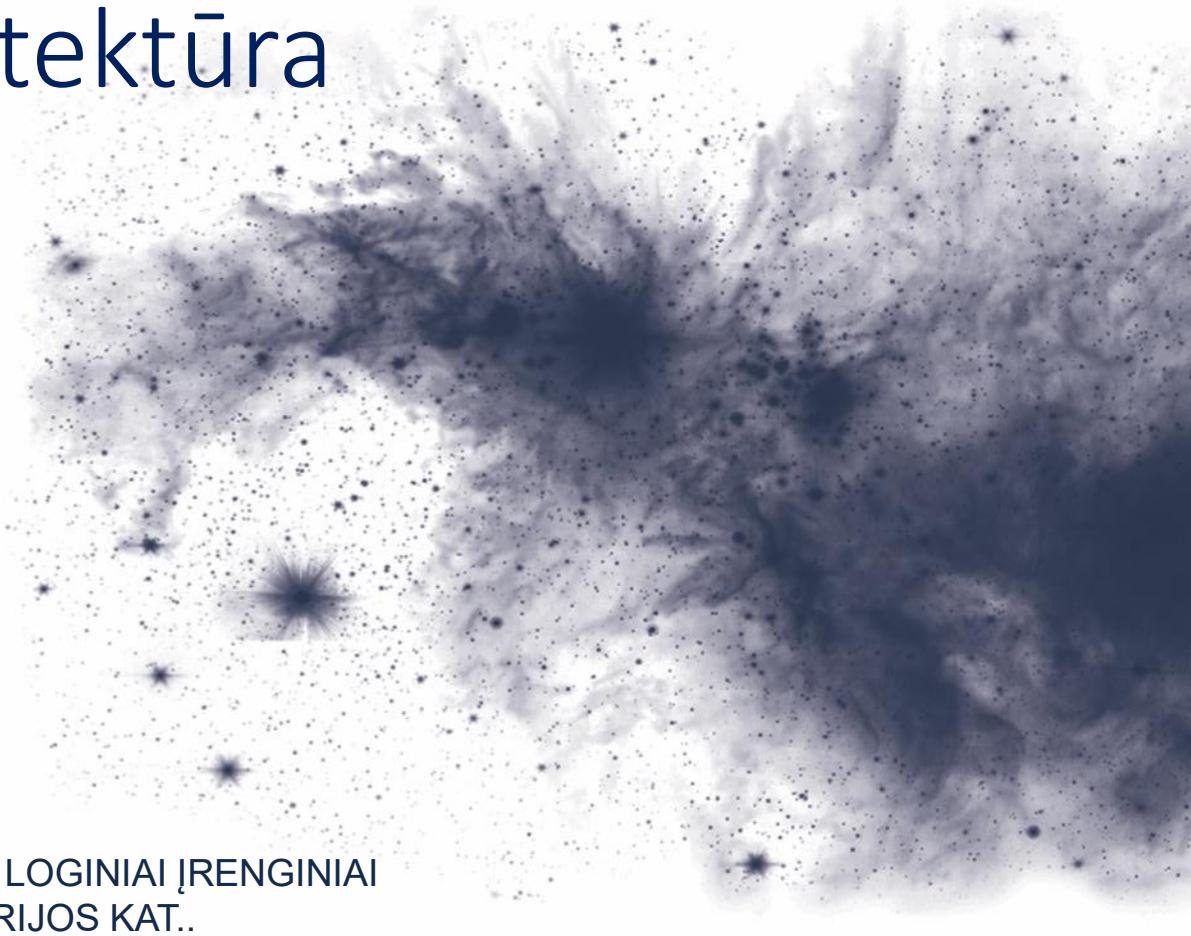


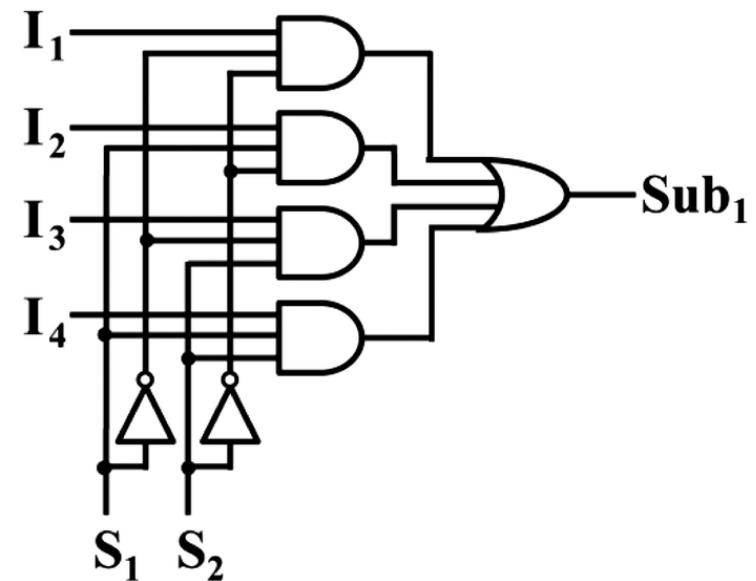
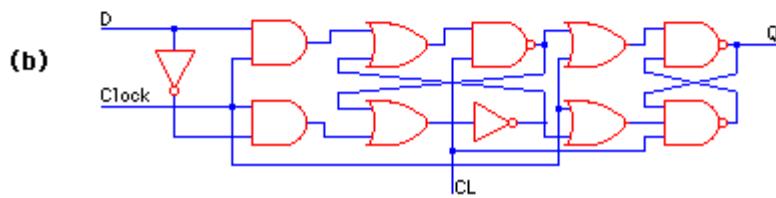
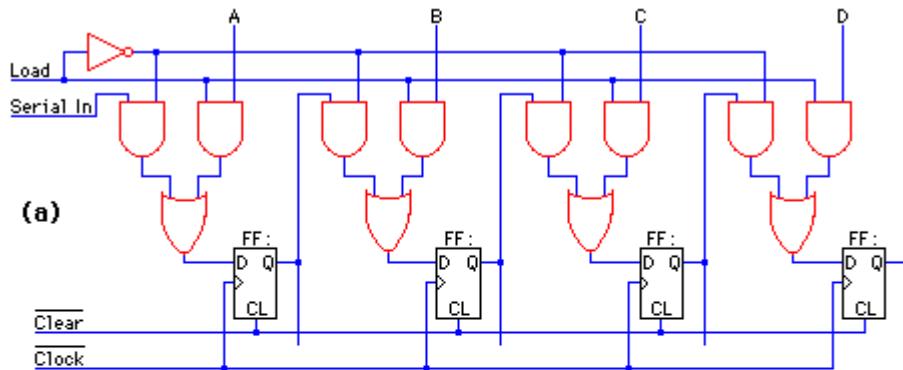
# Programuojami loginiai įrenginiai FPGA architektūra



**R. Ramanauskas**

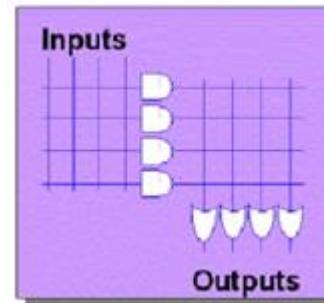
T170B114 PROGRAMUOJAMI LOGINIAI ĮRENGINIAI  
KTU ELEKTRONIKOS INŽINERIJOS KAT..  
2019

# Programuojami loginiai įrenginiai

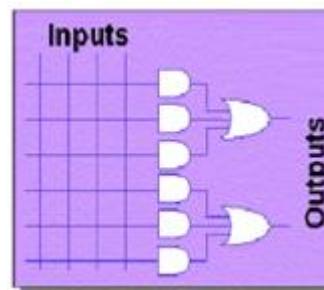


# Programuojami loginiai įrenginiai

- PLD - Programmable Logic Device
  - Programuojamos loginės matricos (PLM)  
Programmable Logic Array (PLA)

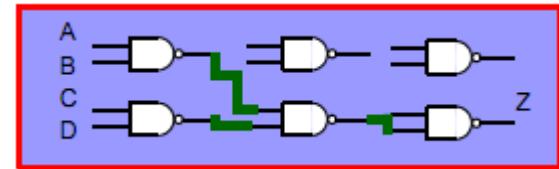
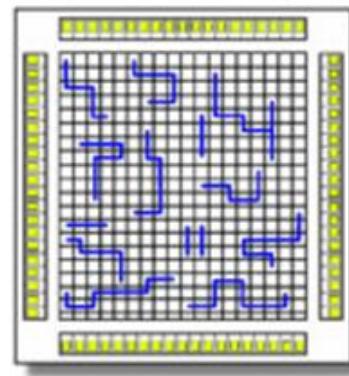
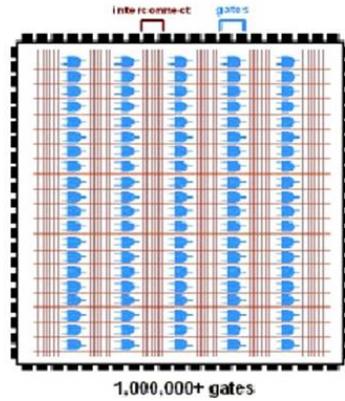


- Programuojama matricinė logika (PML)  
Programmable Array Logic (PAL)



# Programuojami loginiai įrenginiai

- Ventilių matricos (VM)  
Gate Array Logic (GAL)



# Programuojami loginiai įrenginiai



- CPLD - Complex Programmable Logic Device

Sudarytas iš keleto PLD tarpusavyje sujungtų programuojamais sujungimais (programuojamomis jungtimis)

- FPGA - Field Programmable Gate Arrays

Lauku programuojami ventilių masyvai

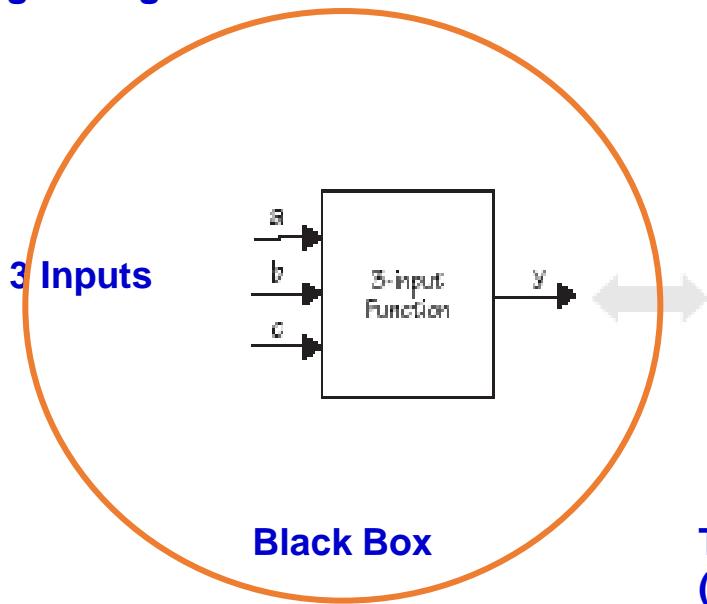
# Programuojami loginiai įrenginiai



- Funcinis lankstumas
- Algoritmų lygiagretinimas
- Eksplatacinės savybės
- Laikas iki gamybos
- Ilgalaikis techninis palaikymas
- Kaina

# Loginės funkcijos

Digital Logic Function



Product AND ( $\&$ )  
Sum OR ( $\mid$ )

<i>a</i>	<i>b</i>	<i>c</i>	<i>y</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$y = (\bar{a} \& \bar{b} \& c) \mid (\bar{a} \& b \& c) \mid (a \& \bar{b} \& \bar{c}) \mid (a \& \bar{b} \& c)$

Sum-of-Products Expression

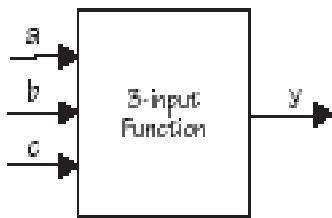
Truth Table  
(Look Up Table LUT)

SUM of PRODUCTS

# Loginės funkcijos

Digital Logic Function

3 Inputs



Product AND (&  
Sum OR (|)

$$y = (\bar{a} \& \bar{b} \& c) | (\bar{a} \& b \& c) | (a \& \bar{b} \& \bar{c}) | (a \& \bar{b} \& c)$$

Sum-of-Products Expression

Black Box

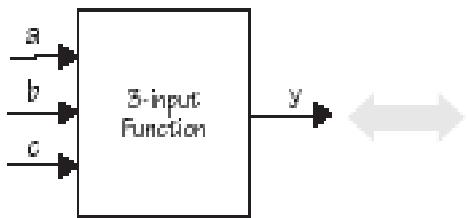
Truth Table  
(Look Up Table LUT)

SUM of PRODUCTS

# Loginės funkcijos

Digital Logic Function

3 Inputs



Product AND (&  
Sum OR (|)

<i>a</i>	<i>b</i>	<i>c</i>	<i>y</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$y = (\bar{a} \& \bar{b} \& c) | (\bar{a} \& b \& c) | (a \& \bar{b} \& \bar{c}) | (a \& \bar{b} \& c)$

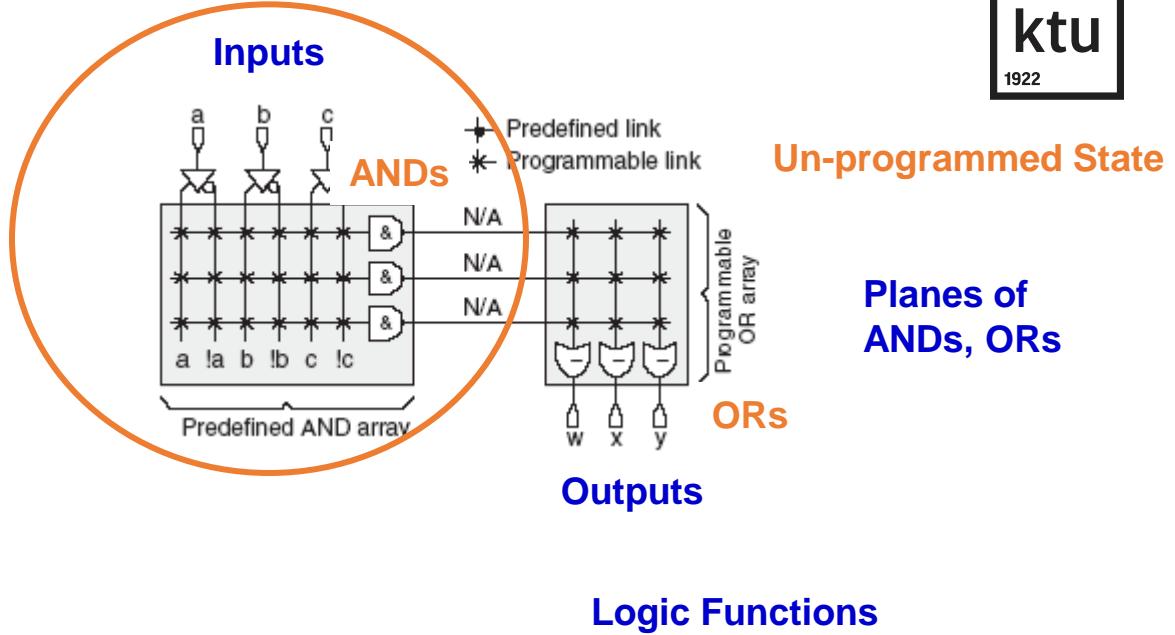
Sum-of-Products Expression

Black Box

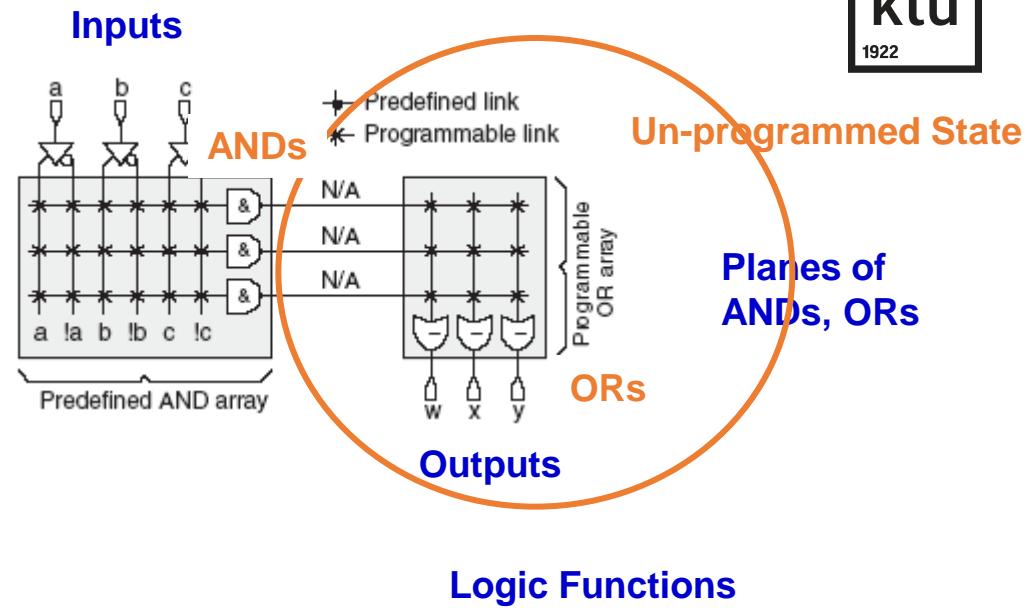
Truth Table  
(Look Up Table LUT)

SUM of PRODUCTS

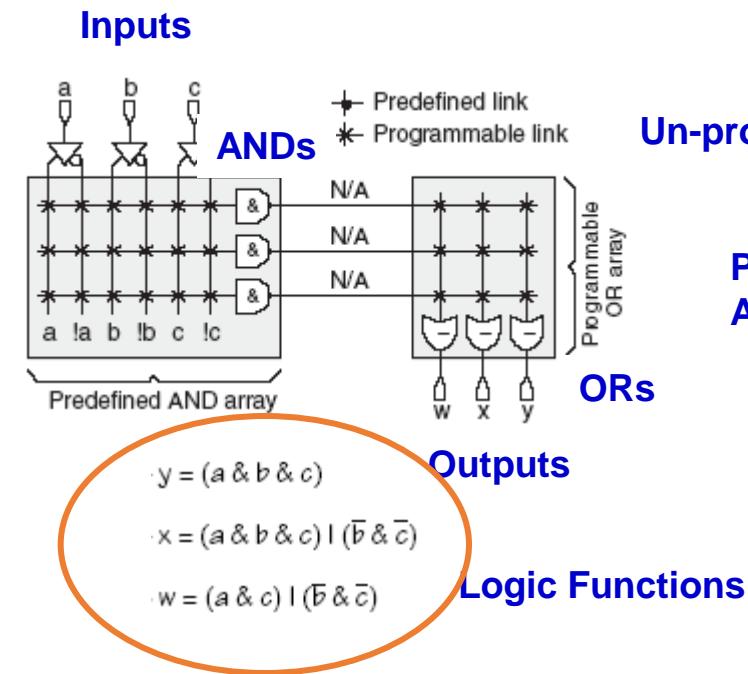
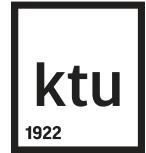
# Loginēs funkcijos realizacija



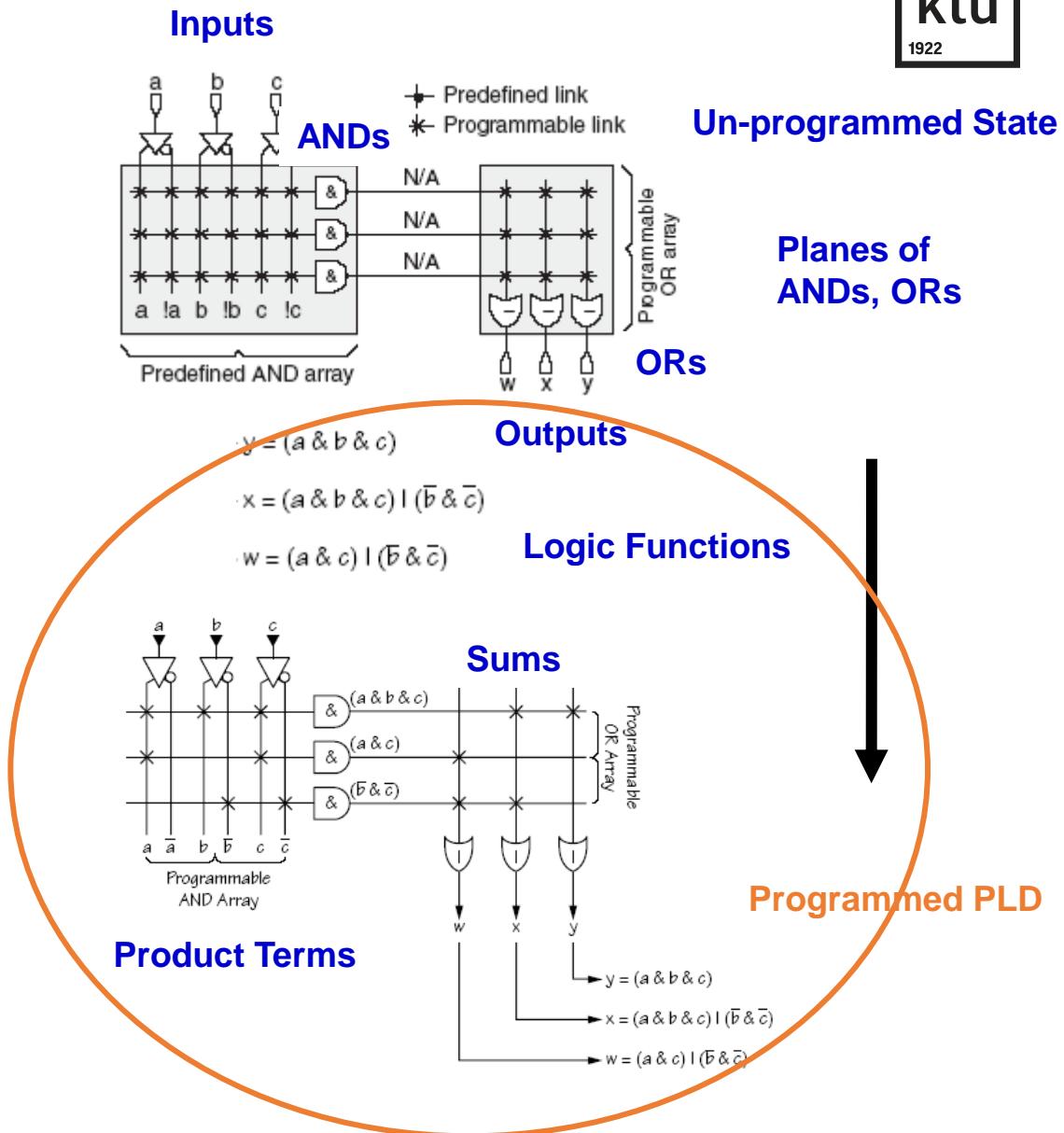
# Loginēs funkcijos realizacija



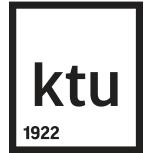
# Loginēs funkcijos realizacija



# Loginės funkcijos realizacija



# Loginēs funkcijos realizacija

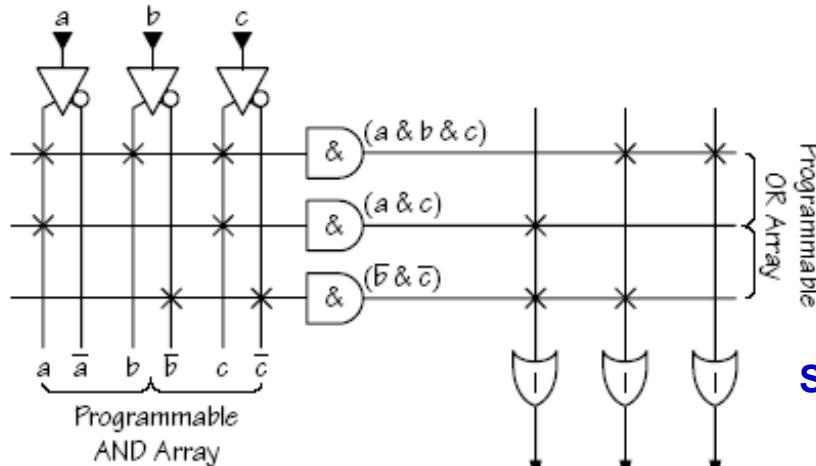


## Logic Functions

$$y = (a \& b \& c)$$

$$x = (a \& b \& c) \mid (\bar{b} \& \bar{c})$$

$$w = (a \& c) \mid (\bar{b} \& \bar{c})$$



## Product Terms

Programmed PLD

## Sums

$$\begin{aligned} y &= (a \& b \& c) \\ x &= (a \& b \& c) \mid (\bar{b} \& \bar{c}) \\ w &= (a \& c) \mid (\bar{b} \& \bar{c}) \end{aligned}$$

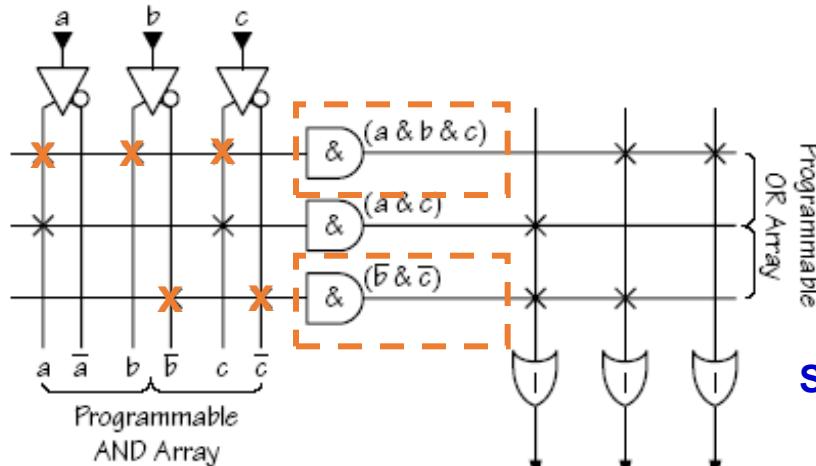
# Loginēs funkcijos realizacija

## Logic Functions

$$y = (a \& b \& c)$$

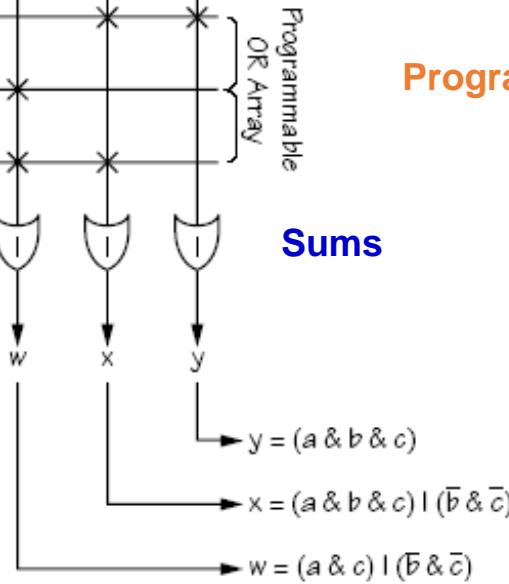
$$x = (a \& b \& c) \mid (\bar{b} \& \bar{c})$$

$$w = (a \& c) \mid (\bar{b} \& \bar{c})$$



## Product Terms

Programmed PLD



## Sums

# Loginės funkcijos realizacija

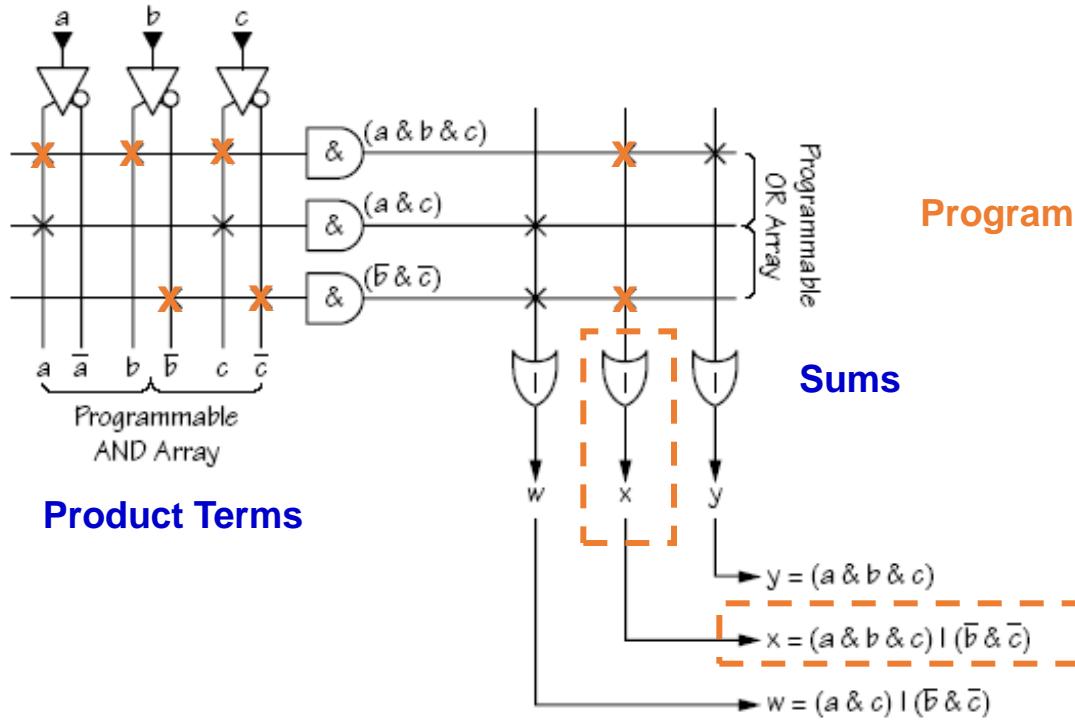
## Logic Functions

$$y = (a \& b \& c)$$

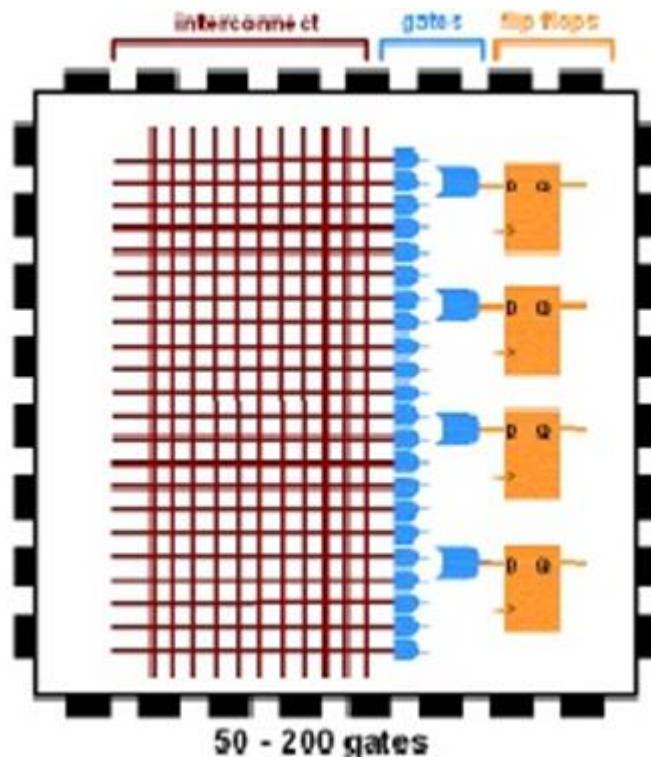
$$x = (a \& b \& c) \mid (\bar{b} \& \bar{c})$$

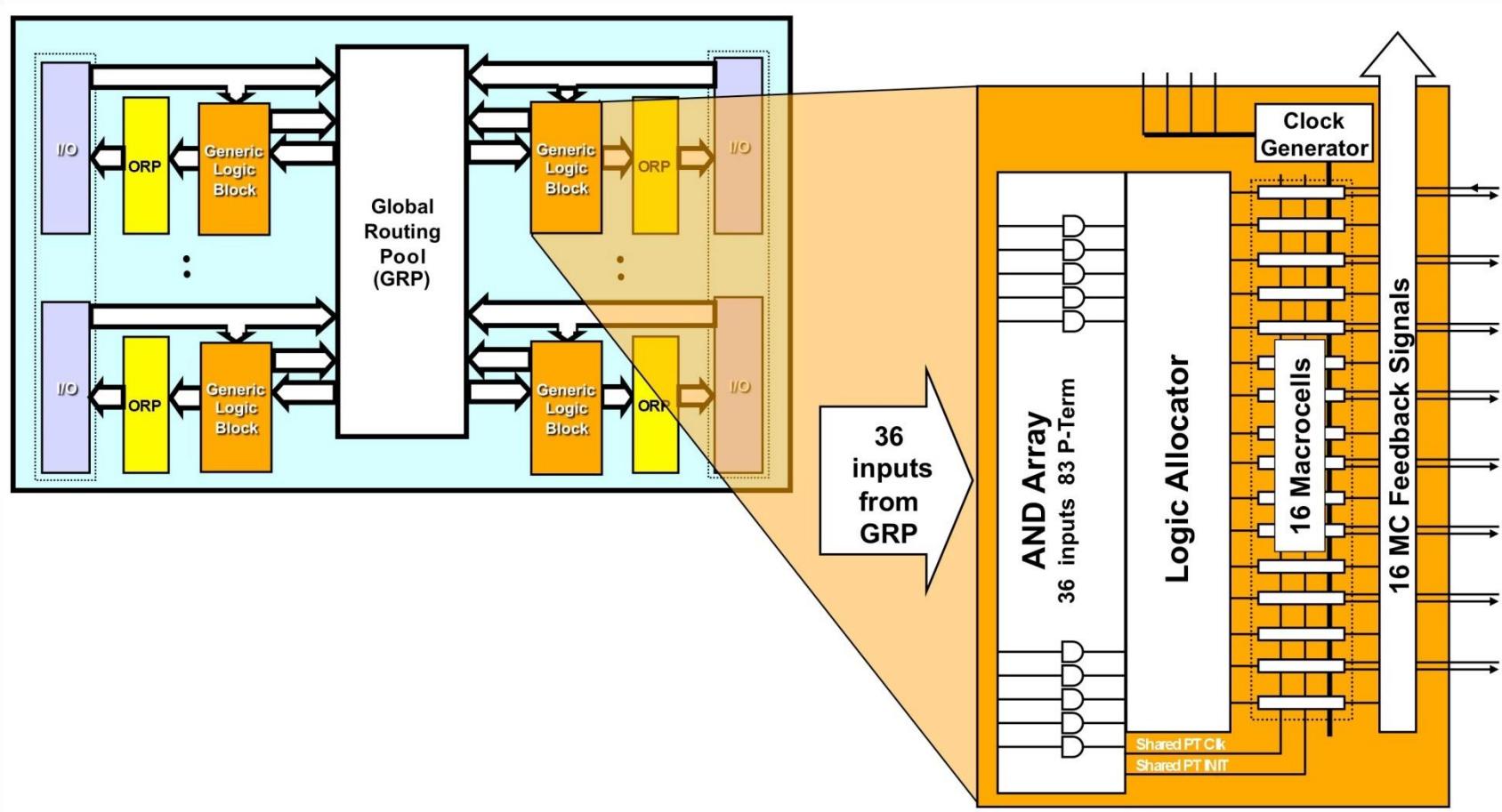
$$w = (a \& c) \mid (\bar{b} \& \bar{c})$$

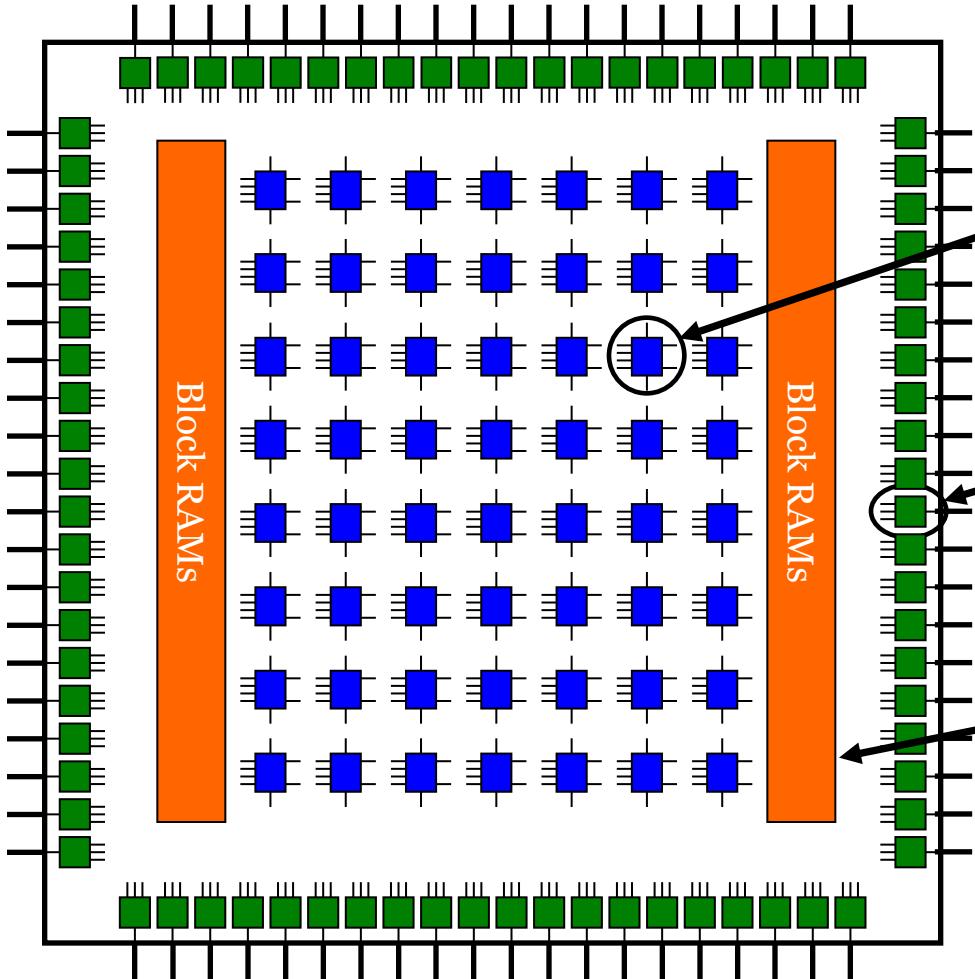
## GLUE LOGIC



# CPLD





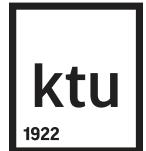


**Konfiguruojami  
Loginiai  
Blokai**

**I/O  
Blokai**

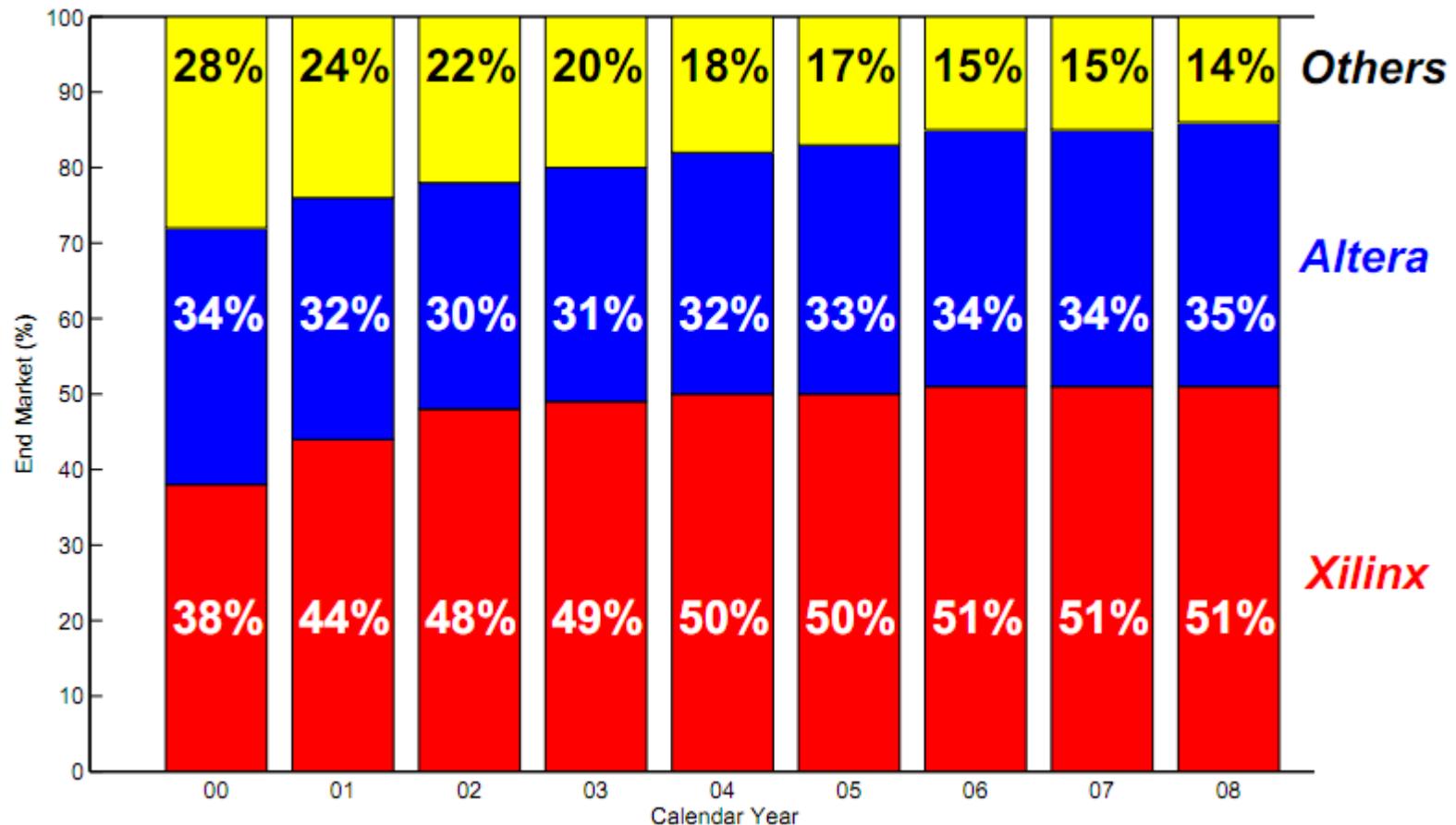
**RAM  
Blokai**

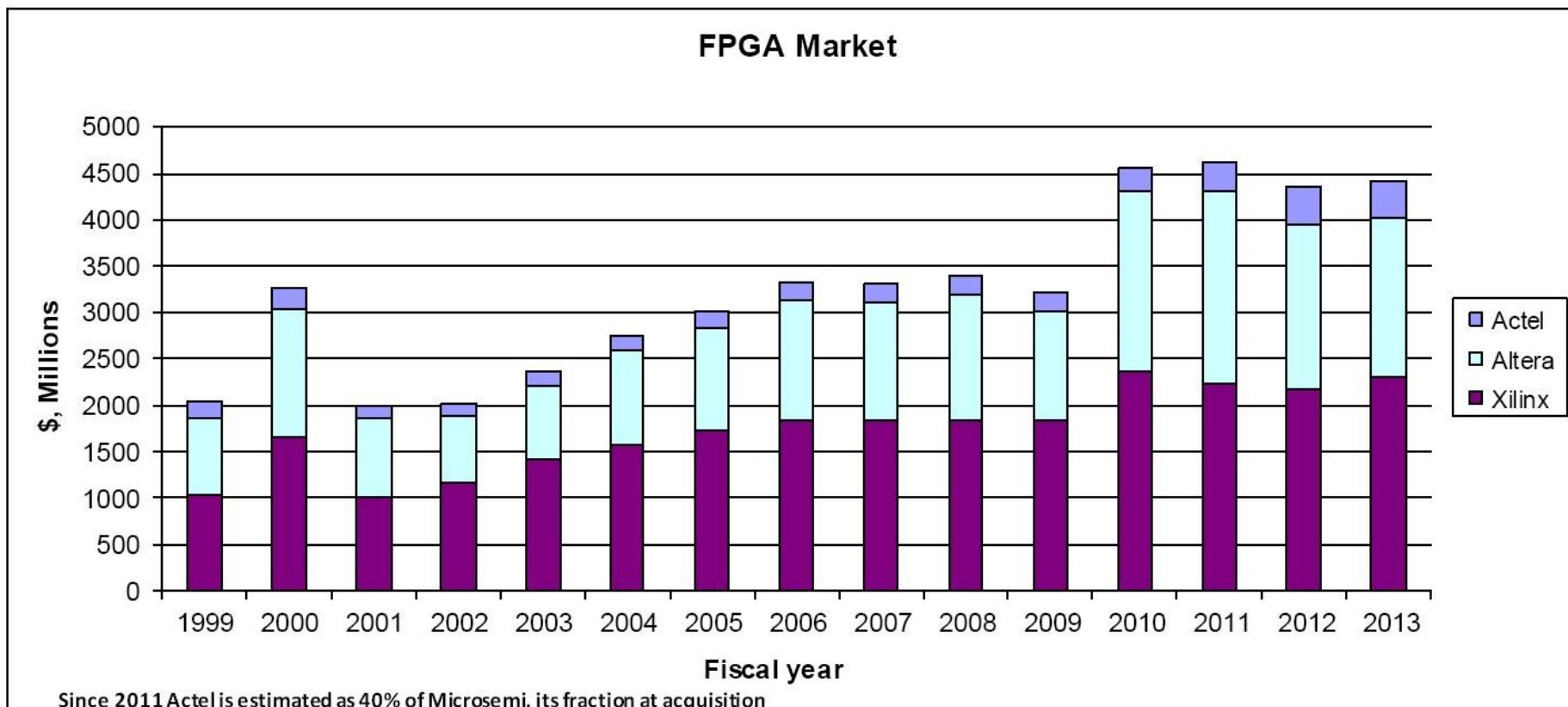
# Gamintojai



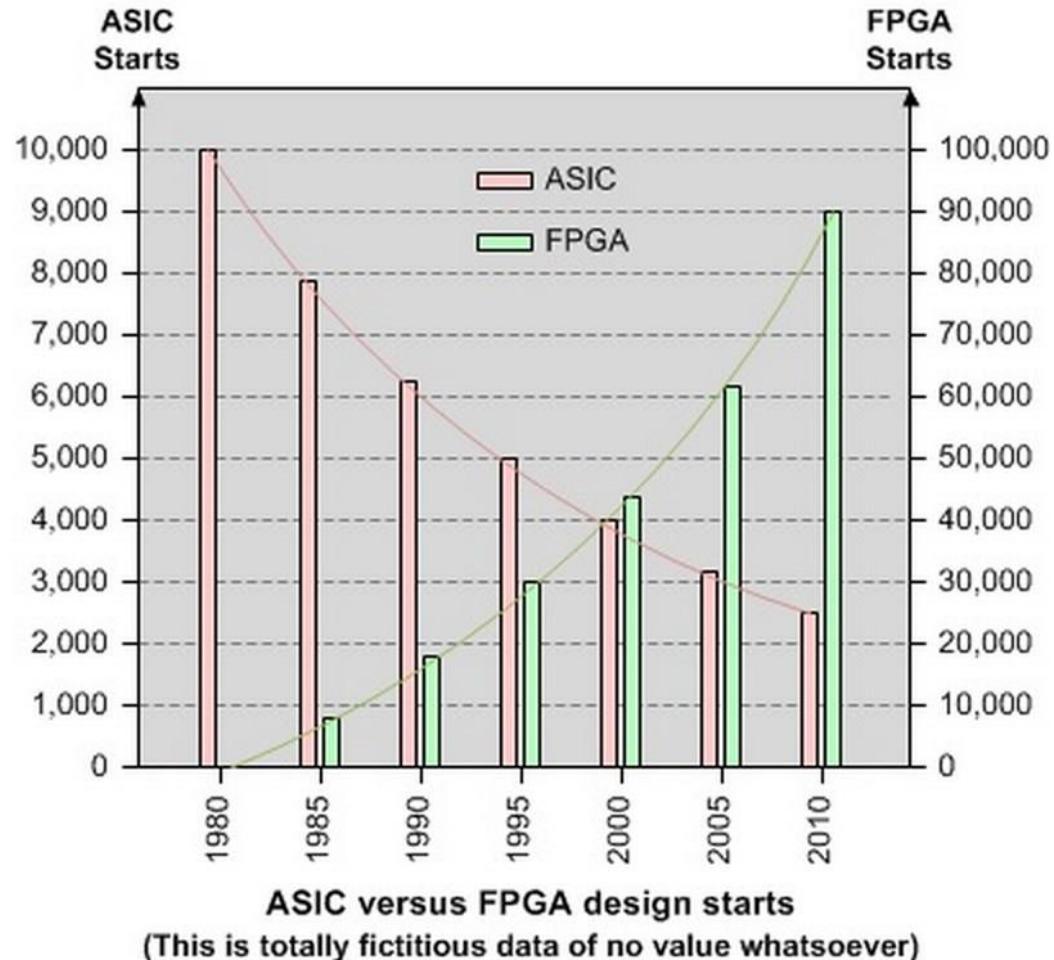
- Pagrindiniai gamintojai:
  - Xilinx: Spartan, Virtex
  - Altera: Cyclone, Arria, Stratix
  - Lattice Semiconductor
  - Actel
  - Atmel
  - ...

# Gamintojai

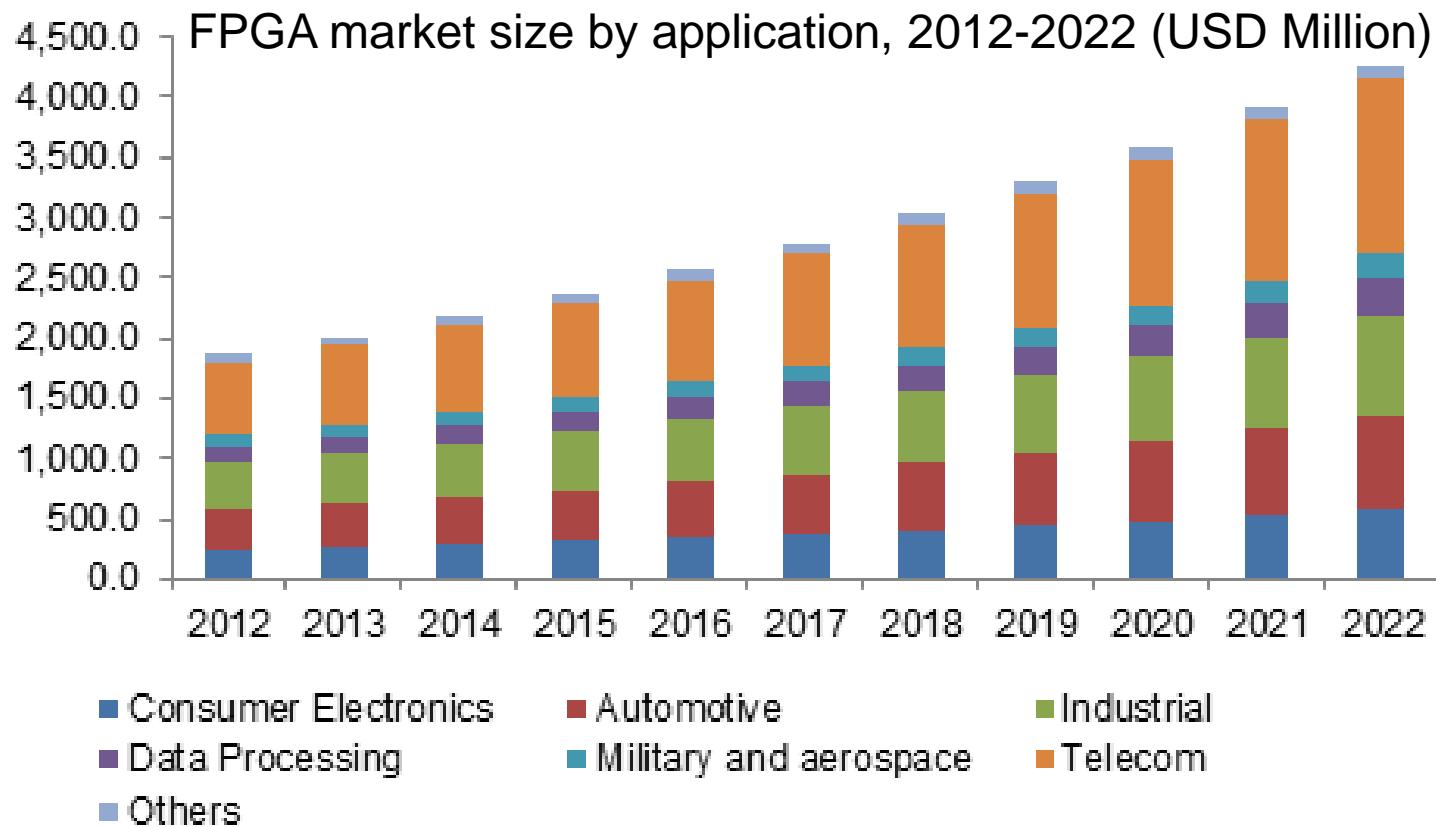




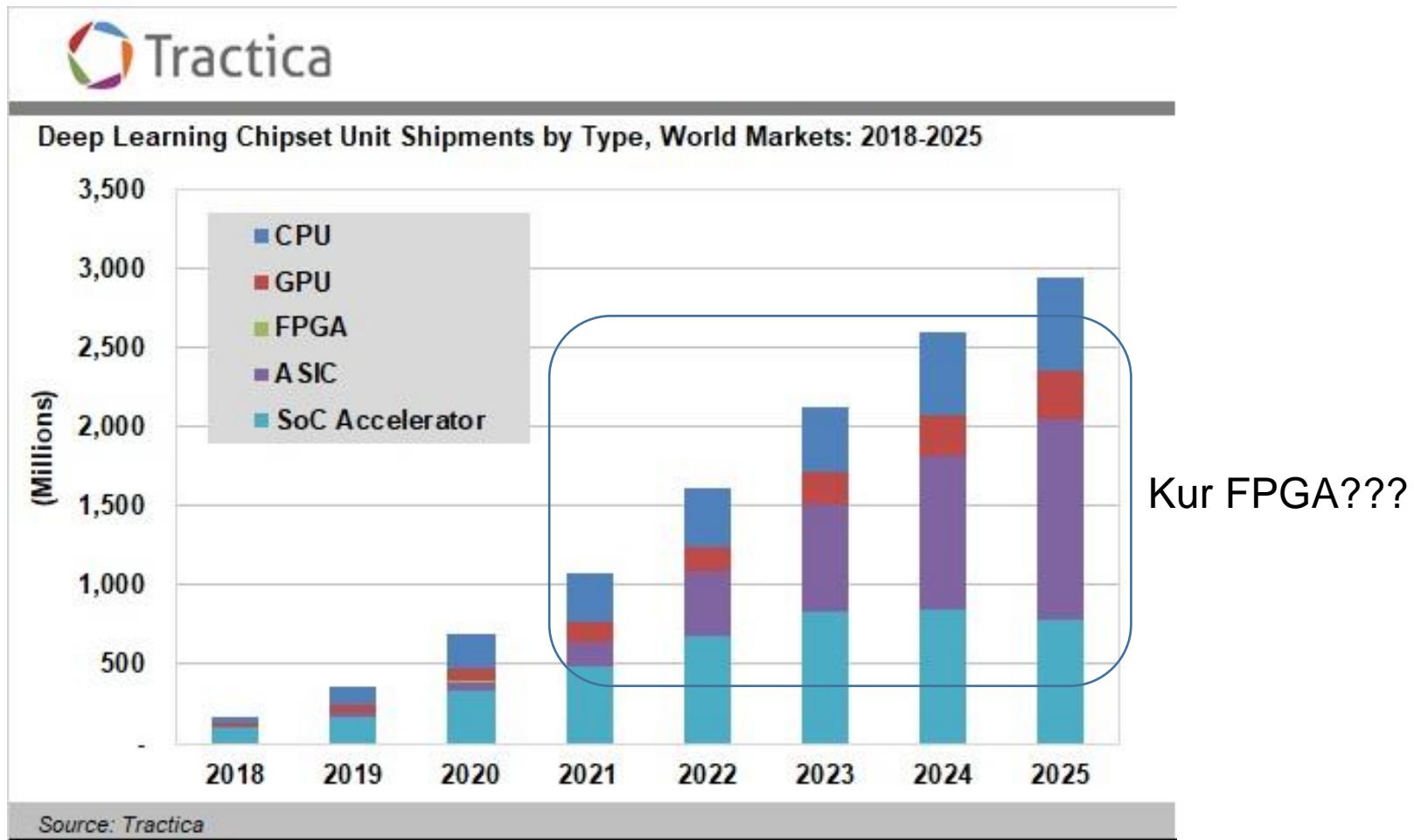
# Pramonės tendencijos



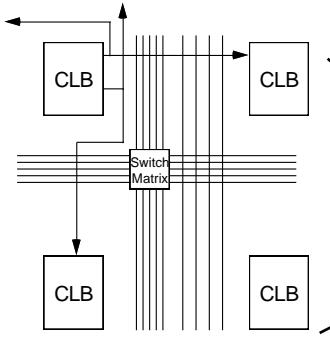
# Pramonės tendencijos



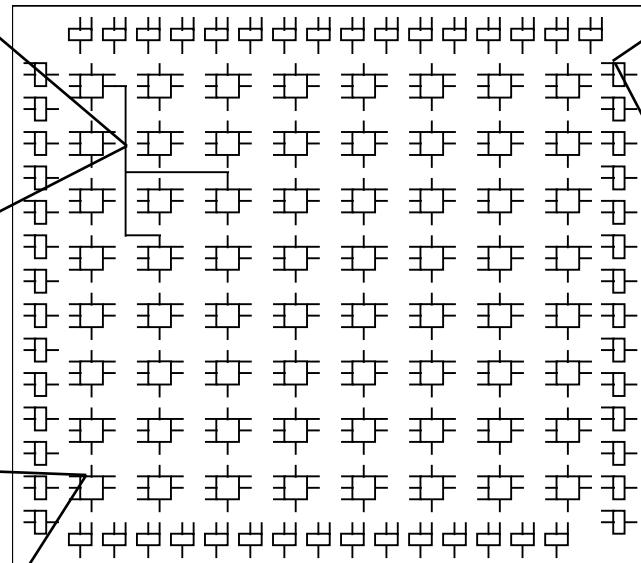
# Pramonės tendencijos



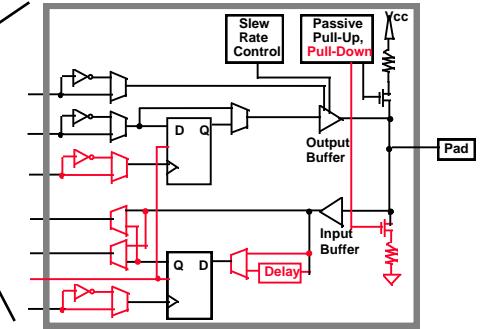
# FPGA struktūra



**Programmable  
Interconnect**



**Configurable  
Logic Blocks (CLBs)**



**I/O Blocks (IOBs)**

# FPGA struktūra



- Programuojami sujungimai
- Konfiguruojami loginiai blokai (CLB)
  - Komutacinė matrica
  - Loginės celės (slice)
    - ✖ SLICEM
      - LUT
      - Memory
      - SR16
    - ✖ SLISEL
      - LUT
- I/O blokai (I/O Blocks)
- Atminties blokai (RAM Blocks)
- Daugintuvų blokai (Multipliers)
- Sinchro valdiklio blokas (DCM)
- Kita
  - DSP blokai

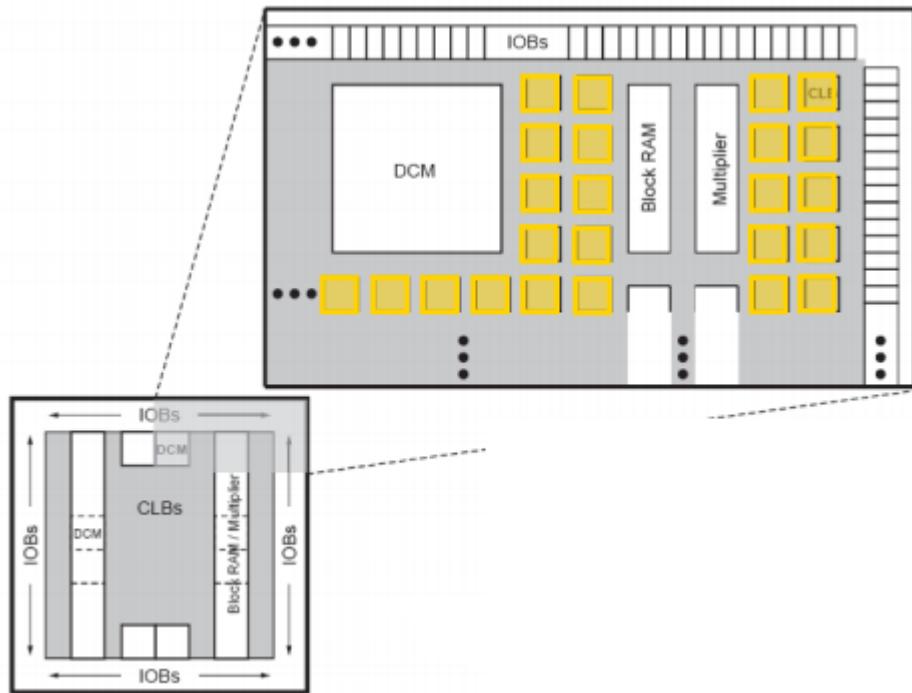
## Konfiguruojami loginiai blokai

CLB (Configurable Logic Blocks)

- Kristale daugiau kaip 1000 KLB;
- Kiekvieną KLB sudaro 2-4 loginės celės, kurias savo ruožtu sudaro loginis grandynas, programuojami registrai (D, T, JK), multipleksoriai, vidinių ir išorinių signalų jungiamiji grandynai.

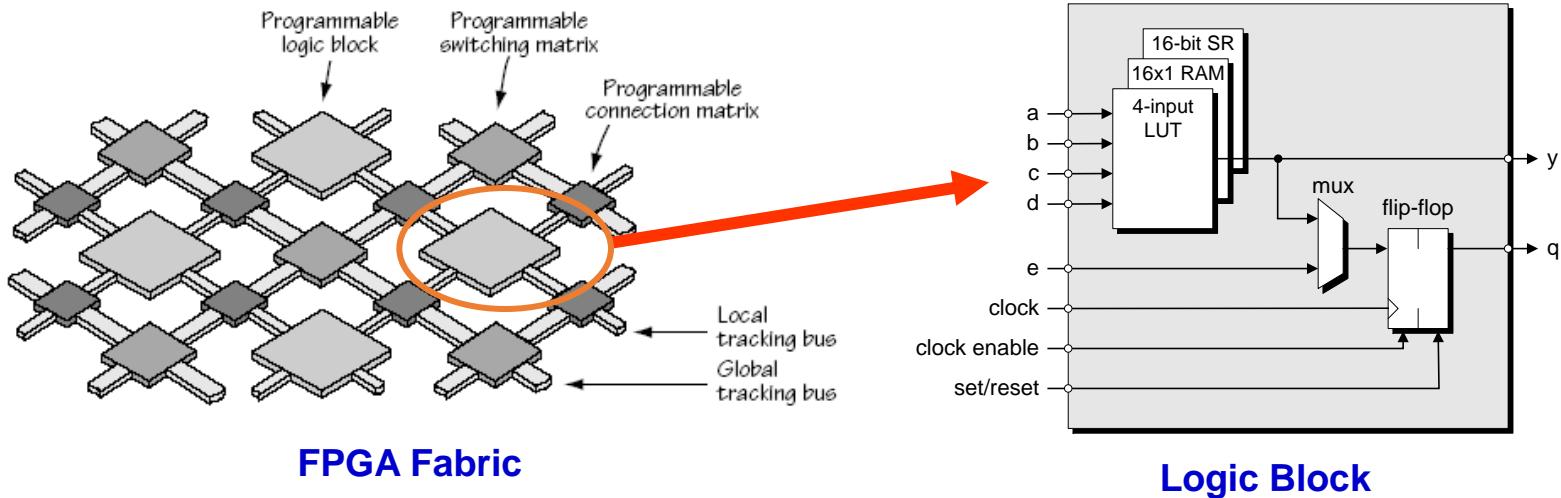
Atliekamos funkcijos:

- loginės operacijos;
- signalo užlaikymo (flip-flop);
- postumio registro;
- paskirstytos atminties;

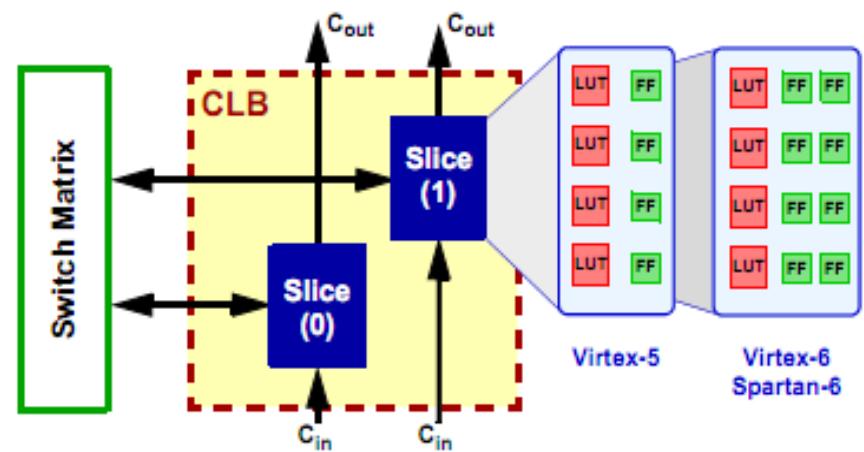
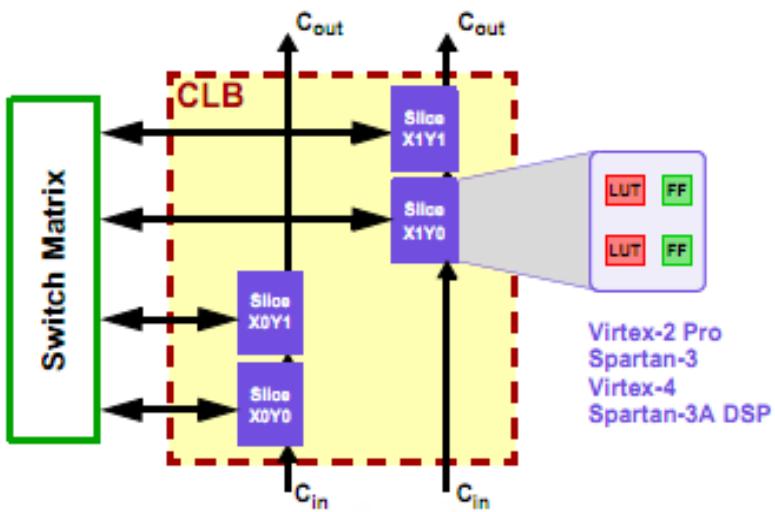


# FPGA.CLB (Xilinx)

## Konfiguruojami loginiai blokai CLB (Configurable Logic Blocks)

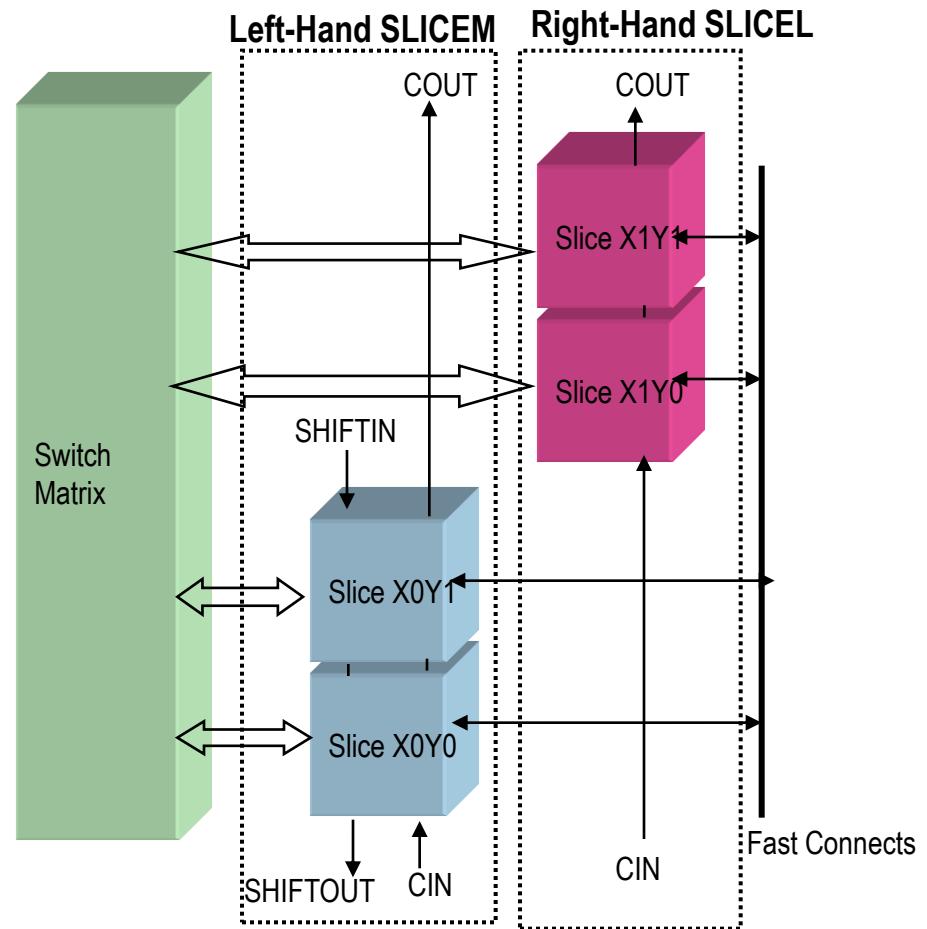


## Konfiguruojamie loginiai blokai CLB (Configurable Logic Blocks)

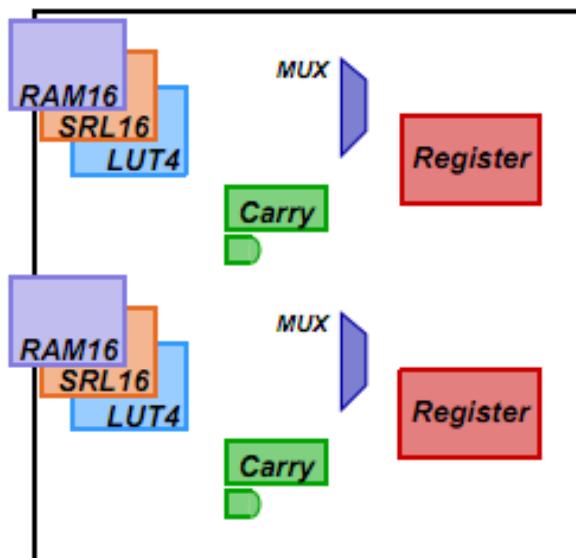


# Spartam-3E.CLB.Slice

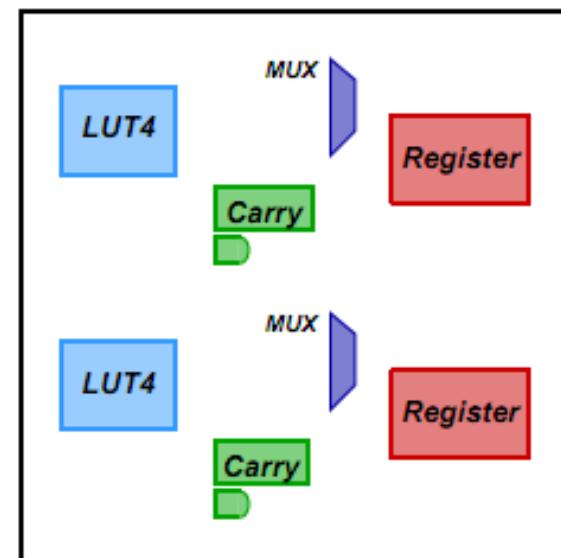
- Spartan™-3 kristale yra keturios loginės celės (slice)
- Celės grupuojamos poromis:
  - SLICEM (Memory)
    - ✧ LUT
    - Loginės funkcijos
    - Paskirstytoji atmintis
    - Postūmio registras SRL16
  - ✧ Multipleksoriai
  - ✧ Registrai (užlaikymo, flip-flop)
- SLICEL (Logic)
  - ✧ LUT
  - Loginės funkcijos
  - ✧ Multipleksoriai
  - ✧ Registrai (užlaikymo, flip-flop)



# SLICEM ir SLICEL sandara

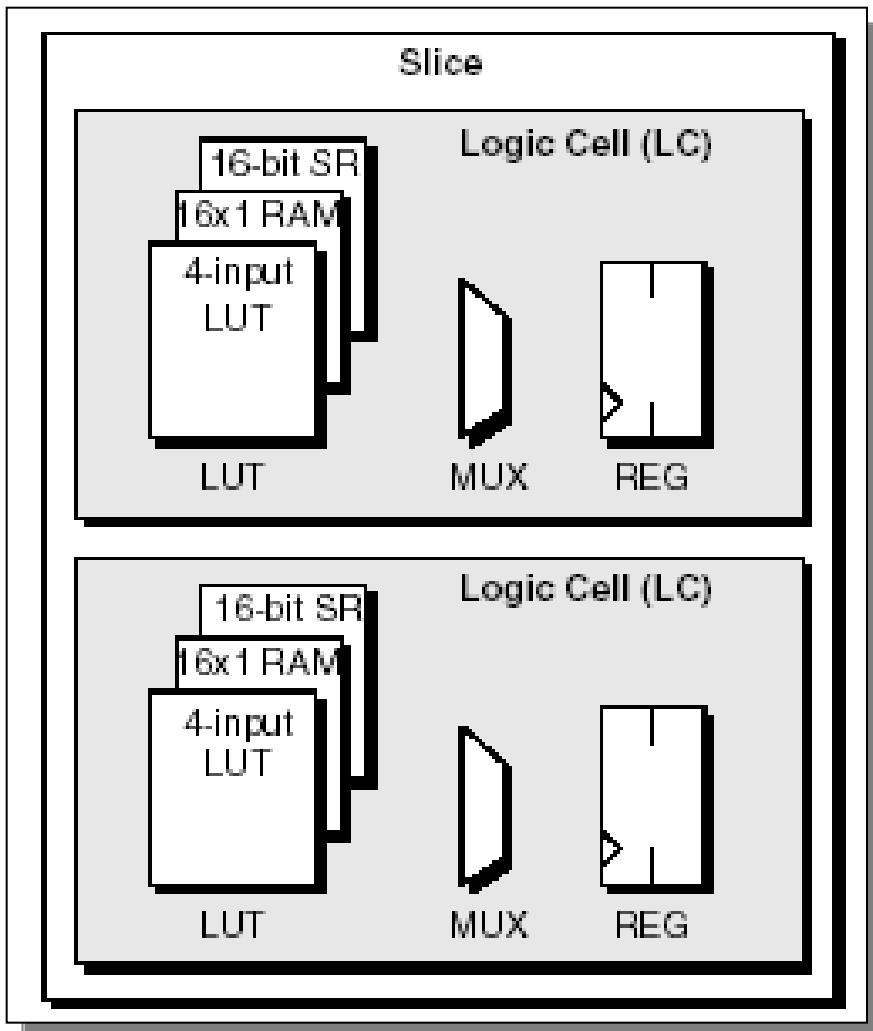


SLICEM



SLICEL / SLICEX (*SLICEX omits carry logic*)

# FPGA. Slice (Xilinx)



Abiem loginėms celėms turi bendrus:

- taktinį signalą (clock) ,
- Taktavimo leidimo signalą(clock enable),
- nustatymo/numetimo (set/reset) signalą.

Įėjimų ir išėjimų signalai yra atskirai kiekvienai LC. Signalai šiame paveikslėlyje neparodyti.

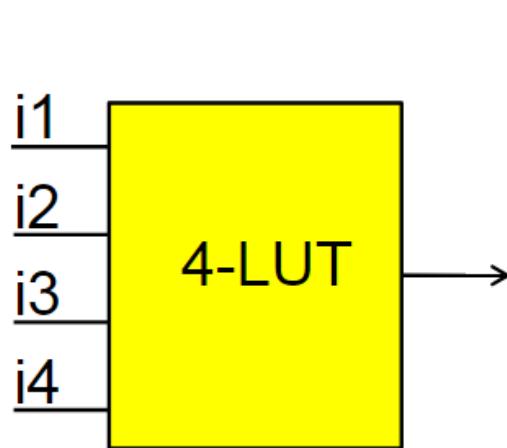
# FPGA. Loginiai blokai



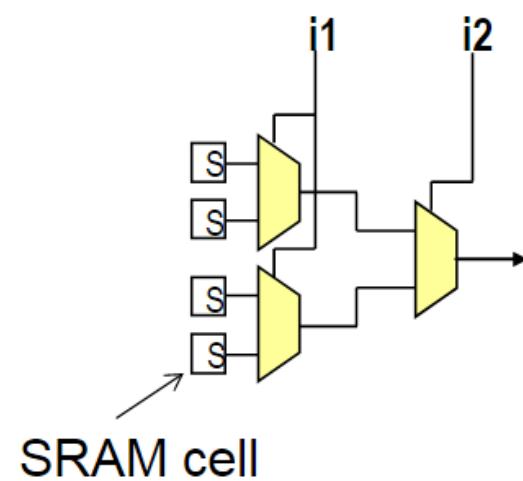
Loginės funkcijos realizuojamos loginiuose blokuose (Altera-LC, Xilinx – CLB). Dažniausiai naudijami 2, 3 ar 4 įjimų (bitų) loginiai blokai, kurie gali būti realizuoti teisingumo lentelių (LUT) arba multipleksorių (MUX) pagrindu.

Sudėtingos funkcijos realizuojamos skaidant jas į mažesnes ir apjungiant programuojamais sujungimais.

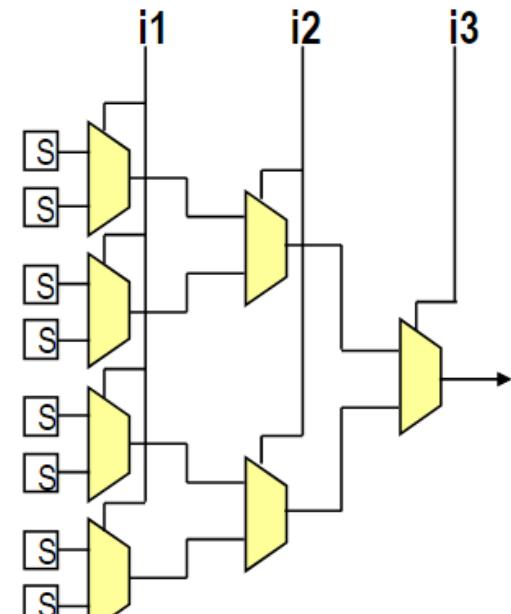
# FPGA. Loginiai blokai



Can realize *any*  
4-input function



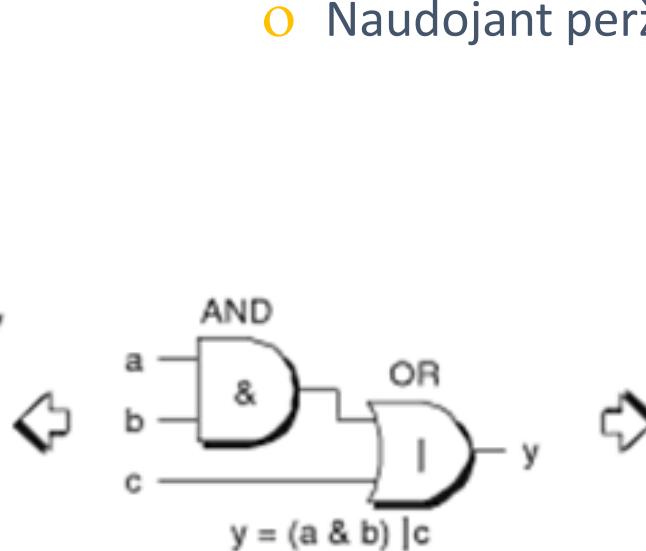
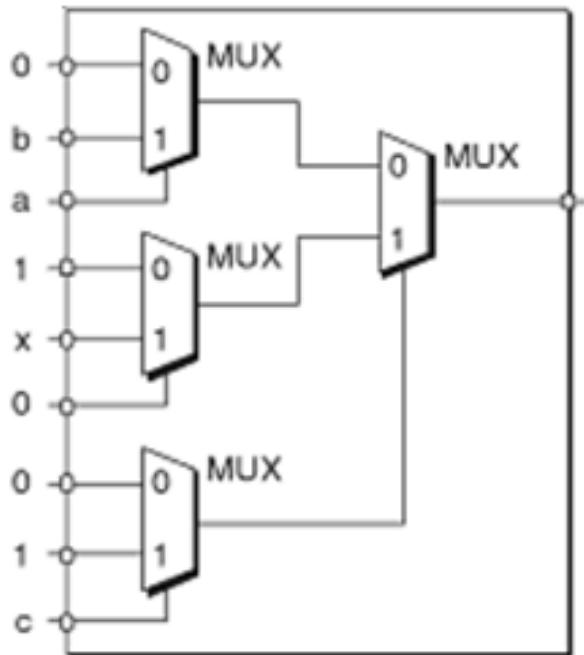
2-LUT



3-LUT

# FPGA. Loginių funkcijų realizavimas

- Funkcija  $y = (a \& b) | c$  gali būti realizuota dviem būdais:
  - Multipleksorių pagalba
  - Naudojant peržiūros lentelę (LUT)



Truth table			
a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

# FPGA. Loginių funkcijų realizavimas



- MUX – loginės funkcijos realizuojamos optimaliau, tačiau jų yra didelis vėlinimas, perduodant srautinius duomenis.
- LUT – RAM tipo atmintis, kurios vėlinimas neprikauuso nuo realizuojamos funkcijos. Greitesnė nei MUX.

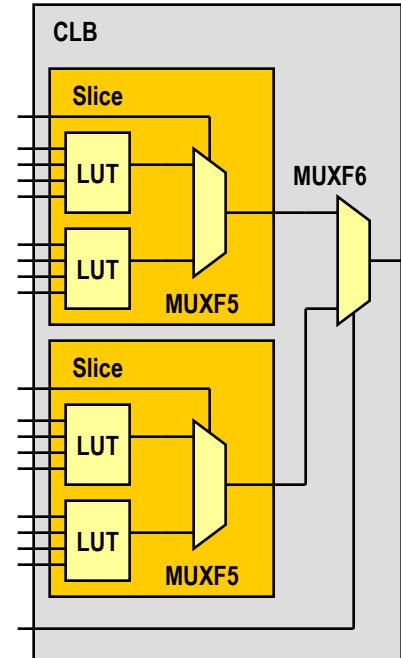
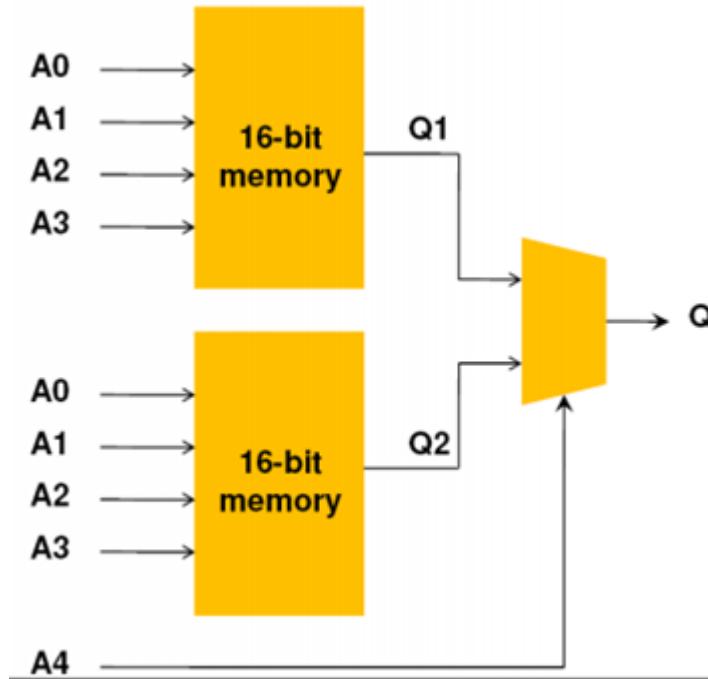
# 3-, 4-, 5- ar 6- įėjimų LUT lentelės



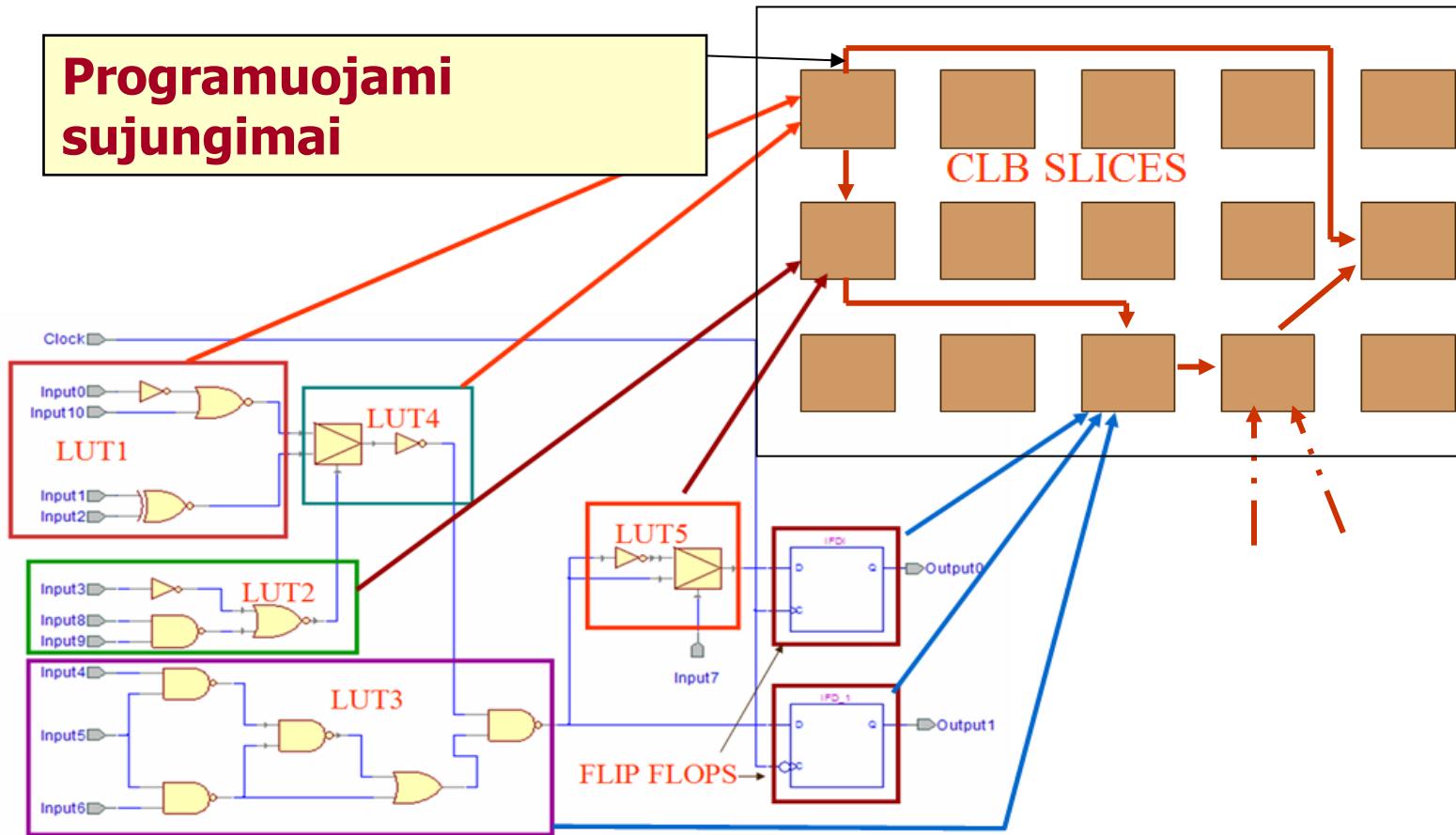
- Kuo didesnis įėjimų skaičius, tuo sudėtingesnę loginę funkciją galima realizuoti LUT lentele. Tačiau kiekvienas papildomas įėjimas dvigubina reikiamą SRAM celių skaičių.
- Pirmosios FPGA buvo su 3-jų iėjimų LUT lentelėmis. Vėliau buvo išdirbtos 4, 5, 6 įėjimų LUT. Konsensusas šiai dienai yra 4-ių įėjimų LUT.

# FPGA. Loginių funkcijų realizavimas

- Galimos mišrios LUT-MUX realizacijos funkcijų jėjimų skaičiaus padidinimui



# FPGA. Loginių funkcijų realizavimas



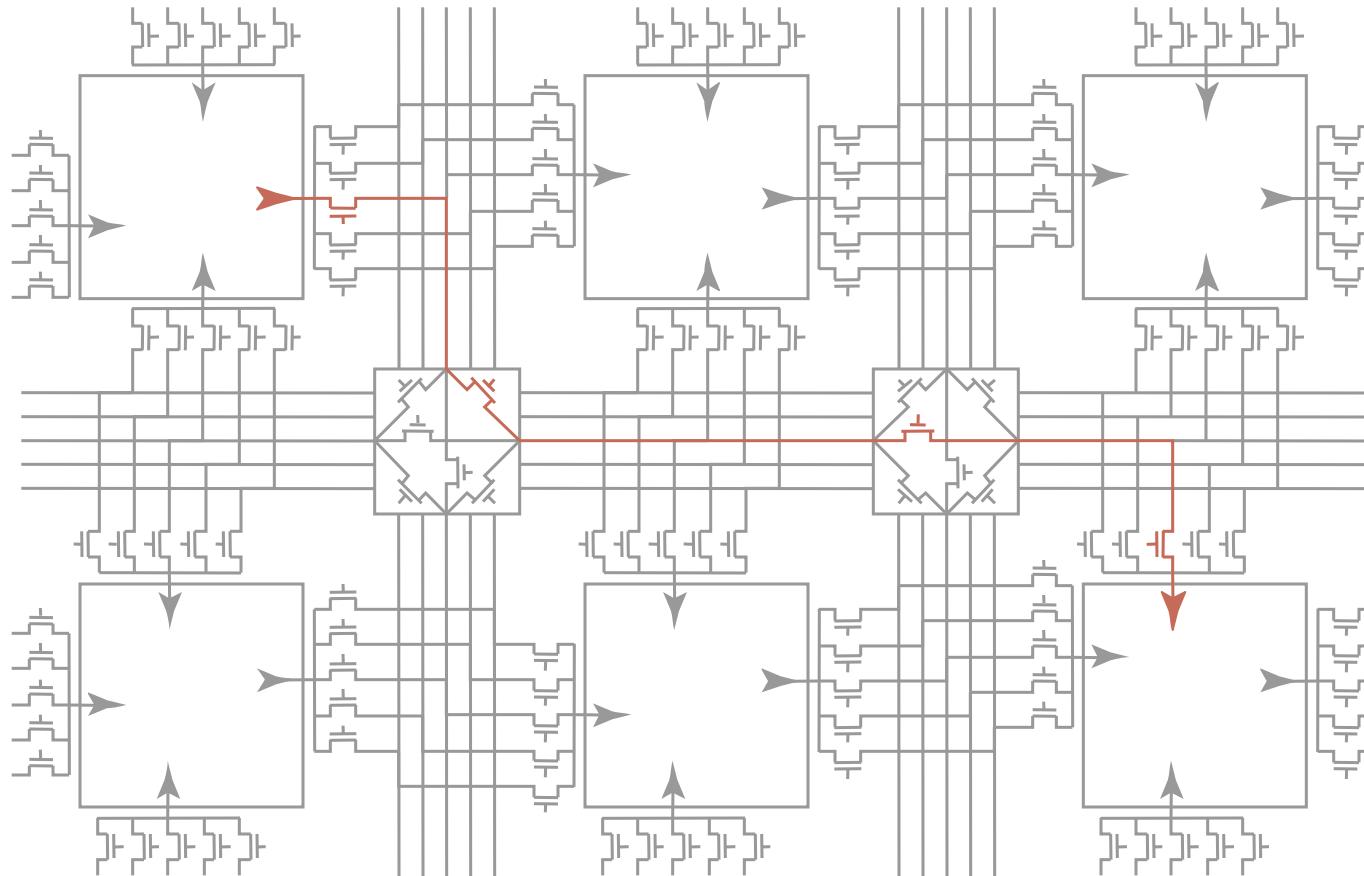
# Hierarchinė klasifikacija



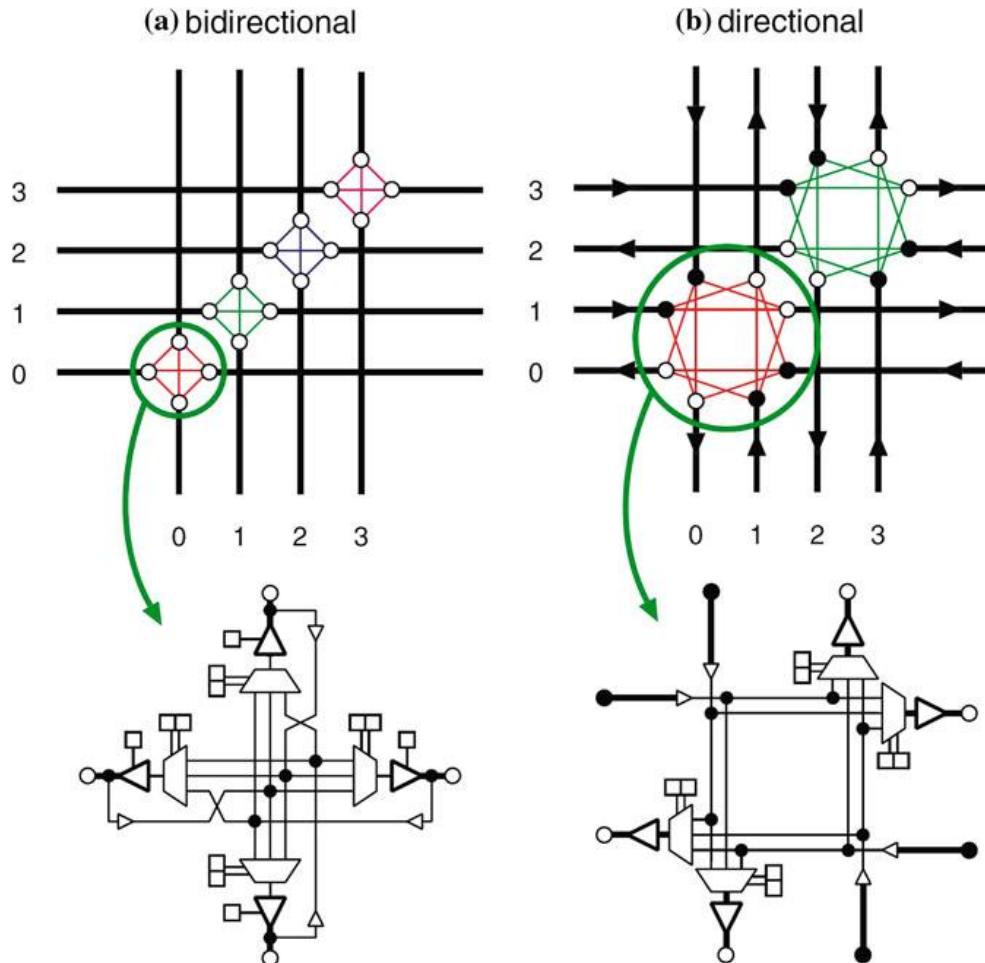
LC->Riekė(dvi LC)->CLB(keturios riekės) yra naudinga todėl, kad ji gerai dera su analogiška sujungimų hierarchija. Tokiu būdu yra:

- Greiti ryšio kanalai tarp LC;
- Truputi lėtesni tarp riekių viename CLB;
- Lėčiausi ryšio kanalai tarp atskirų CLB.

# FPGA. Programuojami sujungimai

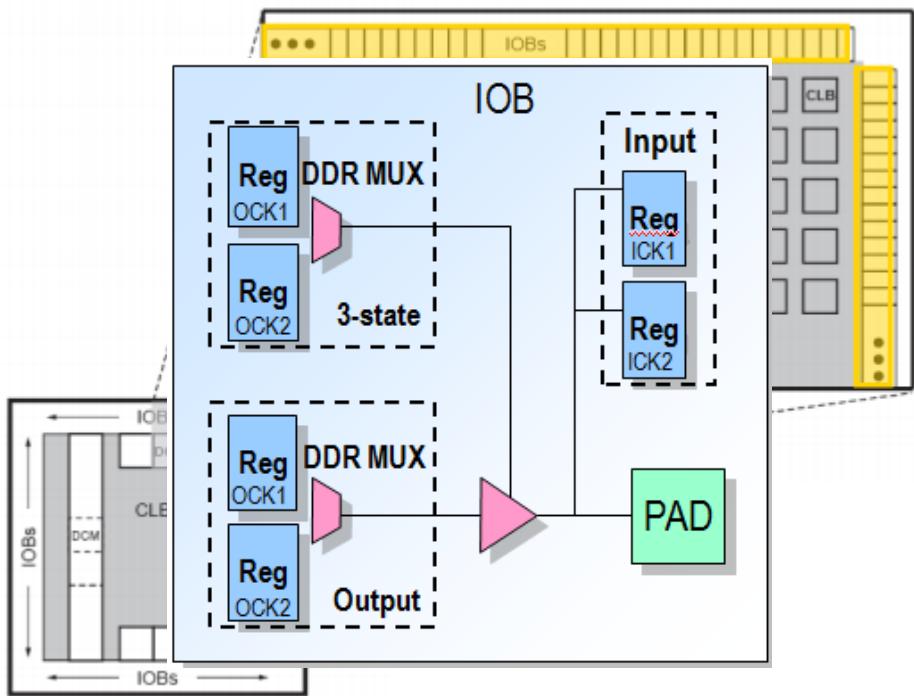


# FPGA. Programuojami sujungimai



## I/O blokai

- Jungia vidinius loginius grandynus su "išoriniu pasauly";
- Duomenys gali būti siunčiami viena ar/ir dvejomis kryptimis ;
- Gali būti įjungiamama aukšto impedanso būsena;
- Įėjimo signali gai būti vėlinami;
- Palaikomi įvairus duomenų perdavimo standartai.



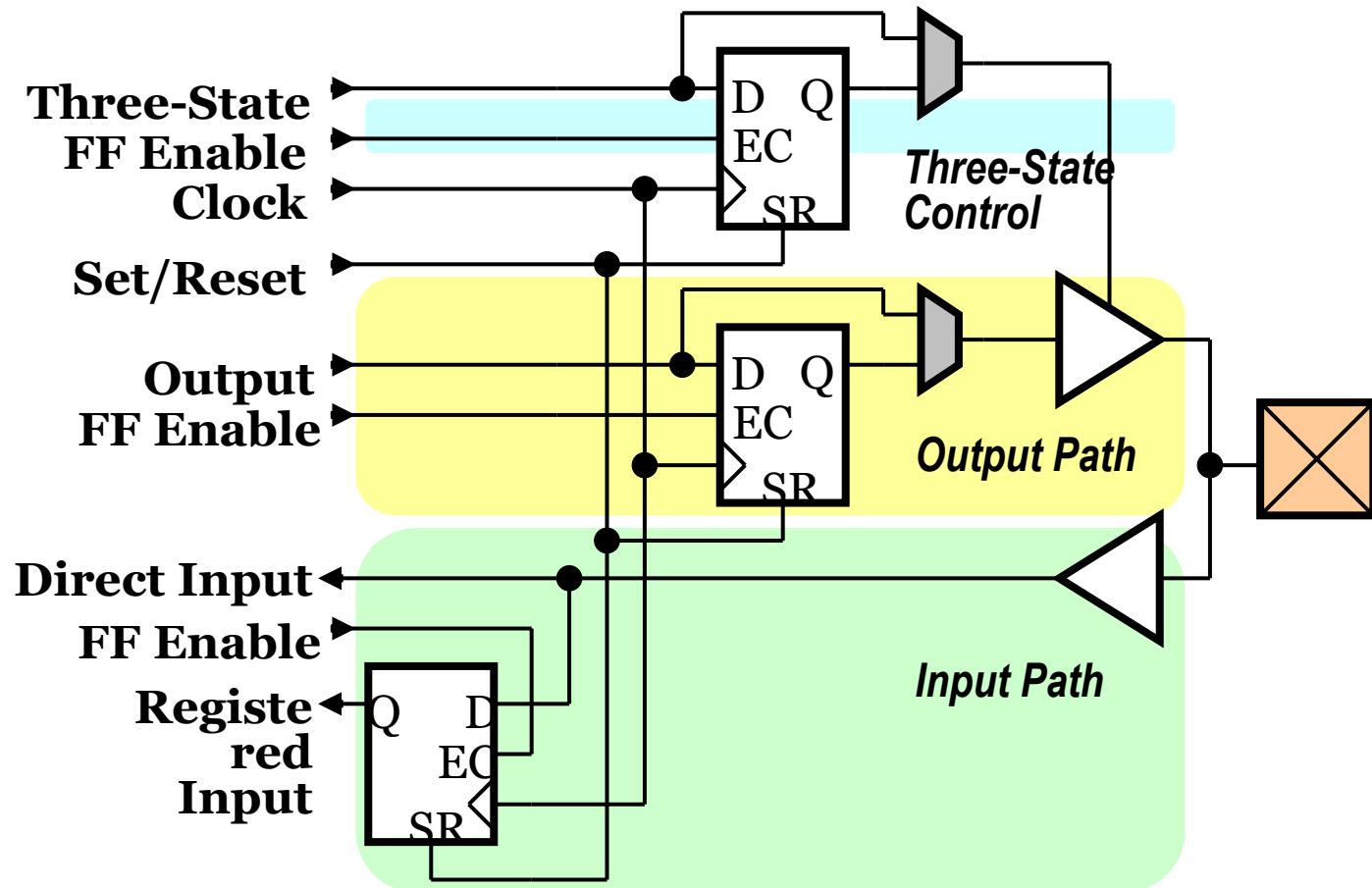
## I/O blokai

- Jungia vidinius loginius grandynus su “išoriniu pasauliu”;
- Duomenys gali būti siunčiami viena ar/ir dvejomis kryptimis ;
- Gali būti įjungiamama aukšto impedanso būsena;
- Įėjimo signali gai būti vėlinami;
- Palaikomi įvairus duomenų perdavimo standartai.

Single-Ended IOSTANDARD	V <sub>CCO</sub> Supply/Compatibility				
	1.2V	1.5V	1.8V	2.5V	3.3V
LVTTL	-	-	-	-	Input/ Output
LVCMOS33	-	-	-	-	Input/ Output
LVCMOS25	-	-	-	Input/ Output	Input
LVCMOS18	-	-	Input/ Output	Input	Input
LVCMOS15	-	Input/ Output	Input	Input	Input
LVCMOS12	Input/ Output	Input	Input	Input	Input
PCI33_3	-	-	-	-	Input/ Output
PCI66_3	-	-	-	-	Input/ Output

# FPGA I/O Blocks

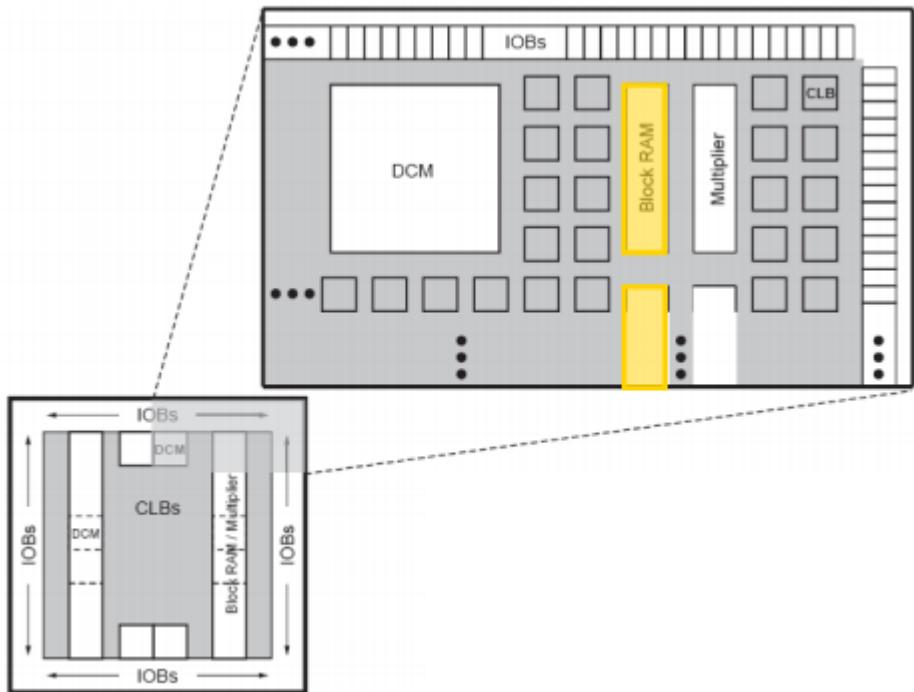
## I/O blokai



## RAM blokai

### BRAM (Block RAM)

- Matricioje gali būti iki kelių dešimčių atminties blokų (Spartan 3E yra 20 RAM blokų);
- Kiekvienas atminties bloko talpa gali saugoti iki 18 kBitų duomenų (Spartan 3E);



## RAM blokai

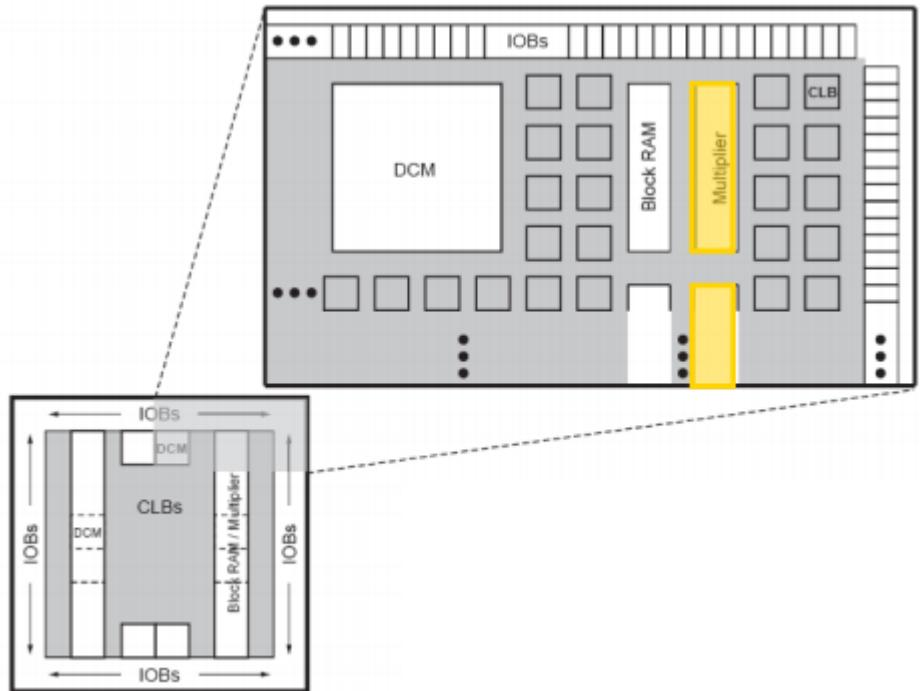
BRAM (Block RAM)

- Atminties naudojimo sritys:
  - Duomenų paketų (pvz., komunikacijų buferiai,
  - Skaitmeninio signalų apdorojimo uždaviniai (pvz., FIR filtro koeficientai ir duomenys)
  - Buferiai kompresijai ir kodavimui (pvz., interleaving)
  - FIFO naudojimas skirtingu dažnio domenų (angl. domain) sujungimui
  - Video ir vaizdų apdorojimo buferiai
  - Procesoriaus kodo saugojimas
  - Bendros paskirties atmintis

## Daugintuvų blokai

### Multiplier

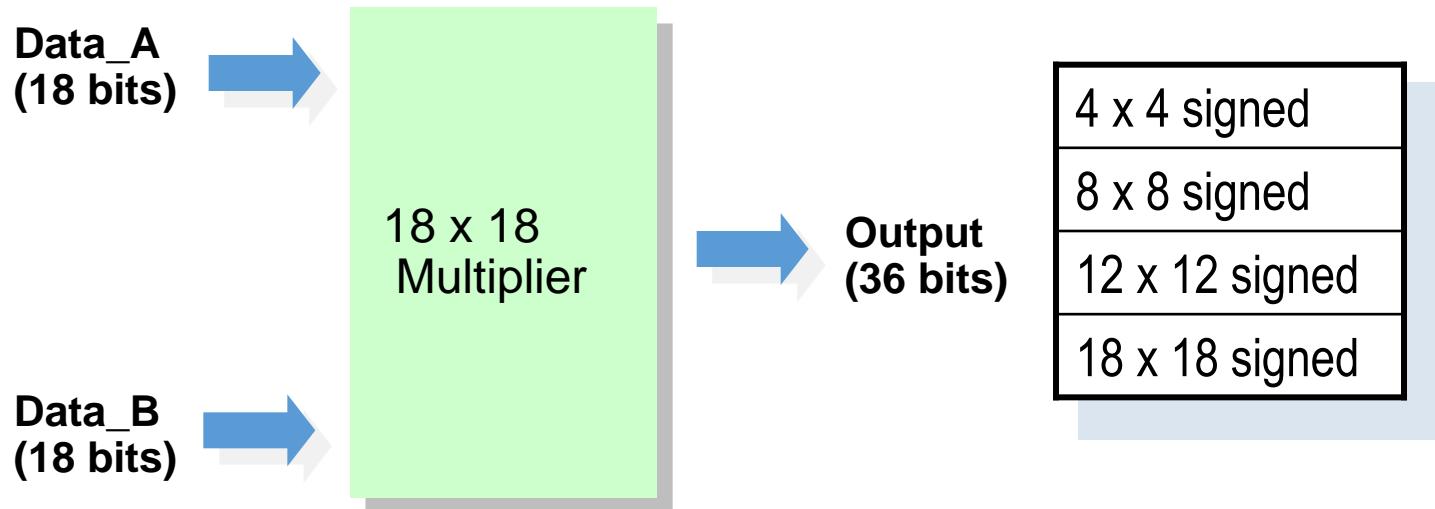
- Išdėstyti šalia RAM blokų;
- Gali sudauginti du 18 bitų skaičius;



## Daugintuvų blokai

### Multiplier

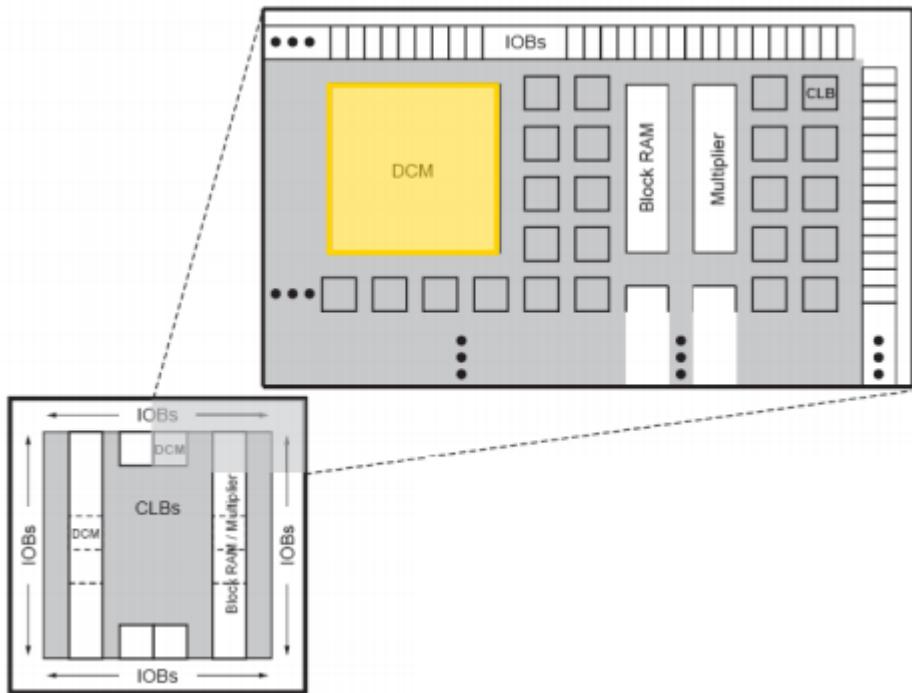
- Išdėstyti šalia RAM blokų;
- Gali sudauginti du 18 bitų skaičius;



## Taktinių impulsų valdiklis DCM (Digital Clock Manager)

TIV sudedmosios dalys:

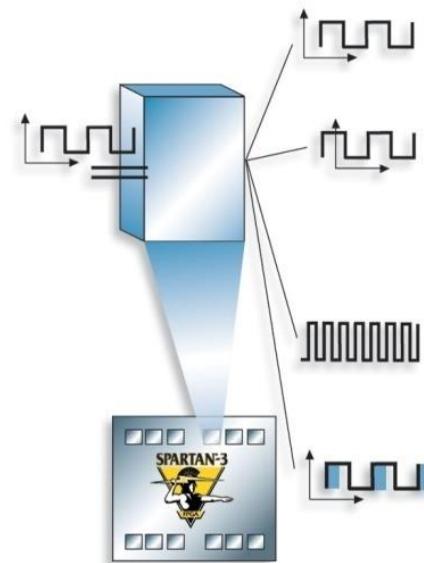
- Fiksuoto vėlinimo grandynai (DLL - Delay-Locked Loop)
- Skaitmeniniai dažnio sintezatoriai (DFS - Digital Frequency Synthesizer)
- Skaitmeninai fazės sukiliai (DPS - Digital Phase Shifter)
- Fiksujotos fazės grandynai (PLL - Phase-Locked Loop)



## Taktinių impulsų valdiklis DCM (Digital Clock Manager)

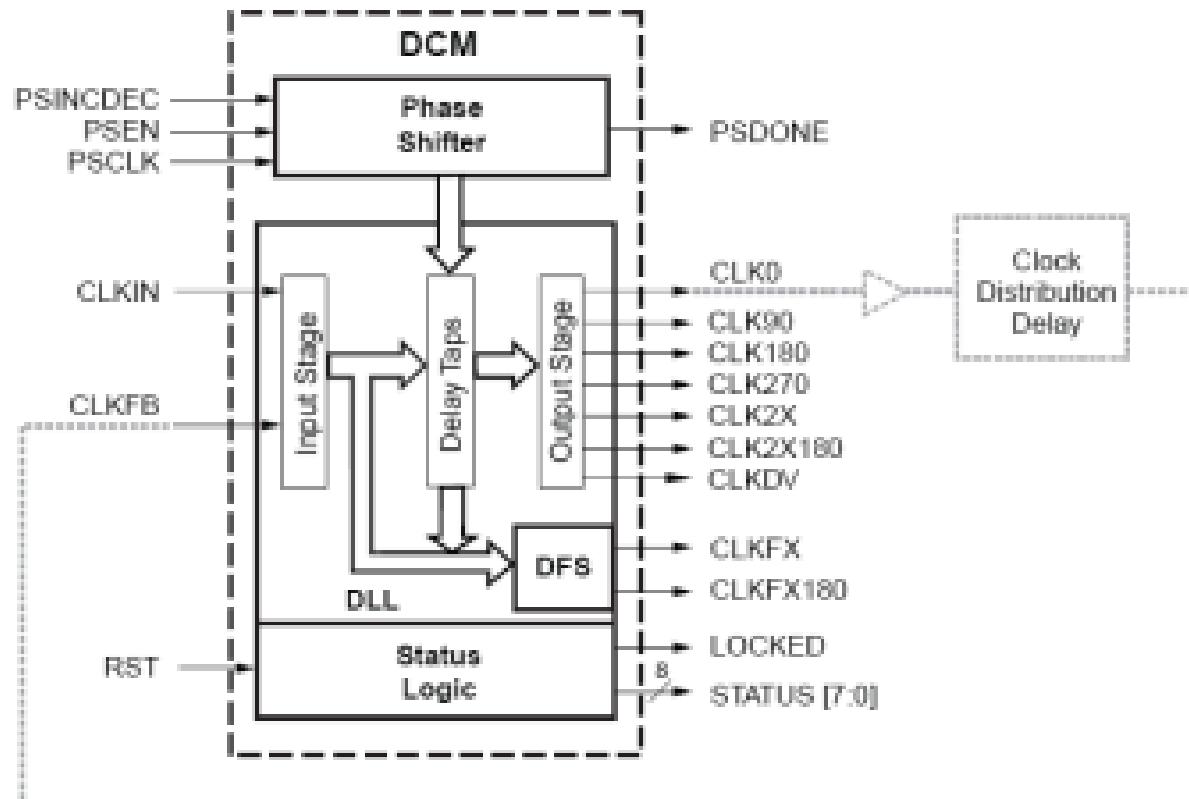
Atliekamos funkcijos:

- generuoja reikiama dažnio ir fazės taktinius impulsus. Atraminiu gali būti naudojamas išorinių generatorių arba kitų DCM signalus;
- naudojami sveikų skaičių daugikliai/ dalikliai ( Pvz.,  $1.8x \rightarrow 9/5$ );
- fazės sukimasis;
- frontų virpėjimų mažinimas.



## Taktinių impulsų valdiklis DCM (Digital Clock Manager)

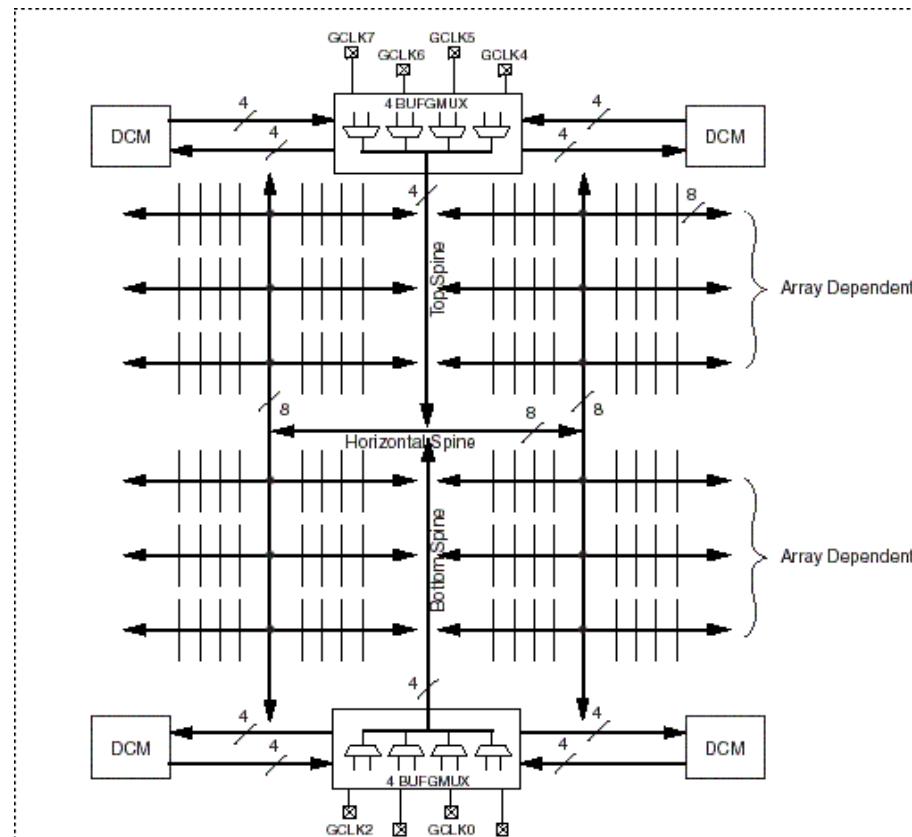
Xilinx Spartan 3



## Taktinių impulsų valdiklis

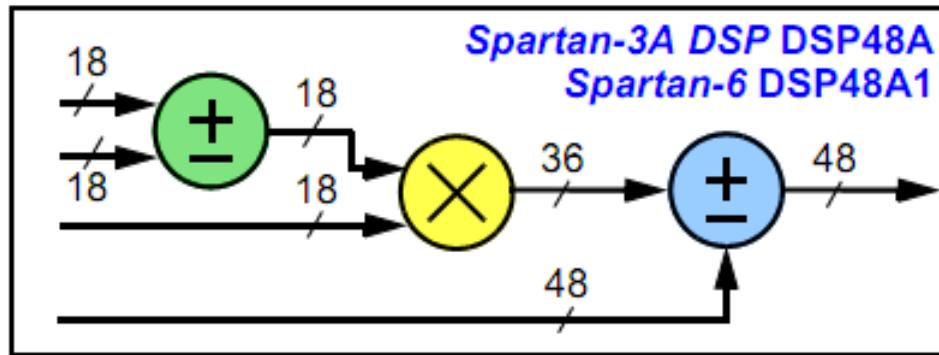
DCM (Digital Clock Manager)

Xilinx Spartan 3



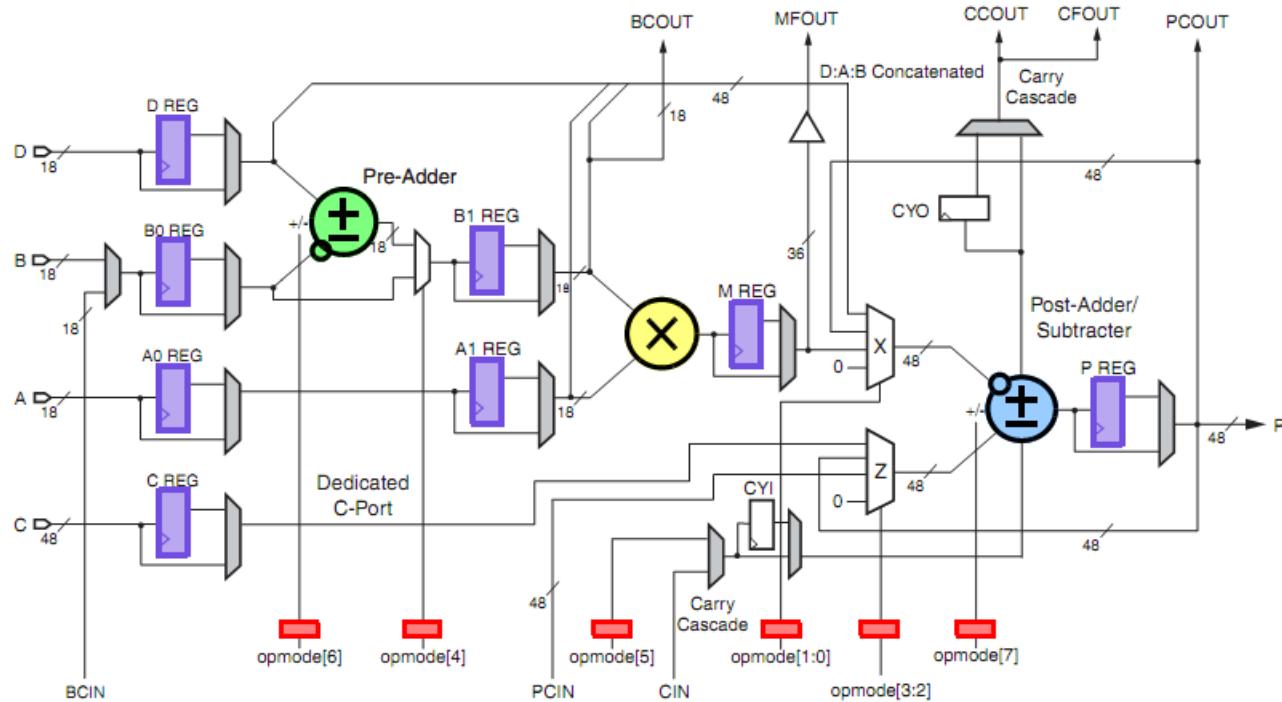
## DSP blokai

Greiti aritmetiniai blokai (daugintuvai ir sumatoriai, registrai , multipleksoriai)



## DSP blokai

Greiti aritmetiniai blokai (daugintuvai ir sumatoriai, registrai , multipleksoriai)



# Spartan 3A ir Spartan 3A-DSP palyginimas



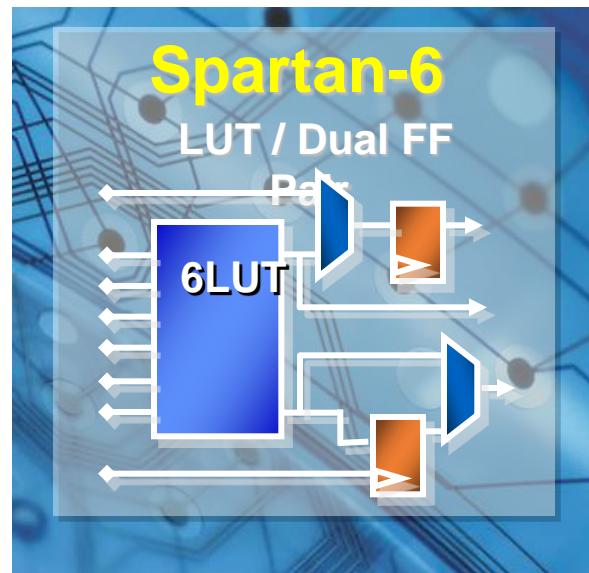
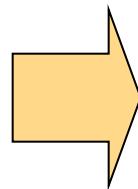
	Spartan-3	Spartan-DSP	
	Spartan-3A	Spartan-3A DSP	
	XC3S1400A	XC3SD1800A	XC3SD3400A
XtremeDSP DSP48A Slices	-	84	126
Dedicated Multipliers	32	DSP48As	DSP48As
Block Ram Blocks	32	84	126
Block RAM (Kb)	576	1,512	2,268
Distributed RAM (Kb)	176	260	373
FFs/LUTs	22,528	33,280	47,744
Logic Cells	25,344	37,440	53,712
DCMs	8	8	8
Max Diff I/O Pairs	227	227	213
CS484 19x19mm (0.8mm pitch)	-	309	309
*FG676 27x27mm (1.0mm pitch)	502	519	469

# Spartan serijos raida

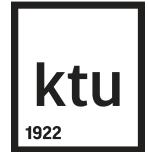


Feature	Extended Spartan-3A (90nm)	Spartan-6 (45nm)
Logic Cells (Kbit)	Up to 55K	Up to 150K
LUT Design	4-input LUT + FF	6-input LUT + 2FF
Block RAM (Mbit)	Up to 2 Mbit	Up to 5 Mbit
Transceiver Count / Speed	no	Up to 8 / Up to 3.125 Gbps
Voltage Scaling	No (1.2V only)	Yes (1.2V, 1.0V)
Static Power (typ mW)	11 mW (smallest density)	Up to 60% less!
Memory Interface	400 Mbps	DDR3 800 Mbps
Max Differential IO	640 Mbps	1050 Mbps
Multipliers/DSP	Up to 126 Multipliers / DSP	Up to 184 DSP48 Blocks
Memory Controllers	no	Up to 4 Hard Blocks
Clock Management	DCM Only	DCM & PLL
PCI Express Endpoint	no	Yes, Gen 1
Security	Device DNA Only	Device DNA & AES

# Spartan serijos raida



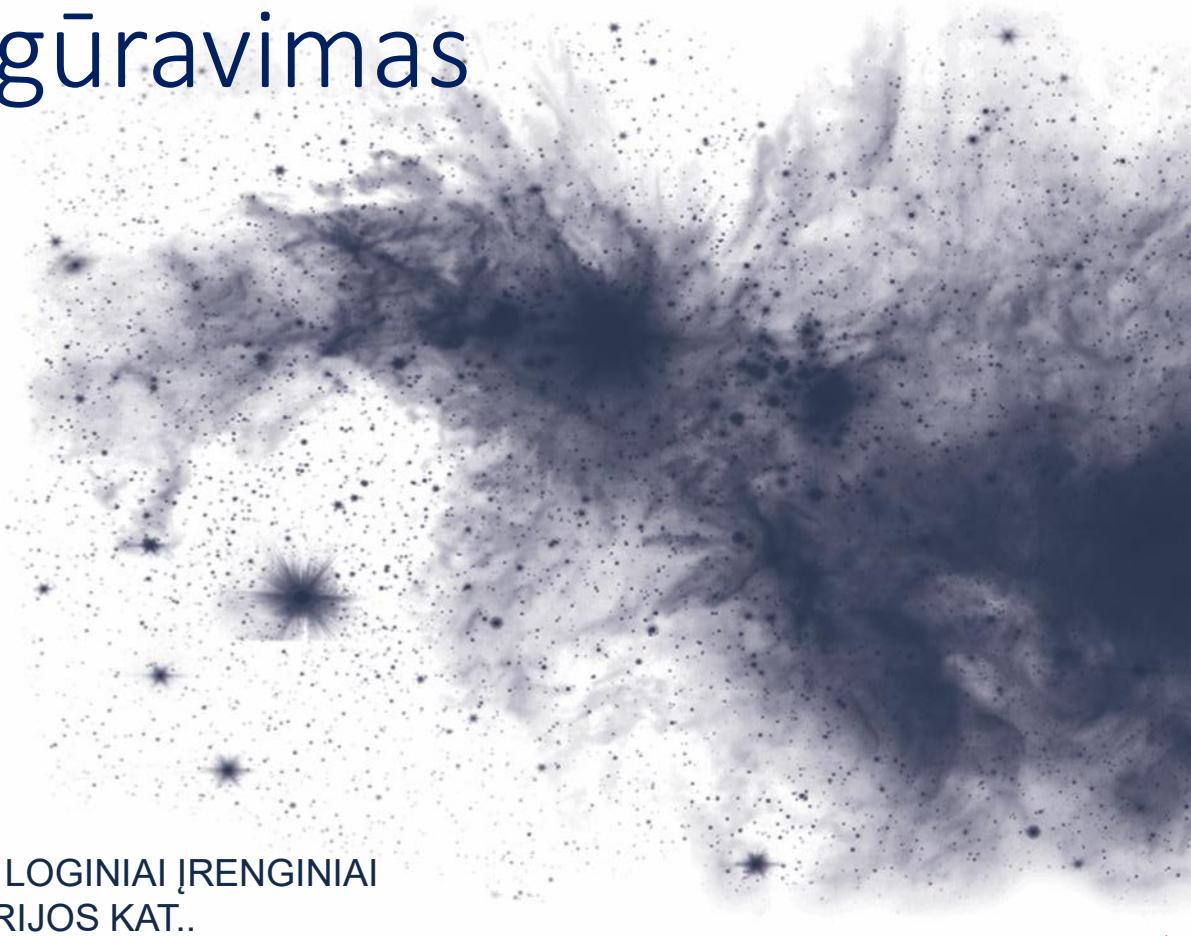
# FPGA senėjimas



Laikoma, kad 1-eri FPGA metai atitinka 15 žmogaus metų.

Kai konkreti FPGA rinkoje egzistuoja jau 2 metus (ekvivalentiška 30 žmogaus metų), galima laikyti, kad tai jau gerai išdirbtas produktas, pasiekęs savo galimybių piką. 4-erių metų FPGA jau galima laikyti moraliai senstelėjusia.

# Programuojamos mikrosistemos FPGA konfigūravimas

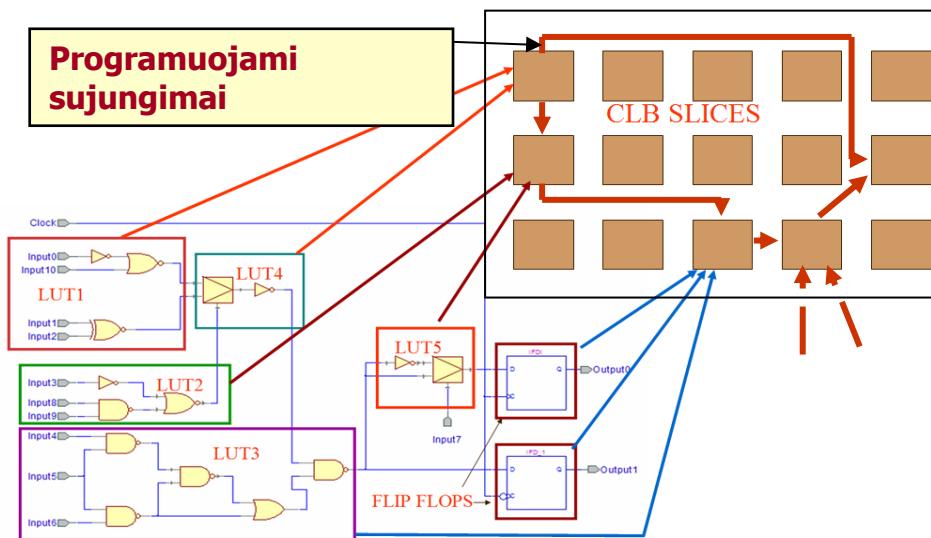
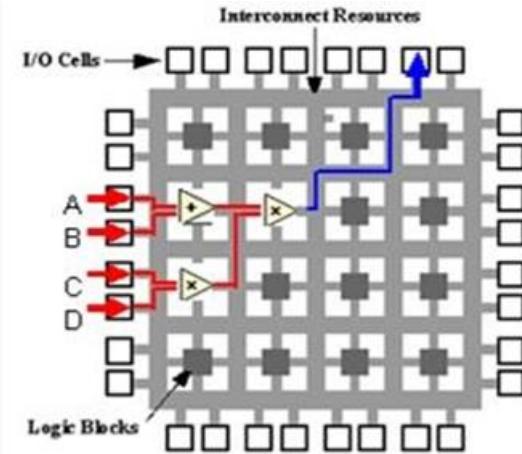


**R. Ramanauskas**

T170B114 PROGRAMUOJAMI LOGINIAI ĮRENGINIAI  
KTU ELEKTRONIKOS INŽINERIJOS KAT..  
2019

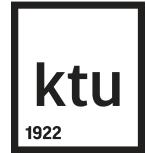
# FPGA. Konfigūravimas

- Tam, kad programuojamoji struktūra atliktų reikiama funkcią, sudaroma loginė schema, kuri išpildoma naudojant atitinkamus konkrečios FPGA (Spartan-3E) vidinius blokus (LUT, MUX, registrus, daugintuvus ir t.t.) ir juos tinkamai sujungus programuojamais sujungimais.



- FPGA - programuojami ventilių masyvai. Programavimas – ventilių masyvo elementų būsenų nustatymas. Kiekvieno elemento būsenai nusakyti reikalingas 1 bitas atminties.
- Konfigūruojami elementai:
  - Konfigūruojamo loginio bloko funkcija (LUT lentelės);
  - Komutacines matricos sujungimai
- Naudojant gamintojo programinę įrangą, schema (funkcija) transformuojama į binarinį failą (su plėtiniu .bit), kurio pagalba konfigūruojami matricos elementai.
- BIT failas gali būti ‘užkrautas’ tiesiogiai į FPGA (pvz., per JTAG) arba gali būti išrašytas į papildomą atmintinę (PROM ar atminties kortelę).

# FPGA. Konfigūravimas



## VHDL kodas

```
architecture MLU_DATAFLOW of MLU is
signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

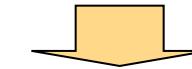
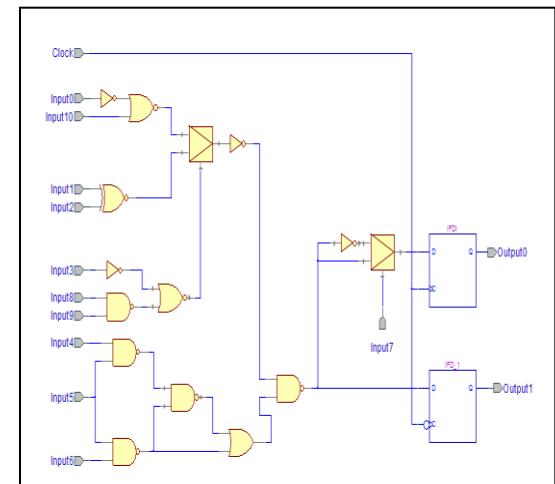
begin
    A1<=A when (NEG_A='0') else
        not A;
    B1<=B when (NEG_B='0') else
        not B;
    Y<=Y1 when (NEG_Y='0') else
        not Y1;

    MUX_0<=A1 and B1;
    MUX_1<=A1 or B1;
    MUX_2<=A1 xor B1;
    MUX_3<=A1 xnor B1;

    with (L1 & L0) select
        Y1<=MUX_0 when "00",
                    MUX_1 when "01",
                    MUX_2 when "10",
                    MUX_3 when others;

end MLU_DATAFLOW;
```

## Struktūrinė schema

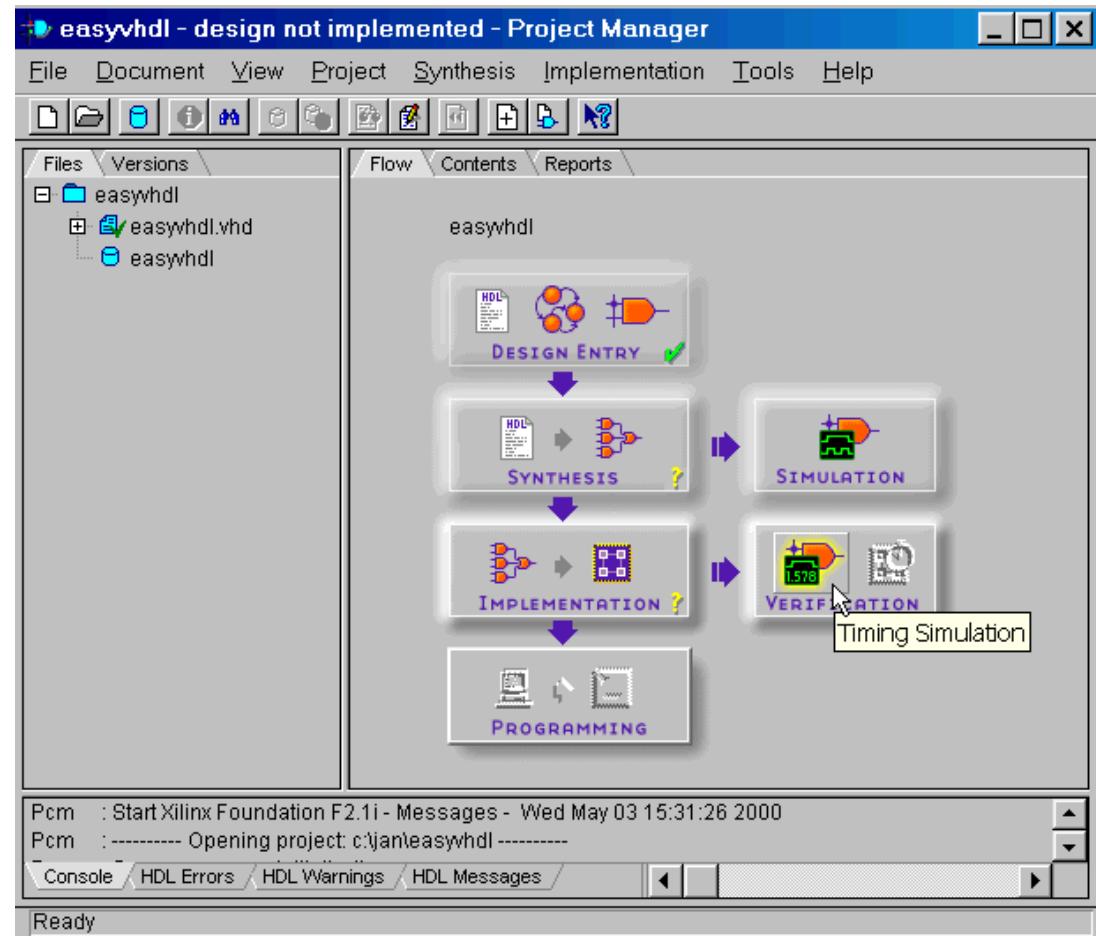


JEDEC file  
01001101010  
11010110101  
01101111001  
10110001011  
00001100010  
10111001010  
10011101100

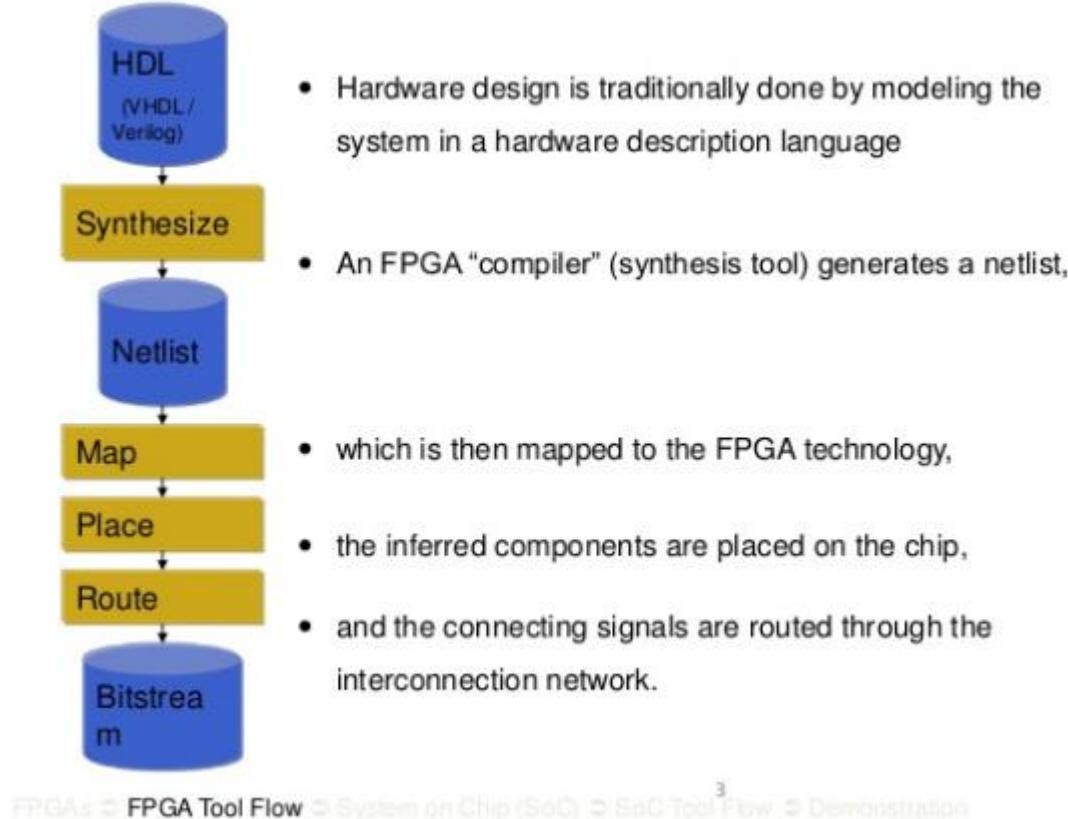
# FPGA. Konfigūravimas



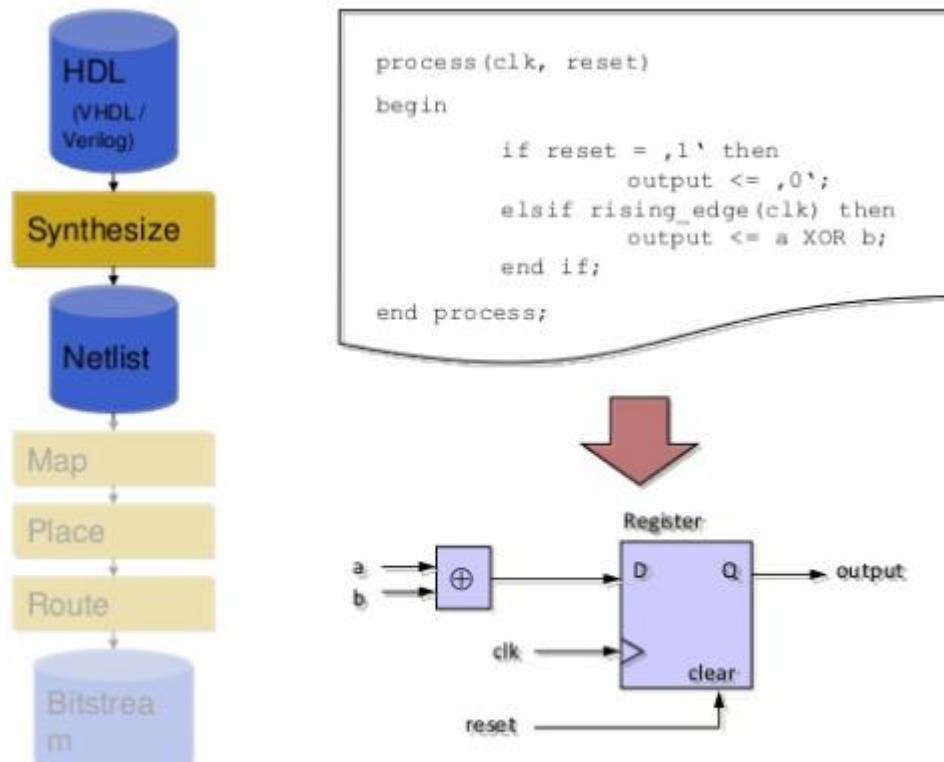
- Synthesize - XST
- Implement Design
- Translate
- Map
- Place & Route
- Generate Programming File



# FPGA. Konfigūravimas

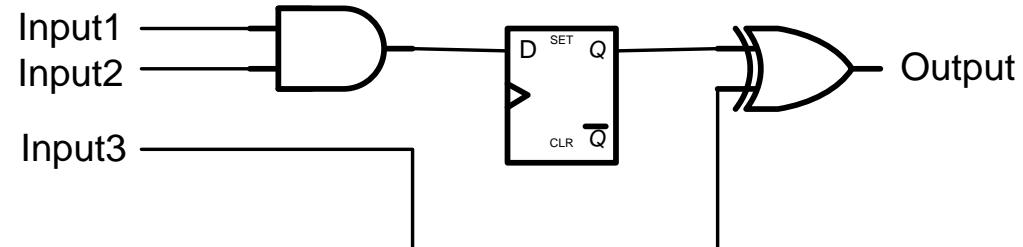
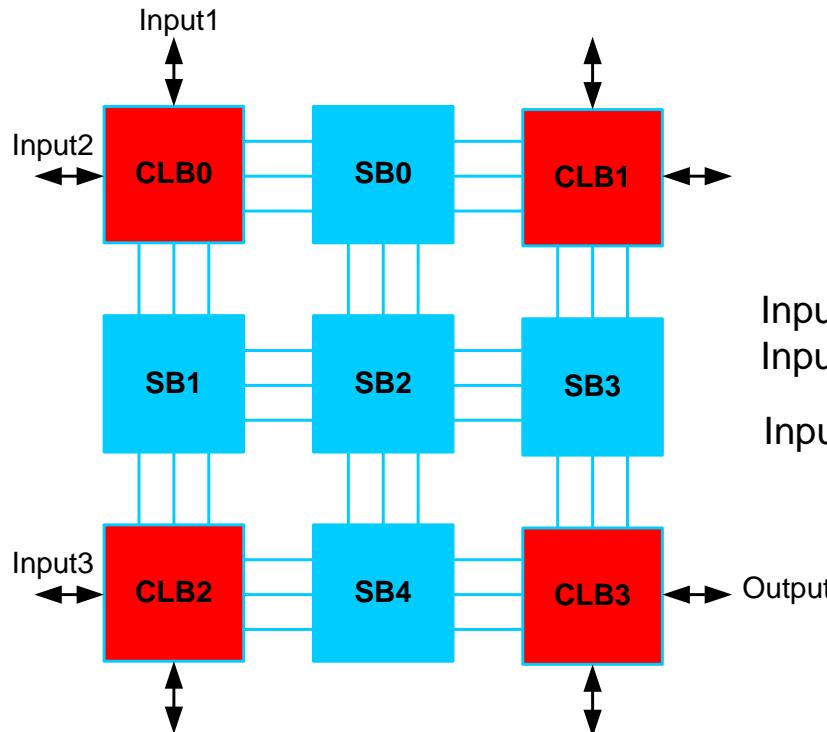


# FPGA. Konfigūravimas

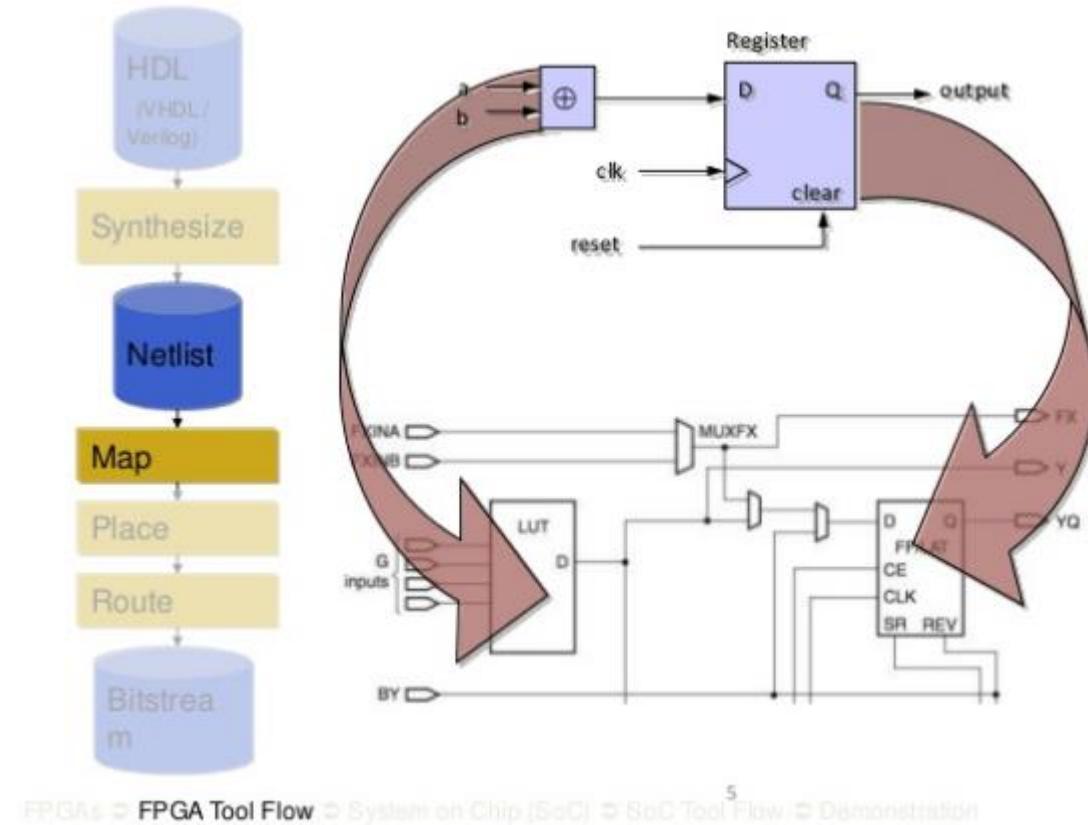


# FPGA. Konfigūravimas

- Tarkime reikia įgyvendinti žemiau pateiktą schemą naudojant 2x2 FPGA matricą turinčią 2 įėjimų LUT kiekviename konfigūruojamam loginiam bloke (CLB).

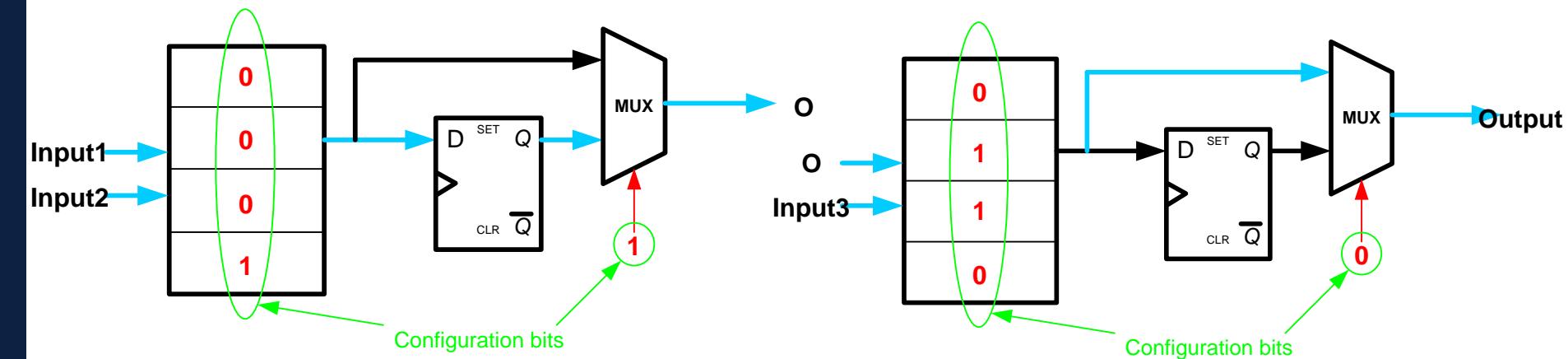
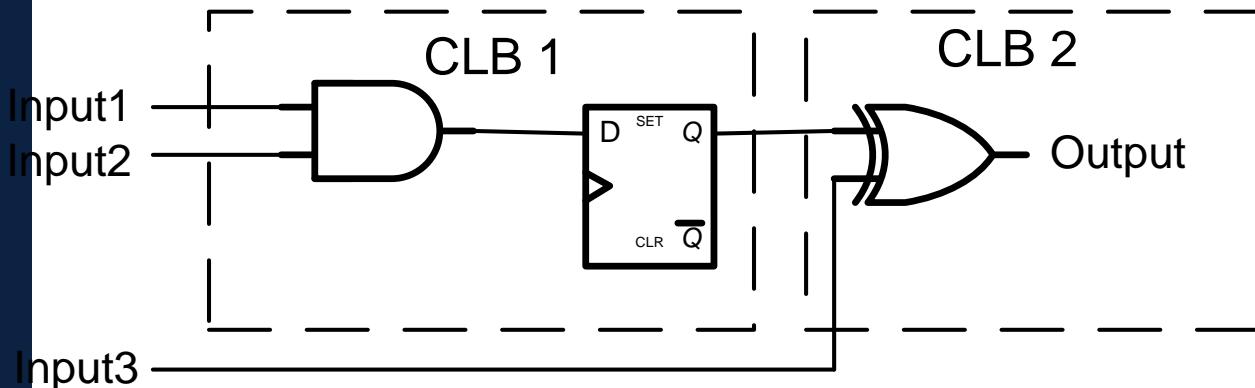


# FPGA. Konfigūravimas



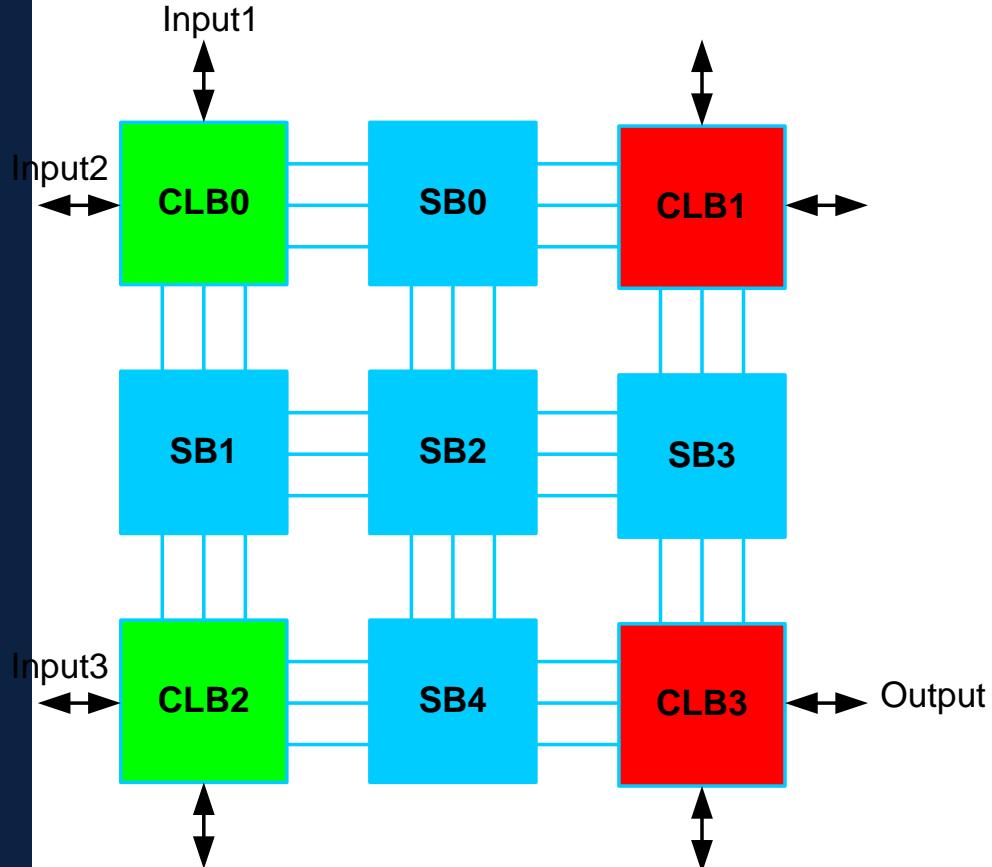
5

# FPGA. Konfigūravimas. Paskirstymas



Paskirstymas yra procesas, kurio metu ventilių lygio aprašas transformuojamas į LUT (angl. *Look Up Table*) aprašą.

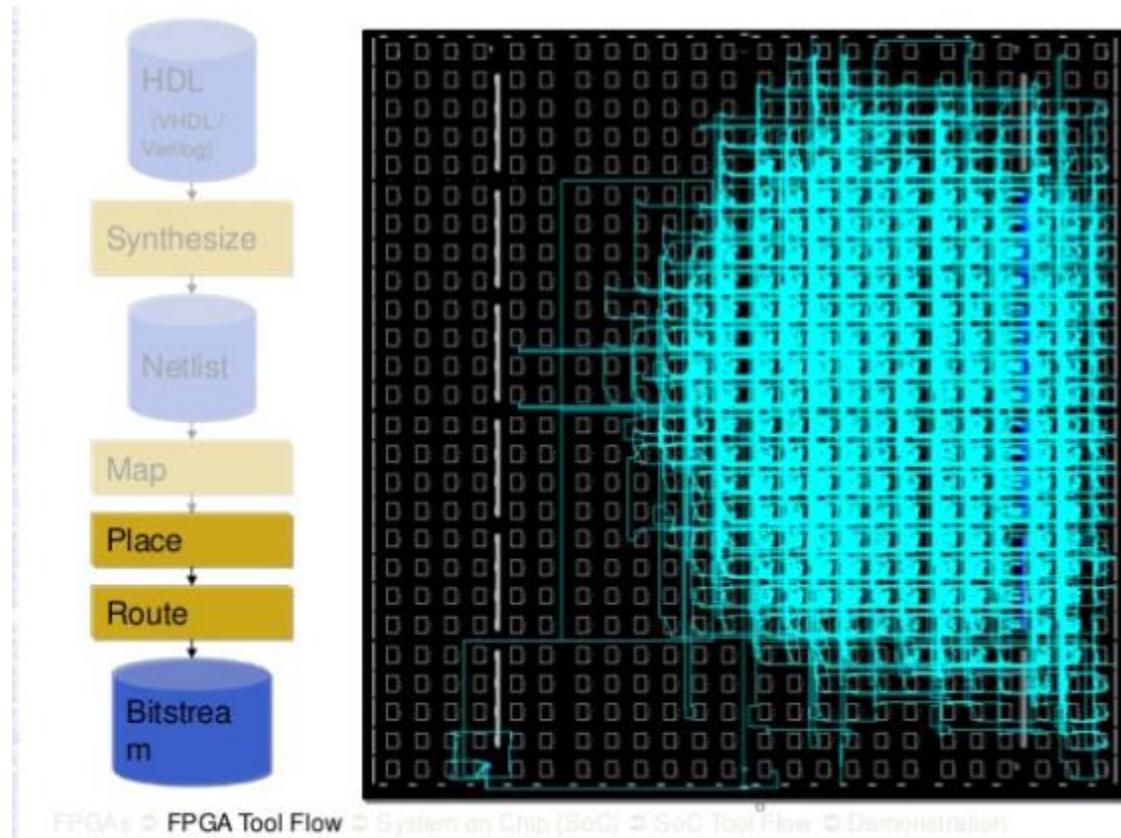
# FPGA. Konfigūravimas. Pakavimas



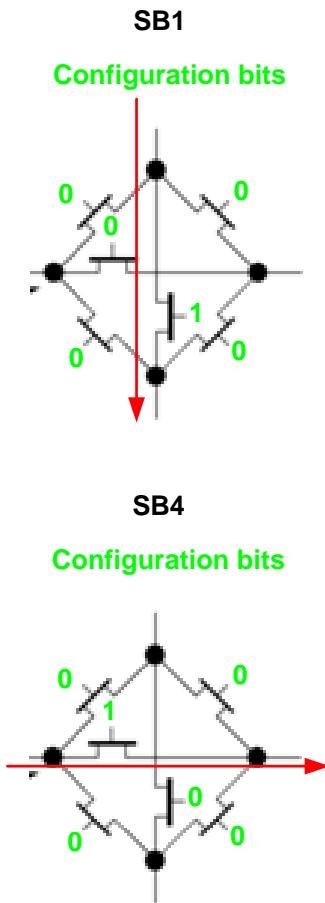
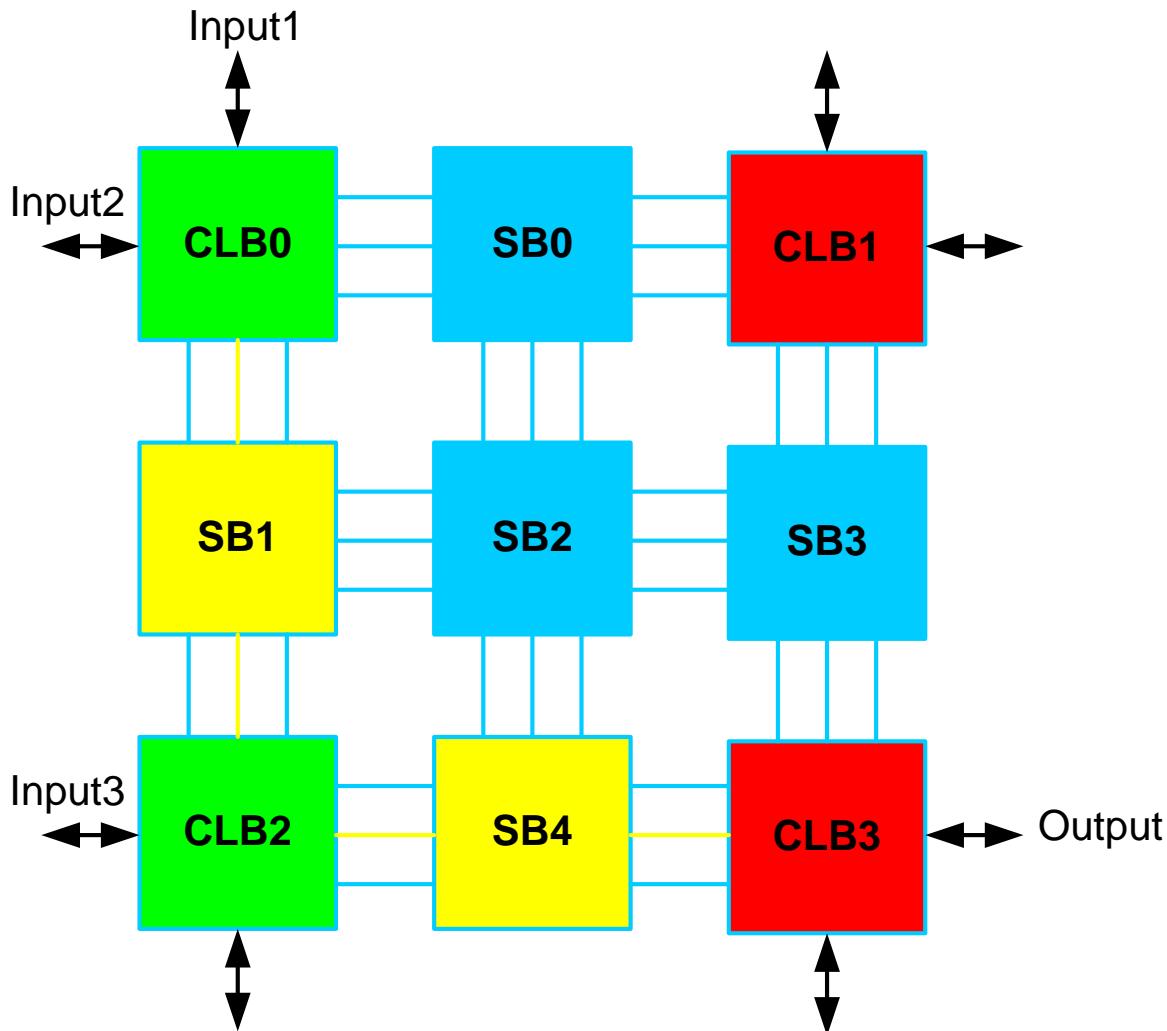
Pakavimo fazėje LUT lentelės ir registrai pakuojami (apjungiami) į konfigūruojamus loginius blokus (CLB). Pakavimo uždavinys yra nuspresti, kurias LUT lenteles iš kurio CLB imti.

# FPGA. Konfigūravimas

## Paskirstymas. Maršruto parinkimas



# FPGA. Sujungimų konfigūravimas



- Paskirstymas į sudėtingus programuojamus loginius blokus
- Paskirstymas į išterptinės atminties (RAM) blokus
- Paskirstymas į CPLD makroceles.

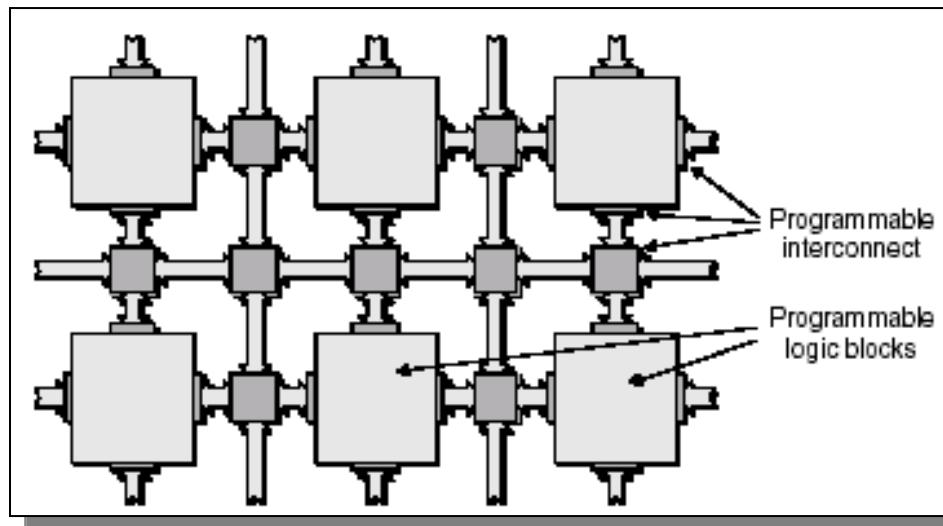
# FPGA. Konfigūracinio failo sudarymas



- Konfigūraciniame faile nurodomos visų (net ir nenaudojamų) elementų būsenos
  - CLB0: 00011
  - CLB1: 01100
  - CLB2: XXXXX
  - CLB3: ?????
  - SBo: 000000
  - SB1: 000010
  - SB2: 000000
  - SB3: 000000
  - SB4: 000001

# FPGA architektūros grūdėtumas

Grūdėtumas – tai vieno programuojamo loginio bloko (LB) vidinės sandaros (loginių elementų skaičiaus, įvairovės ir t.t.) savybė. Čia reikia turėti omenyje, kad FPGA yra sudaryta iš daugelio LB, tarpusavyje sujungtų konfigūruojamais sujungimais.



- Smulkus grūdėtumas (angl. fine-grained)
- Grubus grūdėtumas (angl. coarse-grained)

# Smulkus grūdėtumas



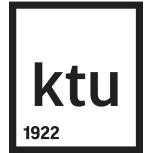
- Tokia architektūra, kai kiekvienas loginis blokas gali realizuoti labai paprastą loginę funkciją (3-jų įejimų loginis ventilis IR, ARBA arba atminties elementas, D-trigeris, D-fiksatorius)
- Tokie įrenginiai efektyvūs realizuojant sujungimų logiką (angl. glue logic), vykdant sistolinius algoritmus (kai galima panaudoti daug lygiagrečių skaičiavimų). Gerai dera su tradicine loginės sintezės teorija, t.y. jos pagalba sintezuojamos efektyvios struktūros.

# Grubus grūdėtumas



- Loginis blokas yra sudarytas iš salyginai didelio skaičiaus elementų (4-ių įėjimų LUT lentelės, 4 multipleksoriai, 4 D trigeriai ir pernešimų logika)
- Pradžioje vyravo smulkios architektūros įrenginiai, tačiau dabar jau didžioji dauguma FPGA komponentų realizuojami tik grubios architektūros pagrindu.

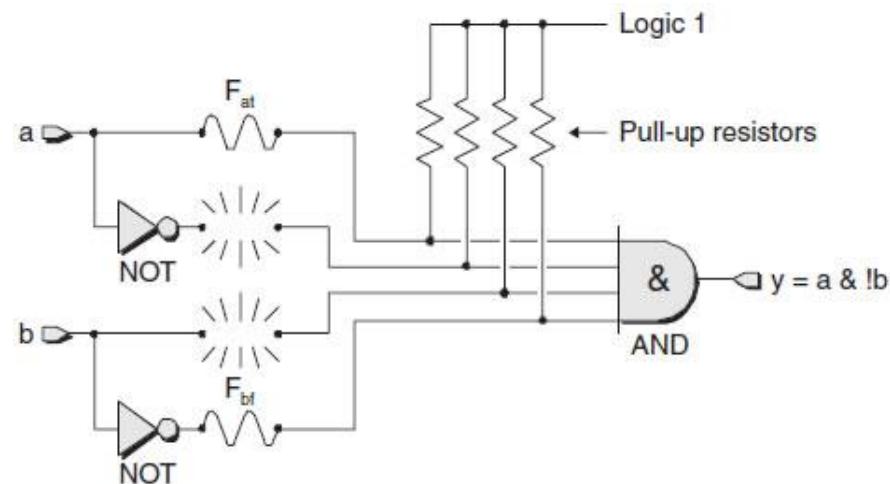
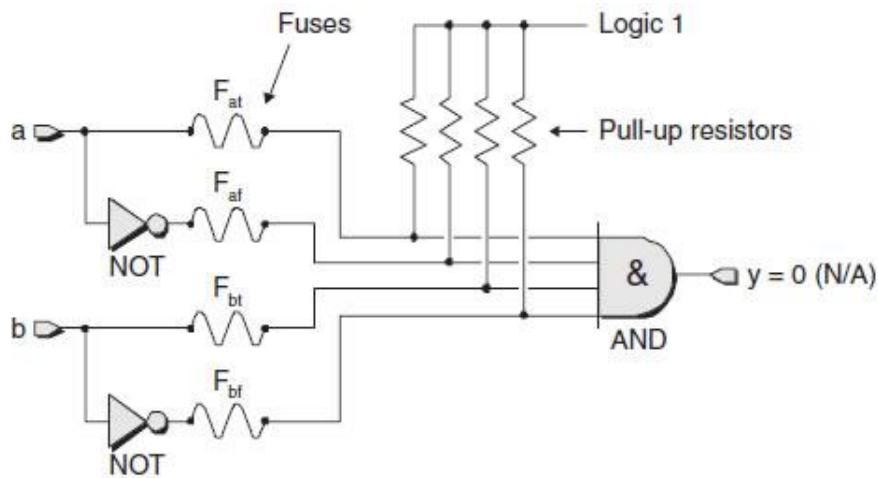
# Grubus ir vidutinis grūdėtumas



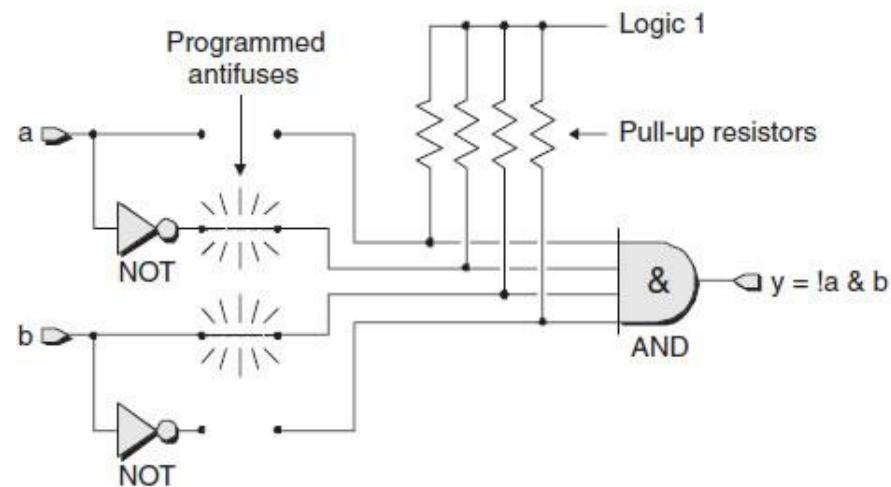
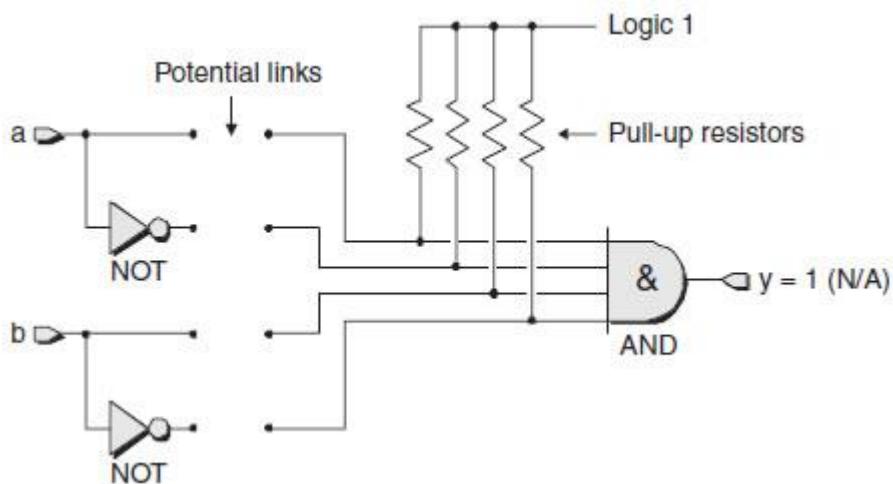
- Yra įrenginių, kurių loginis blokas yra labai sudėtingas, pvz., FFT skaičiavimo blokas arba net bendros paskirties procesorių. Todėl kartais LUT lentelių architektūros FPGA vadinamos vidutinio grūdėtumo, paliekant teisę grubiomis vadintis aukščiau paminėtus įrenginius.
- Didėjant grūdėtumo lygiui mažėja sujungimų ir išnaudojamas kristalo plotas

- Saugikliai (fuse)
- Antisaugikliai (Anti-fuse)
- SRAM
- EEPROM (FLASH)
- EPROM

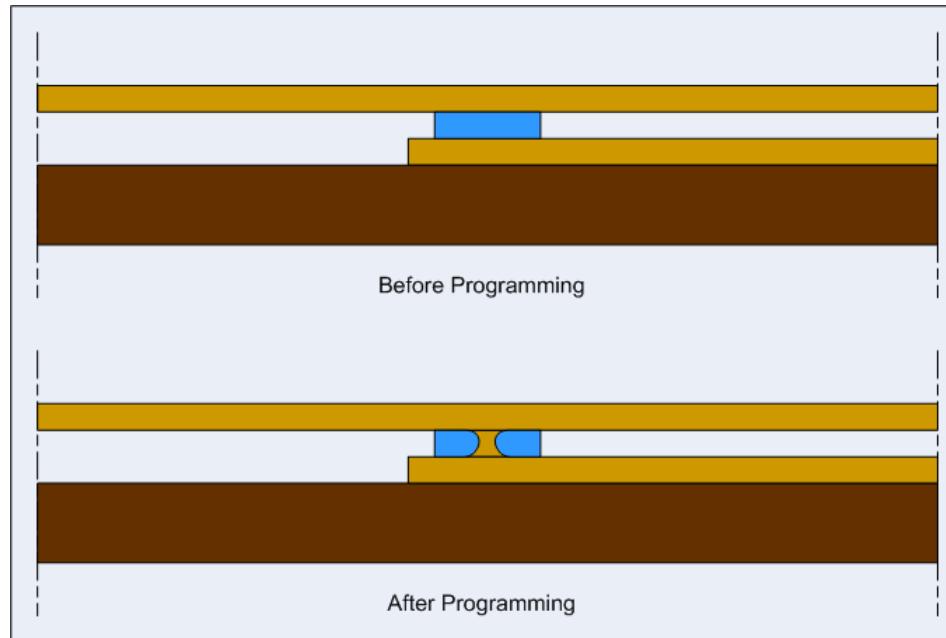
- Gamybos metu visi įmanomi sujungimai padaromi aktyvūs. Programavimo metu naudojant gerokai aukštesnę įtampą nei darbiniame režime ( $\sim 18$  V) nereikalingi sujungimai nudeginami.



- Prieš programavimą visi sujungimai būna atjungti (varža  $> 100 \text{ k}\Omega$ )  
Programuojant suardoma dielektrinė struktūra ir kontakto varža sumažėja iki  $500 \Omega$



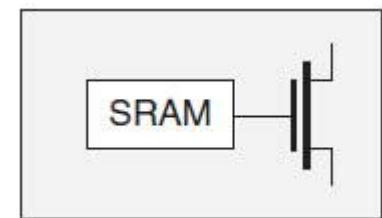
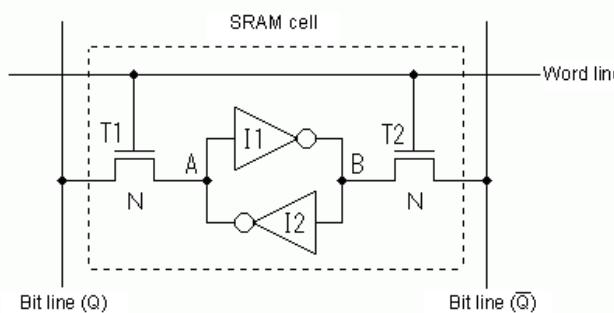
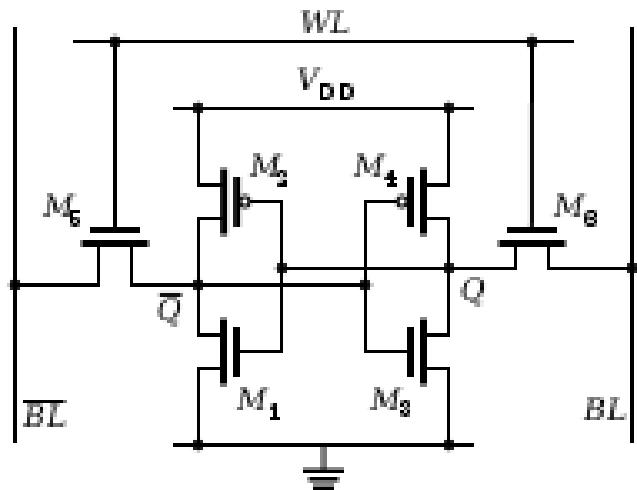
- Prieš programavimą visi sujungimai būna atjungti (varža  $> 100 \text{ k}\Omega$ )  
Programuojant suardoma dielektrinė struktūra ir kontakto varža sumažėja iki  $500 \text{ }\Omega$



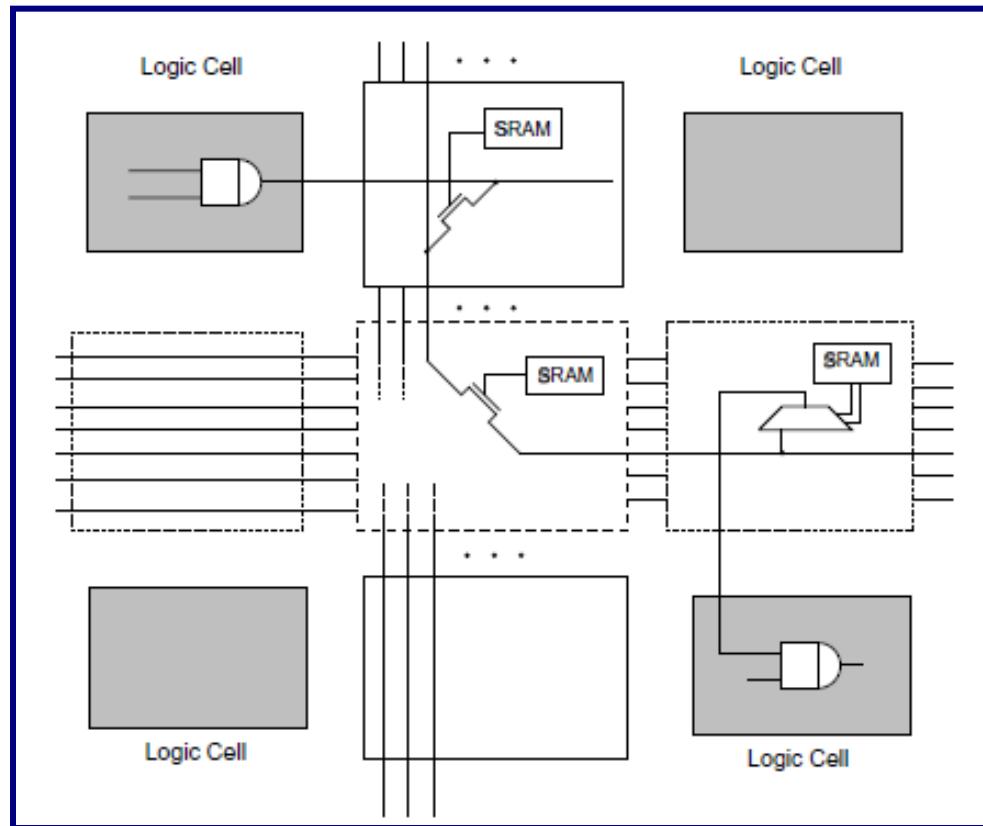
- Privalumai:
  - Maži gabaritai ( $\sim 1\mu\text{m}$ )
  - Maža “off” talpa ir maža “on” varža. Tai salygoja mažą energijos sunaudojimą
  - Nereikia perprogramuoti išjungus maitinimą
  - Nereikia išorinės atminties programos saugojimui
  - Atsparūs radiacijai
- Trūkumai:
  - Sudėtinga gamybos technologija
  - Nėra galimybės perprogramuoti

# FPGA. Konfigūravimo technologijos. SRAM

- Kiekvienas bitas SRAM technologijoje saugomas keturių tranzistorių struktūroje. Ši struktūra turi dvi stabilais būsenas kurios atitinka **0** ir **1**. Papildomai dar reikia dviejų tranzistorių šios struktūros aptarnavimui. Taigi tipinei SRAM celei reikia 6 MOSFET tranzistorių vienio bito saugojimui.



# FPGA. Konfigūravimo technologijos. SRAM

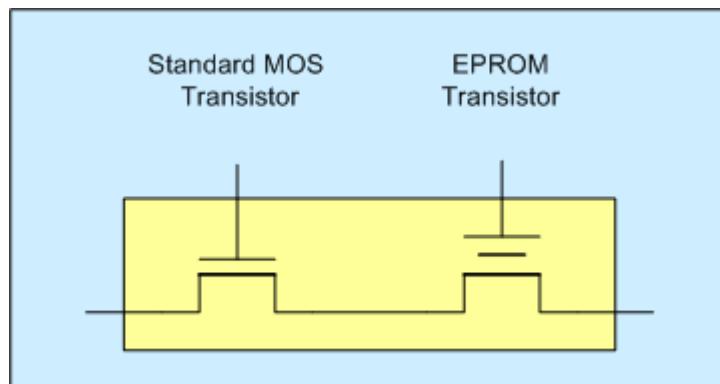
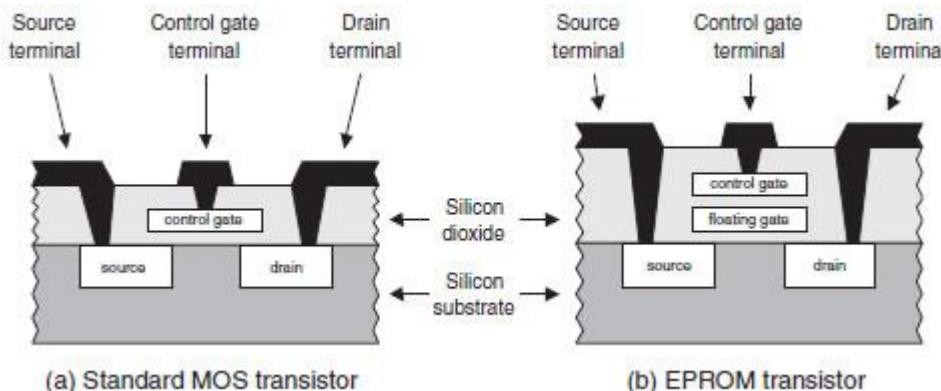


Dviejų loginių IR sujungimas panaudojant 3 SRAM celes, kurios valdo komutavimo matricos linijų sujungimą ir multipleksorių.

- Privalumai:
  - Labai tinkama projektavimui ir prototipams
  - Perrašymų skaičius neribotas
  - SRAM celės kuriamos ta pačia technologija, kaip ir likusi lusto dalis
  - Paprasta konstrukcija
- Trūkumai:
  - Reikia užkrauti konfigūraciją kiekvieną kartą padavus maitinimą (tam reikia papildomos energetiškai nepriklausomos atminties arba mikroprocesoriaus).
  - Dideli SRAM celės gabaritai (6-12 tranzistorių)
  - Didelis (lyginant su kitomis technologijomis) galios suvartojimas dėl nuotėkio srovii tranzistoriuose

# FPGA. Konfigūravimo technologijos. EEPROM

- Elektriškai trinama programuojamoji atmintis  
(electrically erasable programmable read-only memory)



Programavimo metu tarp užtūros ir santakos paduodama gerokai didesnė įtampa už darbinę. ‘Plaukiojanti’ užtūra išgyja neigiamą krūvį ir tranzistorius lieka atidarytas iki ‘ištrynimo’. Atrasis tranzistorius naudojamas informacijos ištrynimui (EPROM reikia UV spindulių)

- Privalumai:
  - Energetiškai nepriklausomi
  - Pasiruošę darbui tik padavus maitinimą
  - E2PROM celė fiziškai mažesnė už SRAM celę. Todėl galimi trumpesni sujungimai.
  - Mažesnė nuotėkio srovė, dėl mažesnio tranzistorių skaičiaus
- Trūkumai:
  - Sudėtinga gamybos technologija.
  - Didelė “on” varža
  - Vėlinimas dviem kartais blogesnis negu SRAM
  - Reikalinga didesnė rašymo įtampa (on-chip charge pump)
  - Rašymas apie 3 kart ilgesnis negu SRAM technologijos.

# FPGA. Konfigūravimo technologijų palyginimas



Feature	EEPROM	Anti-fuse	SRAM
Reprogrammable	Yes	No	Yes
Reprogramming Speed	Medium		Fast
Volatile	No	No	Yes
External Configuration File	No	No	Yes
Prototyping	Good	Bad	Good
Instant-On	Yes	Yes	No
Security	Good	Good	Poor
Configuration Cell	Medium	Small	Large
Power Consumption	Medium	Low	Medium
Radiation Susceptibility	Low	High	Low

# Programinės įrangos saugumo lygiai



Level 0 (ZERO). No special security features added to the system. It is easy to comprise the system with low cost tools.

- Level 1 (LOW). Some security features in place. They are relatively easily defeated with common laboratory or shop tools.
- Level 2 (MODLOW). The system has some security against non-invasive attacks; it is protected for some invasive attacks. More expensive tools are required, as well as specialised knowledge.
- Level 3 (MOD). The system has some security against non-invasive and invasive attacks. Special tools and equipment are required, as well as some special skills and knowledge. The attack may become time-consuming but will eventually be successful.
- Level 4 (MODH). The system has strong security against attacks. Equipment is available but is expensive to buy and operate. Special skills and knowledge are required to use the equipment for an attack. More than one operation may be required so that several adversaries with complementary skills would have to work on the attack sequence. The attack could be unsuccessful.
- Level 5 (HIGH). The security features are very strong. All known attacks have been unsuccessful. Some research by a team of specialists is necessary. Highly specialised equipment is necessary, some of which might have to be designed and built. The success of the attack is uncertain.

**Table 1 Security level of classical integrated circuits**

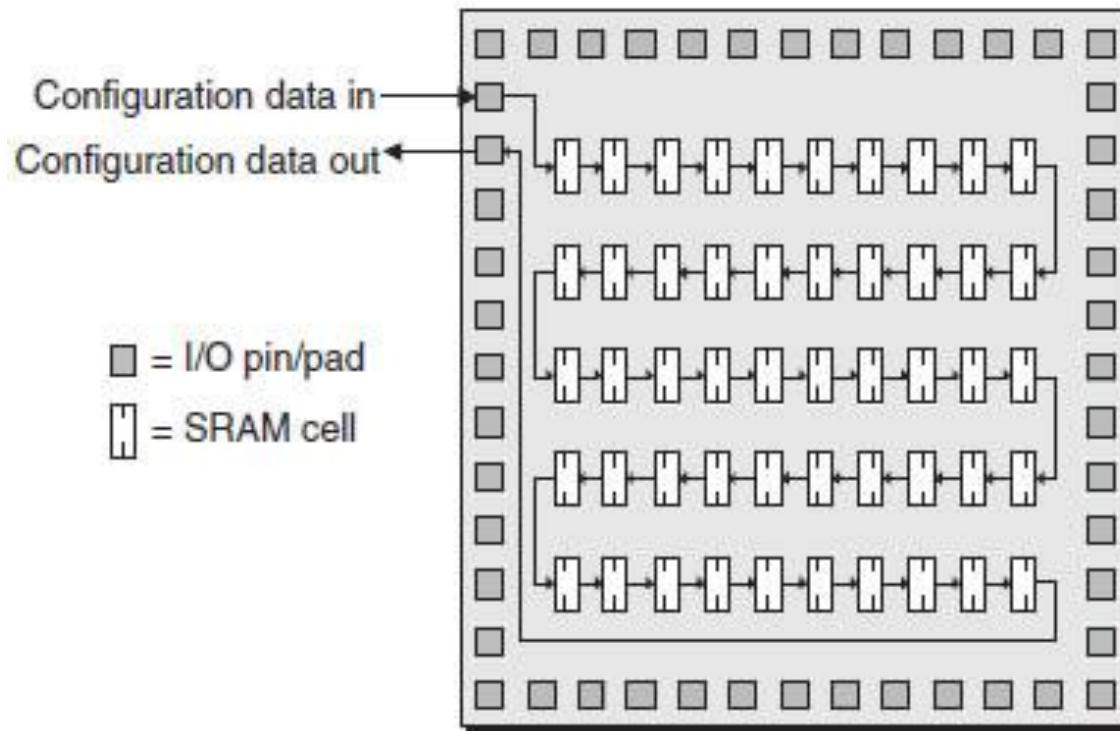
<i>Integrated circuit</i>	<i>Security level</i>
Conventional SRAM FPGA	0
ASIC gate array	3
Cell-based ASIC	3
SRAM FPGA with bitstream encryption	3
Flash FPGA	4
Antifuse FPGA	4

# FPGA. SRAM konfigūravimas



- SRAM pagrindu veikianti FPGA turi būti programuojama neišimant jos iš sistemos – “in system programming”. Ji perprogramuojama kiekvieną kartą įjungiant maitinimą. Tam kad visa sistema veiktų autonomiškai, šalia FPGA montuojama ir atminties mikroschema.
- Visos SRAM celės jungiamos į vieną postūmio registrą ir programos failas į jį įrašomas nuosekliai. Esant reikalui galima jungti kelias FPGA nuosekliai ir jas programuoti vienu metu. Tokiu pat būdu galima ir nuskaityti įrašytą konfigūraciją.
- SRAM celės gali būti grupuojamos į “freimus”

# FPGA. SRAM konfigūravimas



# FPGA konfigūravimo procesas



- Atliekant konfiguravimą būtina žinoti apie:
  - Konfiguravimo kontaktus – konfiguravimo režimui nustatyti
    - ✖ Kai kurie konfiguravimo kontaktai yra įėjimai (būsenos jungikliai), kai kurie išėjimai (būsenos indikatoriai)
  - Konfigūravimo režimus
    - ✖ Priklausomai nuo konfiguravimo režimo reikia atitinkamai ‘suvaldyti’ konfiguravimo kontaktus ir kitą valdymo logiką (grandynus)
    - ✖ Skirtingi deerinimo (debugging) režimai salygoja skirtingus konfiguravimo režimus, kurie gali būti naudojami netgi vienu metu

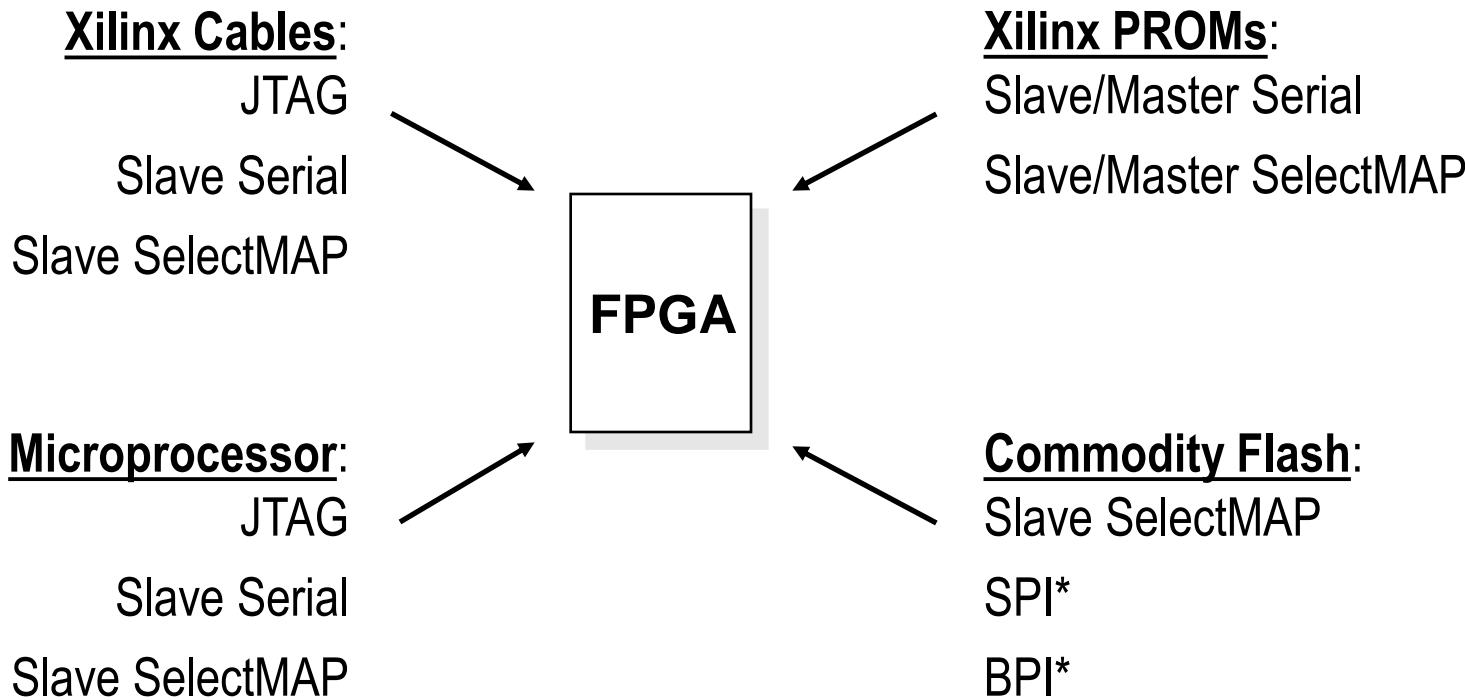
# FPGA konfigūravimo schemas



- Schemas:
  - **Aktyvi** – konfigūruojama FPGA pati generuoja valdymo ir sinchronizavimo signalus (naudojami nuoseklūs konfigūravimo įrenginiai EPCSx)
  - **Pasyvi** – naudojamas išorinis konfigūravimo įrenginys, mikroprocesorius, kuris valdo konfigūravimo procesą, PK prijungtas užkrovimo kabeliu. Konfigūravimo įrenginys arba mikroprocesorius siunčia konfigūravimo duomenis iš laikmenos (konfigūravimo įrenginio, RAM, kitos atminties) į FPGA
  - **JTAG** – naudojama JTAG sasaja
- Režimai:
  - Nuoseklus – konfigūravimo duomenys perduodami nuosekliai
  - Lygiagretus – lygiagrečiai (8 arba 16 bitų magistrale)

Šaltinis: [http://www.altera.com/literature/hb/cyc/cyc\\_c51013.pdf](http://www.altera.com/literature/hb/cyc/cyc_c51013.pdf)

# FPGA konfigūravimo režimai



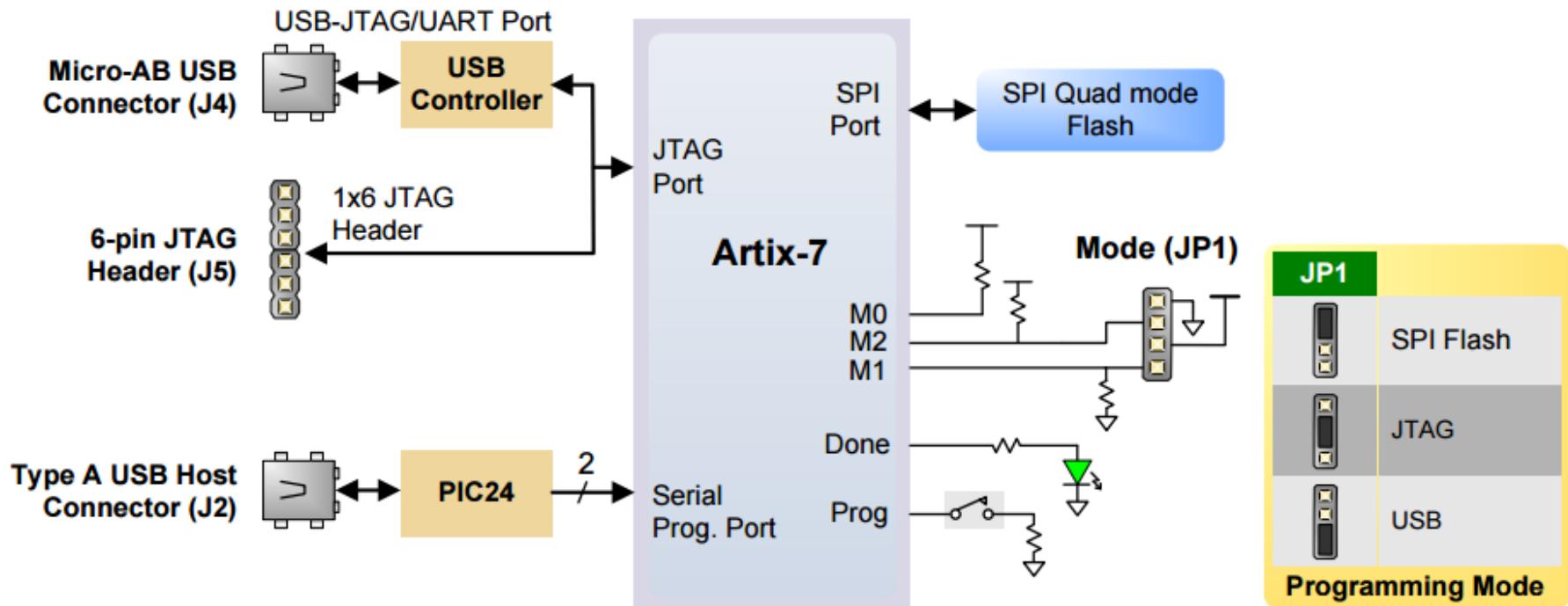
\*SPI and BPI support is available in Spartan™-6, Virtex™-6, and some older FPGA families

# FPGA konfigūravimo režimai



- Nuoseklus (Serial)
  - JTAG
    - ✖ Rekomenduojamas visiem prototipų projektavimo etapams. Naudojamas specialus JTAG kabelis
  - Master Serial
    - ✖ Valdymo signalus generuoja FPGA, naudojami nuoseklios sasajos atmintinės (Platform Flash PROM)
  - Slave Serial
    - ✖ Valdymo signalus generuoja išorinis įrenginys (mikroprocesorius, PC ar CPLD)
  - SPI (Serial Peripheral Interface)
    - ✖ Prosesą valdo FPGA. Naudojamas SPI Flash PROM
- Lygiagretus (naudojamos 8-bitų arba 16-bitų magistralės)
  - Master SelectMAP
    - ✖ Prosesą valdo FPGA
  - Slave SelectMAP
    - ✖ Valdymo signalus generuoja išorinis įrenginys
  - BPI (Byte-Wide Peripheral Interface)
    - ✖ Prosesą valdo FPGA. Naudojamas NOR Flash

# FPGA konfigūravimo režimai



# FPGA konfigūravimo režimai



- Konfiguruojant FPGA gali būti taikomi keli programavimo režimai. Naudojamas režimas nurodomas nustatant atitinkamas būsenas tam skirtuose išvaduose.

Configuration Mode	Preconfiguration Pull-ups	M0	M1	M2	CCLK Direction	Data Width	Serial D <sub>OUT</sub>
Master Serial mode	No	0	0	0	Out	1	Yes
	Yes	0	0	1			
Slave Parallel mode	Yes	0	1	0	In	8	No
	No	0	1	1			
Boundary-Scan mode	Yes	1	0	0	N/A	1	No
	No	1	0	1			
Slave Serial mode	Yes	1	1	0	In	1	Yes
	No	1	1	1			

# Konfigūravimo kontaktai



- Konfiguravimo metu naudojami specializuoti kontaktai
  - Mode pins
  - PROGRAM\_B
  - CCLK (configuration clock)
  - INIT\_B
  - DONE
  - DIN
  - DOUT
  - ...
- Kai kurių kontaktų signalų kryptis (iėjimas ar išėjimas) priklauso nuo konfiguravimo režimo (Pvz. CCLK)
- Kai kurie kontaktai naudojami tik spec. Konfiguravimo režimuose

# Konfigūravimo režimų valdymo signalai



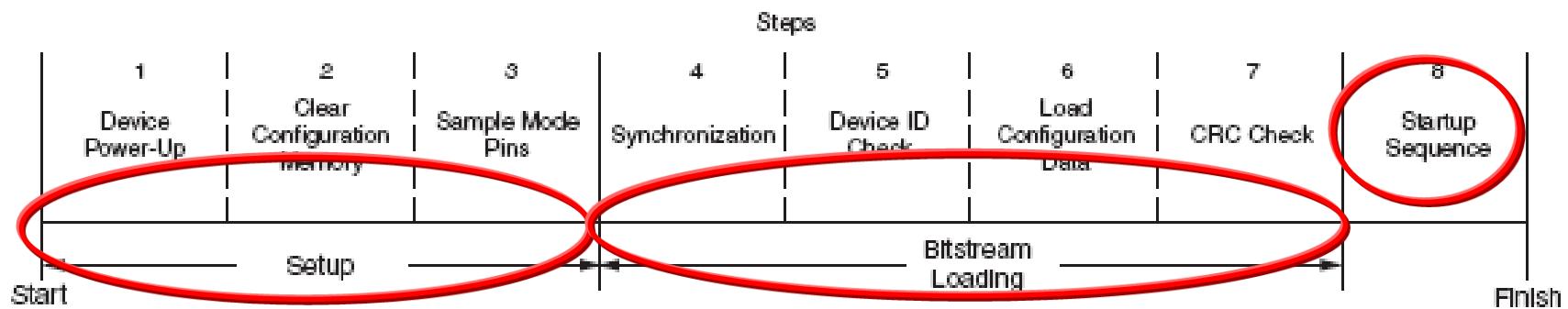
- Mode pins
  - (3) Input pin(s) that select which configuration mode is being used
- PROGRAM\_B
  - Input that initiates configuration
  - Active Low
- CCLK (configuration clock)
  - Input or output (depending on configuration mode)
  - Frequency up to 100 MHz (dependent on the FPGA, see configuration user guide)
- INIT\_B
  - Open-drain bi-directional pin
  - Error and power stabilization flag
  - Active Low
- DONE
  - Open-drain bi-directional pin
  - Indicates completion of configuration process

# Konfigūravimo režimų valdymo signalai



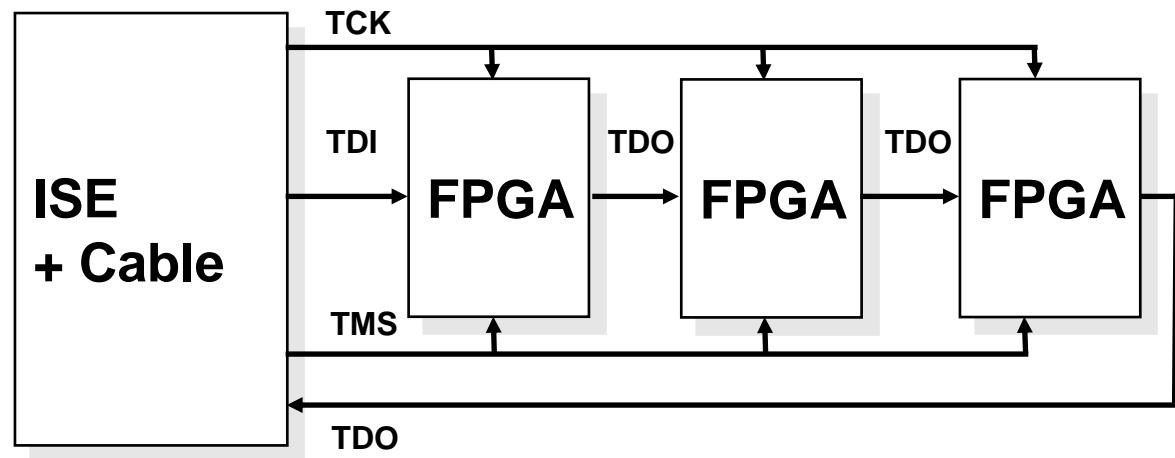
- DIN
  - Serial input for configuration data
- DOUT
  - Output to the next device in a daisy chain
  - Used in daisy chains only
- ...other pins are used for specific configuration modes
- Note that some configuration pins are dual purpose
  - They become user I/O after configuration is complete

# Konfigūravimo seka



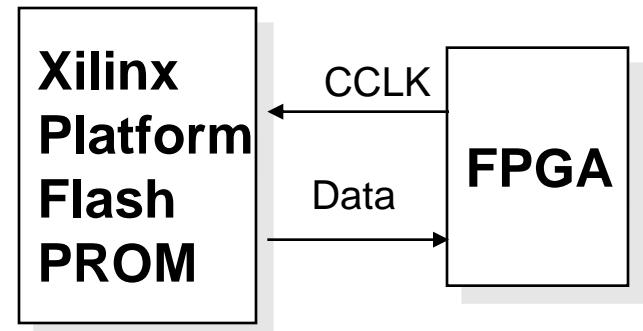
# JTAG konfigūravimo režimas

- Konfigūracinis failas (bitstream) saugomas kompiuteryje ir naudojant spec. programinę įrangą (ISE™) persiunčiamas tiesiai į FPGA
- Valdymo signalai siunčiami lygiagrečiai, konfigūracija - nuosekliai



# Master Serial Configuration Mode

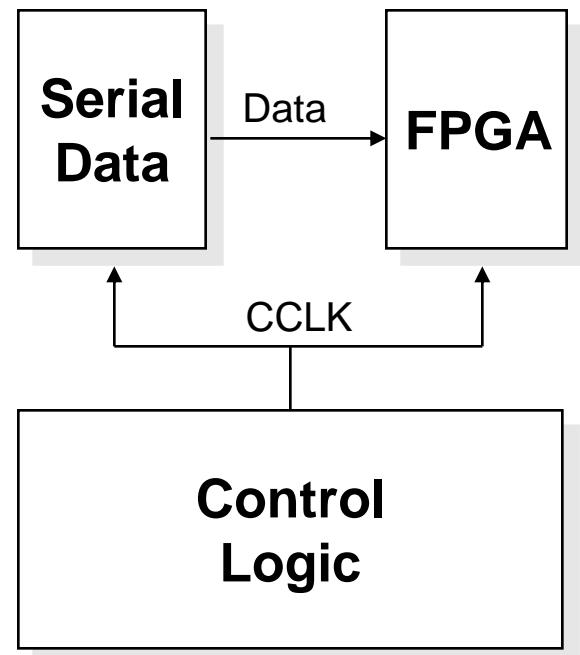
- Master Serial mode
  - FPGA išduoda CCLK signalą
  - Įrašomas 1 bitas per CCLK
  - Duomenys saugomi išorinėje atmintyje (serial PROM)
  - Lėtas bet lengviausiai realizuojamas/valdomas
- FPGA valdymo kontrolė
  - Visi konfigūravimo režimo nustatymo signalai žemame lygyje



# Slave Serial Configuration Mode



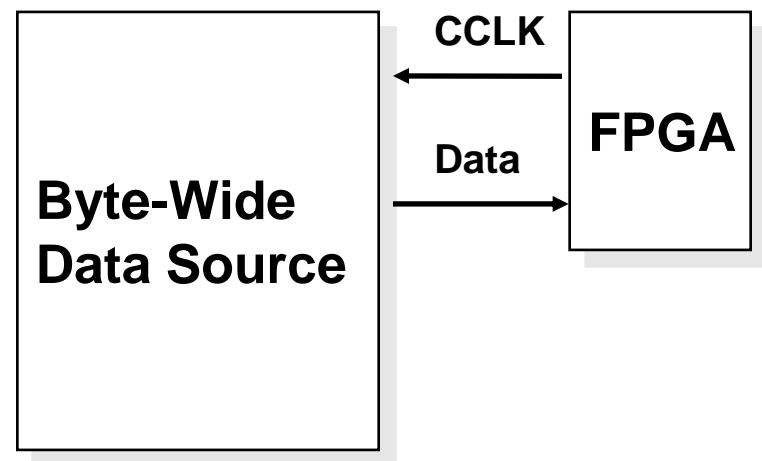
- Reikia išorinio CCLK signalų generatoriaus:
  - Mikroprocesorius arba mikrokontroleris
  - Nuoseklus Xilinx duomenų kabelis/jungtis
  - Kitas FPGA
  - Nuoseklioji FPGA jungtis
- Įrašomas 1 bitas per viena CCLK ciklą
- Visi konfigūravimo režimo valdymo signalai nustatyti į aukštą lygi.



# Master SelectMap Mode (Master Parallel)

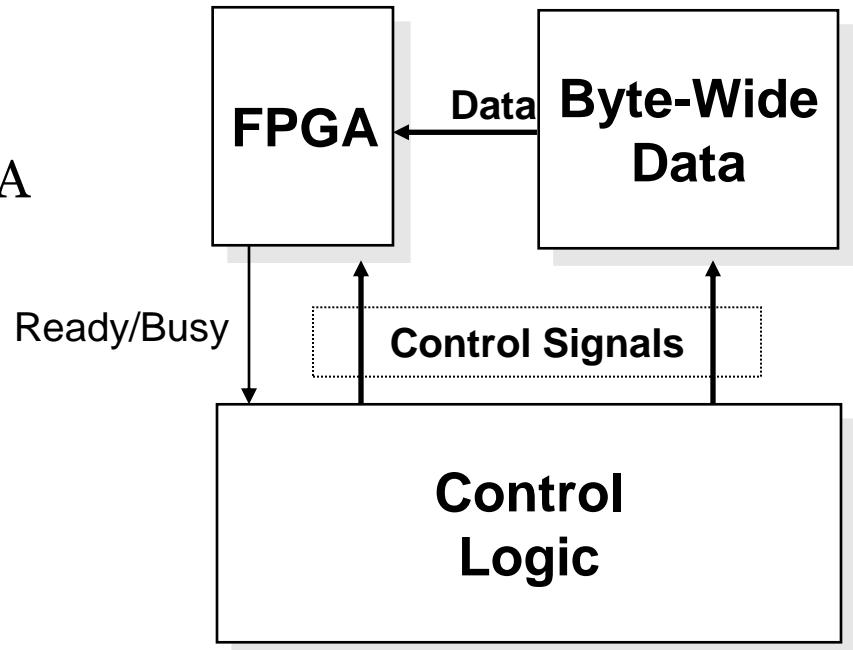


- FPGA atlieka valdymo funkciją
  - FPGA valdo adresų magistralę
  - Perduodamas 1 baitas 1 adresui
    - ✖ 8 CCLK ciklai baitui
- Didelis konfigūravimo greitis



# Slave SelectMap Mode (Slave Parallel)

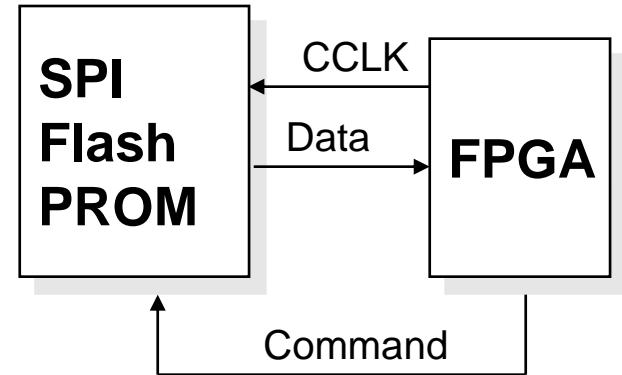
- Reikia išorinio valdiklio (mikroprocesorius, Xilinx EPC, kitas FPGA, CPLD)
- Ready/Busy bendravimo protokolas
- Perduodamas 1 baitas
- Išorinis įrenginys formuoja tiek atminties tiek konfigūruojamo FPGA valdymo signalus
- Naudojant *Xilinx Platform Flash XL* nereikia naudoti papildomo konfigūravimo įrenginio



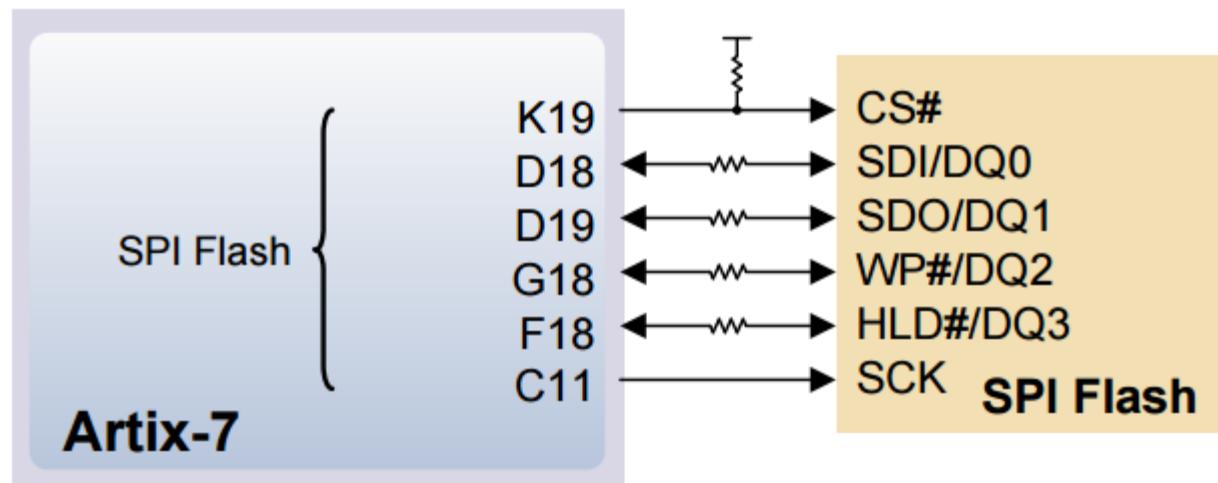
# Serial Peripheral Interface Mode (SPI)



- FPGA configures itself from an attached industry-standard SPI serial Flash PROM
  - FPGA issues a command to Flash and it responds with the data
  - Can be used in multi-boot applications where multiple bitstreams can be loaded by the FPGA
- Data is loaded 1 bit per CCLK (slow)
- There are no standards for the commands
  - Commands are vendor specific
  - Vendor Select (VS) pins tell the FPGA which commands to issue
  - Spartan™-6 supports x2 and x4 modes
  - See Data Sheet or Configuration User Guide for list of supported vendors
  - Excellent choice for embedded applications

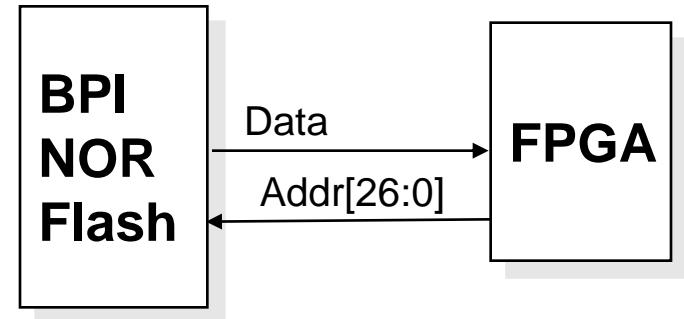


# Serial Peripheral Interface Mode (SPI)



# Byte-Wide Peripheral Interface Mode (BPI)

- FPGA issues an address to a BPI Flash, which responds with the data
  - Uses standard parallel NOR Flash interface
  - No clock is needed because the FPGA contains the control logic
- Usually used in embedded applications
  - Flash is easily used as addressable memory with address and data buses
  - Supported for Virtex™-5, Virtex-6, Spartan™-3E, and Spartan-6 FPGAs
- Xilinx Platform Flash XL is a 128 Mb parallel NOR and works in BPI and SelectMAP modes
  - Spartan-6 BPI mode is shared with SelectMAP mode



- FPGA konfigūravimas vyksta įjungus maitinimą iš išorinių duomenų laikmenų.
- Pasyvaus (slave) konfigūravimo režimai reikalauja išorinės valdymo schemos.
- SPI ir BPI konfigūravimo režimams valdyti naudojama integruotos valdymo schemos, todėl reikalauja mažiausiai derinimo resursų.
- JTAG sasaja pagalba labiausiai tinkama derinimo priemonė prototipo stadijoje.

# Xilinx FPGA tipai ir taikymų sritys

---

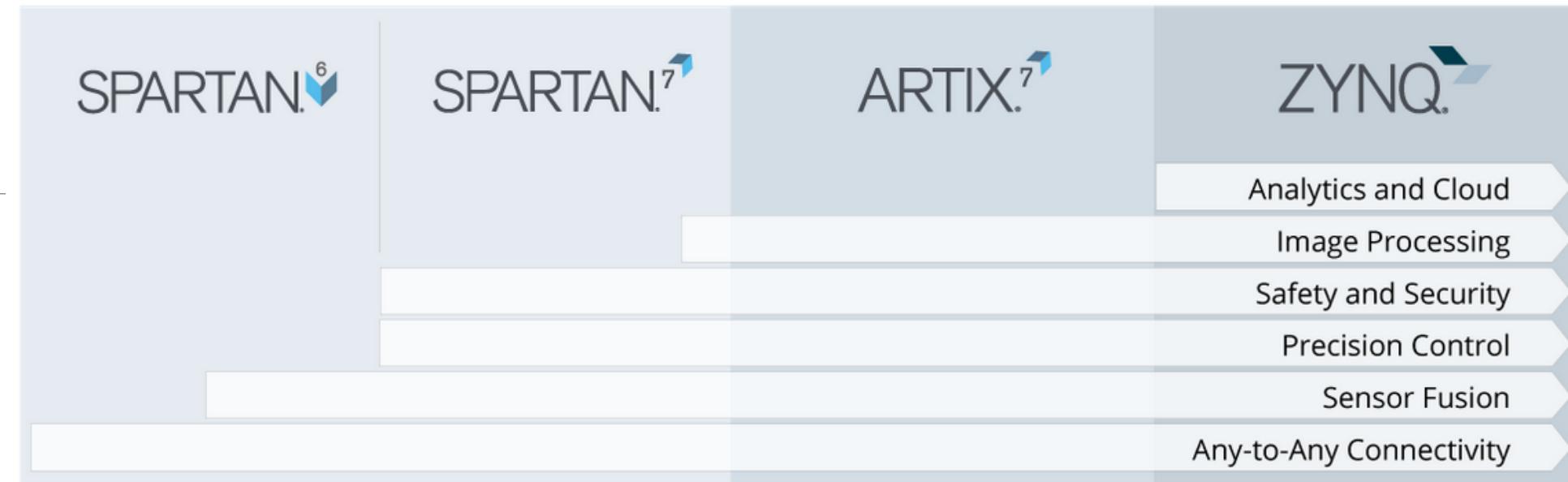
Ž.NAKUTIS

T170B114 PROGRAMUOJAMI LOGINIAI ĮRENGINIAI  
KTU ELEKTRONIKOS INŽINERIJOS KAT.  
2019

I/O Optimized

Transceiver Optimized

System Optimized



*Xilinx Cost-Optimized Portfolio*

## Key Application Examples

Spartan-6 FPGA	Spartan-7 FPGA	Artix-7 FPGA	Zynq-7000 All Programmable SoCs
<ul style="list-style-type: none"><li>• Full-HD Intelligent Digital Signage</li><li>• Industrial Networks</li><li>• Vehicle Networking and Connectivity</li><li>• High Resolution Video and Graphics</li></ul>	<ul style="list-style-type: none"><li>• Machine Vision Interfacing</li><li>• Single-Axis Motor Control</li><li>• Cryptographic Engine for V2X Applications</li><li>• Adaptive LED Lighting System</li><li>• Automotive Data Format / Standard Conversion</li></ul>	<ul style="list-style-type: none"><li>• Low Cost Ultrasound</li><li>• Wireless Backhaul Artix-7 Solution</li><li>• Programmable Logic Controller</li><li>• Software Defined Radio</li><li>• Multi-Protocol Machine Vision Camera</li></ul>	<ul style="list-style-type: none"><li>• Machine Vision</li><li>• Multi-Axis Motor Control</li><li>• Multi-Camera Driver Assistance Platform</li><li>• Ethernet Based Backhaul Solution</li><li>• Monitors and Projectors</li><li>• Multi-function Printers</li><li>• Situational Awareness</li><li>• Video Surveillance</li></ul>

# Cost-Optimized Portfolio Maximum Capacity Comparison

	Spartan-6 FPGA	Spartan-7 FPGA	Artix-7 FPGA	Zynq All Programmable SoCs Z-7007S, Z-7012S, Z-7014S, Z-7010, Z-7015, Z-7020
Logic Cells	150K	100K	215K	85K
Block RAM	4.8 Mb	4.2 Mb	13 Mb	4.9 Mb
DSP Slices	180	160	740	220
Transceiver Count	8	--	16	4
Transceiver Speed	3.2 Gb/s	--	6.6 Gb/s	6.6 Gb/s (Z-7012S, Z-7015)
Memory Interface (DDR3)	800 Mb/s	800 Mb/s	1,066 Mb/s	1,066 Mb/s
PCI Express Interface	Gen1x1	--	Gen2x4	Gen2x4 (Z-7012S, Z-7015)
Analog Mixed Signal (AMS) / XDAC	--	Dual 12-bit 1MSPS ADC with on-chip temp/supply sensors		
I/O Pins	576	400	500	328
I/O Standard Support (40+ protocols supported)	LVDS, Mini-LVDS, Diff HSTL, Diff SSTL, DisplayPort, XAUI, CPRI/OBSAI, V-by-One, Triple Rate SDI, 6G-SDI (Artix-7 FPGA/Zynq-7000 AP SoCs)			

<https://www.xilinx.com/support/documentation/backgrounder/cost-optimized-backgrounder.pdf>

# Spartan-6 šeima

---

The Spartan-6 FPGA family is I/O optimized for connectivity and is the well-established, low-cost market leader. With 25+ design kits and evaluation boards, the entire family is in volume production for immediate implementation, with over 80 million units shipped. Because it is biased towards raw connectivity, the family has best-in-class I/O and form factor related features, including:

- Highest I/O to logic cell ratio
- Most total package offerings
- Smallest form-factor offerings
- Support for 40+ protocols



Figure 2: New OS Support on ISE Tool Suite for Spartan-6 Devices

This makes it an ideal bridging and companion chip solution. Based on Samsung's 45nm low power (LP) process, it can also be a cost-effective algorithm engine, e.g., for video or motor control, with a competitive performance/watt profile.

# Xilinx FPGA ir SoC produktai

Cost-Optimized Portfolio	7 Series	UltraScale	UltraScale+
Spartan-7	Spartan-6	Kintex UltraScale	Kintex UltraScale+
Artix-7	Zynq-7000	Virtex UltraScale	Virtex UltraScale+

**45nm**

SPARTAN.<sup>6</sup>

**28nm**

VIRTEX.<sup>7</sup>  
KINTEX.<sup>7</sup>  
ARTIX.<sup>7</sup>  
SPARTAN.<sup>7</sup>

**20nm**

VIRTEX.<sup>7</sup>  
UltraSCALE  
KINTEX.<sup>7</sup>  
UltraSCALE

**16nm**

VIRTEX.<sup>7</sup>  
UltraSCALE+  
KINTEX.<sup>7</sup>  
UltraSCALE+

# FPGA šeimos pagal taikymus

---

SPARTAN<sup>6</sup>

*I/O Optimized*

- 

- Sensor Fusion

SPARTAN<sup>7</sup>

*I/O Optimized*

- Any-to-Any Connectivity

- Sensor Fusion

- Precision Control

- Safety and Security

ARTIX<sup>7</sup>

*Transceiver Optimized*

- Any-to-Any Connectivity

- Sensor Fusion

- Precision Control

- Safety and Security

- Image Processing

ZYNQ<sup>®</sup>

*System Optimized*

- Any-to-Any Connectivity

- Sensor Fusion

- Precision Control

- Safety and Security

- Image Processing

- Analytics and Cloud

# Xilinx CPLD (Complex Programmable Logic Devices) CoolRunner-II serija

Part Number		XC2C32A	XC2C64A	XC2C128	XC2C256	XC2C384	XC2C512
Logic Resources	System Gates	750	1,500	3,000	6,000	9,000	12,000
	Macrocells	32	64	128	256	384	512
	Product Terms Per Macrocell	56	56	56	56	56	56
Clock	Global Clocks	3	3	3	3	3	3
	Product Term Clocks Per Function Block	16	16	16	16	16	16
I/O Resources	Maximum I/O	33	64	100	184	240	270
	Input Voltage Compatible	1.5 / 1.8 / 2.5 / 3.3					
	Output Voltage Compatible	1.5 / 1.8 / 2.5 / 3.3					
	Min. Pin-to-Pin Logic Delay (ns)	3.8	4.6	5.7	5.7	7.1	7.1
Speed Grades	Commercial Speed Grades (Fastest to Slowest)	-4, -6	-5, -7	-6, -7	-6, -7	-7, -10	-7, -10
	Industrial Speed Grades (Fastest to Slowest)	-6	-7	-7	-7	-10	-7 <sup>(1)</sup> , -10





# Programuojamų loginių įrenginių (FPGA) projektavimo procesas (*Xilinx Vivado pagrindu*)

---

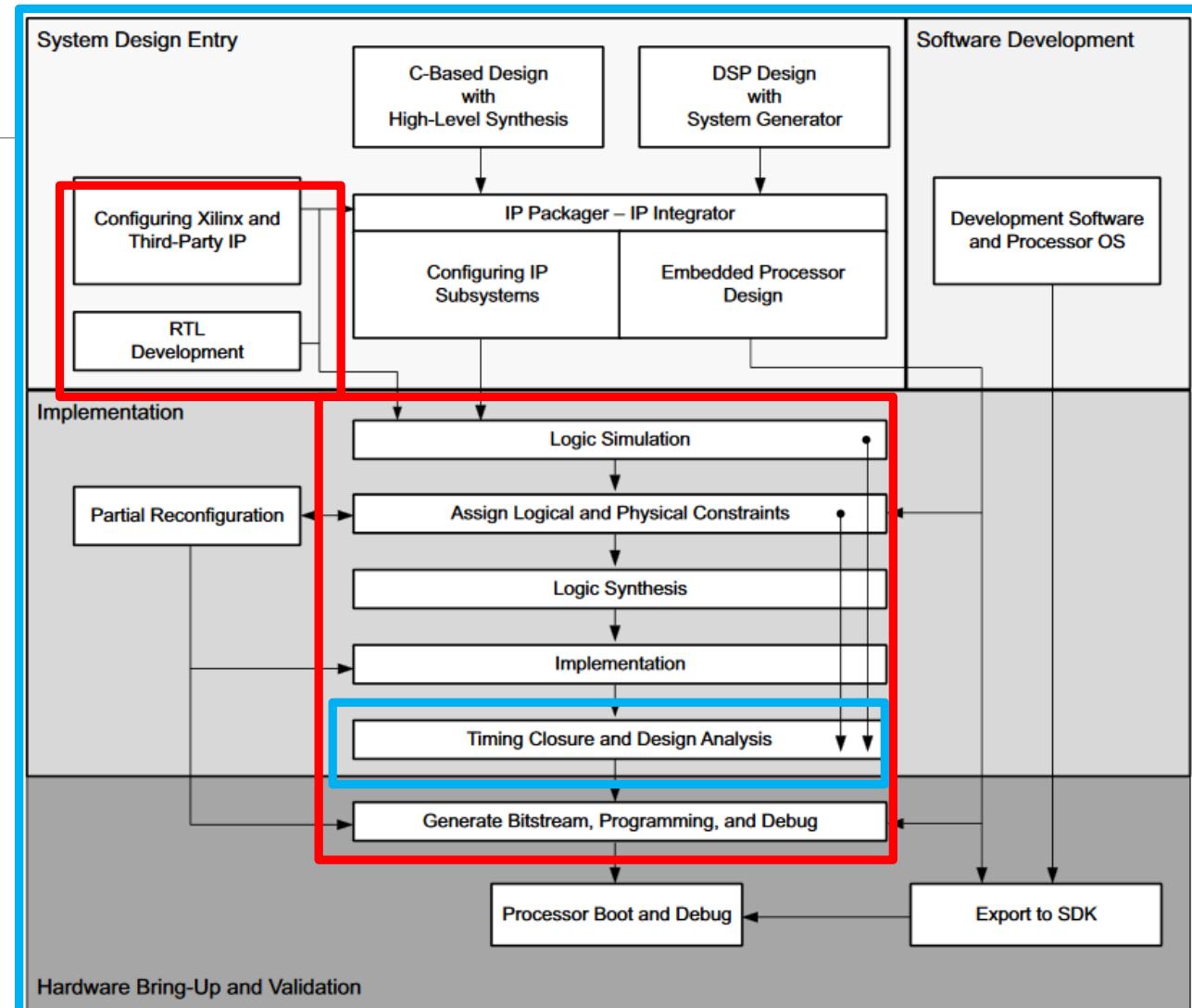
Ž.NAKUTIS

T170B114 PROGRAMUOJAMI LOGINIAI ĮRENGINIAI  
KTU ELEKTRONIKOS INŽINERIJOS KAT.  
2022

# Xilinx projektavimo metodika (UltraFast)

Šio studijo  
modulio  
paskirtis

Magistrantūros  
studijose



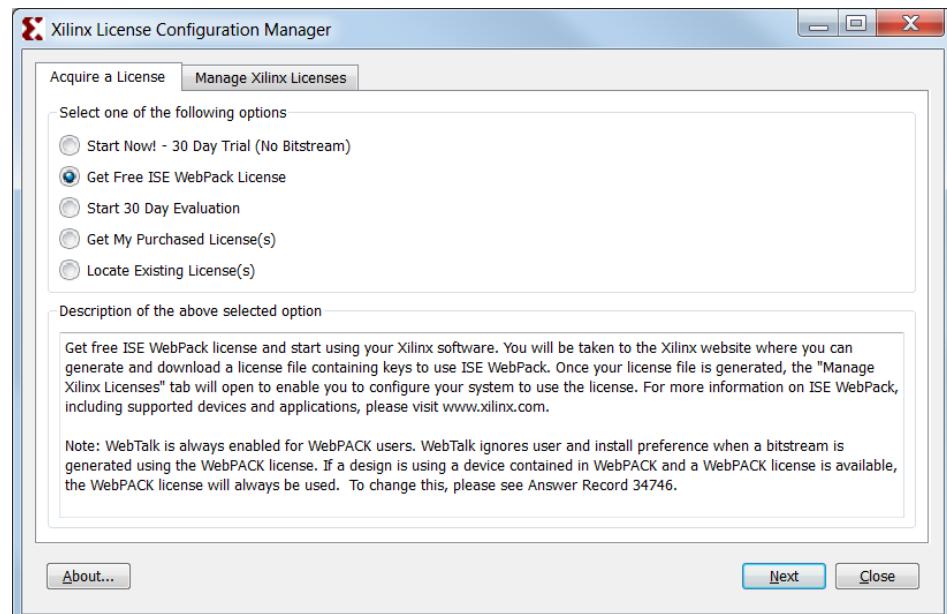
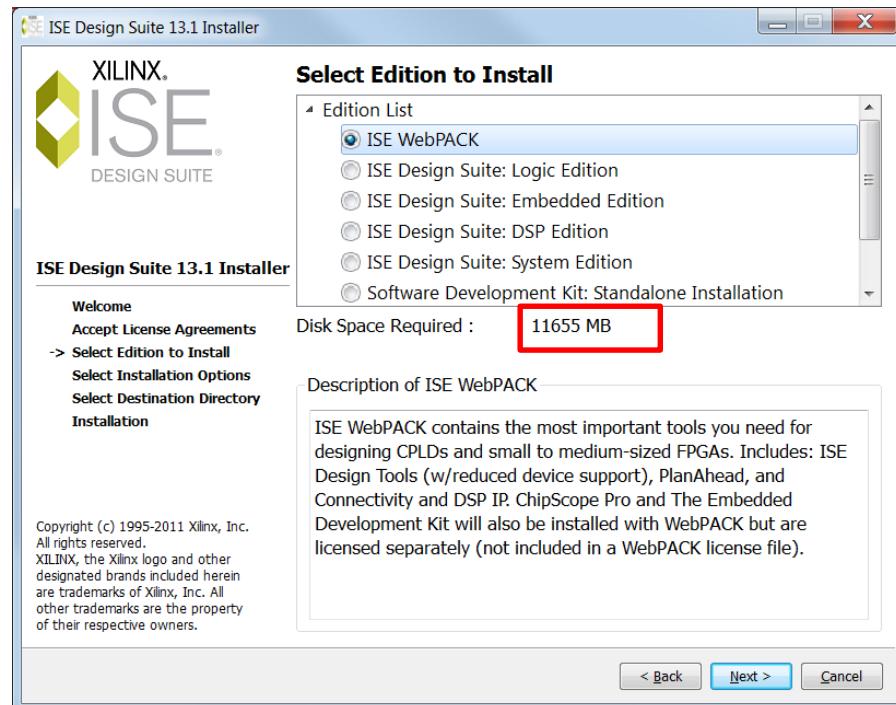
# Projektavimo įrankiai

---

- ❑ HDL sintezės įrankiai
  - ❑ XST (Xilinx Synthesis Tool) integruotas ISE/Vivado aplinkoje
- ❑ Modeliavimas (simuliavimas) ir verifikavimas
  - ❑ Isim vidinis ISE aplinkos simulatorius
  - ❑ ModelSim versija teikiama kartu su Vivado
- ❑ Realizavimas (angl. Implementation)
- ❑ Projekto įvedimo įrankiai (Schemų įvedimas, projekto valdymas ir kt.)
- ❑ Laikinė analizė (angl. Timming analysis), galios suvartojimo analizė

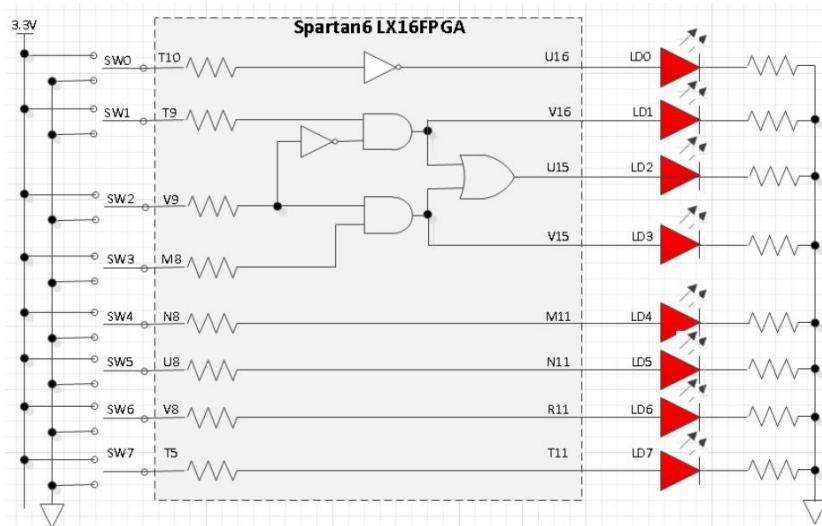
<https://forums.xilinx.com/t5/Design-Tools/ct-p/DESIGN>

# Xilinx ISE WebPack paketo „dydis“



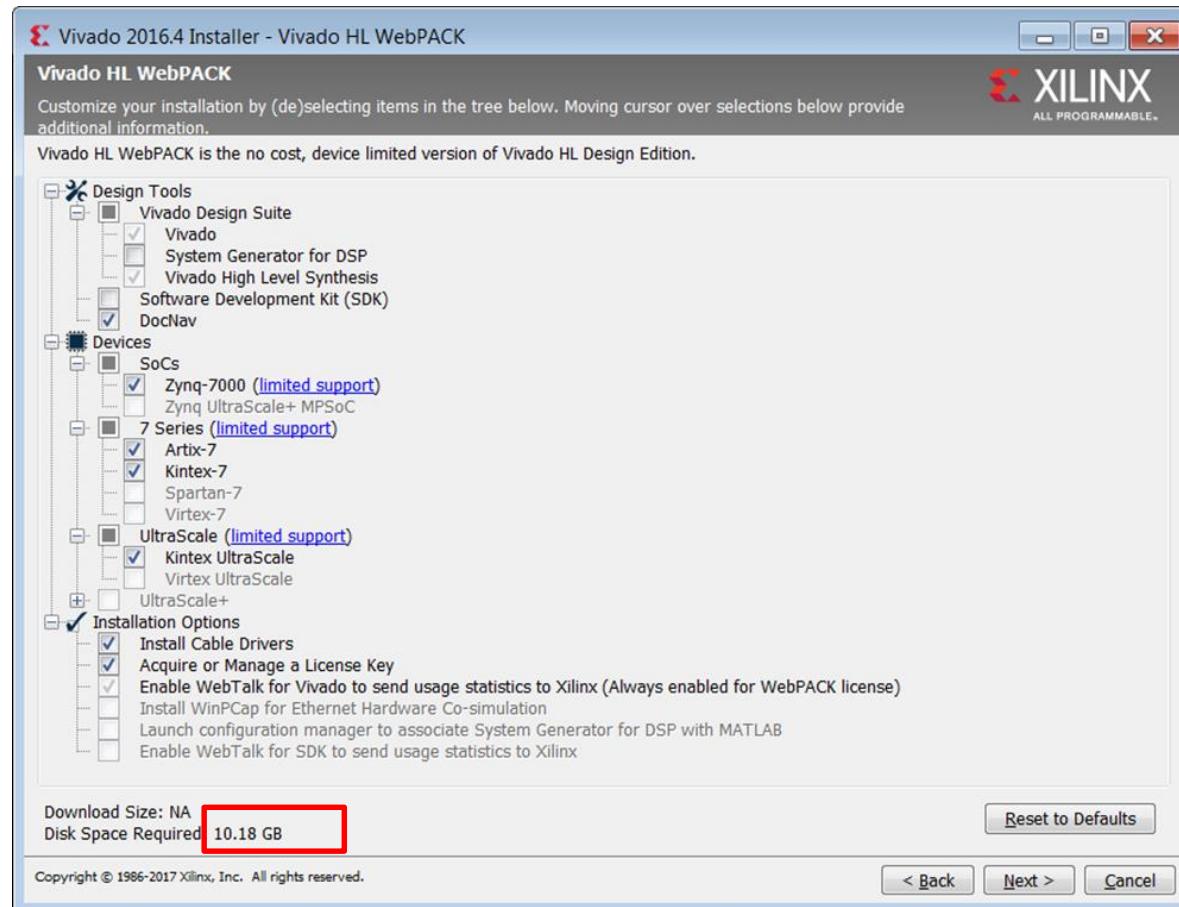
# Projekto aprašo įvedimo variantai (Design entry)

- HDL (Hardware Description Language) kalba, tokia kaip VHDL ar Verilog
- Schema (SCH). Tik ISE aplinkoje.
- EDIF arba NGC/NGO failai (jeigu sintezė bus atliekama nenaudojant Xilinx Project Navigator)

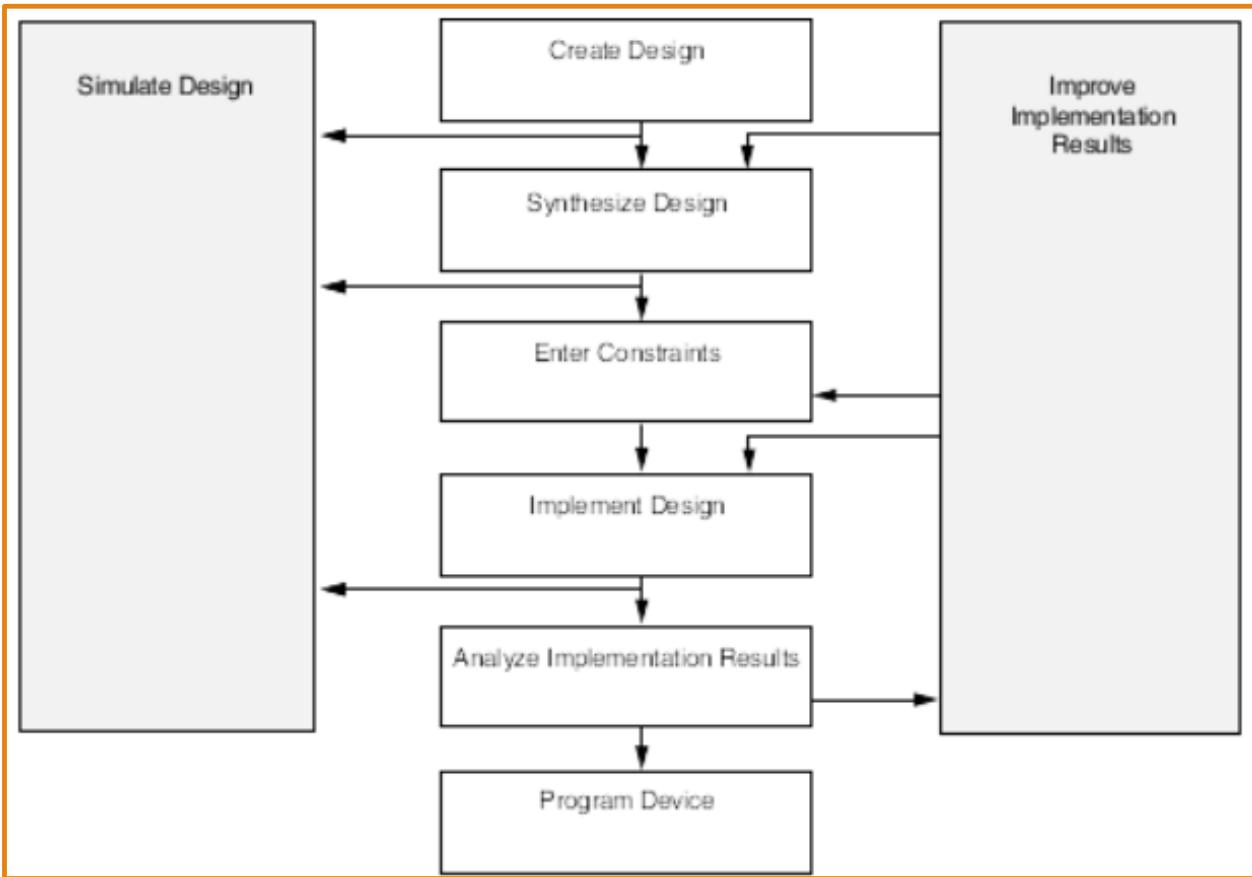


```
2 //////////////////////////////////////////////////////////////////
3 // Module Name: tutorial_tb
4 //////////////////////////////////////////////////////////////////
5 module tutorial_tb(
6   );
7   );
8
9   reg [7:0] switches;
10  wire [7:0] leds;
11  reg [7:0] e_led;
12
13 integer i;
14
15 tutorial tut1(.led(leds),.swt(switches));
16
17 function [7:0] expected_led;
18   input [7:0] swt;
19 begin
20   expected_led[0] = ~swt[0];
21   expected_led[1] = swt[1] & ~swt[2];
22   expected_led[3] = swt[2] & swt[3];
23   expected_led[2] = expected_led[1] | expected_led[3];
24   expected_led[7:4] = swt[7:4];
25 end
26 endfunction
27
```

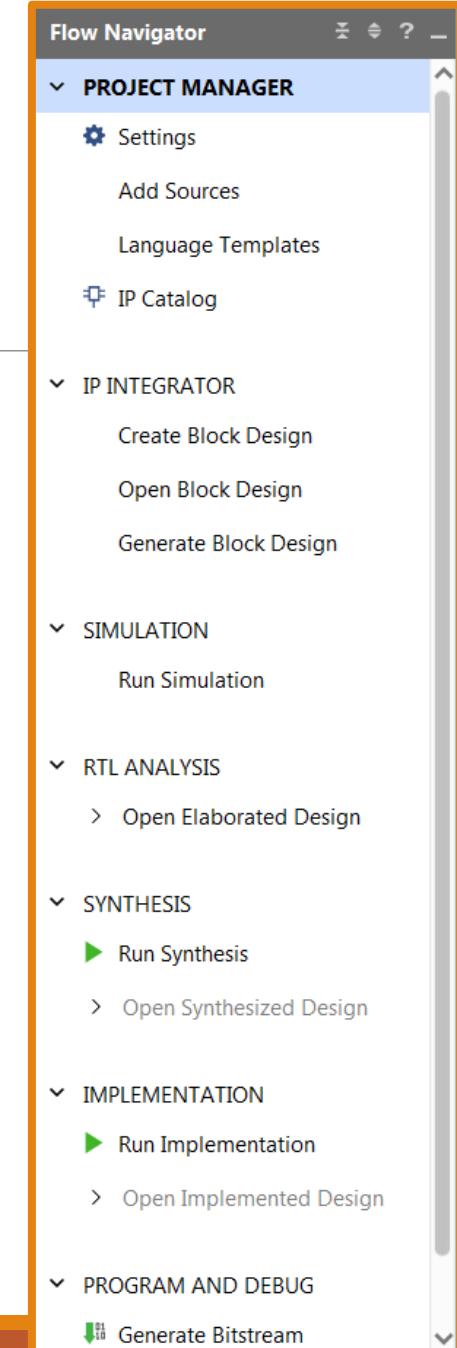
# Vivado aplinka pakeičia ISE nuo 2014 metų



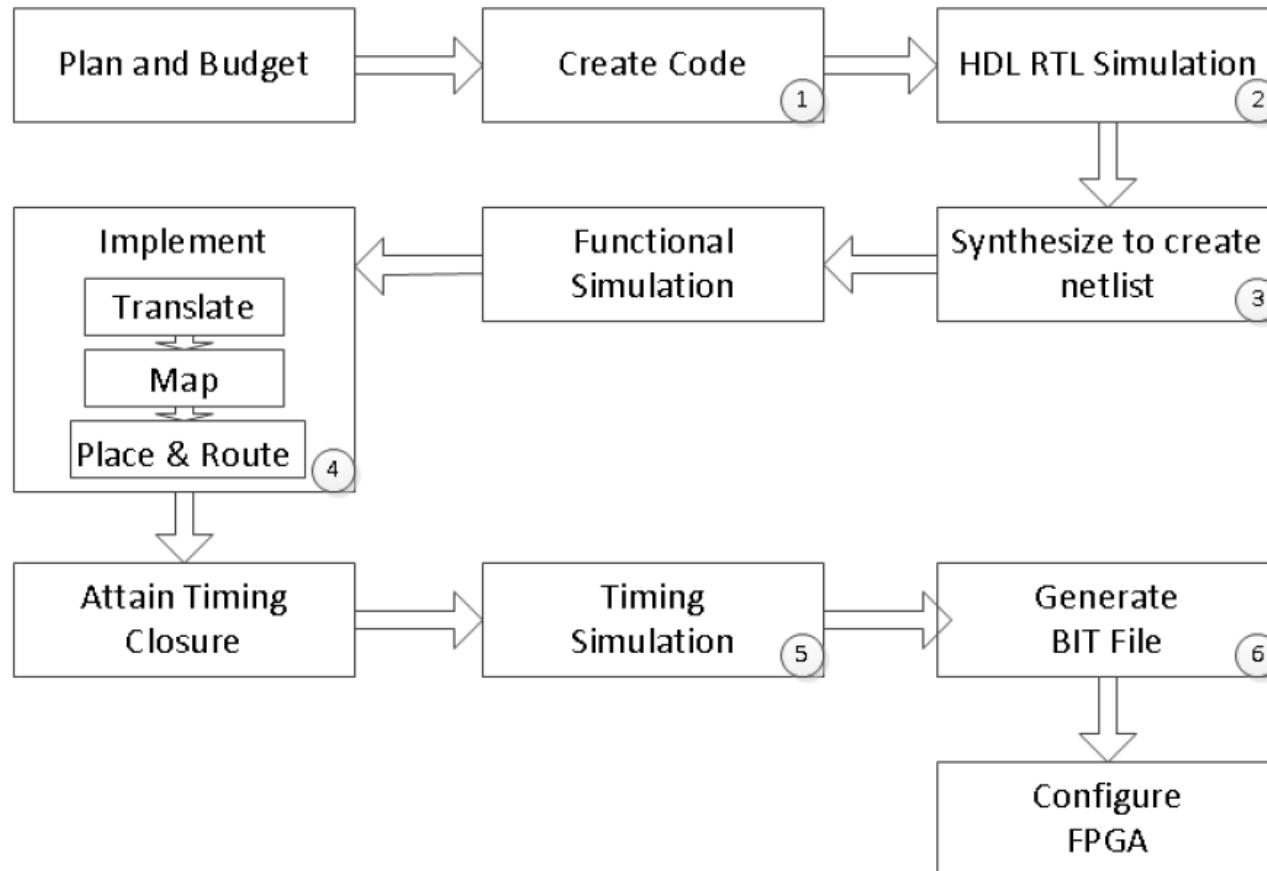
# Projektavimo procesas (eiga) (angl. Design flow)



[http://www.xilinx.com/support/documentation/sw\\_manuals/xilinx11/ise\\_c\\_fpga\\_design\\_flow\\_overview.htm](http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_fpga_design_flow_overview.htm)



# Tipiniai projektavimo etapai (ir atitinkami įrankiai)

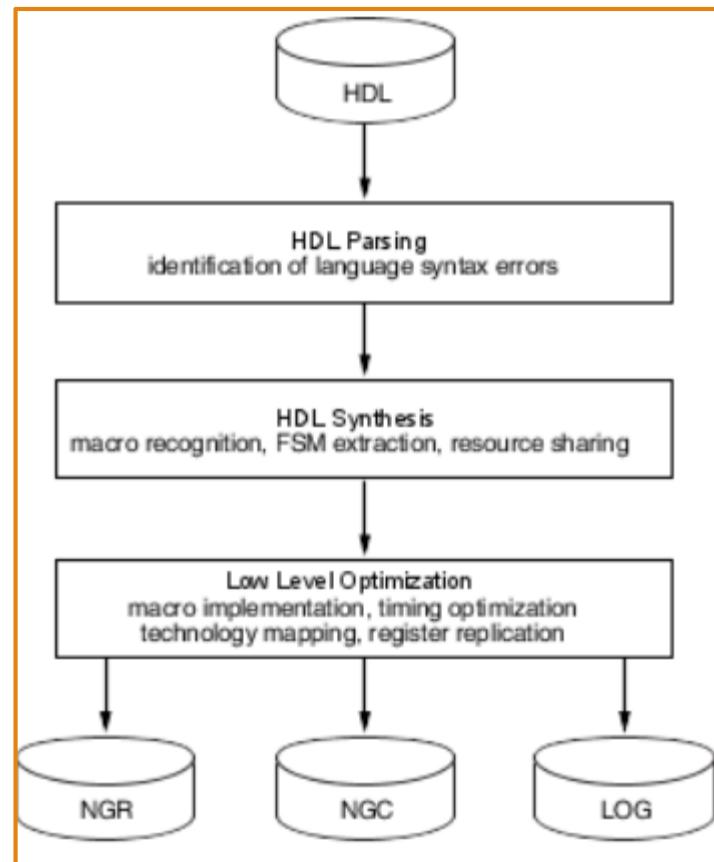


# Sintezė

---

- ❑ Sintezės metu projektas yra kompiliuojamas, konvertuojant HDL aprašus į grandinės failus, priklausomus nuo FPGA architektūros (angl. *architecture dependent netlist*)
- ❑ ISE/Vivado aplinkoje palaikomas vidinis sintezės įrankis XST ir trečiųjų šalių įrankiai (Synplify, Synplify Pro, Precision)

# XST (Xilinx Synthesis Tool) sintezės žingsniai



# Sintezės strategijos (optimizavimo kriterijai) Vivado aplinkoje

The screenshot shows the Vivado Settings dialog window. The left sidebar lists 'Project Settings' (General, Simulation, Elaboration, **Synthesis**, Implementation, Bitstream, IP) and 'Tool Settings' (Project, IP Defaults, Board Repository, Source File, Display, WebTalk, Help, Text Editor, 3rd Party Simulators, Colors, Selection Rules, Shortcuts, Strategies, Window Behavior). The main area is titled 'Synthesis: Specify various settings associated to Synthesis'. It includes sections for 'Constraints' (Default constraint set: constrs\_1 (active)) and 'Report Options' (Strategy: Vivado Synthesis Default Reports (Vivado Synthesis 2019)). Under 'Options', there is a checkbox for 'Write Incremental Synthesis' which is unchecked. The 'Incremental synthesis:' dropdown is set to 'Not set'. A dropdown menu for 'Strategy:' is open, showing 'Vivado Synthesis Defaults (Vivado Synthesis...)' as the selected option. This dropdown also lists 'User Defined Strategies' and 'Vivado Strategies' (Vivado Synthesis Defaults, Flow\_AreaOptimized\_high, Flow\_AreaOptimized\_medium, Flow\_AreaMultThresholdDSP, Flow\_AlternateRoutability, Flow\_PerfOptimized\_high, Flow\_PerfThresholdCarry, Flow\_RuntimeOptimized). At the bottom, a note says 'Select an option above to'.

# Modeliavimas

---

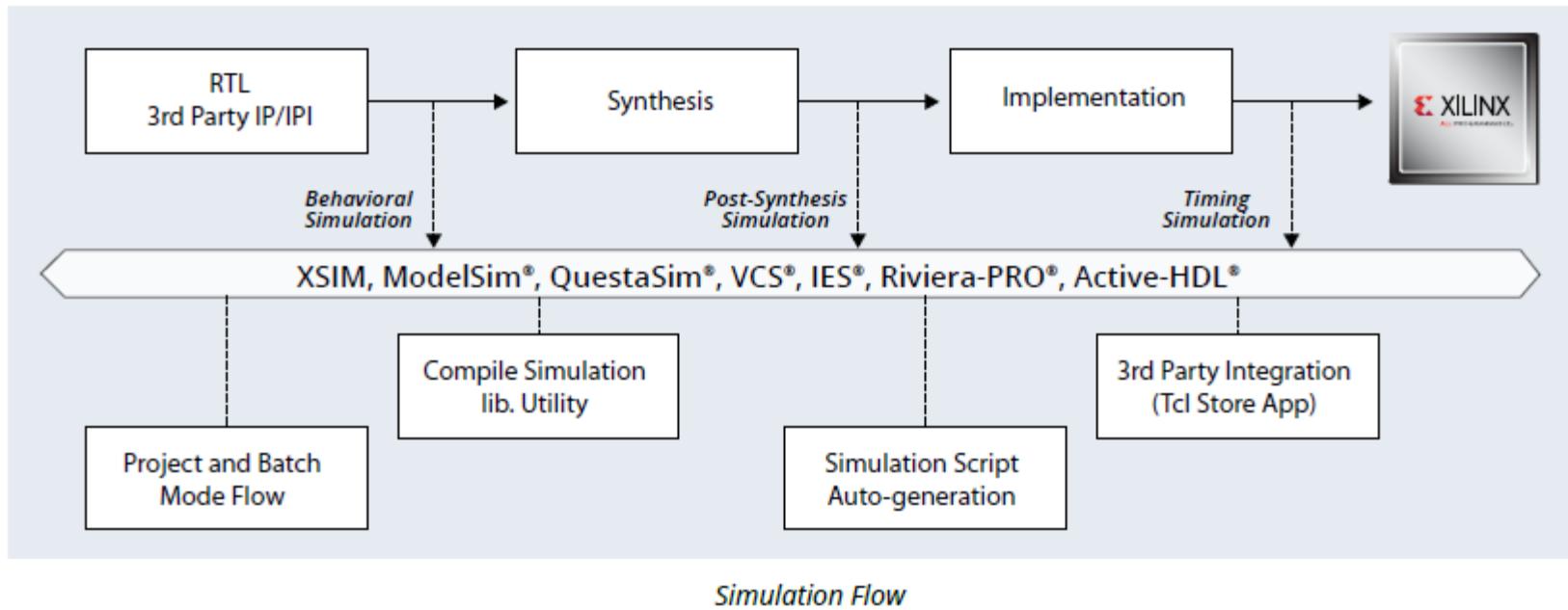
Įvairiuose projektavimo proceso etapuose projekto funkcionalumas gali būti tikrinamas panaudojant simulatorius.

ISE aplinkoje galima panaudoti vidinj simulatorių ISim arba ModelSim simulatorių. Simuliavimą galima atlikti ir ne ISE aplinkoje, o naudojant bet kokj kitą simulatorių.

Modeliavimo tipai:

- Elgsenos (angl. behavioral)
- Funkcinis (angl. functional) : Post-synthesis, post-implementation
- Laikinis (angl. timing analysis)

# Modeliavimas Vivado aplinkoje



# Vivado ir suderinami simuliatoriai

---

Vendor	Simulators	Interactive	Scripts
Aldec	Active-HDL, Riviera-PRO	✓	✓
Cadence	Incisive Enterprise Simulator (IES)	✓	✓
Mentor Graphics	ModelSim, Questa Advanced Simulator	✓	✓
Synopsys	VCS	✓	✓
Xilinx	<a href="#">Vivado Simulator</a>	✓	✓

# Vivado Simulatorius (integruotas į Vivado aplinką)



# Vivado simulatoriaus ypatybės

---

- ❑ Mišrus kalbų palaikymas (Verilog, SystemVerilog ir VHDL).
- ❑ Nemokamas ir neapribotų galimybių

# Tipiniai modeliavimo žingsniai

---

1. Modeliavimo bibliotekų kompiliavimas (nebūtinas naudojant ISim)
2. Projekto ir virtualiųjų bandymo stendų sukūrimas
3. Funkcinis modeliavimas
4. Projekto realizavimas ir laikinio modeliavimo grandinės (netlist) sukūrimas
5. Laikinis modeliavimas

# Ribojimų įvedimas (angl. constraints entry)

---

Ribojimais nurodomi laikiniai, talpinimo ir kiti reikalavimai. ISE aplinkoje yra realizuoti redaktoriai padedantys įvesti laikinius ir I/O prievedė bei išdėstymo ribojimus.

Ribojimų tipai:

1. Laikiniai (timing constraints)
2. Talpinimo (placement constraints)
3. Sintezės (synthesis constraints)

# Ribojimų įvedimo būdai

---

- ❑ Sukurti UCF (User Constraints File) failus, panaudojant:
  - ❑ Constraints Editor
  - ❑ ISE Text Editor
  - ❑ PlanAhead™ software (for FPGAs)
  - ❑ Pinout and Area Constraints Editor (PACE) (for CPLDs)
  - ❑ Commercially available text editors
- ❑ Sukurti HDL failus, panaudojant tekstinį redaktorių
- ❑ Sukurti XST ribojimų failą (XCF), panaudojant tekstinį redaktorių

# Realizavimas (angl. Implementation)

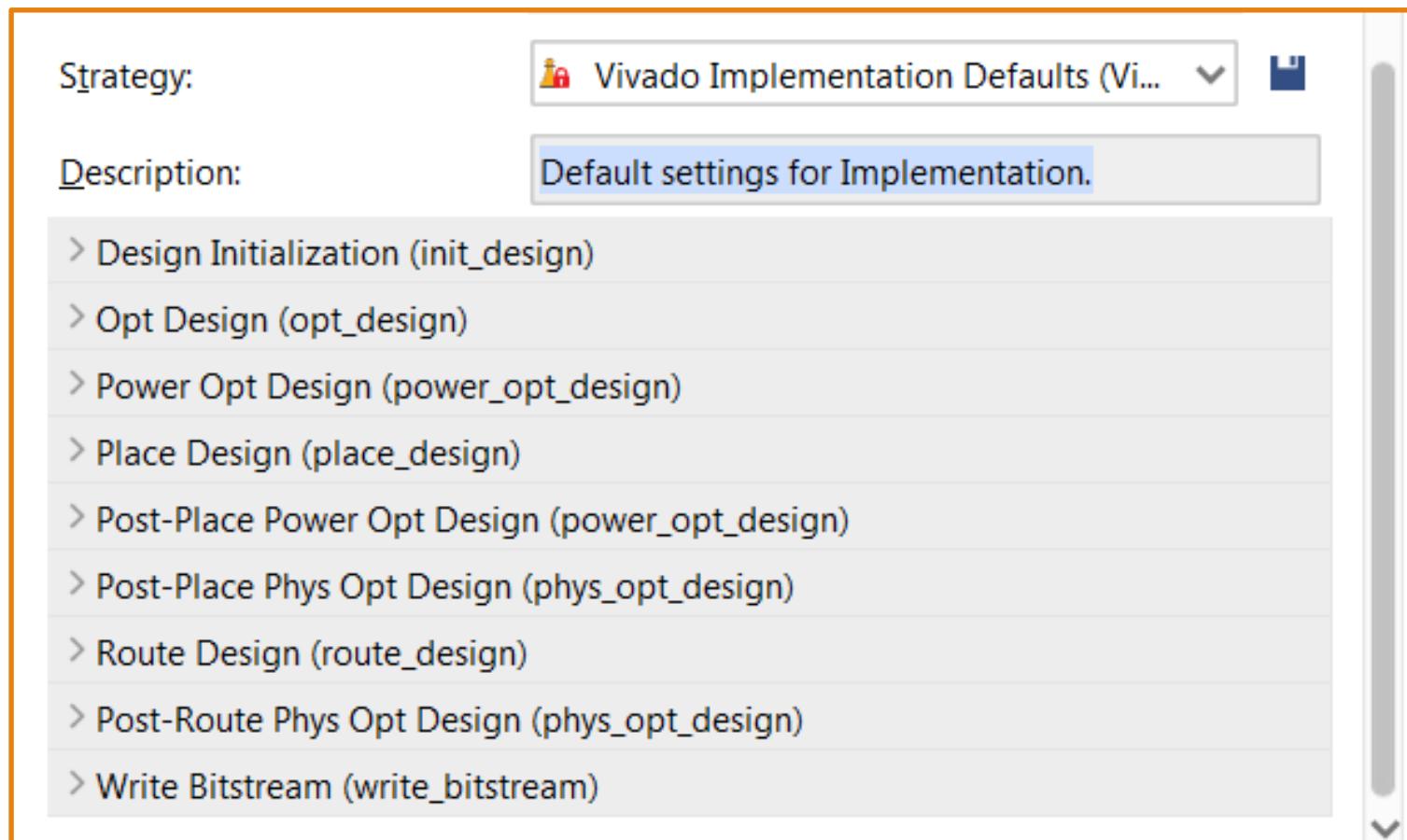
---

Po sintezės etapo vykdomas realizavimo žingsnis, kurio metu loginis projekto aprašas konvertuojamas į specifini formato failą, kuris gali būti nusiunčiamas į tikslinį įrenginį (angl. target device).

**Realizavimo žingsniai:**

- 1. Transliavimas** (Translate - merges the incoming netlists and constraints into a Xilinx® design file).
- 2. Išdėstymas** (Map - fits the design into the available resources on the target device, and optionally, places the design).
- 3. Talpinimas ir trasavimas** (Place and Route - places and routes the design to the timing constraints).
- 4. Konfigūracijos failo generavimas** (Generate Programming File - creates a bitstream file that can be downloaded to the device)

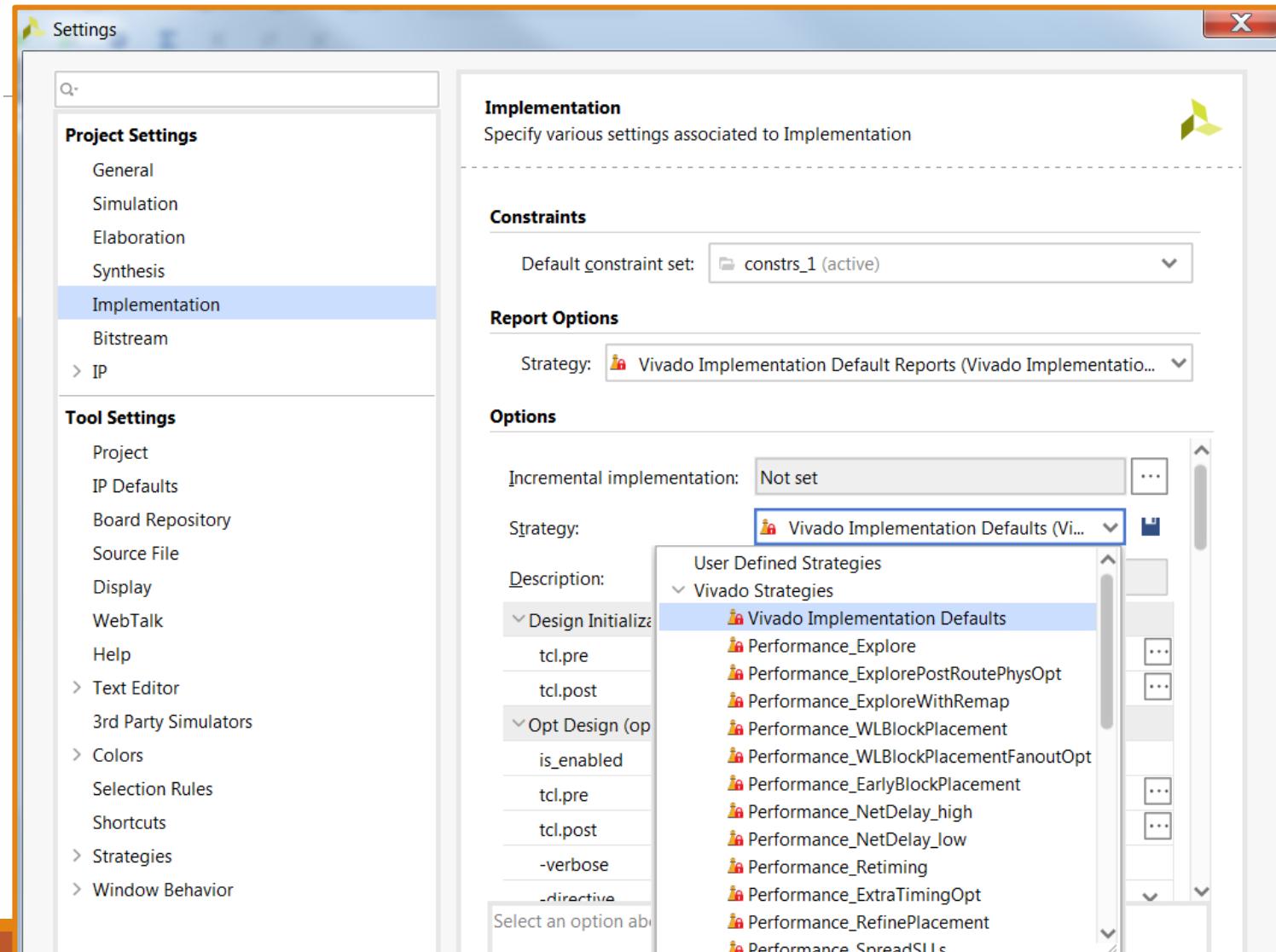
# Vivado realizavimo žingsniai



The screenshot shows the 'Vivado Implementation Defaults' dialog box. The 'Strategy' dropdown is set to 'Vivado Implementation Defaults (Vi...)' with a lock icon. The 'Description' field contains the placeholder text 'Default settings for Implementation.' Below these fields is a list of nine expandable items:

- > Design Initialization (init\_design)
- > Opt Design (opt\_design)
- > Power Opt Design (power\_opt\_design)
- > Place Design (place\_design)
- > Post-Place Power Opt Design (power\_opt\_design)
- > Post-Place Phys Opt Design (phys\_opt\_design)
- > Route Design (route\_design)
- > Post-Route Phys Opt Design (phys\_opt\_design)
- > Write Bitstream (write\_bitstream)

# Realizavimo strategijos (optimizavimo kriterijai) Vivado aplinkoje



# Realizavimo ataskaitų analizė (angl. *Implementation analysis*)

---

Po realizavimo etapo galima analizuoti projektą našumo, resursų panaudojimo, spartos, galios suvartojimo požiūriais.

Realizavimo analizės būdai:

- ❑ Realizavimo ataskaitų peržiūra (pranešimai, resursų panaudojimo ataskaitos ir t.t.)
- ❑ Laikinės analizės rezultatų peržiūra
- ❑ Suvartojamos galios modeliavimas
- ❑ ChipScope Pro tool įrankio panaudojimas testavimui po konfigūracijos nusiuntimo į fizinį mikrograndyną (FPGA)

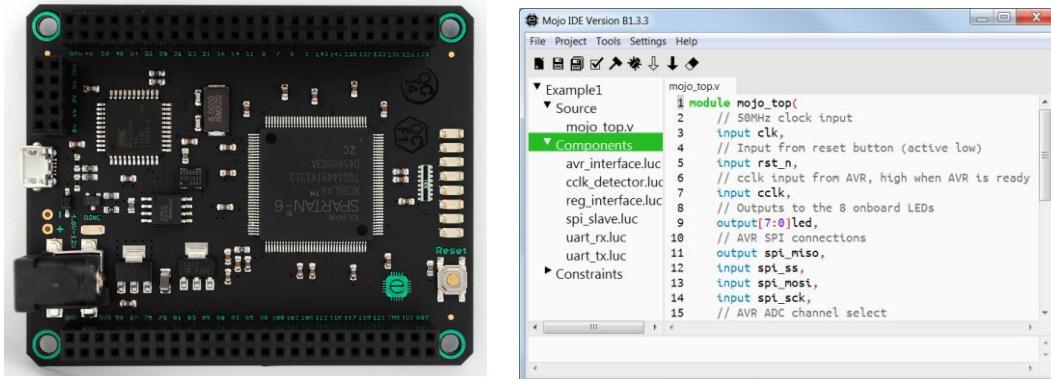
# Resursų panaudojimo ataskaitos pavyzdys

The screenshot shows the Utilization report in the Xilinx Vivado interface. The left sidebar displays a hierarchical tree of design components, with the top node 'mydesign\_wrapper' expanded to show its sub-components: 'mydesign\_i' (mydesign), 'SRoot' (mydesign\_cord), and 'U0' (mydesign\_cord). The main pane displays utilization statistics for these components across various resources. A red box highlights the first row of the utilization table, which corresponds to the 'mydesign\_wrapper' component.

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IPADs (10)	BUFI0 (20)
mydesign_wrapper	50	56	27	47	3	56	19	1
mydesign_i (mydesign)	50	56	27	47	3	0	0	0
SRoot (mydesign_cord)	50	56	27	47	3	0	0	0
U0 (mydesign_cord)	50	56	27	47	3	0	0	0

# FPGA mokymosi platformos, iniciatyvos

Mojo projektas <https://embeddedmicro.com/tutorials/mojo>



Papillo projektas <https://www.sparkfun.com/products/retired/11838>



# FPGA maketyų gamintojai

---

- ❑ Digilent Inc. <https://store.digilentinc.com/>
- ❑ Terrasic Inc. <https://www.terasic.com.tw/en/> (Intel/ Altera partneris)

# Xilinx įrankiai projektavimui platesniame kontekste

- ❑ Programinės įrangos
- ❑ Aparatinės įrangos
- ❑ Sisteminio lygmens
- ❑ Akseleratoriai ir kt.

Software Zone	Hardware Zone	System Zone	Acceleration Zone
SDAccel Development Environment	Vivado Design Suite - HLx Editions	MathWorks	<b>Accelerated Cloud Services</b>
SDSoC Development Environment	ISE Design Suite	National Instruments	<b>reVISION Zone</b>
SDNet Development Environment	UltraFast Design Methodology		<b>Machine Learning</b>
Embedded Development	Intellectual Property		
	Boards, Kits, and Modules		

# Diskrečiujų schemų sintezės elementai

*(modulis T170B114)*

2022

Kauno technologijos universitetas  
Elektronikos inžinerijos katedra

Prof. dr. Žilvinas Nakutis

# Apie ką turėtumėte sužinoti ir išmokti

- Mintermus ir maxtermus
- Sandaugų sumos ir sumų sandaugos loginių schemų formas
- Karno (Carnough maps, K-maps) diagramų metodo naudojimą sintezei
- Espresso algoritmą

# Sintezės uždavinys

- Transliuoti (pakeisti) aukštesnio abstrakcijos lygmens (pvz., algoritminj) aprašą į žemesnio lygmens aprašą (pvz., loginius ventilius)
- Loginė sintezė – RTL aprašo (VHDL, Verilog) transformavimas į ventilius

# Loginio įrenginio aprašų įvairovė

## What is the clearest way to describe a circuit

### Differential encoding

#### 1. In words:

If both inputs are 1, change both outputs.

If one input is 1 change an output as follows:

    If the previous outputs are equal

        change the output with input 0;

    If the previous outputs are unequal  
        change the output with input 1.

If both inputs are 0, change nothing.

#### 2. With equations:

$$I_{out} = \overline{I \oplus Q} (I \oplus I_{prev} + (I \oplus Q)(Q \oplus Q_{prev})$$

$$Q_{out} = \overline{I \oplus Q} (Q \oplus Q_{prev} + (I \oplus Q)(I \oplus I_{prev})$$

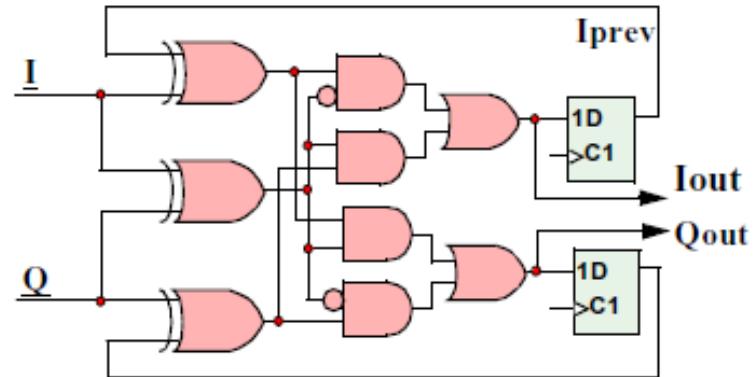
#### 3. By a schematic; does not give much insight.

#### 4. By tables; two forms are shown.

    An output vs. input table.

    An output-change vs. input table.

Which representation would you choose?



		(I,Q)prev			
		00	01	11	10
IQ	00	00	01	11	10
	01	10	00	01	11
11	11	10	00	01	
	10	01	11	10	00

Iout,Qout

		(I,Q)prev	
		=	≠
IQ	00	...	...
	01	Δ -	- Δ
11	Δ Δ	Δ Δ	
	10	- Δ	Δ -

ΔIout,ΔQout

# Sintezės uždavinys

Skaitmeninės logikos sintezė – tai loginės grandinės sukūrimas pagal užduotą aprašą loginėmis lygtimis arba teisingumo lentele.

## Užduotis:

sukurkite įrenginio grandinę,  
duotai teisingumo lentelėi

*(Digital design with CPLD  
applications and VHDL, 2000  
psl. 68).*

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

# Sandaugų sumos (*angl. Sum-of-Products*) ir sumų sandaugos (*angl. Product-of-Sums*) formos

- **Sandaugos narys** (*angl. Product term*) narys loginėje išraiškoje, kuriame logiškai sudauginami vienas ar daugiau kintamujų (signalų), pvz., A & B.
- **Minterm narys** (*angl. Minterm*) - tai sandaugos narys, kuriame sudauginami visi galimi kintamieji arba jų invertuoti nariai
- **Sumos narys** (*angl. Sum term*) narys loginėje išraiškoje, kuriame logiškai sumuojami vienas ar daugiau kintamujų (signalų), pvz., A | B | D
- **Maxterm narys** (*angl. Maxterm*) tai sumos narys, kuriame sumuojami visi galimi kintamieji arba jų invertuoti nariai
- **Sandaugų suma** (*angl. Sum-of-products (SOP)*), pvz.,  
$$(A \& B \& C) + (!A \& !B \& C) + (A \& B \& !C)$$
- **Sumų sandauga** (*angl. Product-of-sums (POS)*) pvz.,  
$$(A+B+C) \& (A+!B+!C) \& (A+B+!C)$$

# Sandaugų sumos forma (SOP)

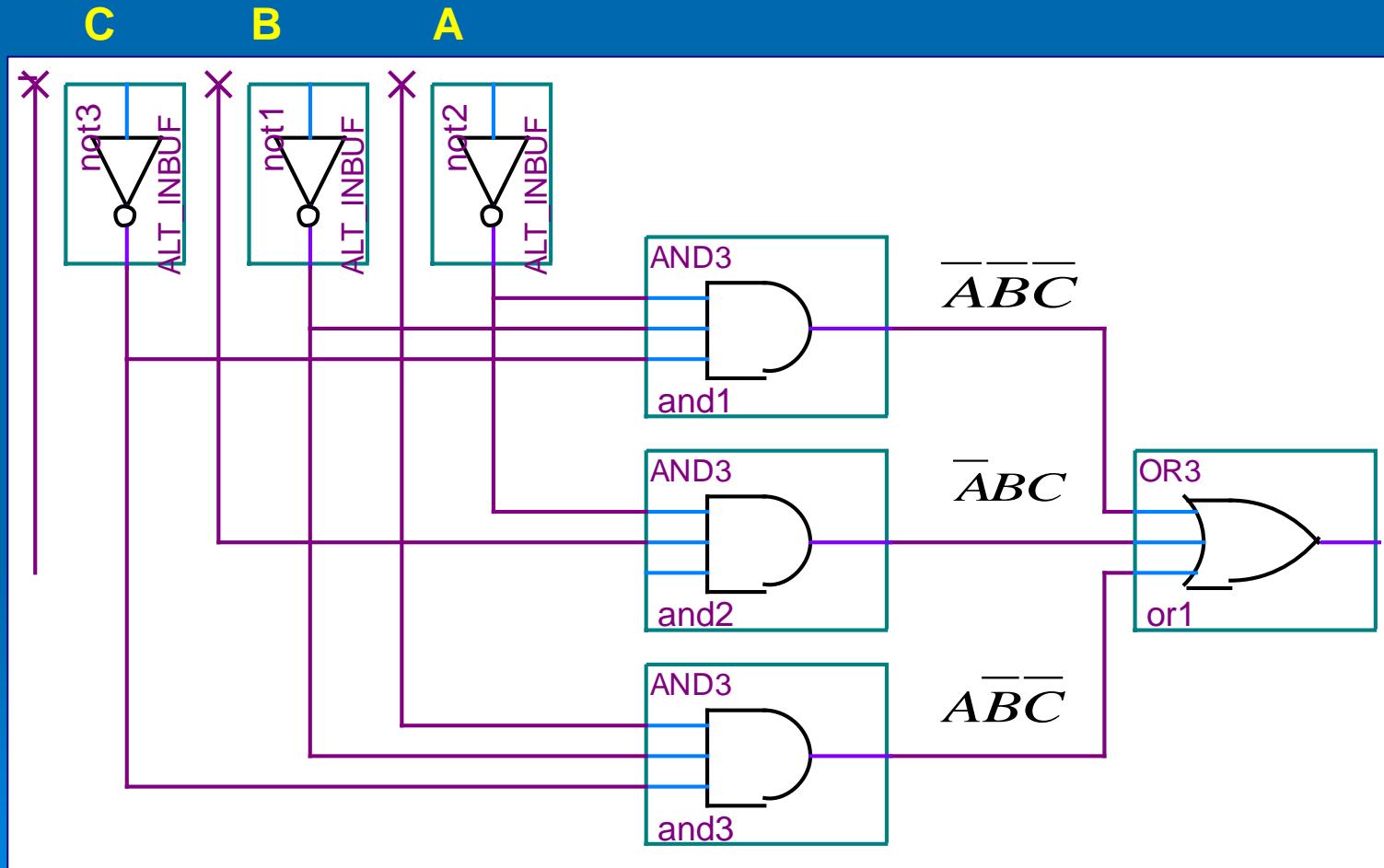
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Mintermai Y=1

1.  $\bar{A} \bar{B} \bar{C}$
2.  $\bar{A} B C$
3.  $A \bar{B} \bar{C}$

$$Y = \bar{A} \bar{B} \bar{C} + \bar{A} B C + A \bar{B} \bar{C}$$

# Sandaugų sumos formos schema



# Užduotis: XOR ir NXOR elementų sintezė

$A$	$B$	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

$A$	$B$	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

# Sumų sandaugos formos (POS)

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Mintermai !Y=1

$$\begin{array}{l} \overline{A} \overline{B} C \\ \overline{A} B \overline{C} \\ A \overline{B} C \\ A B \overline{C} \\ A B C \end{array}$$

$$\overline{Y} = \overline{A} \overline{B} C + \overline{A} B \overline{C} + A \overline{B} C + A B \overline{C} + A B C$$

Abiejų pusių invertavimas ir De Morgano taisyklė

$$\begin{aligned} \overline{x + y + z} &= \overline{x} \overline{y} \overline{z} \\ \overline{xyz} &= \overline{x} + \overline{y} + \overline{z} \end{aligned}$$

$$Y = (A + B + \overline{C})(A + \overline{B} + C)(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C})$$

Pasiūlykite POS formos lygties sudarymo algoritmą tiesiai iš teisingumo lentelės

# SOP ir POS formų sudarymo apibendrinimas

A	B	C	D	Y	Minterms	Maxterms
0	0	0	0	1	$\overline{A} \overline{B} \overline{C} \overline{D}$	
0	0	0	1	1	$\overline{A} \overline{B} \overline{C} D$	
0	0	1	0	0		$A + B + \overline{C} + D$
0	0	1	1	1	$\overline{A} \overline{B} C D$	
0	1	0	0	0		$A + \overline{B} + C + D$
0	1	0	1	0		$A + \overline{B} + C + \overline{D}$
0	1	1	0	0		$A + \overline{B} + \overline{C} + D$
0	1	1	1	0		$A + \overline{B} + \overline{C} + \overline{D}$
1	0	0	0	1	$A \overline{B} \overline{C} \overline{D}$	
1	0	0	1	0		$\overline{A} + B + C + \overline{D}$
1	0	1	0	1	$A \overline{B} C \overline{D}$	
1	0	1	1	0		$\overline{A} + B + \overline{C} + \overline{D}$
1	1	0	0	1	$A B \overline{C} \overline{D}$	
1	1	0	1	1	$A B \overline{C} D$	
1	1	1	0	1	$A B C \overline{D}$	
1	1	1	1	0		$\overline{A} + \overline{B} + \overline{C} + \overline{D}$

SOP form:

$$Y = \overline{A} \overline{B} \overline{C} \overline{D} + \overline{A} \overline{B} \overline{C} D + \overline{A} \overline{B} C \overline{D} + A \overline{B} \overline{C} \overline{D} + A \overline{B} C \overline{D} + A B \overline{C} \overline{D} \\ + A B \overline{C} D + A B C \overline{D}$$

POS form:

$$Y = (A + B + \overline{C} + D)(A + \overline{B} + C + D)(A + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + \overline{C} + D) \\ (A + \overline{B} + \overline{C} + \overline{D})(A + B + C + D)(A + B + \overline{C} + D) \\ (A + B + \overline{C} + D)$$

Reikia  
prastinti

# Bulio algebros savybės

Bulio algebros žinojimas leidžia suprastinti SOP ir POS formas bei minimizuoti reikiama ventilių skaičių.

- Komutatyvumas:  
 $A \& B = B \& A$ ,  $A + B = B + A$
- Asociatyvumas:  
 $(A \& B) \& C = A \& (B \& C)$ ,  
 $(A + B) + C = A + (B + C)$ ,
- Distributyvumas:  
 $A \& (B + C) = A \& B + A \& C$ .

Užduotis: Patikrinkite XOR ( $\wedge$ ) funkcijos asociatyvumą.

# Keleto kintamuųjų Bulio algebros teoremos

- De Morgano teoremos:

$$\begin{aligned}\overline{xy} &= \overline{x} + \overline{y} \\ \overline{x + y} &= \overline{\overline{x} + \overline{y}}\end{aligned}$$

- Kita teorema:

$$x + xy = x$$

$$\begin{aligned}x + xy &= x(1 + y) \quad (\text{Distributive property}) \\ &= x \cdot 1 \quad (1 + y = 1; \text{Theorem}) \\ &= x\end{aligned}$$

# Bulio (bitinės) algebros teoremos

AND identity function	$A \cdot 1 = A$
OR identity function	$A + 0 = A$
Output reset	$A \cdot 0 = 0$
Output set	$A + 1 = 1$
Identity law	$A = A$
AND complementary law	$A \cdot !A = 0$
OR complementary law	$A + !A = 1$
AND idempotent law	$A \cdot A = A$
OR idempotent law	$A + A = A$
AND commutative law	$A \cdot B = B \cdot A$
OR commutative law	$A + B = B + A$
AND associative law	$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$
OR associative law	$(A + B) + C = A + (B + C) = A + B + C$
AND distributive law	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
OR distributive law	$A + (B \cdot C) = (A + B) \cdot (A + C)$
De Morgan's NOR theorem	$!(A + B) = !A \cdot !B$
De Morgan's NAND theorem	$!(A \cdot B) = !A + !B$

# Formų prastinimas, panaudojant Karno diagramas

- Tam, kad iš teisingumo lentelės optimaliai generuoti loginę funkciją, patogiau naudotis **Karno diagramų** metodu.
- Karno (pagal prancūzų matematiką *Maurice Karnaugh*) diagrama (angl. *Karnaugh map* arba *K-map*, *KV-map* (pagal kitą atradėją *Edward W. Veitch*)) – tai kitas teisingumo lentelės pavidalas, kuriame kiekvienas loginių kintamujų derinys – teisingumo lentelės eilutė arba mintermas - vaizduojamas kvadrateliu.
- *Edward W. Veitch* šį metodą panaudojo 1952, o *Maurice Karnaugh* – 1953 metais.
- Karno diagramos sudaromos taip, kad gretimuose jos kvadrateliuose skirtusi tik vieno loginio kintamojo reikšmė, o visų kitų sutaptų.

Predko, Myke. *Digital Electronics Demystified*, McGraw-Hill, 2004 (Ebrary bibliotekoje)

<http://www.facstaff.bucknell.edu/mastascu/eLessonsHTML/Logic/Logic3.html>

# Key terms

- **Karnaugh map** . A graphical tool for finding the maximum SOP or POS simplification of a Boolean expression. A Karnaugh map works by arranging the terms of an expression in such a way that variables can be canceled by grouping minterms or maxterms.
- **Cell**. The smallest unit of a Karnaugh map, corresponding to one line of a truth table. The input variables are the cell's coordinates, and the output variable is the cell's contents.
- **Adjacent cell** . Two cells are adjacent if there is only one variable that is different between the coordinates of the two cells. For example, the cells for minterms  $ABC$  and  $A\bar{B}C$  are adjacent.
- **Pair** . A group of two adjacent cells in a Karnaugh map. A pair cancels one variable in a K-map simplification.
- **Quad** . A group of four adjacent cells in a Karnaugh map. A quad cancels two variables in a K-map simplification.
- **Octet** . A group of eight adjacent cells in a Karnaugh map. An octet cancels three variables in a K-map simplification.

# Karno diagrammos sudarymas

Pradinė funkcija

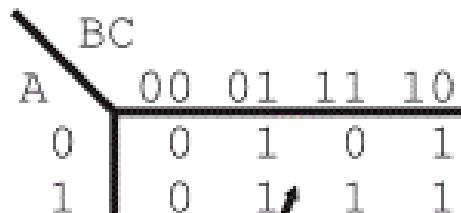
$$\begin{aligned} \text{Output} = & (\bar{A} \cdot \bar{B} \cdot C) + (\bar{A} \cdot B \cdot \bar{C}) \\ & + (A \cdot \bar{B} \cdot \bar{C}) + (A \cdot B \cdot C) \\ & + (A \cdot \bar{B} \cdot C) \end{aligned}$$

Original Truth Table

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	1	0
0	1	0	1
1	1	0	1
1	1	1	1
1	0	1	1
1	0	0	0

Column to Strip Out

Equivalent Karnaugh Map



Outputs in a  
2 Dimensional  
Field

# Karno diagramos sudarymo taisyklė

- Gretimos diagramos stulpelius ir eilutes atitinkantys jėjimo signalai privalo skirtis tik vieno bitu vertė.

A \ BC	00	01	11	01
0	0	1	0	1
1	0	1	1	1

Teisingas pavyzdys

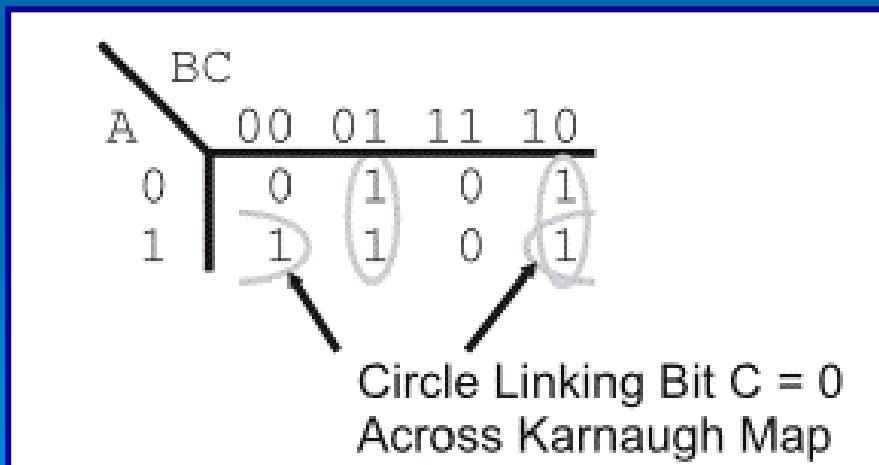
A \ BC	00	01	10	11
0	0	1	1	0
1	0	1	1	1

Skiriasi dviejų bitų vertės

# Kontūrų (circle) išskyrimo taisyklės

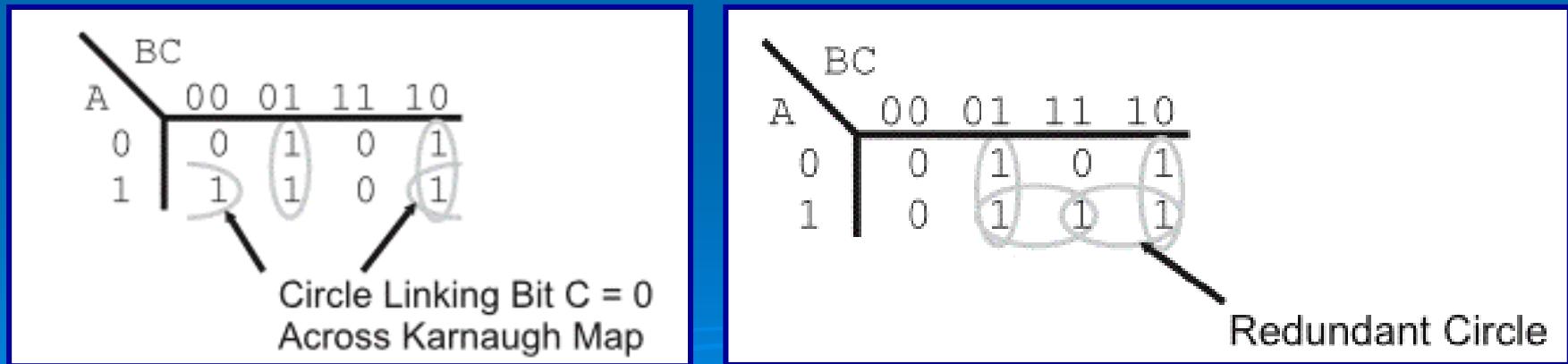
- Į vieną kontūrą galima apjungti tik kartotinį 2 laipsnyje gretimų bitų skaičių (pvz., negalima apjungti 3-jų bitų). Taigi galima apibraukti 1, 2, 4, 8 ir t.t. lentelės langelius

Pastaba: Gretimi langeliai yra ir tie, kurie yra priešingose diagramos pusėse.



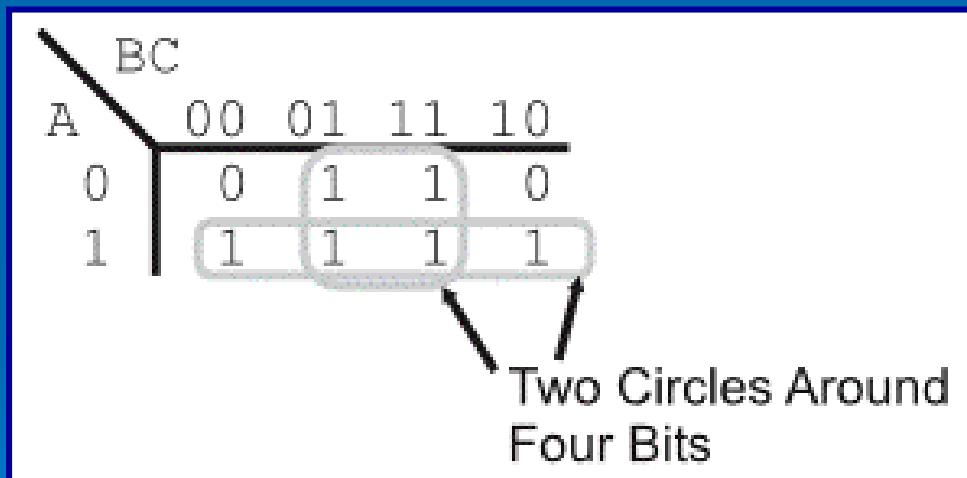
# Kontūrų išskyrimo taisyklės

2. Kontūrai gali kirstis, t.y. tas pats bitas gali papulti į du ar daugiau kontūrų. Jeigu egzistuoja kontūras, kuris apima tik bitus, jau patenkančius į kitus kontūrus, tai toks kontūras yra perteklinis (*angl. redundant*)



# Kontūrų išskyrimo taisyklės

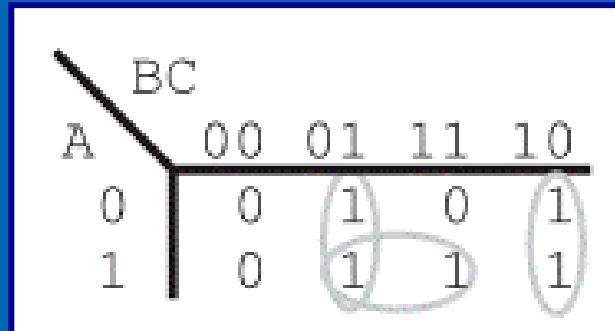
3. Apibraukiame ne tik po du, bet ir  $2^n$  bitų kontūrus



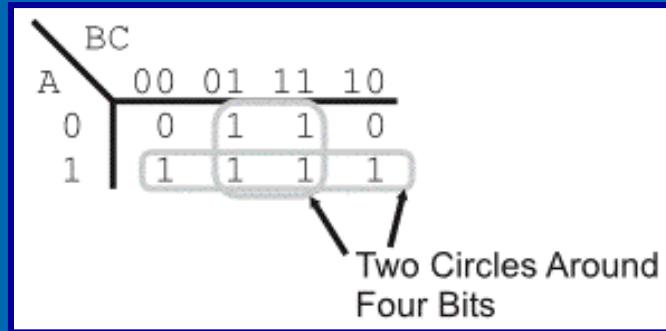
# Iš Karno diagramos užrašome optimizuotą lygtį

- Loginė funkcija užrašoma sandaugų suma (SOP), kurios kiekvienas dėmuo-sandauga atitinka vieną apibrėžtą kontūrą.
- I sandaugą, atitinkančią kontūrą, rašomi tik tie loginiai kintamieji, kurių reikšmės visuose tuo kontūru apibrėžtuose kvadrateliuose yra vienodos – lygios arba 0 arba 1.
- Jei ta reikšmė yra lygi nuliui, j sandaugą išrašoma invertuota kintamojo reikšmė, jei vienetui – tiesioginė.

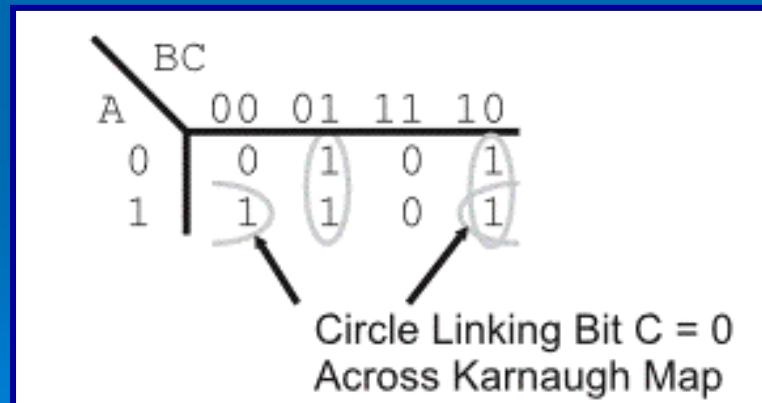
# Iš Karko diagrammos užrašome optimizuotą lygtį



$$Y = (\neg B \cdot C) + (B \cdot \neg C) + (A \cdot C)$$



$$Y = A + C$$



# Karno diagrammos panaudojimo pavyzdys

P	D	W1	W2	Alarm Response
0	0	0	0	
0	0	0	1	
0	0	1	1	
0	0	1	0	Sound Alarm
0	1	1	0	Sound Alarm
0	1	1	1	
0	1	0	1	
0	1	0	0	
1	1	0	0	
1	1	0	1	Sound Alarm
1	1	1	1	Sound Alarm
1	1	1	0	Sound Alarm
1	0	1	0	Sound Alarm
1	0	1	1	Sound Alarm
1	0	0	1	Sound Alarm
1	0	0	0	

→

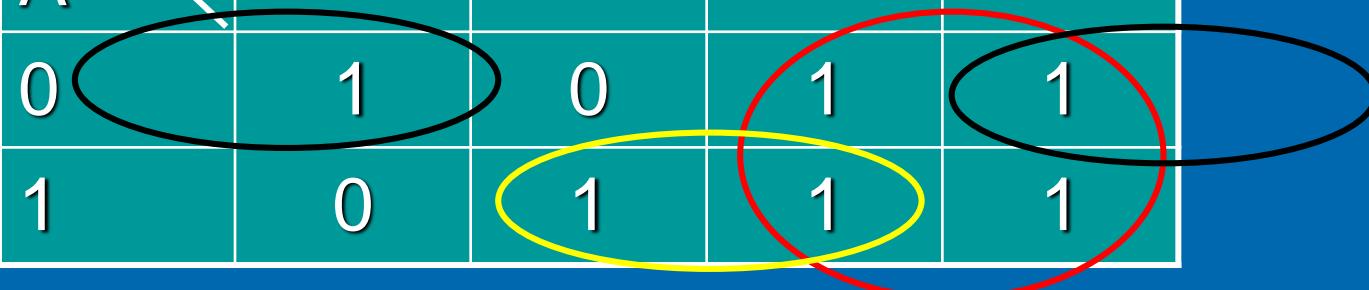
Alarm Karnaugh Map

PD	W1W2			
	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	0	1	1	1
10	0	1	1	1

$$\text{Alarm Response} = (P \cdot W2) + (W1 \cdot !W2)$$

# Užduotis Nr.1 su Karno diagramomis

		BC	00	01	11	10
		A	1	0	1	1
0	0	1	0	1	1	1
	1	0	1	1	1	1



# Užduotis Nr.2 su Karno diagramomis

CD \\	00	01	11	10
AB \\	00			
00			1	1
01	1		1	1
11	1		1	1
10		1	1	

# Karno diagramų taikymas funkcijoms su nereikšmingomis loginių kintamujų kombinacijomis (don't care output)

BC		00	01	11	10
A		0	0	1	0
0	1	0	0	1	0
1	1	1	1	1	X

$$Y = (A^*!B) + (B^*C)$$

Kai apjungiami tik vienetai

$$Y = (B^*C) + A$$

Kai apjungiamos ir nereikšmingos būsenos

Pastaba: abi lygtys teisingos

# Karno diagramų trūkumai

- Naudojama dažniausiai rankiniam optimizavimui, nes sunkiai kompiuterizuojasi
- Dirbant rankiniu būdu galima greitai pridaryti klaidą
- Tinka iki 6 įėjimo kintamujų, o realiai iki 4

# Kiti loginių funkcijų minimizavimo algoritmai

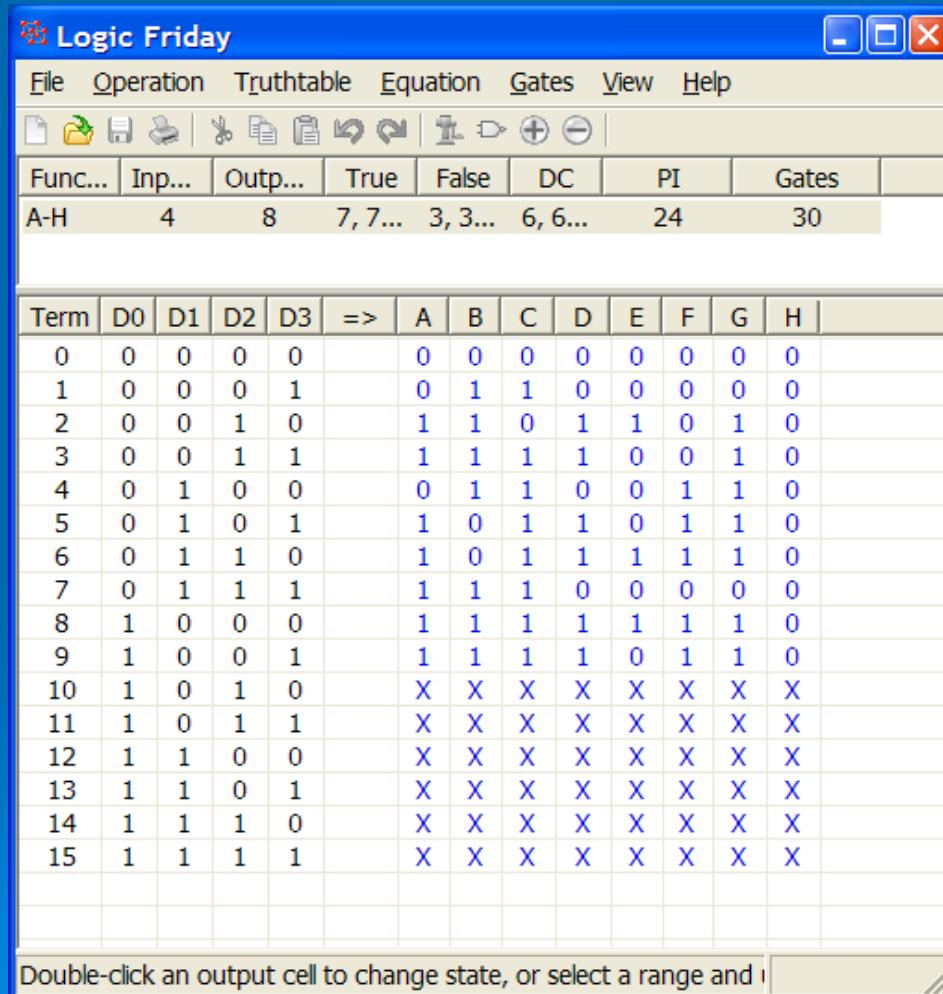
- Quine–McCluskey algoritmas ([http://en.wikipedia.org/wiki/Quine-McCluskey\\_algorithm](http://en.wikipedia.org/wiki/Quine-McCluskey_algorithm)):
  - Formaliai patogiau aprašomas matematiškai, kad galėtų būti realizuotas kompiuterine programa
  - Tinka nedideliam jėjimų/išėjimų skaičiui, nes skaičiavimų apimtys auga eksponentiškai didėjant jėjimo/išėjimo skaičiui
- Espresso heuristinis loginis minimizatorius - de-facto standartas

# Espresso heuristinis loginis minimizatorius



- Sukurtas Brayton'o Berkeley universitete apie 1980 metus
- Tai jau kompiuterinė programa, naudojanti heuristinius minimizavimo algoritmus
- Minimizavimo rezultatas ne visada garantuotai optimalus
- Lyginant su kitais algoritmais skaičiavimų apimtys daug mažesnės
- Praktiškai nėra ribojimų jėjimų/išėjimų skaičiui. Išspresti uždaviniai su keliomis dešimtimis in/out
- Sintezuoja schemą, atsižvelgiant į nurodytus bazinius ventilius (angl. *map to available gates*), todėl gerai tinka CPLD ir FPGA sintezei
- Algoritmas turi daug atmainų
- Naudojama daugelyje sintezės programų

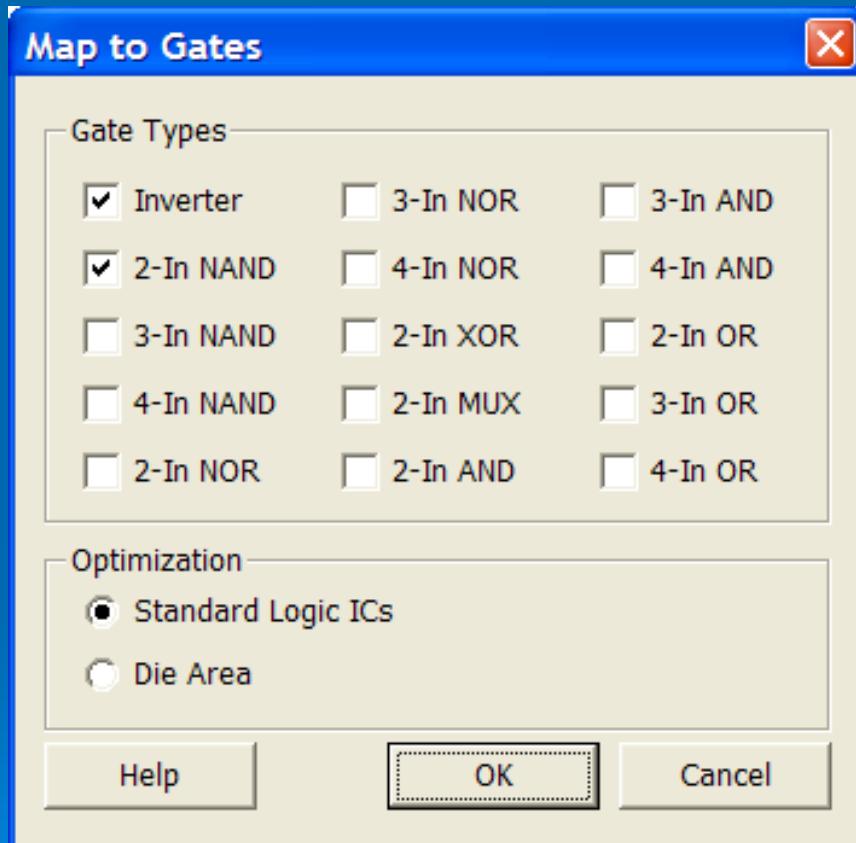
# Logic Friday – nemokama programa loginei sintezei Espresso algoritmo pagrindu



Pavyzdys: Teisingumo lentelė 7-ių segmentų indikatoriaus dešifratoriui

Download: <https://logic-friday.software.informer.com/1.1/>

# Bazinių elementų pasirinkimas



➤ Norime  
realizuoti su  
**2NAND** ir  
**NOT**  
elementais

# Sintezės rezultatas (30 ventilių)

Logic Friday

File Operation Truthtable Equation Gates View Help

Function Inputs Outputs True False DC PI Gates

A-H 4 8 7, 7, 8, 6, 3, 5, 7, 0 3, 3, 2, 4, 7, 5, 3, 10 6, 6, 6, 6, 6, 6, 6, ... 24 30

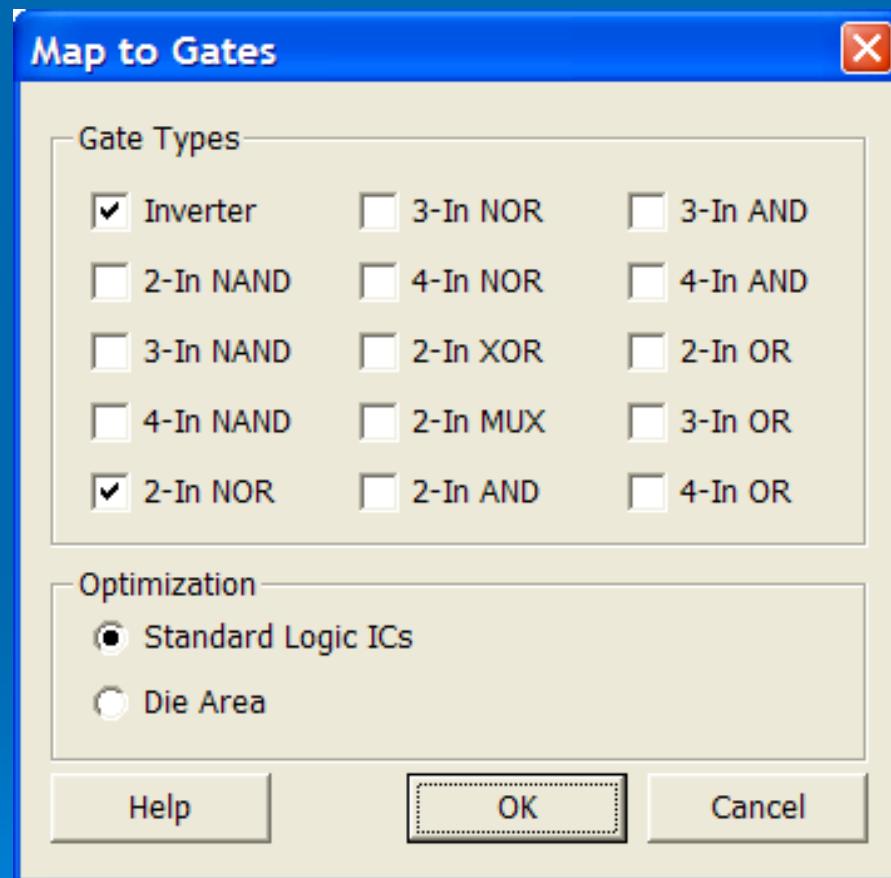
D0	D1	D2	D3	=>	A	B	C
0	0	0	1		1	1	
0	0	1	0		1	1	
0	0	1	1		1	1	1
0	1	0	0		1	1	
0	1	0	1		1	1	
0	1	1	0		1	1	
0	1	1	1		1	1	1
1	0	0	0		1	1	1
1	0	0	1		1	1	1
1	0	1	0	X	X	X	
1	0	1	1	X	X	X	
1	1	0	0	X	X	X	
1	1	0	1	X	X	X	
1	1	1	0	X	X	X	
1	1	1	1	X	X	X	

```

E = D0' D1' D2 D3' + D0' D1 D2 D3' + D0 D1' D2' D3';
F = D0' D1 D2' D3' + D0' D1 D2' D3 + D0' D1 D2 D3' + D0 D1' D2' D3' + D0 D1' D2' D3;
G = D0' D1' D2 D3' + D0' D1' D2 D3 + D0' D1 D2' D3' + D0' D1 D2' D3 + D0' D1 D2 D3' + D0 D1' D2' D3;
D2' D3' + D0 D1' D2' D3;
H = 0;

```

# Bazinių elementų pasirinkimas



➤ Norime realizuoti su **2NOR** ir NOT elementais

# Sintezės rezultatas (36 ventiliai)

Logic Friday

File Operation Truthtable Equation Gates View Help

Func... Inp... Outp... True False DC PI Gates

A-H 4 8 7, 7... 3, 3... 6, 6... 24 36

D0	D1	D2	D3	=>	A	B
0	0	0	1		1	
0	0	1	0		1	1
0	0	1	1		1	1
0	1	0	0		1	
0	1	0	1		1	
0	1	1	0		1	
0	1	1	1		1	1
1	0	0	0		1	1
1	0	0	1		1	1
1	0	1	0	X	X	
1	0	1	1	X	X	
1	1	0	0	X	X	
1	1	0	1	X	X	
1	1	1	0	X	X	
1	1	1	1	X	X	

```
E = D0' D1 D2 D3' + D0 D1' D2' D3';
F = D0' D1' D2 D3' + D0' D1 D2' D3' + D0' D1 D2 D3 + D0' D1 D2' D3' + D0 D1' D2' D3' + D0 D1' D2' D3;
G = D0' D1' D2 D3' + D0' D1' D2 D3 + D0' D1 D2' D3' + D0' D1 D2' D3 + D0' D1 D2 D3' + D0 D1' D2' D3' + D0 D1' D2 D3;
H = 0;
```

Ready

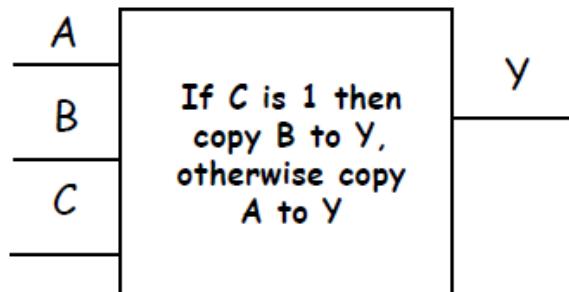
# Daugiapakopė (Multi-level) sintezė

- Pradžioje sintezės priemonės kuria daugiasluoksnį loginių lygčių tinklą iš RTL (tarpregistrių perdavimų lygmens) aprašo
- Sukurtas tinklas yra optimizuojamas nuo technologijos nepriklausančiomis priemonėmis
- Galiausiai, optimizuotos loginės lygtys transformuojamos į loginių ventilių grandinę. Vykdomas talpinimo (angl. *mapping*) uždavinys, kuris apsprendžiamas esamų ventilių (**IR**, ARBA, ir kt.), vėlinimų, suvartojomos galios, užimamo ploto ir t.t.

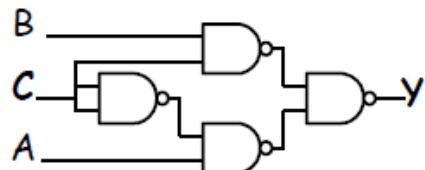
# Sintezé FPGA platformai

- Panaudojant multipleksorius
- Panaudojant peržiūros lenteles (angl. Look Up Tables)

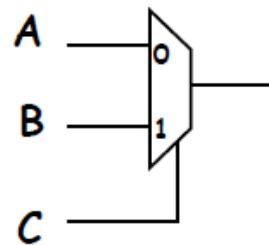
# Loginė sintezė, panaudojant multipleksorius (MUX)



2-input Multiplexer



schematic

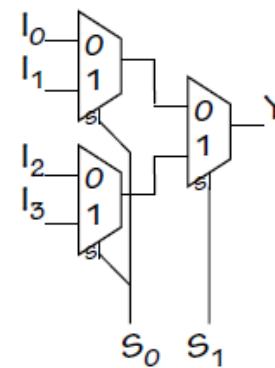


Gate  
symbol

Truth Table

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A 4-input Mux  
implemented as  
a tree



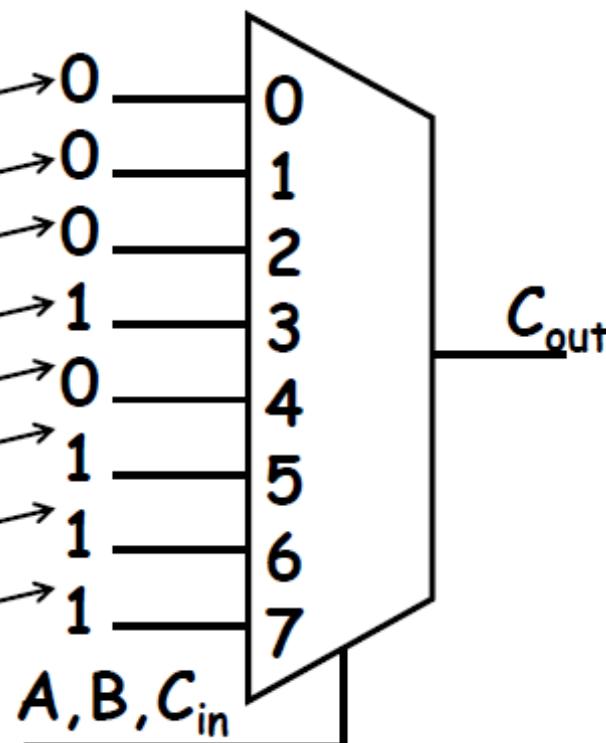
# Kombinacionės logikos realizavimas su MUX

Consider implementation of some arbitrary Boolean function,  $F(A,B)$

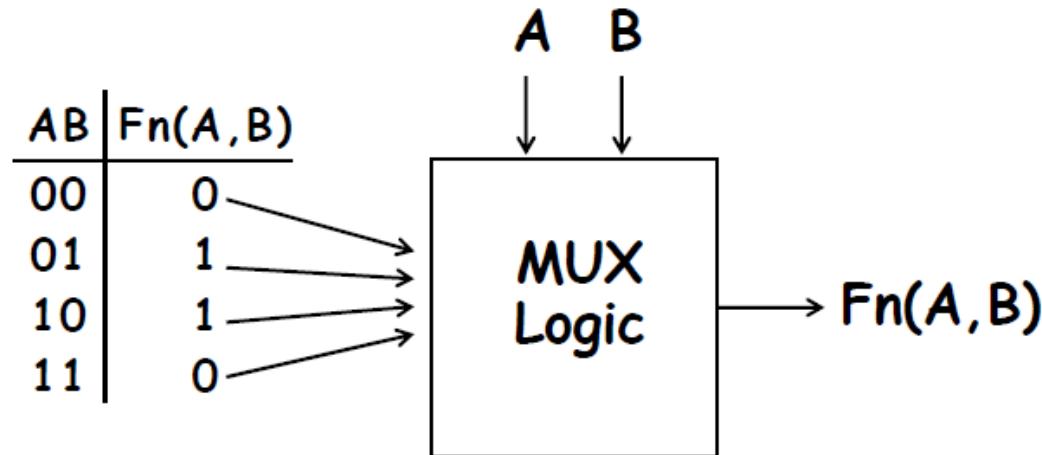
... using a MULTIPLEXER  
as the only circuit element:

A	B	$C_{in}$	$C_{out}$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Full-Adder  
Carry Out Logic



# Multipleksoriai yra universalūs



Generalizing:

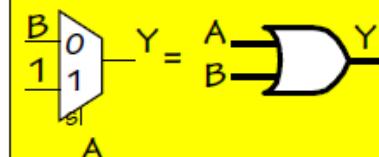
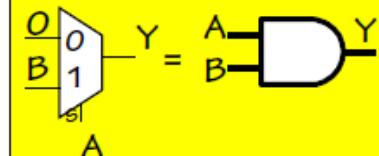
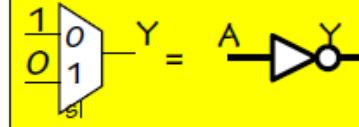
In theory, we can build any 1-output combinational logic block with multiplexers.

For an N-input function we need a  $2^N$  input mux.

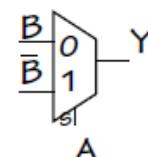
BIG Multiplexers?

How about 10-input function? 20-input?

Muxes are UNIVERSAL!

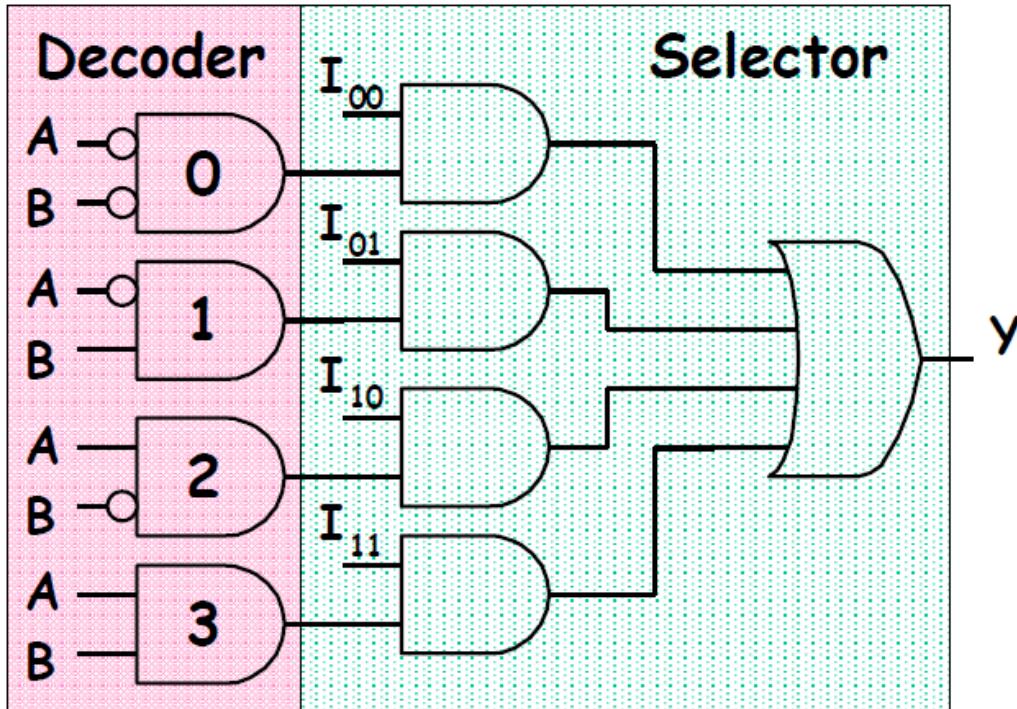


In future technologies  
muxes might be the  
“natural gate”.



# MUX apibendrinta struktūra

A decoder generates all possible product terms for a set of inputs



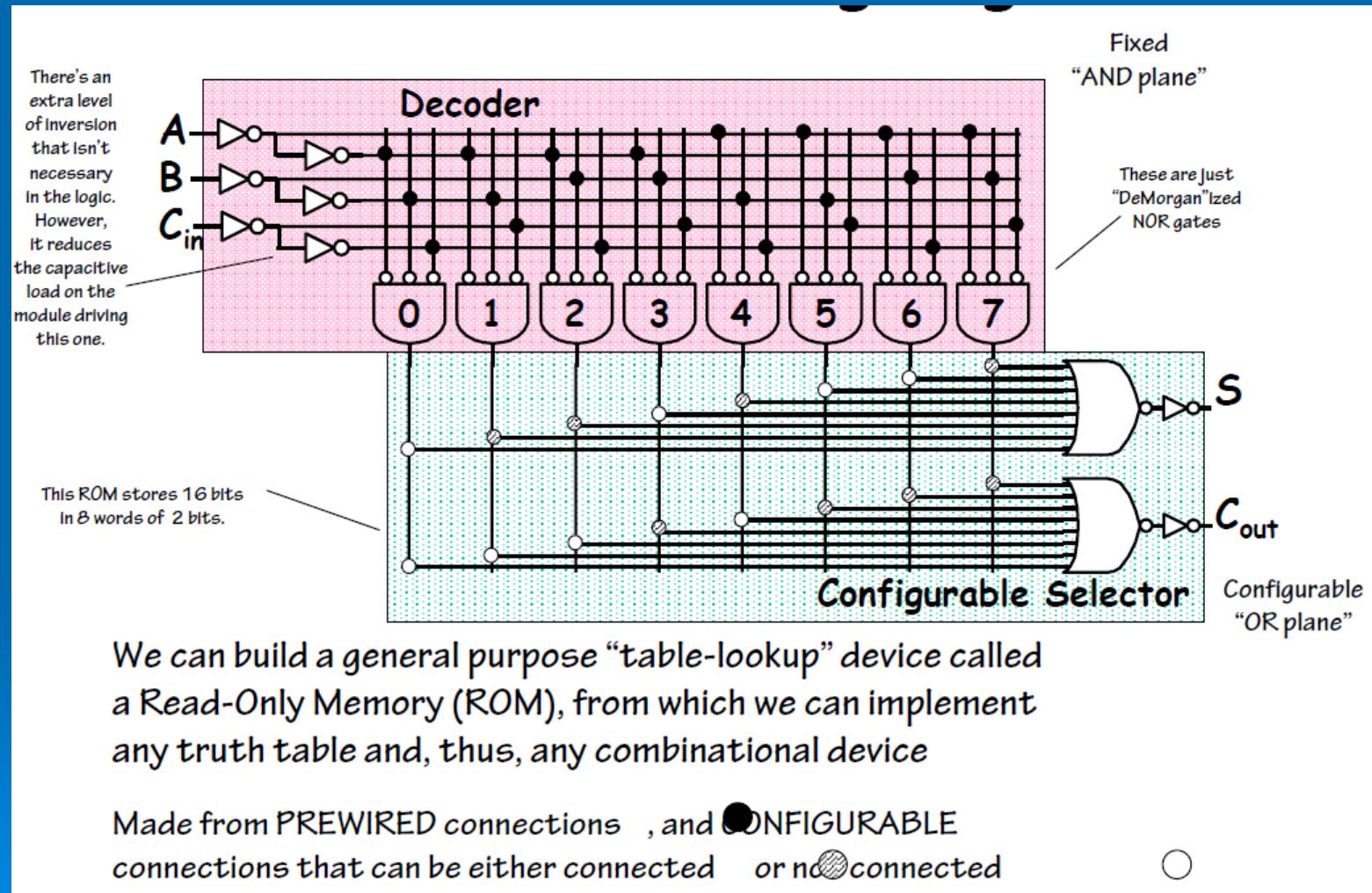
Multiplexers can be partitioned into two sections.

A DECODER that identifies the desired input, and

a SELECTOR that enables that input onto the output.

Hmmm, by sharing the decoder part of the logic MUXs could be adapted to make lookup tables with any number of outputs

# Paskirstyta dekoduojanti logika

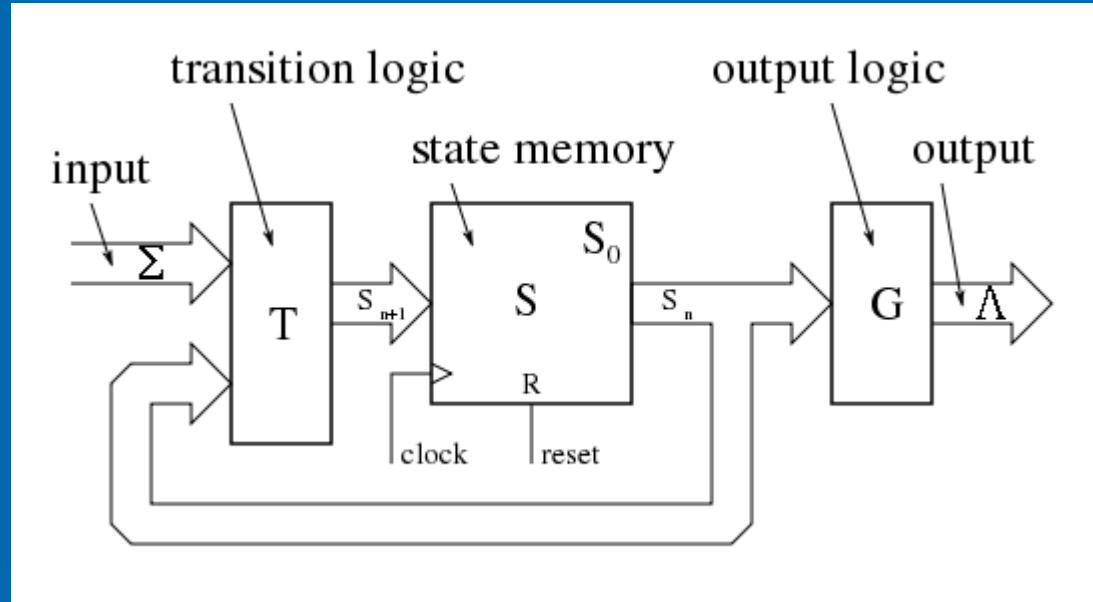


# Nuosekliosios logikos sintezė

## ➤ Naudojama automatų teorijos:

- **Milio (Mealy, George H. (September 1955).** "A Method for Synthesizing Sequential Circuits". *Bell Systems Technical Journal* 34: 1045–1079. )
- **Muro (Moore, Edward F (1956).** "Gedanken-experiments on Sequential Machines". *Automata Studies, Annals of Mathematical Studies* (Princeton, N.J.: Princeton University Press) (34): 129–153.)

# Muro automato (angl. FSM – Final State Machine) struktūra



➤ Palyginkite su CPLD makrocelės struktūra

# Formalus Muro automato aprašas

- Muro automatas aprašomas 6 parametru rinkiniu (  $S$ ,  $S_0$ ,  $\Sigma$ ,  $\Lambda$ ,  $T$ ,  $G$  ):
  - Baigtine būsenų aibe (  $S$  )
  - Pradine būsena  $S_0$  iš aibės (  $S$  )
  - Baigtine aibe, sudarančia jėjimų alfabetą (  $\Sigma$  )
  - Baigtine aibe, sudarančia išėjimų alfabetą(  $\Lambda$  )
  - Perėjimo (angl. transition) funkcija ( $T : S \times \Sigma \rightarrow S$ ), kuri transformuoja būseną ir jėjimų alfabetą į kitą būseną
  - Išėjimo (angl. output) funkcija ( $G : S \rightarrow \Lambda$ ), kuri transformuoja (angl. mapping) kiekvieną būseną į išėjimo alfabetą
- Muro automato būsenų skaičius visada būna ne mažesnis negu Milio automato būsenų skaičius tam pačiam įrenginiui aprašyti.

# **KAUNO TECHNOLOGIJOS UNIVERSITETAS**



**Elektros ir elektronikos fakultetas  
Elektronikos inžinerijos katedra**

## **VHDL kalbos pradmenys**

**Metodinė medžiaga  
Programuojami loginiai įrenginiai (T170B114)**

**Ž. Nakutis  
2024**

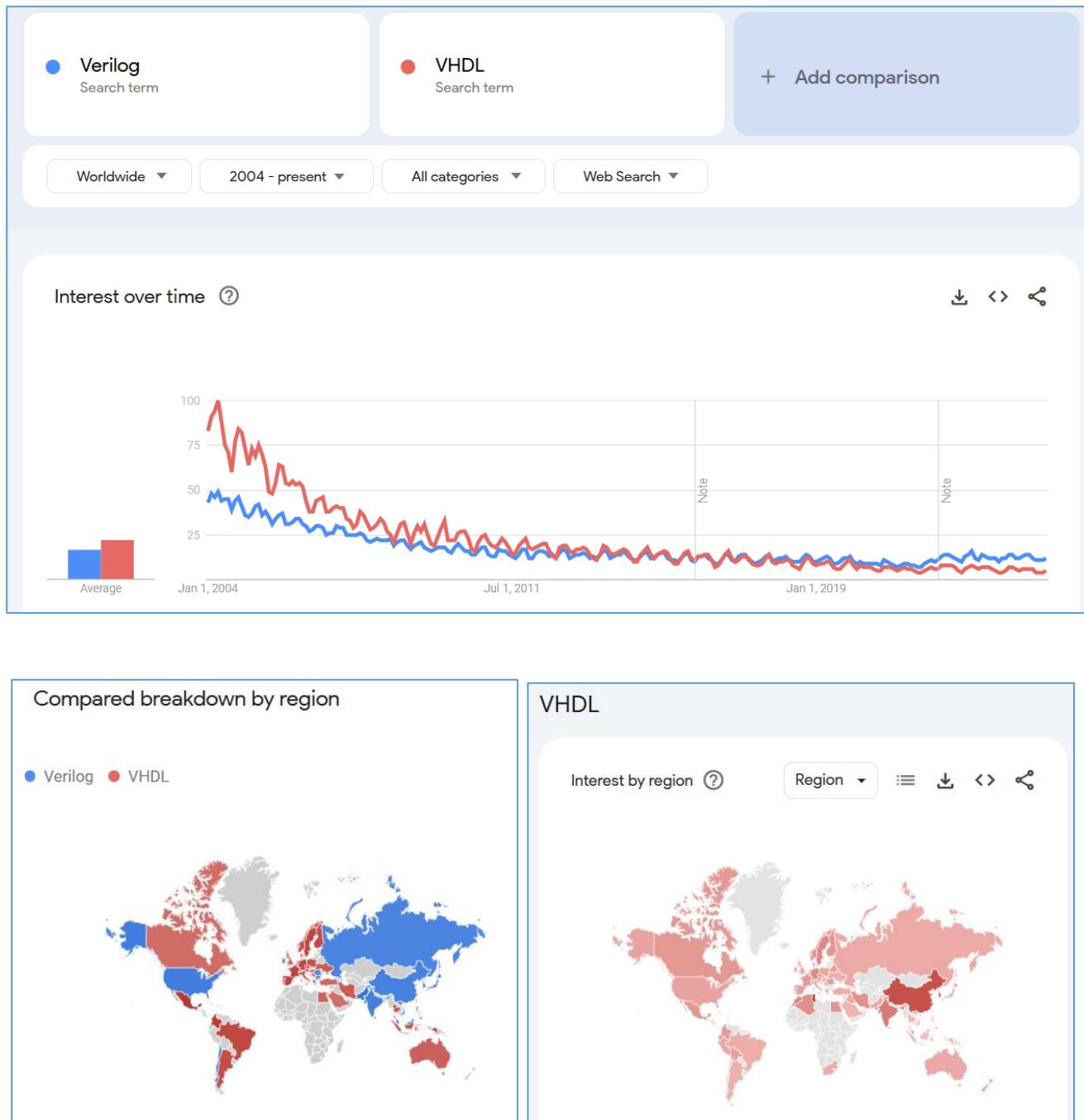
## Turinys

<b>1</b>	<b>KALBOS PASKIRTIS .....</b>	<b>4</b>
<b>2</b>	<b>PAGRINDINĖS KALBOS KONSTRUKCIJOS .....</b>	<b>5</b>
<b>3</b>	<b>DUOMENŲ TIPAI .....</b>	<b>7</b>
3.1	SKALIARINIAI TIPAI.....	7
3.1.1	<i>Sąrašo tipai .....</i>	7
3.1.2	<i>Skaitiniai tipai .....</i>	8
3.1.3	<i>Fiziniai tipai .....</i>	8
3.2	SUDÉTINIAI TIPAI .....	9
3.2.1	<i>Deklaravimas .....</i>	9
3.2.2	<i>Priskyrimas.....</i>	9
3.3	APIBRĖŽTI IR NEAPIBRĖŽTI IEEE 1164 TIPAI .....	10
3.4	TIPŲ KONVERTAVIMAS .....	11
3.5	DUOMENŲ TIPŲ ATRIBUTAI.....	13
<b>4</b>	<b>DUOMENŲ OBJEKTAI.....</b>	<b>13</b>
4.1	KONSTANTOS .....	14
4.2	KINTAMIEJI.....	14
4.3	SIGNALAI .....	14
<b>5</b>	<b>KALBOS KONSTRUKCIJOS .....</b>	<b>15</b>
5.1	PRISKYRIMAS .....	15
5.2	OPERACIJOS IR IŠRAIŠKOS .....	15
5.2.1	<i>Operatoriai.....</i>	15
5.2.2	<i>Konkatenacija.....</i>	17
5.2.3	<i>Redukavimas.....</i>	18
5.3	NUOSEKLIOSIOS KONSTRUKCIJOS .....	19
5.3.1	<i>Sąlyginis sakiny if .....</i>	19
5.3.2	<i>case ir case? sakiniai .....</i>	19
5.3.3	<i>Ciklai.....</i>	20
5.4	LYGIAGREČIOSIOS KONSTRUKCIJOS .....	23
5.4.1	<i>process sakiny.....</i>	23
5.4.2	<i>Lygiagretusis signalų priskyrimas.....</i>	24
5.4.3	<i>process saknio naudojimas sinchroninės logikos aprašymui .....</i>	25
5.5	BAIGTINIŲ BŪSENŲ AUTOMATŲ APRAŠYMAS .....	25
5.6	FUNKCIJOS IR PROCEDŪROS .....	27
<b>6</b>	<b>VHDL KALBOS NAUDOJIMAS SKAITMENINIŲ ĮRENGINIŲ MODELIAVIMUI .....</b>	<b>27</b>
6.1	VIRTUALIEJI BANDYMO STENDAI (TESTBENCH) .....	27
6.1.1	<i>Tik stimulų stendas .....</i>	28
6.1.2	<i>Pilnasis stendas .....</i>	31
6.2	SIMULIAVIMO TIKSLUMO NUSTATYMAS .....	34
6.3	MODELIAVIMUI SKIRTOS KALBOS KONSTRUKCIJOS .....	34
6.3.1	<i>Stimulų generavimo pavyzdžiai .....</i>	34
6.3.2	<i>Šablonų sąrašas Vivado aplinkoje.....</i>	38
6.4	VHDL SIMULATORIAI.....	38
<b>7</b>	<b>SINTEZUOJAMOS LOGIKOS APRAŠAI .....</b>	<b>39</b>
7.1	KOMBINACINĖS LOGIKOS APRAŠAI .....	39
7.1.1	<i>Loginiai ventiliai .....</i>	39
7.1.2	<i>Multipleksorius.....</i>	40
7.1.3	<i>Prioritetinis enkoderis.....</i>	40
7.1.4	<i>7 segmentų indikatoriaus dekoderis .....</i>	42
7.2	NUOSEKLIOSIOS LOGIKOS APRAŠAI .....	42
7.2.1	<i>Bistabilieji elementai.....</i>	43
7.2.2	<i>Postūmio registrai .....</i>	44
7.2.3	<i>Skaitikliai.....</i>	52
<b>8</b>	<b>PROGRAMAVIMO STILIAI ARBA APRAŠŲ LYGMENYS.....</b>	<b>55</b>

<b>9</b>	<b>VHDL KALBA PROJEKTAVIMO PAKETUOSE .....</b>	<b>56</b>
9.1	XILINX PROJEKTAVIMO PRIEMONĖSE .....	56
9.1.1	<i>Kodo šablonai (templates).....</i>	56
9.1.2	<i>IP šerdžių katalogas .....</i>	57
9.2	EDA PLAYGROUND APLINKA .....	58
9.2.1	<i>Grafinė aplinka .....</i>	58
9.2.2	<i>Nestandardinių bibliotekų naudojimas.....</i>	59
9.2.3	<i>Laikinių diagramų atvaizdavimas .....</i>	59
<b>10</b>	<b>PAGRINDINIAI LITERATŪROS ŠALTINIAI .....</b>	<b>64</b>
<b>11</b>	<b>PAPILDOMI LITERATŪROS ŠALTINIAI .....</b>	<b>64</b>

# 1 Kalbos paskirtis

VHDL (angl. *VLSI Hardware Description Language*) kalba yra skirta skaitmeninių sistemų modeliavimui (simuliavimui) ir sintezei. Šia kalba galima aprašyti ir modeliuoti skaitmenines grandines, panaudojant kalbos arba vartotojo apibrėžtas primityvas ir vėlinimo bei taktavimo reikalavimus. Taip pat projektuojamą sistemą galima žadinti vartotojo aprašytais stimulais testavimui ir verifikavimui atliki.



1 pav. Google trends paieškų intensyvumas (atlikta 2024-09-15)  
<https://trends.google.com/trends/explore?date=all&q=Verilog,VHDL>

## 2 Pagrindinės kalbos konstrukcijos

VHDL kalba aprašyta loginė grandinė vadinama **objektu** (angl. *entity*) arba elementu. Loginė grandinė gali būti labai paprasta, pvz., loginis ventilis IR, arba labai sudėtinga, pvz., mikroprocesorius. Objekto aprašymas yra apibūdinamas dviejų tipų aprašomais: **sąsajos** (angl. *entity interface*) aprašu ir vienu arba keletu **architektūrų** (angl. *architectural body*) aprašu. Žemiau yra pateiktas dviejų jėjimų loginio ventilio IR objekto aprašas *myand2*, sudarytas iš sąsajos aprašo ir jo architektūros *Behavioral* aprašo.

### 1 progr. (byla myand2.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity myand2 is
    Port ( A : in STD_LOGIC;
            B : in STD_LOGIC;
            Y : out STD_LOGIC);

-- Teisingumo lentelė
-----
-- | A  B  | Y |
-- | 1  1  | 1 |
-- | 0  x  | 0 |
-- | x  0  | 0 |
-- x - bet kuri loginė reikšmė (0 arba 1)

end myand2;

architecture Behavioral of myand2 is

begin
Y<= A and B;

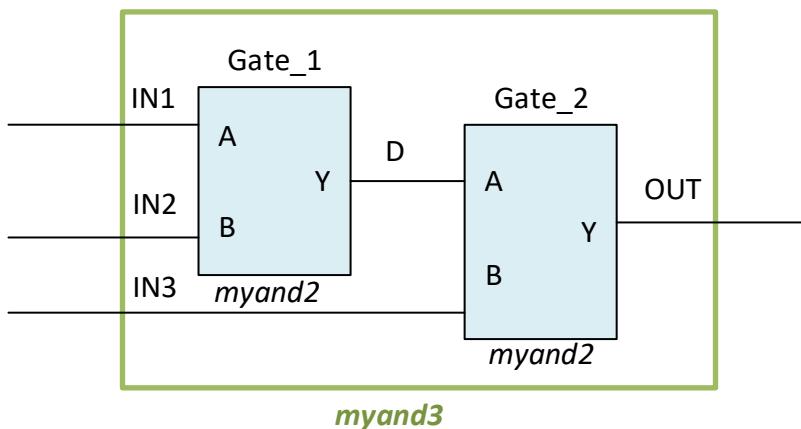
end Behavioral;
```

Courier New Baltic font 10

Sąsajos aprašas deklaruoją objekto jėjimus ir išėjimus. Sąsajos signalo aprašas apima signalo režimą (įvesties (in) arba išvesties (out)) bei signalo tipą. Pavyzdyste 1 signalai A ir B yra skirti įvedimui, o Y – išvedimui. Visų signalų tipas yra std\_logic (pavienis signalas). Sąsajos apraše yra įprasta komentarų pavidalu dokumentuoti objekto funkcionalumo aprašymą, pvz., pateikiant teisingumo lentelę. Komentarai VHDL kalbos programose prasideda dviem simboliais --.

Objekto architektūros aprašas specifikuoja objekto elgseną arba struktūrą, realizuojančią objektą iš paprastesnių komponentų. Pradiniuose projekte etapuose vykdant modeliavimą pradedama nuo objekto elgsenos (algoritminio) aprašo. Vis tik aprašo, kuriamie gali būti naudojamos aukšto abstrakcijos lygmens konstrukcijos tokios kaip ciklai, daugelio šakų sąlyginiai sakinai, būsenų automatai ir pan., ryšys su realiu aparatiniu realizavimu būna gana miglotas. Elgsenos aprašai neatsako iš klausimus kaip lengva bus atlikti skaitmeninio įrenginio (objekto) sintezę, koks bus grandinės vėlinimas ir t.t.

2 apraše yra pateiktas struktūrinis objekto AND3 architektūros apraše *Structural*, kuriamie yra sukuriami du objekto *myand2 egzemplioriai* (angl. *instance*), kurie paženklinami unikaliomis žymėmis atitinkamai *Gate\_1* ir *Gate\_2*. Objekto *myand2* aprašo dviejų jėjimų loginį ventilį IR, kuris yra vienas iš dažniausiai aparatiškai realizuojamų diskretinės elektronikos komponentų. Konstrukcija **port map**, kuriant egzempliorių (angl. *instantiation*) yra naudojama susieti išorinius signalus (duomenų linijas, laidininkus) su komponento sąsajos apraše deklaruotais vidiniais įvesties ir išvesties prievadais.

**2 pav. Scheminis programos (myand3) atvaizdas****2 progr. (byla myand3.vhd)**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity myand3 is
    Port ( IN1,IN2,IN3 : in STD_LOGIC;
           OUT : out STD_LOGIC);

-- Teisingumo lentelė
-----
-- | A  B  C | Y |
-- | 1  1  1 | 1 |
-- | 0  x  x | 0 |
-- | x  x  0 | 0 |
-- | x  0  x | 0 |
-- x - bet kuri loginė reikšmė (0 arba 1)

end myand3;

architecture Algoritmic of myand3 is
begin
-- loginis aprašas (atsižvelgiant į teisingumo lentelę)
    Y <= '1' when (IN1='1' and IN2='1' and IN3='1') else '0';
end Algoritmic;

architecture Structural of myand3 is
COMPONENT myand2_gate
    PORT(
        A : IN std_logic;
        B : IN std_logic;
        Y : OUT std_logic
    );
END COMPONENT;
    SIGNAL D: STD_LOGIC;
    for all: myand2_gate use entity work.myand2(Behavioral);
begin

-- dviejų objekto myand2 egzempliorių sukūrimas
Gate_1: myand2_gate PORT MAP(
    A => IN1,
    B => IN2,
    Y => D);

Gate_2: myand2_gate PORT MAP(
    A => D,
    B => IN3,
    Y => OUT);

end Structural;

```

### 3 Duomenų tipai

**Duomenų tipas** – tai galimų reikšmių aibės vardas. Objekto duomenų tipas apibrėžia kokias reikšmes gali įgauti konkretaus tipo objektas ir kokie veiksmai gali būti atliekami su šiuo objektu. Pavyzdžiui, sveikujų skaičių tipas *integer* nurodo, kad galimos objekto reikšmės yra sveiki skaičiai (...,-2,-1,0,1,2,...), o galimi veiksmai (arba operacijos) yra suma, daugyba ir t.t.

**Duomenų subtipas** – tai tipas papildytas ribojimais (angl. *constraints*). Pavyzdžiui, tipas natural yra integer subtipas, kurio ribojimas nurodo, kad skaičiai yra tik teigiami (0,1,2,...).

VHDL kalboje naudojami tipai klasifikuojami į tokias grupes:

1. Skaliariniai (angl. scalar)
  - 1.1. Sąrašo (išvardijimo [2]) tipas (angl. *enumeration*) – diskretiniai elementai
  - 1.2. Sveikasis tipas (angl. *integer*) – diskretiniai, skaičiai
  - 1.3. Fizinis tipas (angl. *physical*) – skaičiai
  - 1.4. Slankaus kablelio arba realus tipas (angl. *floating*) –skaičiai
2. Sudėtiniai (angl. *composite*)
  - 2.1. Masyvai (angl. *array*) – visi elementai yra vieno tipo
  - 2.2. Įrašai (angl. *records*) – elementai gali būti skirtinį tipą
3. Prieigos (angl. *access*)
4. Failų (angl. *file*)

VHDL programose galima naudoti:

1. Standartinius tipus (BIT, BOOLEAN, INTEGER)
2. Apibrėžtus bibliotekose, pvz., IEEE bibliotekos 1164 pakete yra apibrėžiami tokie tipai kaip STD\_LOGIC ir STD\_LOGIC\_VECTOR.

Tam, kad programoje naudoti IEEE 1164 tipus, jos pradžioje reikia įtraukti biblioteką:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

#### 3.1 Skaliariniai tipai

##### 3.1.1 Sąrašo tipai

Sąrašo tipas yra skaliarinis tipas, kuris apibrėžiamas, pateikiant galimą reikšmių sąrašą eilės tvarka. Elementais sąraše gali būti identifikatoriai arba simboliai. Su objektais, kuriems priskirti vidiniai tipai *Bit* ir *Boolean*, galima atlikti logines operacijas *not*, *and*, *or*, *nand*, *nor* ir *xor*.

##### 3 progr. Sąrašo tipų pavyzdžiai

```
-- kai kurie vidiniai VHDL sąrašo tipai (standard package)
type Bit is ('0', '1');
type Boolean is (FALSE, TRUE);

-- IEEE 1164 standarte apibrėžtas tipas, naudojamas atliekant sintezę
type STD_ULOGIC('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');

-- vartotojo apibrėžti tipai
type STATE is (START, IDLE, STATE1, STATE2, STATE3); -- identifikatorių sąrašas
type TRISTATE is ('Z', '0', '1'); -- simbolių sąrašas
```

VHDL sąrašo tipai palaikomi Xilinx/Vivado aplinkoje		
Sąrašo (enumerated) tipai	Biblioteka, kurioje apibrėžtas	Leistinos reikšmės
bit	standard package	0 (logic zero) 1 (logic 1)
boolean	standard package	false true
std_logic	IEEE std_logic_1164 package	Žr. std_logic leidžiamas reikšmes.

<https://docs.xilinx.com/r/en-US/ug901-vivado-synthesis/VHDL-Data-Types>

### 3.1.2 Skaitiniai tipai

VHDL kalboje yra apibrėžti du vidiniai (angl. *built-in*) skaitiniai tipai **Integer** ir **Real**, su kuriais galima vykdyti sumavimo, skirtumo, daugybos ir dalybos operacijas. Šių tipų skaičių diapazonai priklauso nuo naudojamo kompiuterio bitų skaičiaus. VHDL kalba reikalauja, kad minimalus diapazonas būtų nuo -2147483647 iki +2147483647 ( $=2^{32}/2-1$ ).

#### 4 progr. Skaitinių tipų pavyzdžiai

```
-- Integer
type COUNTER is range 0 to 100;
subtype LOW_RANGE is COUNTER range 0 to 50;
type INDEX is range 7 downto 0;

-- Real
type PROBABILITY is range 0.0 to 1.0;
```

Skaitinių tipų objektams priskiriamos reikšmės gali būti nurodomos bet kuria skaičiavimo sistema nuo dvejetainės iki šešioliktainės. Pagal nutylėjimą naudojama dešimtainė sistema.

#### 5 progr. Skaitinių tipų objekto reikšmių priskyrimas

```
A <= 123;          -- dešimtainė sistema
B <= 16#FF#;      -- šešioliktainė sistema
C <= 2#1010_1010#; -- dvejetainė sistema
Money := 1_000_000.0; -- skaičiai su dešimtainiu kableliu
```

**Pastaba:** *Real* tipo objektai negali būti sintezuojami Xilinx Vivado aplinkoje, o naudojami tik modeliavimui.

### 3.1.3 Fiziniai tipai

Fizinis tipas yra sudarytas iš duomenų tipo ir su juo susieto vieneto, kuris naudojamas aprašyti fiziniams matavimams, pvz., laikui, ilgiui, įtampai ir t.t. Vienintelis vidinis VHDL fizinis tipas yra *Time*, kuris apibrėžtas kaip parodyta 6 programe.

#### 6 progr. Fizinio tipo pavyzdys

```
type Time is range
units
  fs;                  -- femtosekundės
  ps = 1000 fs;        -- pikosekundės
  ns = 1000 ps;        -- nanosekundės
  us = 1000 ns;        -- mikrosekundės
  ms = 1000 us;        -- milisekundės
  sec = 1000 ms;       -- sekundės
  min = 60 sec;        -- minutės
  hr = 60 min;         -- valandos
```

```
end units;
```

## 3.2 Sudėtiniai tipai

### 3.2.1 Deklaravimas

Sudėtinį duomenų tipą reikšmės yra sudėtinės, pvz., vienmačių sveikų skaičių masyvo reikšmė gali būti (12, 0 , 15, 128). Domenų tipas *Bit\_Vector* (žr. 7 progr.) yra bitų masyvas. Jis naudojamas aprašyti duomenims, saugomiems registre arba perduodamiems lygiagrečia magistrale. Pažymėjimas < > reiškia, kad diapazonas yra neribojamas. Masyvo dydis gali būti nurodomas skliausteliuose po raktinio žodžio *array*.

VHDL Bit Vector tipai		
Tipas	Aprašytas bibliotekoje	Modelis
bit_vector	Standard	Bitų vektorius: type Bit_Vector is array (Natural range <>) of Bit;
std_logic_vector	IEEE std_logic_1164	std_logic elementų vektorius

### 7 progr. Masyvo tipų pavyzdžiai

```
type REGISTER_32_Bit is array (31 downto 0) of Bit; -- mažėjantis indeksas
type REGISTER_16_Bit is array (0 to 15) of Bit; -- didėjantis indeksas

type ARRAY_2D is array (Natural range <>, Natural range <>) of Bit; -- dvimatis masyvas
```

Naudojant masyvų apibrėžime *downto* atitinka mažėjančiam išlygiavimui ([little endian](#)). Tai reiškia mažiausio svorio bitas (LSB) saugomas jauniausioje (dešiniausioje) pozicijoje: L *downto* R, L atitinka (MSB), R LSB. to atitinka didėjantį išlygiavimą ([big endian](#)): L to R, L atitinka LSB, o R - MSB.

```
signal big_endian : std_logic_vector(0 to 7) := (0 => '1', others => '0');
signal little_endian : std_logic_vector(7 downto 0) := (0 => '1', others => '0');
```

Dažniausiai VHDL programose naudojamas *downto*. Nors *downto* ir to abu leistini, tačiau svarbu, kad vienoje programe būtų naudojamas vienas būdas, siekiant išvengti klaidų. Daugiau informacijos: <https://insights.sigasi.com/tech/to-downto-ranges-vhdl/>

### 3.2.2 Priskyrimas

Sudėtinį duomenų tipą (string, bit\_vector, std\_logic\_vector) objektams priskiriamą reikšmę nurodoma dvigubose kabutėse. Skaitinė reikšmė gali būti nurodyta tik dvejetainė (pagal nutylėjimą), aštuntaine arba šešioliktainė sistema. Žemiau pateiktose pavyzdžiuose atkreipkite dėmesį į raktinio žodžio **others** naudojimą priskiriant reikšmes sudėtiniams tipams.

### 8 progr. Sudėtinų tipų objektų reikšmių priskyrimas

```
constant FLAG : bit_vector(0 to 7) := "11111111"; -- pagal nutylėjimą dvejetainė sistema
constant MSG : string := "Hello";
```

```

BUS_8_BIT <= B"1111_1111"; -- dvejetainė sistema
BUS_9_BIT <= O"353"; -- aštuntainė sistema
BUS_16_BIT <= X"AA55"; -- šešioliktainė sistema

signal my_array : std_logic_vector(7 downto 0);

-- vieno elemento priskyrimas
my_array(0) <= '0';
-- dalies bitų priskyrimas
my_array(3 downto 0) <= "0001";
-- pozicinis priskyrimas (positional association)
my_array <= ("0", "0", "0", "0", "0", "0", "0", "1");
-- vardinis priskyrimas (named association variants)
my_array <= (0 => '1', others => '0');
my_array <= (0 => '1', 1 => '1', others => '0');
my_array <= (0 | 1 => '1', others => '0');
my_array <= (1 downto 0 => '1', others => '0');

```

<https://insights.sigasi.com/tech/to-downto-ranges-vhdl/>

### 3.3 Apibrėžti ir neapibrėžti IEEE 1164 tipai

Pirminis duomenų tipas aprašytas IEEE 1164 standarte yra std\_ulogic (*standard unresolved logic*). Šio tipo kintamieji gali įgyti 9 reikšmes (nurodomos viengubose kabutėse):

```

TYPE std_ulogic IS (
    'U',  -- Uninitialized
    'X',  -- Forcing Unknown
    '0',  -- Forcing 0
    '1',  -- Forcing 1
    'Z',  -- High Impedance
    'W',  -- Weak Unknown
    'L',  -- Weak 0
    'H',  -- Weak 1
    '-'   -- Don't care
);

-----
-- *** industry standard Logic type ***
-----
SUBTYPE std_logic IS resolved std_ulogic;

```

*std\_logic* tipas aprašomas kaip išvestinis iš *std\_ulogic* tipo, pridedant išsprendimo funkciją (angl. *resolution function*). **Išsprendimo funkcija** apibrėžia priskiriamą reikšmę, kai jėjime lygiagrečiai veikia daugiau negu vienas jėjimo signalas. IEEE 1164 standarto aprašą galima parsisiusti, sekant nuorodą [IEEE Standard Multivalue Logic System for VHDL Model Interoperability \(Std\\_logic\\_1164\)](#) [IEEE Std 1164-1993](#).

VHDL kalboje yra išskiriami neapibrėžti (angl. *unresolved*, pavyzdys *std\_ulogic*) ir apibrėžti (angl. *resolved*, pvz., *std\_logic*) tipai. Kai neapibrėžto tipo (*std\_ulogic*) signalui lygiagrečiai priskiriamos dvi reikšmės (angl. *two signal drivers*), tuomet kompiliuoojant projektą generuoojama klaida. Kai apibrėžto tipo (*std\_logic*) signalui lygiagrečiai priskiriamos dvi reikšmės (angl. *conflicting drivers*), tuomet priskiriamoji reikšmė nustatoma pagal išsprendimo lentelę (angl. *resolution table*). Taigi, jeigu įrenginio VHDL apraše *std\_logic* tipo signalui lygiagrečiai priskiriamos '0' ir '1', tai jis įgauna reikšmę 'X' (žr. lentelę). Kai *std\_logic* tipo signalui lygiagrečiai priskiriamos '0' ir 'Z', tai jis įgauna reikšmę '0'. Išsprendimo lentelėje iš tikrujų yra aprašyta, kad veikiant 'Z' reikšmei (aukštas impedansas) ir kitai galimai reikšmei signalas įgauna antrają reikšmę. Taip modeliuojamas diskretinės

elektronikos ventilių veikimas, kai prie magistralės yra prijungta daugiau negu vienas siųstuvas, kurie perduoda loginį lygi imtuvui. Jeigu visi siųstuvai išskyrus vieną yra Z (aukšto impedanso) būsenoje, tai vienas iš siųstuvų gali perduoti reikšmę į klausantį jėjimą.

std\_logic tipo išsprendimo lentelė (angl. *resolution table*):

Šaltinis: [https://vhdlwhiz.com/std\\_logic-vs-std\\_ulogic/](https://vhdlwhiz.com/std_logic-vs-std_ulogic/)

Galimos operacijos su konkretaus tipo signalais taip pat apibrėžiamos šiame standarte ir jį realizuojančiuose paketuose. Pavyzdžiui, loginės operacijos IR (and) aprašas std\_logic tipui atrodo taip:

```
-- overloaded logical operators ( with optimizing hints )

FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
    BEGIN
        RETURN (and_table(l, r));
    END "and";

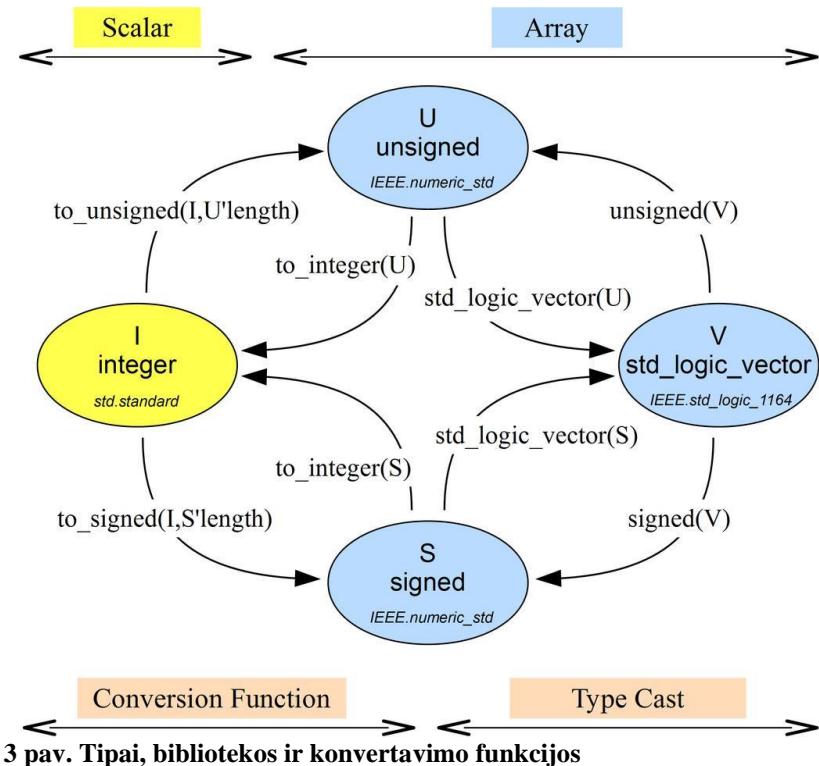
-- truth table for "and" function
CONSTANT and_table : stdlogic_table := (
    -- | U X 0 1 Z W L H - | |
    -- +-----+-----+
    ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | 0 |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ) -- | - |
);

```

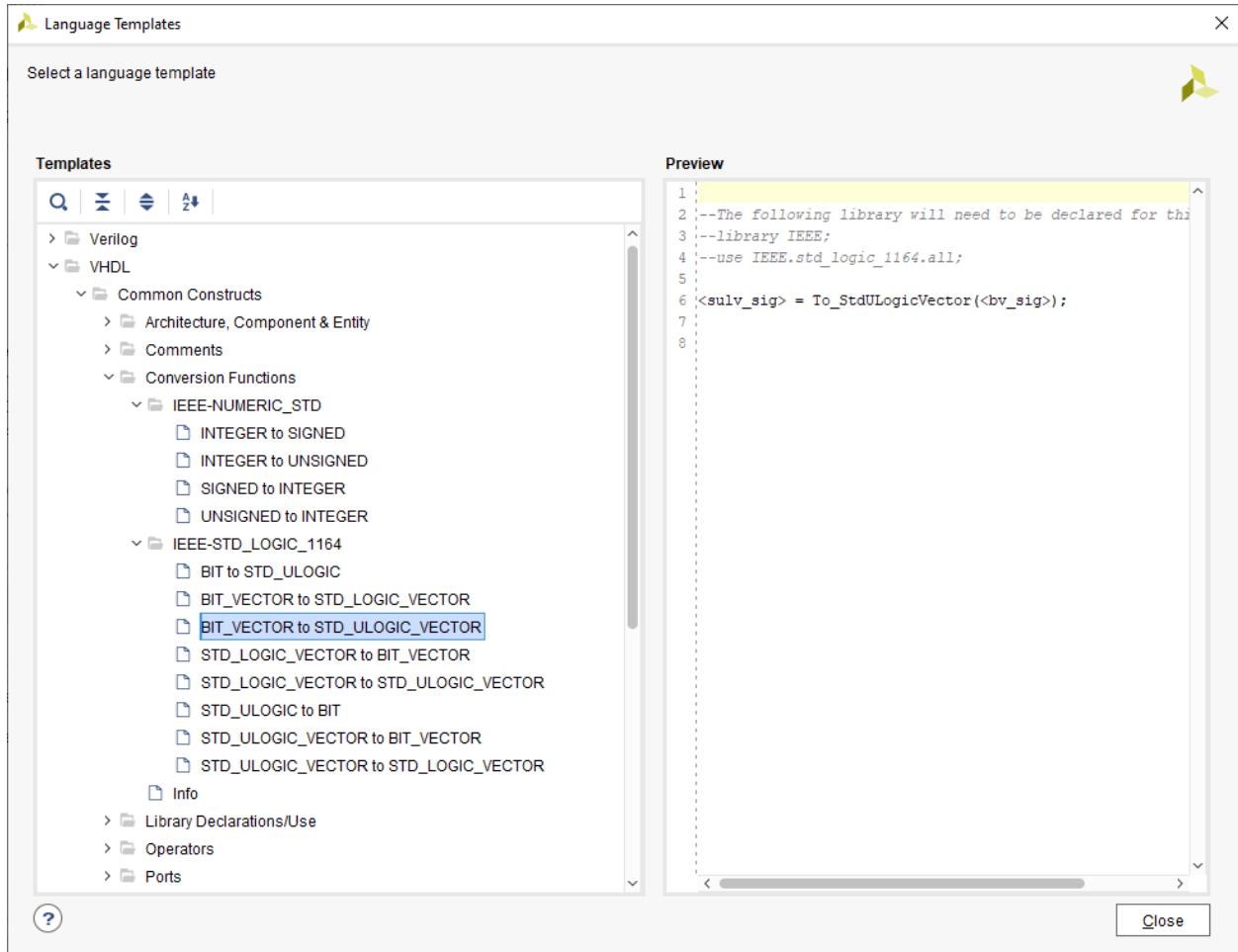
### **3.4 Tipų konvertavimas**

Dažniausiai pasitaijantys tipu konvertavimai yra aprašyti:

<https://www.nandland.com/vhdl/tips/tip-convert-numeric-std-logic-vector-to-integer.html>



3 pav. Tipai, bibliotekos ir konvertavimo funkcijos



4 pav. Tipų konvertavimo šablonių Xilinx Vivado aplinkoje

### 3.5 Duomenų tipų atributai

VHDL kalboje duomenų tipai gali turėti atributus, kurie nurodomi po apostrofo (žr. 9 progr.). Pavyzdžiui atributas `length` grąžina masyvo ilgį. 9 progr. pateiktas procesas PROCESS\_RANGE suskaičiuoja kiek vienetų yra masyve DBUS, panaudojant atributą `range` masyvo elementų skenavimui.

#### 9 progr. Masyvo tipo atributai ir elementų prieiga

```
signal DBUS: Bit_Vector (15 downto 0);

-- atributai
DBUS'right = 0 -- dešiniausio elemento indeksas
DBUS'left = 15 -- kairiausio elemento indeksas
DBUS'high = 15 -- vyriausio elemento indeksas
DBUS'low = 0 -- jauniausio elemento indeksas
DBUS'length = 16 -- masyvo ilgis
DBUS'range = 15 downto 0 -- masyvo indeksų diapazonas

-- masyvo elementų prieiga
DBUS(1) := DBUS(3); -- trečiojo elemento reikšmės kopijavimas į pirmajį elementą

PROCESS RANGE: process (DBUS)
    variable COUNT: Integer :=0;
begin
    COUNT :=0;
    for I in DBUS'range loop
        if DBUS(I) = '1' then
            COUNT := COUNT +1;
        end if;
    end process;
```

Bitų sekos vektoriuje invertavimas, panaudojant atributą `left`:

```
for i in 0 to vector'left loop
    inverted_vector(i) <= vector(vector'left - i);
end loop;
```

Daugiau informacijos apie atributus ir jų naudojimą galima rasti adresu [https://www.hdlworks.com/hdl\\_corner/vhdl\\_ref/VHDLContents/AttributesPredefined.htm](https://www.hdlworks.com/hdl_corner/vhdl_ref/VHDLContents/AttributesPredefined.htm).

## 4 Duomenų objektai

Duomenų objektas – tai deklaruoto duomenų tipo reikšmės saugojimo vieta (talpykla). Duomenų objektų klasės VHDL kalboje yra:

- Konstanta** (angl. *constant*) – tai objektas, kurio reikšmė nurodoma kompiliavimo metu ir negali būti pakeista VHDL sakiniu.
- Kintamasis** (angl. *variable*) – tai duomenų objektas, einamoji reikšmė gali būti keičiamas VHDL sakiniu.
- Signalas** (angl. *signal*) – tai duomenų objektas, kuriam apibūdinti reikalinga laiko dimensija. VHDL sakiniuose galima priskirti būsimas reikšmes šiam duomenų objektui, nepakeičiant einamosios reikšmės.

Duomenų objektai turi būti pirma deklaruoti, o tik po to juos galima naudoti priskyrimo sakiniuose.

## 4.1 Konstantos

Konstantų deklaravimo pavyzdžiai yra pateikti 10 programoje. Po raktinio žodžio **constant** rašomas konstantos vardas, toliau dvitaškiu atskiriamas jos tipas, o po simbolio := nurodoma konstantos reikšmė.

### 10 progr. Konstantų deklaravimas

```
-- konstantos
constant HIGH_IMPEDANSE: TRISTATE := 'Z';
constant PI: REAL := 3.14159;
constant C1K: Integer := 2**10; -- ** žymi kėlimą laipsniu
constant MESAGE1: STRING := "Sveiki atvyke!";
```

## 4.2 Kintamieji

VHDL kalbos kintamieji yra panašūs į kitų aukšto lygmens programavimo kalbų, tokų kaip C, C++ ar Java kintamuosius. Kintamieji gali būti deklaruoti tik procesuose arba paprogramėse (angl. *subprograms*).

### 1 lent. Kintamųjų deklaravimas

Deklaravimo vieta	Kintamojo ypatybė	Incializavimas
Procesas	Lokalus šiam procesui. Statinis, t. y. reikšmė priskirta viename proceso iškvietime atsimenama iki kito proceso iškvietimo.	Vieną kartą simuliavimo pradžioje.
Paprogramė	Dinaminis, t. y. neišlaiko reikšmės tarp skirtingu paprogramės iškvietimų.	Kiekvieną kartą iškviečiant paprogramę.

### 11 progr. Kintamųjų deklaravimas

```
-- kintamieji
variable A, B, C: Bit;
variable A_skaitiklis, B_skaitiklis: COUNTER := 10; -- kintamieji su pradine reikšme
variable Event_Timer: Time := 1 ns;
variable Z: Integer := 0;
-- masyvo tipo kintamieji
-- masyvo deklaravimas, šešioliktainio skaičiaus priskyrimas
variable REG1: Bit_Vector (15 downto 0) := X"F21A";
-- dvejetainio skaičiaus priskyrimas
variable REG2: Bit_Vector (15 downto 0) := B"1010_1111_0000_0101";
-- Elementų reikšmės REG2(15)=1, REG2(14)=0, REG2(13)=1, ... REG2(1)=0, REG2(0)=1.

variable ROM_A: ROM_TYPE (0 to 7) :=
(0 => X"FFFF_FFFF",
 5 => X"F000_0001",
 others => X"0000_0000");

-- dvimatis masyvas
variable MATRIX_3X4: ARRAY_2D (0 to 2, 0 to 3) :=
((1',1',0',0'),
 (0',1',1',0'),
 (0',0',1',1'));
```

## 4.3 Signalai

Signalai yra naudojami aprašyti duomenis fizinėse duomenų linijose ir grandinėse. Fizinėse sistemose signalas, pvz., įtampa, negali pasikeisti momentiškai. Todėl VHDL signalų reikšmės taip pat

nekintą momentiškai. Nauja priskirta signalui reikšmė negalioja iki tam tikro ateities momento. Šis ateities momentas gali būti nurodytas tiesiogiai. Jeigu laikas nenurodomas, tai naujoji reikšmė priskiriama po begaliniai trumpo laiko intervalo, vadinamo *delta* laiku.

Signalų deklaravimas yra panašus į kintamųjų deklaravimą (žr. 12 progr.), tačiau signalai negali būti deklaruoti procesuose ir paprogramėse, o deklaruojami tik kaip prievedai (portai) deklaruojant objektus (angl. *entities*) arba architektūrą aprašų deklaravimo sekciijose.

### 12 progr. Kintamųjų deklaravimas

```
-- signalai
signal X, Y, Z: Bit;
signal DOWN_COUNT: COUNTER := COUNTER`right;
-- atributas right nurodo dešiniausią reikšmę

signal ROM_B: ROM_TYPE (0 to 3) :=
(X"1111_FFFF", X"2222_FFFF", X"3333_FFFF", X"4444_FFFF");
```

## 5 Kalbos konstrukcijos

### 5.1 Priskyrimas

Priskyrimo sakiniai (angl. *assignment statement*) yra naudojami kintamųjų ir signalų reikšmių keitimui.

Kintamojo priskyrimo sakiny, kuris užrašomas naudojant skiriamą ženklą **dvitaškis-lygybė** `:=`, priskiria naują reikšmę kintamajam **momentiškai** (blokuojantis priskyrimas).

Signalio priskyrimo sakiny, kuris užrašomas naudojant skiriamą ženklą **daugiau-lygybė** `<=`, nurodo priskirti naują reikšmę signalui **po tam tikro laiko momento ateityje** (neblokuojantis priskyrimas).

### 13 progr. Kintamųjų ir signalų priskyrimo pavyzdžiai (deklaravimas aprašytas anksčiau)

```
-- kintamųjų priskyrimo pavyzdžiai
A := '1';
Z := 1234;
MATRIX_3X4(1,2) := MATRIX_3X4(0,1);
ROM_A(0) := X"AA55"; -- priskyrimas masyvo elementui

-- signalų priskyrimo pavyzdžiai
X <= '0'; -- modeliuojant priskyrimas vykdomas po delta laiko
Y <= '1' after 10 ns; -- naudojama modeliavimui, bet ne sintezei
Z <= '0' after 10 ns, '1' after 20 ns, '0' after 30 ns; -- naudojama modeliavimui,
--bet ne sintezei
ROM_B(0) <= ROM_B(1);
```

### 5.2 Operacijos ir išraiškos

#### 5.2.1 Operatoriai

VHDL kalboje palaikomi standartiniai operatoriai, kurie parodyti 2 lentelėje. Vienos grupės operatorių prioritetai yra vienodi, o prioritetai tarp grupių parodyti 2 lentelėje.

VHDL operacijos apibrėžiamos priklausomai nuo signalo tipo (<https://startingelectronics.org/software/VHDL-CPLD-course/tut13-VHDL-data-types-and-operators/>), pvz., aritmetinės operacijos galimos tik su INTEGER, SIGNED ir UNSIGNED duomenų tipais. Su std\_logic\_vector tipo duomenimis aritmetinės operacijos gali būti atliekamos tik tuo

atveju, jeigu naudojama biblioteka STD\_LOGIC\_SIGNED arba STD\_LOGIC\_UNSIGNED iš IEEE paketo (<https://startingelectronics.org/software/VHDL-CPLD-course/tut13-VHDL-data-types-and-operators/>).

## 2 lent. VHDL kalbos operatoriai

Prioritetas	Grupė	Operatoriai	Ženklas
1 (auksčiausias)	Įvairūs	Kėlimas laipsniu Absoliutinė skaičiaus vertė (modulis) Loginė inversija (tik Bit ir Boolean tipams)	** abs not
2	Daugybos	Aritmetinė saundauga (Integer ir Real tipams) Aritmetinė dalyba (Integer ir Real tipams) Aritmetinės dalybos sveikoji dalis (tik Integer tipui) Aritmetinės dalybos liekana (tik Integer tipui)	* / mod rem
3	Ženklo	Ženklas nekeičiamas Ženklas keičiamas	+
4	Sumavimo	Aritmetinė suma Aritmetinis skirtumas Vienmačio masyvo elementų konkatenacija	+-&
5	Palyginimo (relational operators)	Lygu Nelygu Daugiau negu Daugiau negu arba lygu Mažiau negu Mažiau negu arba lygu	=/=>≥<≤=
6	Palyginimo ?(matching relational operators) (pradedant VHDL-2008 versija)	Lygu  Nelygu  Daugiau negu Daugiau negu arba lygu Mažiau negu Mažiau negu arba lygu	?= supranta simbolį "-" (don't care) ?/= supranta simbolį "-" (don't care) ?> ?≥ ?< ?≤=
7 (žemiausias)	Loginiai	Loginė sandauga (IR) Loginė suma (ARBA) Loginė sandauga su inversija (IR-NE) Loginė suma su inversija (ARBA-NE) Loginė suma moduliui 2	and or nand nor xor
8	Sąlyginis operatorius	Išraiškos rezultatas yra loginis skaičius (1 arba 0)	?

Palyginimo operatorius ?= (angl. *matching relational operator*) gali gražinti bit arba std\_logic tipo rezultatą. Ankstesnėse VHDL versijose palyginimo operatoriai gražindavo tik boolean tipo rezultatus. Todėl naudojant naują palyginimo operatorių kodą galima užrašyti trumpiau, t.y. vietoj

```
if x = y then
    out1 <= '1';
else
    out1 <= '0';
end if;
```

dabar užtenka užrašyti <https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Matching-Relational-Operators>:

```
out1 <= x ?= y;
```

Tokiu būdu paprastėja (trumpėja) dešifratoriaus užrašas:

```
signal DevSel : std_logic;
signal Addr: std_logic_vector(7 downto 0);

DevSel <= Addr ?= X"A5" and Cs1 and not nCs2;
```

[http://www.synthworks.com/papers/VHDL\\_2008\\_end\\_of\\_verbosity\\_2013.pdf](http://www.synthworks.com/papers/VHDL_2008_end_of_verbosity_2013.pdf) (Slides 6-7).

Papildomai palyginimo operatoriai su ? atpažista simbolį „nesvarbu“ (dont‘care).

DevSel <= Addr ?= "111---00"; --rezultatas lygus 1, jeigu Addr trys vyriausi bitai 111, o du jauniausi 00 (suprantant don't care simbolį, tod4l trys viduriniai bitai nesvarbu)

Operatoriai ir tipai, su kuriais jie yra apibrėžti.

Operator type	Predefined operators	Supported synthesizable predefined data types (*)
Logical	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, BOOLEAN, BOOLEAN_VECTOR <sup>(1)</sup> , STD_(U)LOGIC, STD_LOGIC_(U)VECTOR, (UN)SIGNED <sup>(2)</sup> , UFIXED <sup>(1)</sup> , SFIXED <sup>(1)</sup> , FLOAT <sup>(1)</sup>
Arithmetic	+, -, *, /, **, ABS, REM, MOD	INTEGER, NATURAL, POSITIVE, STD_(U)LOGIC_VECTOR <sup>(3)</sup> , (UN)SIGNED <sup>(4)</sup> , UFIXED <sup>(1)</sup> , SFIXED <sup>(1)</sup> , FLOAT <sup>(1)</sup>
Comparison	=, /=, >, <, >=, <=	BIT, BIT_VECTOR, BOOLEAN, BOOLEAN_VECTOR <sup>(1)</sup> , INTEGER, NATURAL, POSITIVE, INTEGER_VECTOR <sup>(1)</sup> , CHARACTER, STRING, STD_(U)LOGIC_VECTOR <sup>(3)</sup> , (UN)SIGNED <sup>(4)</sup> , UFIXED <sup>(1)</sup> , SFIXED <sup>(1)</sup> , FLOAT <sup>(1)</sup>
Shift	SLL, SRL, SLA, SRA, ROL, ROR	BIT_VECTOR, BOOLEAN_VECTOR <sup>(1)</sup> , STD_LOGIC_(U)VECTOR <sup>(3)</sup> , (UN)SIGNED <sup>(4)</sup> , UFIXED <sup>(1)</sup> , SFIXED <sup>(1)</sup>
Concatenation	& (" " and OTHERS too)	BIT_VECTOR, BOOLEAN_VECTOR <sup>(1)</sup> , INTEGER_VECTOR <sup>(1)</sup> , STRING, STD_(U)LOGIC_VECTOR, (UN)SIGNED
Matching comparison <sup>(1)</sup>	?=, ?/=, ?>, ?<, ?>=, ?<=	BIT, BIT_VECTOR <sup>(*)</sup> , BOOLEAN_VECTOR <sup>(*)</sup> , STD_(U)LOGIC, STD_(U)LOGIC_VECTOR, (UN)SIGNED <sup>(2)</sup> , UFIXED <sup>(1)</sup> , SFIXED <sup>(1)</sup> , FLOAT <sup>(1)</sup>
Condition <sup>(1)</sup>	??	BIT, STD_(U)LOGIC
Min/Max and String conversion <sup>(1)</sup>	MINIMUM, MAXIMUM, TO_STRING, etc.	Nearly all VHDL types in standard packages (see appendices)
(*) Note: Some types support only a partial set of operators		(3) Requires package std_logic_(un)signed or numeric_std_unsigned
(1) Introduced or proposed in VHDL 2008		(4) Requires package numeric_std or std_logic_arith
(2) With package numeric_std		

### 5.2.2 Konkatenacija

Konkatenacija, tai dviejų objektų sujungimas į vieną, pvz., dviejų signalų sujungimas į vieną grupę, turinčią bendrą pavadinimą.

```
signal a, b, c, d : std_logic:='0';
signal out : std_logic_vector(1 downto 0);
out<= (a and b) & (c and d);

--tai analogiška tokiam fragmentui
out(1) <= a and b;
out(0) <= c and d;
```

```
signal a: std_logic_vector(0 to 3):="1111";
signal b: std_logic_vector (0 to 7):= (others => '0');
--4 viriausi bitai nustatomi į „0“, o 4 jauniausiai į „1“.
b<="0000" & a;
```

Panaudojimo pavyzdys, kai apjungus keletą bitų, jie naudojami *case* sakinyje (dažnas fragmentas realizuojant multipleksorius):

```
signal out : std_logic_vector(3 downto 0);
signal bit0,bit1,bit2,bit3 :std_logic;

process(bit0,bit1,bit2,bit3)
    variable bitcat : std_logic_vector(3 downto 0);
begin
    bitcat := bit0 & bit1 & bit2 & bit3; --konkatenacija
    case bitcat is
        when "0001" => out <= 1;
        when "0010" => out <= 2;
        when "0011" => out <= 3;
        when others => out <= 4;
    end case;
end process;
```

*Integer* tipo skaičių konkatenacija atliekama netiesiogiai, t. y. konvertuojant į STD LOGIC VECTOR tipą:

```
signal out : std_logic_vector(31 downto 0);
constant a: integer := 123;
constant b: integer := 456;
-- conv_std_logic_vector(signal_name, number_of_bits)
-- formuojamas 32 bitų vektorius iš dviejų integer skaičių
out <= conv_std_logic_vector(a, 16) & conv_std_logic_vector(b, 16);
```

Konkatenacija atliekama ir su eilutės tipo kintamaisiais, pvz.,

```
for i in 0 to 15 loop
    report "X=" & integer'image(to_integer(unsigned(x_in))) & "      Y=" &
std_logic'image(y_out);
end loop;
```

Postūmio registro realizavimą, panaudojant konkatenaciją, žiūrėkite 28 progr. (Postūmio registro realizavimas).

### 5.2.3 Redukavimas

Redukavimo operacija atlieka loginį veiksmą su visais to paties vektoriaus (signalo ar kintamojo) bitais.

```
use std_logic_misc;
-- std_logic_misc bibliotekoje aprašytos redukavimų atliekančios funkcijos
--function AND_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
--function NAND_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
--function OR_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
--function NOR_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
--function XOR_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
--function XNOR_REDUCE(ARG: STD_LOGIC_VECTOR) return UX01;
```

```

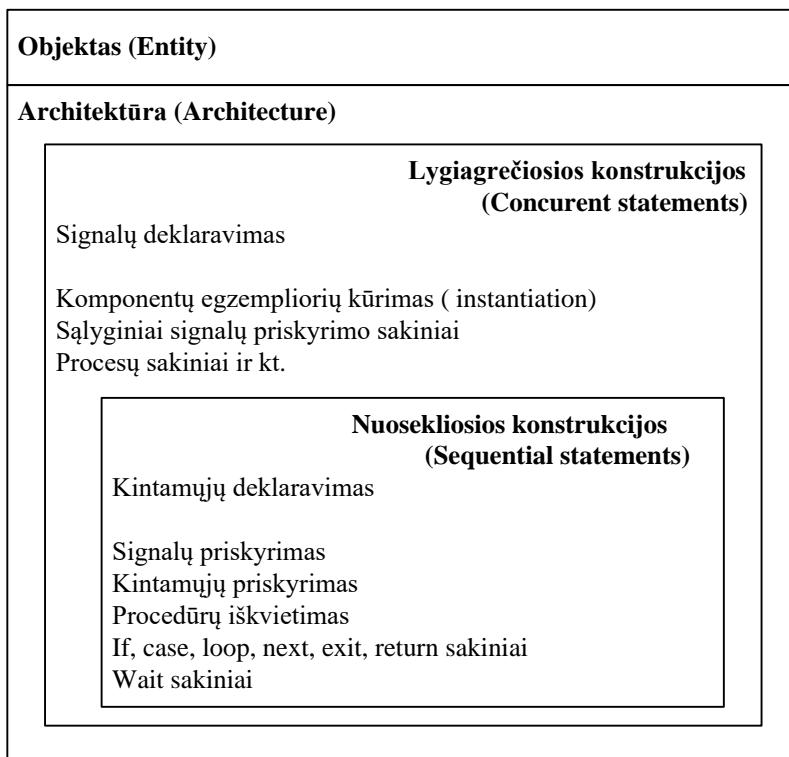
signal OneBit: std_logic;
signal Word: std_logic_vector (7 downto 0);

OneBit <= xor_reduce(Word); -- Word(0) xor Word(1) xor ...
-- xor redukuavimo operacija apskaičiuoja lyginumo bitą (parity bit)
OneBit <= and_reduce(Word); -- Word(0) and Word(1) and ...
OneBit <= or_reduce(Word); -- Word(0) or Word(1) or ...

```

### 5.3 Nuosekliosios konstrukcijos

Nuosekliosios konstrukcijos (angl. *sequential control statements*) yra naudojamos procesuose ir paprogramėse (funkcijose, procedūrose) algoritmams aprašyti. Šios konstrukcijos vykdomos jų surašymo tvarka, t. y. nuosekliai, o ne lygiagrečiai.



5 pav. VHDL aprašo struktūra

#### 5.3.1 Sąlyginis sakinys if

##### 14 progr. Nuosekliųjų konstrukcijų pavyzdžiai

```

-- if konstrukcijos pavyzdys
if A < 0 then
    LEVEL := 1;
elsif A > 1000 then
    LEVEL := 2;
else
    LEVEL := 3;
end if;

```

#### 5.3.2 case ir case? sakiniai

```
-- case konstrukcijos pavyzdys
case Adress is
when 0 => ChipSelect <= 1;
when (1 to 255) => ChipSelect <= 2;
when X"FFFF" => ChipSelect <= 3;
when others ChipSelect <= 0;
end case;
```

VHDL-2008 kalbos versija palaiko ir naują operatorių *case?*, kuriame galima naudoti bet kokia loginę reikšmę apibūdinantį žymėjimą "-" (angl. don't care):

```
case? sel is
when "1---" =>
    out <= "11";
when "01--" =>
    out <= "10";
when "001-" =>
    out <= "01";
when "0001" =>
    out <= "00";
when others =>
    null;
end case;
```

[http://www.doulos.com/knowhow/vhdl\\_designers\\_guide/vhdl\\_2008/vhdl\\_200x\\_small/#matchcase](http://www.doulos.com/knowhow/vhdl_designers_guide/vhdl_2008/vhdl_200x_small/#matchcase)

### 5.3.3 Ciklai

Ciklų veikimas yra viena iš dažnai klaidingai suprantamų HDL kalbų konstrukcijų (<https://www.nandland.com/vhdl/examples/example-for-loop.html>). Ciklai (VHDL for loop konstrukcija) naudojami skirtingai sintezuojamai ir nesintezuojamai (pvz., kuriant bandymų stendus) logikoms. Prieš naudojant ciklus būtina aiškiai suprasti jų veikimą. **Ciklų veikimas HDL kalbose yra visiškai skirtinges negu mikroprocesorių kalbose, tokiose kaip C.**

```
-- ciklų pavyzdžiai
while A<B
    A:= A+1;
end loop;

for I in 1 to 10 loop
A(i):=A(i)+1;
end loop;

loop
    A:= A+1;
exit when A > 10;
end loop;
```

#### 5.3.3.1 Ciklai sintezuojamos logikos aprašuose

Ciklų aprašai yra sintezuojami. **for loop** ciklai naudojami logikos replikavimui (daugelio panašiu logikos blokų sukūrimui)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```

entity Example_For_Loop is
  port (
    i_Clock : std_logic
  );
end Example_For_Loop;

architecture behave of Example_For_Loop is

  signal r_Shift_With_For : std_logic_vector(3 downto 0) := X"1";
  signal r_Shift-Regular : std_logic_vector(3 downto 0) := X"1";

begin

  -- Postūmio registro sukūrimas panaudojant For Loop
  p_Shift_With_For : process (i_Clock)
  begin
    if rising_edge(i_Clock) then
      for ii in 0 to 2 loop
        r_Shift_With_For(ii+1) <= r_Shift_With_For(ii);
      end loop;  -- ii
    end if;
  end process;

  -- Postūmio registro sukūrimas, panaudojant priskyrimo sakinius
  p_Shift_Without_For : process (i_Clock)
  begin
    if rising_edge(i_Clock) then
      r_Shift-Regular(1) <= r_Shift-Regular(0);
      r_Shift-Regular(2) <= r_Shift-Regular(1);
      r_Shift-Regular(3) <= r_Shift-Regular(2);
    end if;
  end process;

end behave;

```

Kombinacinės logikos grupės egzempliorių sukūrimui (angl. expand combinational logic) naudojama **for generate** konstrukcija, kurios paskirtis yra (<https://www.nandland.com/vhdl/examples/example-generate-statement.html>):

Taikymas Nr. 1. Replikuoti logikos dalis aprašytas VHDL kalba,

Taikymas Nr. 2. Ijungti/išjungti logikos blokus.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity example_generate is
  generic (
    g_DEBUG      : natural := 1          -- 0 = no debug, 1 = print debug
  );
end example_generate;

architecture behave of example_generate is

  signal r_VECTOR : std_logic_vector(15 downto 0) := (others => '0');

  signal w_VECTOR_MSB_1 : std_logic_vector(7 downto 0);
  signal w_VECTOR_MSB_2 : std_logic_vector(7 downto 0);

  signal w_VECTOR_TEST : std_logic_vector(15 downto 0);

begin

  -- Taikymas Nr. 1: Logikos replikavimas
  -- Stores just the most significant byte in a new signal
  g_GENERATE_FOR: for ii in 0 to 7 generate
    w_VECTOR_MSB_1(ii) <= r_VECTOR(ii+8);
  end generate g_GENERATE_FOR;

  -- Kodas analogiškas esančiam aukščiau

```

```
-- bet aukščiau esantis yra kompaktiškesnis, lengviau skaitomas, ir atsparesnis klaidoms
w_VECTOR_MSB_2(0) <= r_VECTOR(8);
w_VECTOR_MSB_2(1) <= r_VECTOR(9);
w_VECTOR_MSB_2(2) <= r_VECTOR(10);
w_VECTOR_MSB_2(3) <= r_VECTOR(11);
w_VECTOR_MSB_2(4) <= r_VECTOR(12);
w_VECTOR_MSB_2(5) <= r_VECTOR(13);
w_VECTOR_MSB_2(6) <= r_VECTOR(14);
w_VECTOR_MSB_2(7) <= r_VECTOR(15);

-- Taikymas Nr. 2: Logikos ijjungimas/išjungimas
g_KEEP_DEBUG : if g_DEBUG = 1 generate

    p_TEST: process (r_VECTOR) is
    begin
        w_VECTOR_TEST <= r_VECTOR;
    end process p_TEST;

end generate g_KEEP_DEBUG;

-- Taikymas Nr. 2: Logikos ijjungimas/išjungimas
g_REMOVE_DEBUG : if g_DEBUG = 0 generate
    w_VECTOR_TEST <= (others => '0');
end generate g_REMOVE_DEBUG;

-- Šis kodas nesintezuojamas, visas kitas kodas šiame pavyzdje yra sintezuojamas
p_MAIN_TEST : process is
begin
    r_VECTOR <= X"DEAD";
    wait for 100 ns;
    r_VECTOR <= X"BEEF";
    wait for 100 ns;
    wait;
end process;

end behave;
```

Dar vienas kombinacinių logikos replikavimo pavyzdys (Taikymas Nr. 1):

```
entity REG_BANK is
    port (
        CLK: in STD_LOGIC;
        RESET: in STD_LOGIC;
        DIN: in STD_LOGIC_VECTOR(3 downto 0);
        DOUT: out STD_LOGIC_VECTOR(3 downto 0));
end REG_BANK;

architecture GEN of REG_BANK is
    component REG
        port(D,CLK,RESET : in std_ulogic;
             Q          : out std_ulogic);
    end component;
begin
    GEN_REG:
    for I in 0 to 3 generate
        REGX : REG port map
            (DIN(I), CLK, RESET, DOUT(I));
    end generate GEN_REG;
end GEN;
```

Bitų sekos nuoseklumo pakeitimas (angl. *bit sequence reversing*), panaudojant **for generate** konstrukciją:

```
signal y unsigned (0 to 7);
signal a unsigned (0 to 7);
```

```
gen: for i in 0 to 7 generate
    y(i) <= a(7-i);
end generate;
```

### 5.3.3.2 Ciklai simuliavimo tikslams

Simuliavimo aprašuose ciklai veikia panašiai kaip mikroprocesorinėse programavimo kalbose. Ciklo kūne gali būti naudojami vėlinimo sakiniai, kurie įveda simuliavimo vėlinimus. Pavyzdyme žemiau parodyta, kaip inicializuojama Data reikšmė, priskiriant po vieną reikšmę kas 10 ns. Priskiriame reikšmės taip pat išvedamos į pranešimų langą. Šis kodas nesintezuojamas.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity for_loop_simulation is
end entity for_loop_simulation;

architecture behave of for_loop_simulation is
    type t_Data is array (0 to 5) of integer;
begin

    process is
        variable Data : t_Data; -- sukurti 6-ių integer skaičių masyva
    begin

        for ii in 0 to 5 loop
            Data(ii) := ii*ii;
            report("Data at Index " & integer'image(ii) & " is " &
                   integer'image(Data(ii)));
            wait for 10 ns;
        end loop;

        assert false report "Test Complete" severity failure;
    end process;
end architecture behave;
```

Šio pavyzdžio vykdymo išvedimai konsolės lange būtų:

```
# Time  0: Data at Index 0 is  0
# Time 10: Data at Index 1 is  1
# Time 20: Data at Index 2 is  4
# Time 30: Data at Index 3 is  9
# Time 40: Data at Index 4 is 16
# Time 50: Data at Index 5 is 25
```

## 5.4 Lygiagrečiosios konstrukcijos

Lygiagrečiosios konstrukcijos (angl. *concurrent statements*) naudojamos objektų architektūrų aprašuose signalų elgsenos apibūdinimui.

### 5.4.1 process sakinys

*process* sakinys labai plačiai naudojamas architektūrų aprašuose. *process* sakinys gali būti dviejų formų:

1. Su jautrumo sąrašu (procesas PROC1 15 programoje).

2. Be jautrumo sąrašo (procesas PROC2 15 programoje).

### 15 progr. process sakinio konstrukcijos

```
-- process konstrukcija su jautrumo sąrašu
PROC1: process (S1, S2)
    -- konstantų deklaravimas
    -- kintamuju deklaravimas
    -- paprogramių deklaravimas
    -- NEGALIMAS signalų deklaravimas
begin
    -- Nuosekliosios konstrukcijos
end process PROC1;

-- process konstrukcija be jautrumo sąrašo
PROC2: process
    -- konstantų deklaravimas
    -- kintamuju deklaravimas
    -- paprogramių deklaravimas
    -- NEGALIMAS signalų deklaravimas
begin
    -- Nuosekliosios konstrukcijos
    wait on S1, S2;
end process PROC2;
```

Procesas su jautrumo sąrašu PROC1 yra įvykdomas vieną kartą pradedant simuliavimą ir kiekvieną kartą, esant įvykiui bet kuriame iš signalų, kurie įrašyti jautrumo sąraše (S1 ir S2 signalai PROC1 procese). Kiekvieną kartą aktyvavus procesą tame esantys nuoseklieji sakiniai įvykdomi vienas paskui kitą panašiai kaip procedūrinėse programavimo kalbose C arba C++.

Procesas be jautrumo sąrašo PROC2 yra įvykdomas vieną kartą pradedant simuliavimą ir jo vykdymas tēsiamas iki sakinio *wait*. Įvykdžius paskutinį sakinį, procesas vėl pradedamas vykdyti nuo pirmojo sakinio. **Todėl modeliuojant (ypač kuriant bandymų stendus), procese turi būti bent vienas wait sakinys, kad išvengti begalinio ciklo.**

### 5.4.2 Lygiagretusis signalų priskyrimas

Signalų priskyrimo sakiniai gali būti nuoseklieji arba lygiagretieji:

- *Nuoseklis* priskyrimo sakinys įvykdomas tik tada, kai algoritmas pasiekia priskyrimo sakinį.
- *Lygiagretusis* priskyrimo sakinys vykdomas bet kuriuo momentu, kai tik vyksta dešinėje priskyrimo operatoriaus pusėje esančio signalo reikšmės pasikeitimas (vadinama signalo įvykiu).

16 programoje yra pateikti du ekvivalentiški lygiagretaus priskyrimo signalui C aprašai.

### 16 progr. Lygiagrečiųjų priskyrimo sakiniių pavyzdžiai

```
signal A, B, C: Bit;

-- lygiagretusis signalo priskyrimas
C <= A or B;

-- ekvivalentus užrašas, panaudojant process sakinį
process (A,B)
begin
    C <= A or B;
end process;
```

17 programoje yra pateikti lygiagretaus sąlyginio signalų priskyrimo pavyzdžiai (*when-else* ir *with-select-when* konstrukcijos) kartu su ekvivalentiškais užrašais, naudojant *process* sakinį ir atitinkamas konstrukcijas (*if-else* ir *case-when*).

**17 progr. Lygiagrečiųjų sąlyginių priskyrimo sakinių pavyzdžiai**

```

signal Z,A,B,C,D,S: Bit;
signal op,X,Y: Integer;

-- lygiagretusis signalo priskyrimas
S <= C or D when op=1 else
    C and D when op=2 else
    C xor D;

-- ekvivalentus užrašas, panaudojant process sakinį
process (C,D,op)
begin
    if op=1 then S <=C or D;
    elsif op=2 then S <=C and D;
    else S <=C xor D;
    end if;
end process;

-- lygiagretusis signalo priskyrimas
with (X+Y) select
Z <= A after 5 ns when 0, -- kai X+Y==0
    B after 10 ns when 1,
    C after 15 ns when others;

-- ekvivalentus užrašas panaudojant process sakinį
process (X, Y, A, B, C)
begin
case (X+Y) is
    when 0 => Z <=A after 5 ns;
    when 1 => Z <=B after 10 ns;
    when others => Z <=C after 15 ns;
end process;

```

**5.4.3 process sakinio naudojimas sinchroninės logikos aprašymui**

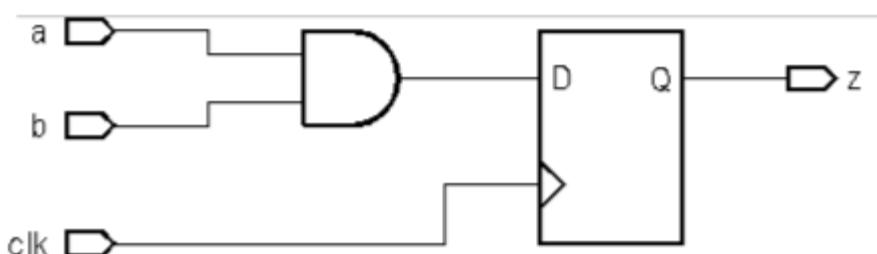
Synchroninės logikos aprašymui process sakinje galima panaudoti *wait* sakinys [12].

**18 progr. Sinchroninės logikos aprašo pavyzdys su wait konstrukcija**

```

proc6: process
begin
wait until clk = '1';
    z <= a and b;
end
process proc6;

```

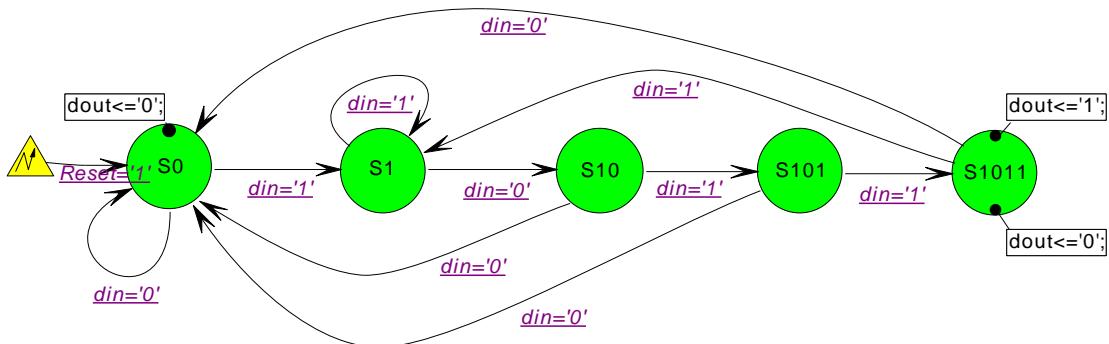


**6 pav. Susintezuota grandinė**

**5.5 Baigtinių būsenų automatu aprašymas**

Baigtinių būsenų automatas BBA (angl. *Finite-state Machine (FSM)*) – tai synchroninis įrenginys, sudarytias iš nuosekliosios ir kombinacinės logikų, kurių išėjimų ir vidinių atminties elementų saugomos reikšmės kinta pagal apibrėžtą būsenų seką, kaip reakcija į taktinius impulsus bei priklausomai nuo jėjimuose veikiančių signalų reikšmių [1,0,11]. BBA skirtomi į Muro (*Moore*) ir Milio (*Mealy*) tipo automatus.

Galima panagrinėti įrenginio, atliekančio bitų sekos „1011“ aptikimą jėjime veikiančiam binarinių signalų sraute, aprašą, realizuotą kaip BBA.



7 pav. Bitų sekos „1011“ aptikimo įrenginio būsenų diagrama

#### 19 progr. Bitų sekos aptikimo įrenginio aprašas VHDL kalba

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity BitSeqDetector is
  port (
    clk: in STD_LOGIC;
    din: in STD_LOGIC;
    Reset: in STD_LOGIC;
    dout: out STD_LOGIC);
end BitSeqDetector;

architecture BitSeqDetector_arch of BitSeqDetector is

-- Būsenos registro tipas
type Sreg0_type is (
  S0, S1, S10, S101, S1011);

signal Sreg0: Sreg0_type; -- Būsenos registras

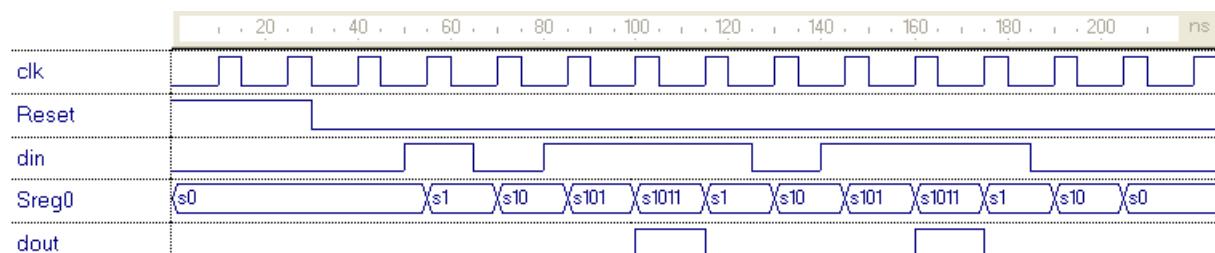
begin
  -----
  -- Automatas: Sreg0
  -----
  Sreg0_machine: process (clk, reset)
  begin
    if Reset='1' then
      Sreg0 <= S0;
      dout <= '0';
    elsif clk'event and clk = '1' then
      case Sreg0 is
        when S10 =>
          if din='1' then
            Sreg0 <= S101;
          elsif din='0' then
            Sreg0 <= S0;
            dout <= '0';
          end if;
        when S101 =>
          if din='1' then
            Sreg0 <= S1011;
          elsif din='0' then
            Sreg0 <= S10;
          end if;
        when S1011 =>
          if din='1' then
            Sreg0 <= S101;
          elsif din='0' then
            Sreg0 <= S0;
            dout <= '0';
          end if;
        when others =>
          Sreg0 <= S0;
      end case;
    end if;
  end process;
end;
  
```

```

when S101 =>
    if din='1' then
        Sreg0 <= S1011;
        dout <= '1'; -- Seka aptikta
    elsif din='0' then
        Sreg0 <= S0;
        dout <= '0';
    end if;
when S1011 =>
    if din='0' then
        Sreg0 <= S0;
        dout <= '0';
    elsif din='1' then
        Sreg0 <= S1;
        dout <= '0';
    end if;
when S0 =>
    if din='0' then
        Sreg0 <= S0;
        dout <= '0';
    elsif din='1' then
        Sreg0 <= S1;
    end if;
when S1 =>
    if din='0' then
        Sreg0 <= S10;
    elsif din='1' then
        Sreg0 <= S1;
    end if;
when others =>
    null;
end case;
end if;
end process;

end BitSeqDetector_arch;

```



8 pav. Bitų sekos „1011“ aptikimo įrenginio modeliavimo laikinės diagramos

## 5.6 Funkcijos ir procedūros

Funkcijos <https://vhdlwhiz.com/function/>

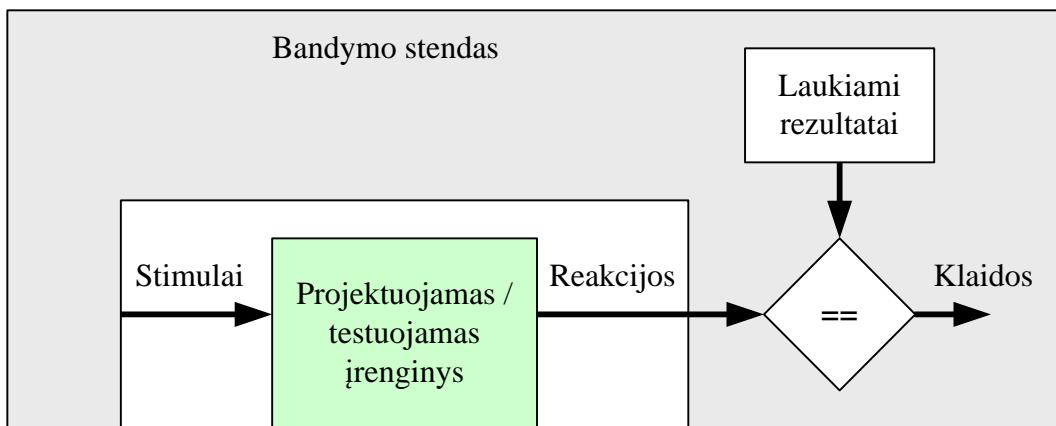
Procedūros <https://vhdlwhiz.com/using-procedure/>

Ir daugiau medžiagos <https://vhdlwhiz.com/basic-vhdl-tutorials/>

# 6 VHDL kalbos naudojimas skaitmeninių įrenginių modeliavimui

## 6.1 Virtualieji bandymo stendai (TestBench)

Sukurti VHDL aprašai (modeliai) turi būti verifikuojami (testuojami). Pirminis verifikavimo žingsnis yra atliekamas vykdant funkcinį modeliavimą (angl. *Functional Simulation*). Tam VHDL kalba aprašomas aukščiausio hierarchijos lygmens objektas (angl. **top-level entity**), vadinamas **virtualiuoju bandymo stendu** (angl. *test bench*).



9 pav. Testuojamas įrenginys (angl. *design under test*) ir bandymo stendas (angl. *testbench*)

Yra daugybė būdų, kaip kurti virtualiuosius bandymo stendus. Galima išskirti kai kurias jų grupes [4]:

- Tik stimulų stendai** – sudaryti tik iš stimulų generatoriaus ir testuojamo įrenginio. Jokie verifikavimo veiksmai ar palyginimai stende nevykdomi. Paprastai šiuo atveju projektuotojas vizualiniu būdu analizuoją simuliatoriaus pateiktus rezultatus (laikines diagramas) esant duotiems stimulams ir pats sprendžia apie funkcionavimo teisingumą.
- Pilni stendai** – sudaryti iš stimulų generatoriaus, testuojamo įrenginio, laukiamų teisingo funkcionavimo rezultatų aprašo bei jo lyginimo su reakcijomis.
- Specializuoti simuliatoriaus stendai** – tai tokie stendai, kurie kuriami pasinaudojant konkretaus simuliatoriaus palaikomis komandų sistemomis (ne VHDL kalba). Šiuolaikinės simuliavimo priemonės tokios kaip Aldec Active-HDL, Altera Quartus simuliatorius, Symphony EDA ir kitos leidžia vartotojui stimulus formuoti grafinėje projektavimo terpėje naudojantis dialogo langais, kuriuose nurodomi stimulų tipai ir jų parametrai. Šiuo atveju jokių HDL kalbos tekstu aprašomų bandymų standų projektuotojas tiesiogiai nekuria.

Aprašant bandymo stendo objekto sąsają, sakinys *port* nenaudojamas, nes visi signalai yra generuojami stendo viduje.

### 6.1.1 Tik stimulų stendas

Galime panagrinėti kaip kuriamas bandymų stendas paprastam trijų iėjimų loginiam ventiliui *myand3*, kurio VHDL aprašas yra pateiktas 2 programoje. Sudarysime tik stimulus formuojantį bandymo stendą *myand3\_tb*, o reakcijų stebėjimui Xilinx XSim simuliatoriaus formuojamomis laikinėmis diagramomis.

#### 20 progr. (byla myand3\_tb.vhd)

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY myand3_tb IS
END myand3_tb;

ARCHITECTURE behavior OF myand3_tb IS

```

```

-- Testuojamo komponento (Angl. Unit Under Test (UUT)) deklaravimas
COMPONENT myand3_gate
  PORT(A,B,C : in STD_LOGIC;
       Y : out STD_LOGIC);
END COMPONENT;

--Išėjimai
signal inputs : std_logic_vector(2 downto 0) := (others => '0');

--Išėjimai
signal Yout : std_logic;

for all: myand3_gate use entity work.myand3(Structural);

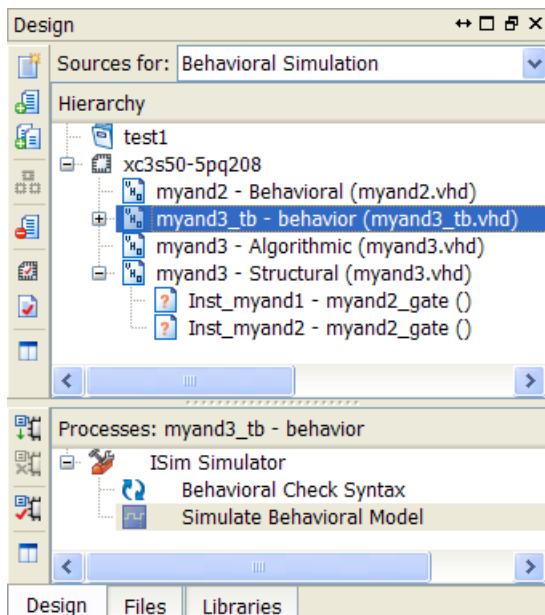
BEGIN

-- Testuojamo modulio egzemplioriaus kūrimas
uut: myand3_gate PORT MAP (
    A => inputs(0),
    B => inputs(1),
    C => inputs(2),
    Y => Yout );

-- Stimulai
stim_proc: process
begin
  inputs <= "000" after 1 ns, -- dvigubose kabutėse dvejetainiai skaičiai
          "001" after 2 ns,
          "010" after 3 ns,
          "011" after 5 ns,
          "100" after 6 ns,
          "101" after 7 ns,
          "110" after 8 ns,
          "111" after 9 ns;
  wait;
end process;
END;

```

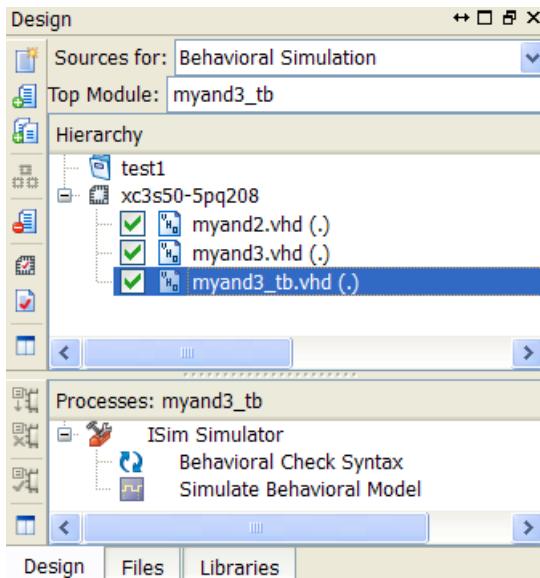
10 pav. parodyta Xilinx ISE projektavimo aplinkoje sukurto projekto bylos.



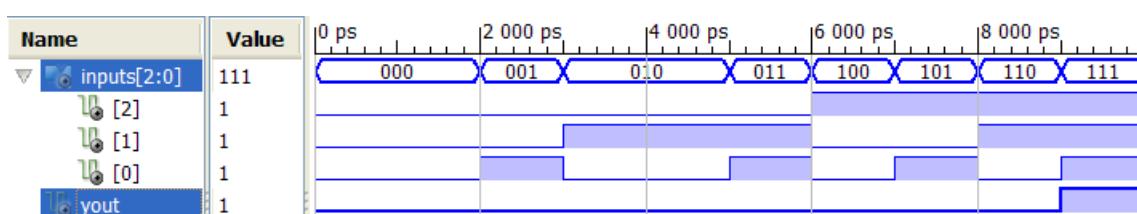
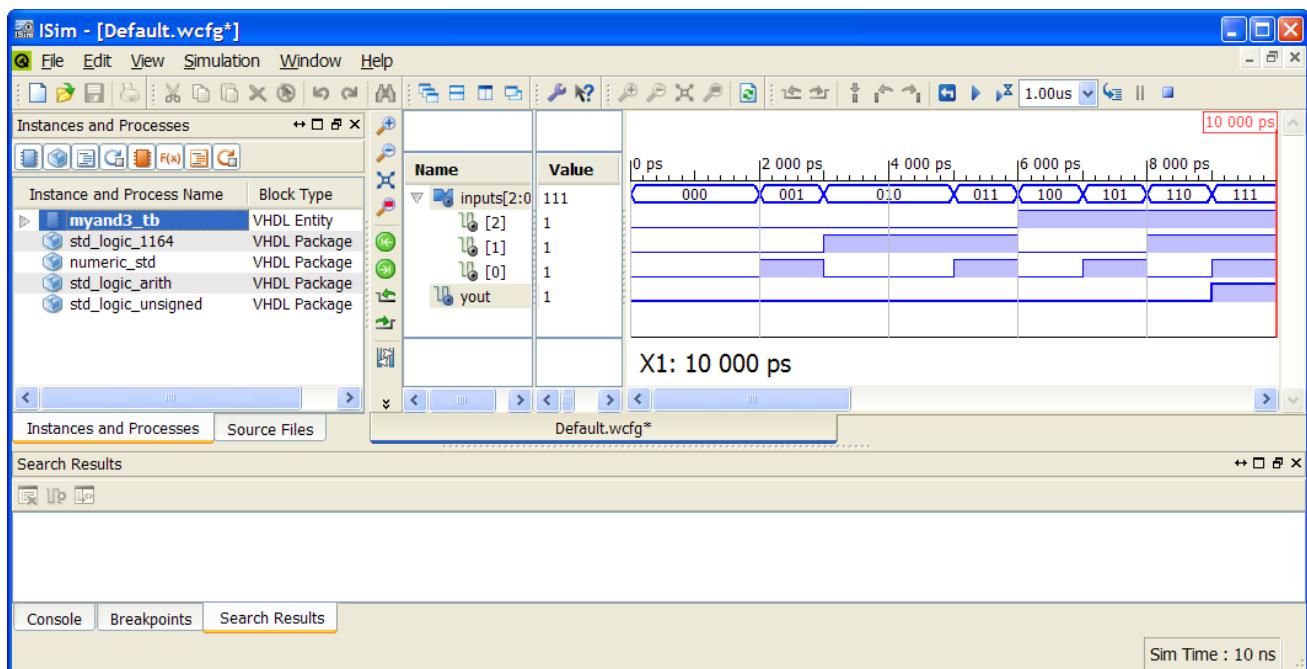
10 pav. Xilinx ISE projektavimo aplinkos Design panelės nustatymai Manual Compile Order režime

Vykdant modeliavimą su Xilinx XSim simuliatoriumi gali reikėti nurodyti projekto bylo kompliliavimo seką (pasirinkus meniu punktą **Project/Manual Compile Order**). 11 pav. parodytas

Design panelės turinys, prieš pradedant simuliavimą. Simuliatorius iškviečiamas du kartus spragtelėjus ant užrašo **Simulate Behavioral Model**. 12 pav. parodyti modeliavimo rezultatai – laikinės stimulų ir reakcijų signalų diagramos.



11 pav. Xilinx ISE projektavimo aplinkos Design panelės nustatymai Manual Compile Order režime



12 pav. Modeliavimo laikinės diagramos (Xilinx XSim simuliatorius)

## 6.1.2 Pilnasis stendas

Galime panagrinėti pavyzdį, iliustruojantį kaip vykdomas projektuojamo įrenginio testavimas, panaudojant pilną virtualų testavimo stendą. 21 programe yra pateiktas dešimtainio skaitiklio aprašas *dut\_counter*. Skaitiklis yra taktuojamas signalu *clock*, o jo einamoji reikšmė yra saugoma *Qout* signale. Skaitiklis skaičiuoja nuo nulio iki devynių, o po devynių jis vėl įgyja reikšmę nulis.

### 21 progr. Dešimtainio skaitiklio aprašas (Projektas FullTB)

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_unsigned.all; -- reikia dėl aritmetinių operacijų
                                -- su STD_LOGIC_VECTOR tipo objektais

entity dut_counter is
    port(
        clock : in STD_LOGIC;
        Qout : out STD_LOGIC_VECTOR(3 downto 0):=X"0"
    );
end dut_counter;

architecture arch_dut_counter of dut_counter is
begin
    process (clock)
        variable Qint : STD_LOGIC_VECTOR (3 downto 0):=X"0";
    begin
        if clock'event and clock = '1' then
            if Qint <9 then
                Qint := Qint+1;
            else
                Qint := "0000";
            end if;
        end if;
        Qout <= Qint;
    end process;
end arch_dut_counter;
```

Nagrinėjant skaitiklio *dut\_counter* bandymų stendo *dut\_counter\_tb* aprašą, reikėtų atkreipti dėmesį į fragmentus, kuriuose:

1. Sukuriamas testuojamo įrenginio (angl. *design under test* arba *unit under test*) egzempliorius;
2. Atidaromi tekstinio formato failai *Test\_Vectors1.txt* (žr. 3 lent.) ir *results.txt*, kurie atitinkamai naudojami stimulų ir laukiamų įrenginio išėjimo reikšmių nuskaitymui bei modeliavimo rezultatų išsaugojimui;
3. Panaudojant VHDL bibliotekos STD paketo TEXTIO funkciją *read* yra skaitomos modeliavimo laiko reikšmės ir laukiamos testuojamo įrenginio išėjimo reikšmės;
4. Panaudojant VHDL bibliotekos STD paketo TEXTIO funkcijas *write* ir *writeline* formuojančios rezultatų failas;
5. Atliekamas laukiamos *expected\_value* ir testuojamo įrenginio išėjime gautos *Qout* reikšmių palyginimas;
6. Funkcijų *assert* ir *report* pagalba, į modeliavimo aplinkos langą yra išvedami pranešimai apie modeliavimo rezultatus (13 pav. galima matyti modeliavimo laikines diagramas ir pranešimus *Console* panelėje);
7. Procesu CLK\_GEN generuojamas stimulus įrenginiui. Šiame pavyzdyme tai yra tik taktinis signalas *clock*.

Tam, kad parodyti kaip aptinkamos testuojamo įrenginio funkcionavimo klaidos, šiame pavyzdyme specialiai buvo įvestas viena klaidinga laukama skaitiklio išėjimo reikšmė laiko momentu 30 ns. Iš tikrujų, praėjus 30 ns nuo modeliavimo pradžios skaitiklio registre einamoji reikšmė turi būti

lygi 3. Vykdant modeliavimą su Xilinx XSim simulatoriumi šis laukiamų ir gautų reikšmių neatitikimas buvo aptiktas ir pranešimai išvesti tiek į rezultatų failą tiek ir į pranešimų panelę *Console*.

Laukiamų ir gautų reikšmių visuma literatūroje dažnai vadinama testiniai vektoriais. Aptikus klaidą, sakoma, kad buvo aptiktas klaudingas testinis vektorius.

## 22 progr. Skaitklio *dut\_counter* bandymų standas

```

library ieee;
use ieee.STD_LOGIC_UNSIGNED.all;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

library STD;
use STD.TEXTIO.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all; -- dėl funkcijos to_hstring

entity dut_counter_tb is
end dut_counter_tb;

architecture TB_ARCHITECTURE of dut_counter_tb is

constant CLK_PERIOD: TIME := 10 ns;
-- Testuojamo komponento deklaravimas
component dut_counter
port(
    clock : in STD_LOGIC;
    Qout : out STD_LOGIC_VECTOR(3 downto 0) );
end component;

-- Stimuli signalai - signalai paduodami į testuojamo įrenginio įėjimus
signal clock : STD_LOGIC:='0';
-- Stebimi signalai - signalai prijungti prie testuojamo įrenginio išėjimų
signal Qout : STD_LOGIC_VECTOR(3 downto 0):=X"0";

file vector_file : TEXT open READ MODE is "Test_Vectors1.txt";
file out_file : TEXT open WRITE_MODE is "results.txt";
begin

-- Testuojamo įrenginio egzemplioriaus kūrimas
UUT : dut_counter
port map (
    clock => clock,
    Qout => Qout
);

STIMULUS: process
variable file_line, error_line : LINE;
variable error_str: string (1 to 40);
variable time_value: real;
variable vector_time: time;
variable expected_value : integer;
variable space: character;
variable value_ok: boolean;

begin -- stimuli procesas
    write(error_line, "    Laikas    Q(lauktas)    --> Q(gautas)");
    writeline(out_file,error_line);
    write(error_line, "=====");
    writeline(out_file,error_line);

    while NOT(endfile(vector_file)) loop
        readline(vector_file,file_line);
        read(file_line,time_value);
        vector_time:=time_value * 1 ns;
        if (now < vector_time) then
            wait for vector_time-now;
        end if;

```

```

read(file_line,expected_value,value_ok);
assert value_ok      REPORT "Blogia išejimo reiksme";

-- formuoti rezultatų failą
write(error_line, now, right, 10, ns);
write(error_line, expected_value,right, 5);
write(error_line, " --> ",right, 15);
write(error_line, to_hstring (ufixed(Qout)),right, 2);
-- naudojama Xilinx ieee_proposed biblioteka fixed_pkg paketas

-- write(error_line, TO_BITVECTOR(Qout),right, 2);
-- naudojama VHDL-2006 TO_HEX_STRING modeliuojant su Active HDL

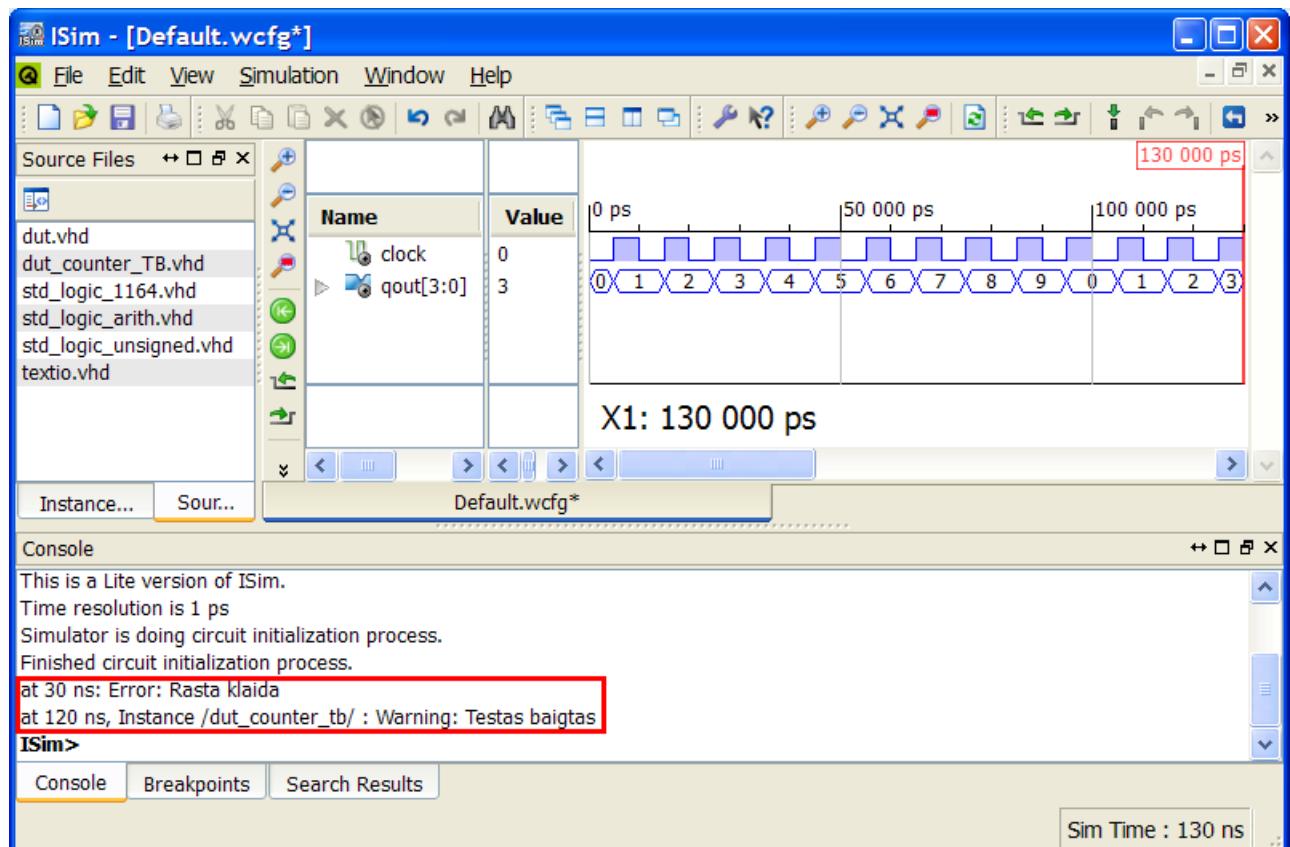
-- palyginti išėjimus
if expected_value/=Qout
    then
        report "Rasta klaida" severity ERROR;
        write(error_line, " (Klaida) ");
    end if;
    writeline(out_file,error_line);

end loop;
assert false REPORT "Testas baigtas" severity WARNING;
FILE_CLOSE(out_file);
wait;
end process; -- stimulų proceso pabaiga

CLK_GEN: process(clock)
begin
    clock<= not clock after CLK_PERIOD/2;
end process;

end TB_ARCHITECTURE;

```



13 pav. Xilinx XSim simuliatoriaus vartotojo sąsaja, atlikus modeliavimą, naudojant bandymų stenda dut\_counter\_tb

### 3 lent. Tekstiniai stimulų ir modeliavimo rezultatų failai

Test_Vectors1.txt	results.txt
0 0	Laikas Q(lauktas) --> Q(gautas)
10 1	=====
20 2	0 ns 0 --> 0.
30 2	10 ns 1 --> 1.
40 4	20 ns 2 --> 2.
50 5	30 ns 2 --> 3. (Klaida)
60 6	40 ns 4 --> 4.
70 7	50 ns 5 --> 5.
80 8	60 ns 6 --> 6.
90 9	70 ns 7 --> 7.
100 0	80 ns 8 --> 8.
110 1	90 ns 9 --> 9.
120 2	100 ns 0 --> 0.
	110 ns 1 --> 1.
	120 ns 2 --> 2.

## 6.2 Simuliavimo tikslumo nustatymas

VHDL kalboje skirtingai nuo Verilog nėra direktyvos timescale, kurios pagalba galima būtų programos tekste nustatyti modeliavimo žingsnį ir tikslumą. Vivado Xsim simuliatoriaus atveju galima naudoti komandinės eilutės opciją **-timeprecision\_vhdl <time\_precision>**, kuria galima pakeisti pagal nutylėjimą naudojamą 1 ps modeliavimo tikslumą.

<https://www.xilinx.com/support/answers/50577.html>

<b>-timeprecision_vhdl arg</b>	Specify time precision for vhdl designs. Default: 1ps.	xelab
--------------------------------	---	-------

Vivado Design Suite User Guide Logic Simulation UG900 (v2018.3) December 14, 2018

## 6.3 Modeliavimui skirtos kalbos konstrukcijos

### 6.3.1 Stimulų generavimo pavyzdžiai

#### 6.3.1.1 Daugiabičio skaičiaus keitimas panaudojant ciklą for-loop

Žemiau esančiame bandymų stende parodyta kaip std\_logic\_vector tipo objekto kintamajam sugeneruoti stimulą, kuris didėja po 1 po pasirinkto vėlinimo panaudojant ciklą for loop.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_MyCalculator is
generic( N : integer := 5);
end tb_MyCalculator;

architecture tb of tb_MyCalculator is

component MyCalculator
    port (A : in std_logic_vector (N-1 downto 0);
          B : in std_logic_vector (N-1 downto 0);
          SUM : out std_logic_vector (N downto 0));
end component;
```

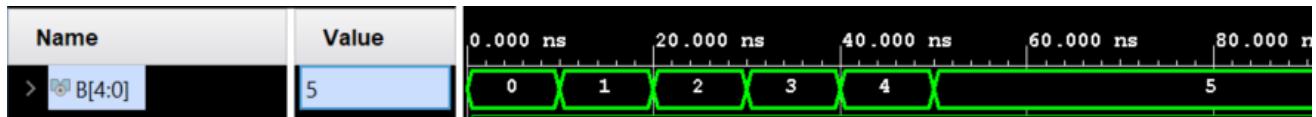
```

signal A    : std_logic_vector (N-1 downto 0);
signal B    : std_logic_vector (N-1 downto 0);
signal SUM  : std_logic_vector (N downto 0);

begin
  dut : MyCalculator
  port map (A    => A,
            B    => B,
            SUM => SUM);

  stimuli : process is
    variable i: integer;
  begin
    A <=std_logic_vector(to_unsigned(2,A'length));
    for i in 0 to 5 loop
      B<=std_logic_vector(to_unsigned(i, B'length));
      wait for 10 ns;
    end loop;
    wait;
  end process;
end tb;

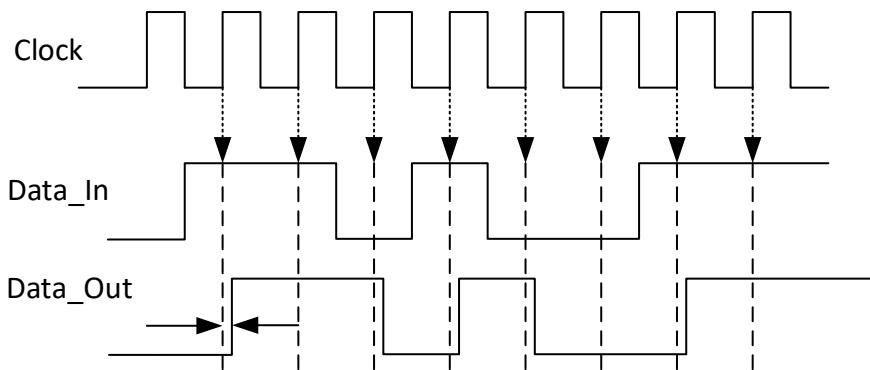
```



14 pav. Stimulo generavimo panaudojant ciklą laikinės diagramos

### 6.3.1.2 Lygiagrečių stimulo duomenų generavimas bandymų stende

Atskiruose procesuose generuojame taktinių impulsų signalą ir įėjimo duomenų signalą. Atkreipkite dėmesį, kad duomenys keičiami krentančio taktinių impulsų fronto metu, tam kad į žadinamą įrenginij taktinio impulso kylantis frontas atitiktų momentą, kai duomenys jau yra stabilūs (nusistovėję).



15 pav. Stimulo generavimo pavyzdys

Objekto aprašas

```
-- (F:/Works/Xilinx/TestBechExample/MyDesign2023/MyDesign2023.srccs/
sources_1/new/mydesign.vhd) -----
library IEEE;
```

```

use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- aritmetinėms operacijoms

entity mydesign is
  port (
    Clock : in STD_LOGIC;
    InDataValid : in STD_LOGIC;
    InData : in STD_LOGIC_VECTOR ( 7 downto 0 );
    OutData : out STD_LOGIC_VECTOR ( 7 downto 0 );
    OutDataValid : out STD_LOGIC
  );
end mydesign;

architecture Behavioral of mydesign is
begin

main: process (Clock)
begin
  if (rising_edge(Clock)) then
    if (InDataValid='1') then
      OutData<= InData+x"2";
      OutDataValid<='1';
    end if; -- if (InDataValid='1')
  end if; -- if (rising_edge(Clock)) then
end process main;

end Behavioral;

```

Bandymų stendo aprašas

```

-- (F:/works/Xilinx/TestBechExample/MyDesign2023/mydesign_tb.vhd) -----
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- aritmetinėms operacijoms

entity mydesign_tb is
-- Port ( );
end mydesign_tb;

architecture Behavioral of mydesign_tb is

constant PERIOD : time := 100 ns;
signal Clock: STD_LOGIC:='0';
signal InDataValid: STD_LOGIC:='0';
signal InData : STD_LOGIC_VECTOR ( 7 downto 0 ):="00000000";
signal OutData : STD_LOGIC_VECTOR ( 7 downto 0 ):="00000000";
signal OutDataValid : STD_LOGIC:='0';

component mydesign is
  port (
    Clock : in STD_LOGIC;
    InDataValid : in STD_LOGIC;
    InData : in STD_LOGIC_VECTOR ( 7 downto 0 );
    OutData : out STD_LOGIC_VECTOR ( 7 downto 0 );
    OutDataValid : out STD_LOGIC
  );
end component mydesign;
begin
mydesign_i: component mydesign

```

```

port map (
    Clock => Clock,
    InData(7 downto 0) => InData(7 downto 0),
    InDataValid => InDataValid,
    OutData(7 downto 0) => OutData(7 downto 0),
    OutDataValid => OutDataValid
);

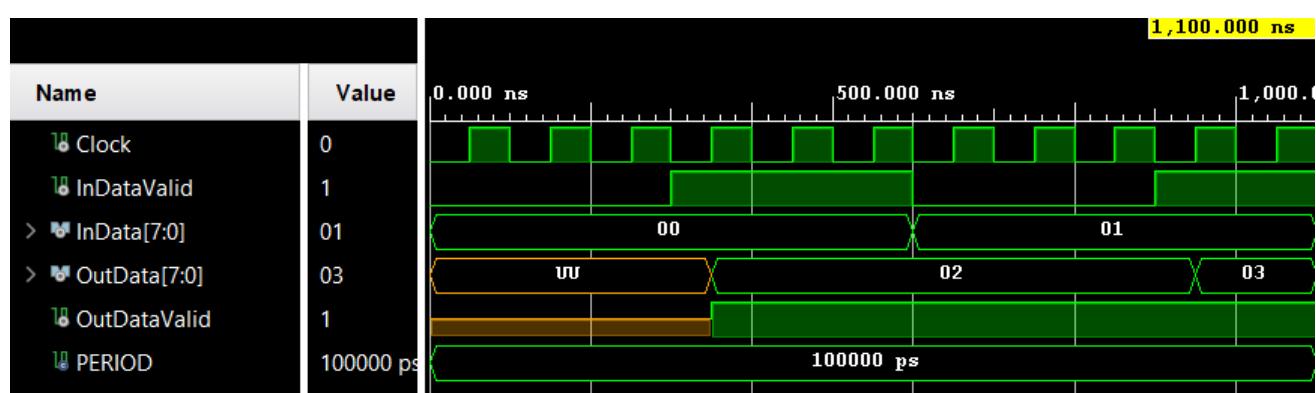
clk_proc: process
begin
    Clock <= '0';
    wait for PERIOD/2;
    Clock <= '1';
    wait for PERIOD/2;
end process clk_proc;

stimuli_proc: process (Clock)
variable div: integer :=0;
begin
    if (falling_edge(Clock)) then
        if div=5 then
            div:=0;
            InData<=InData+x"1";
            InDataValid<='0'; -- data valid at the input
        else
            div:=div+1;
            if div=3 then
                InDataValid<='1'; -- model that data is not yet valid at the input
                end if;
            end if;
        end if;
    end process stimuli_proc;

end Behavioral;

```

Skaitmeninio įrenginio modeliavimo laikinių diagramų pavyzdys, naudojant aukščiau pateiktą bandymų stendo aprašą.



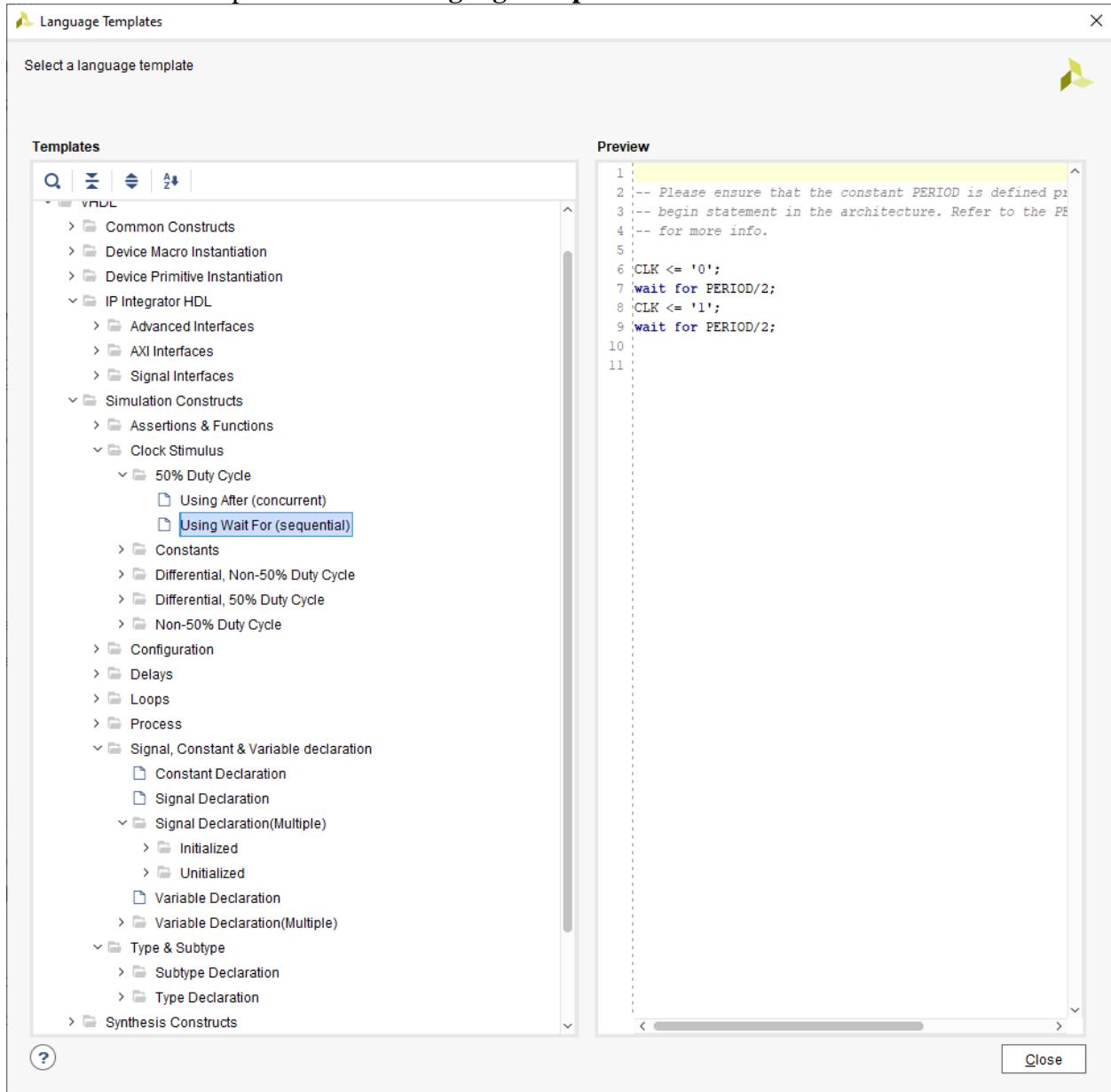
### 6.3.1.3 Nuoseklių stimulo duomenų generavimas bandymų stende

Šiuo atveju stendo apraše arba išoriniame stimulų faile surašomi baitai, kurie stende nuskaitomi ir nuosekliai (pabičiu) perduodami į testuojamą įrenginį. Šis atvejis yra patogus, kai reikia į testuojamą įrenginio nuoseklaus jėjimo, pvz., SPI sąsajos MISO arba MOSI, prie vadus perduoti duomenų paketą.

Pavyzdys yra pateiktas semestro darbo metodiniuose nurodymuose.

### 6.3.2 Šablonų sąrašas Vivado aplinkoje

Pasiekiamas meniu punktu **Tools /Language templates**.



## 6.4 VHDL simuliatoriai

Aldec Active-HDL/Riviera

Mentor Graphics ModelSim

**Symphony EDA VHDL Simili (<http://www.symphonyeda.com>)**

Synopsys VCS

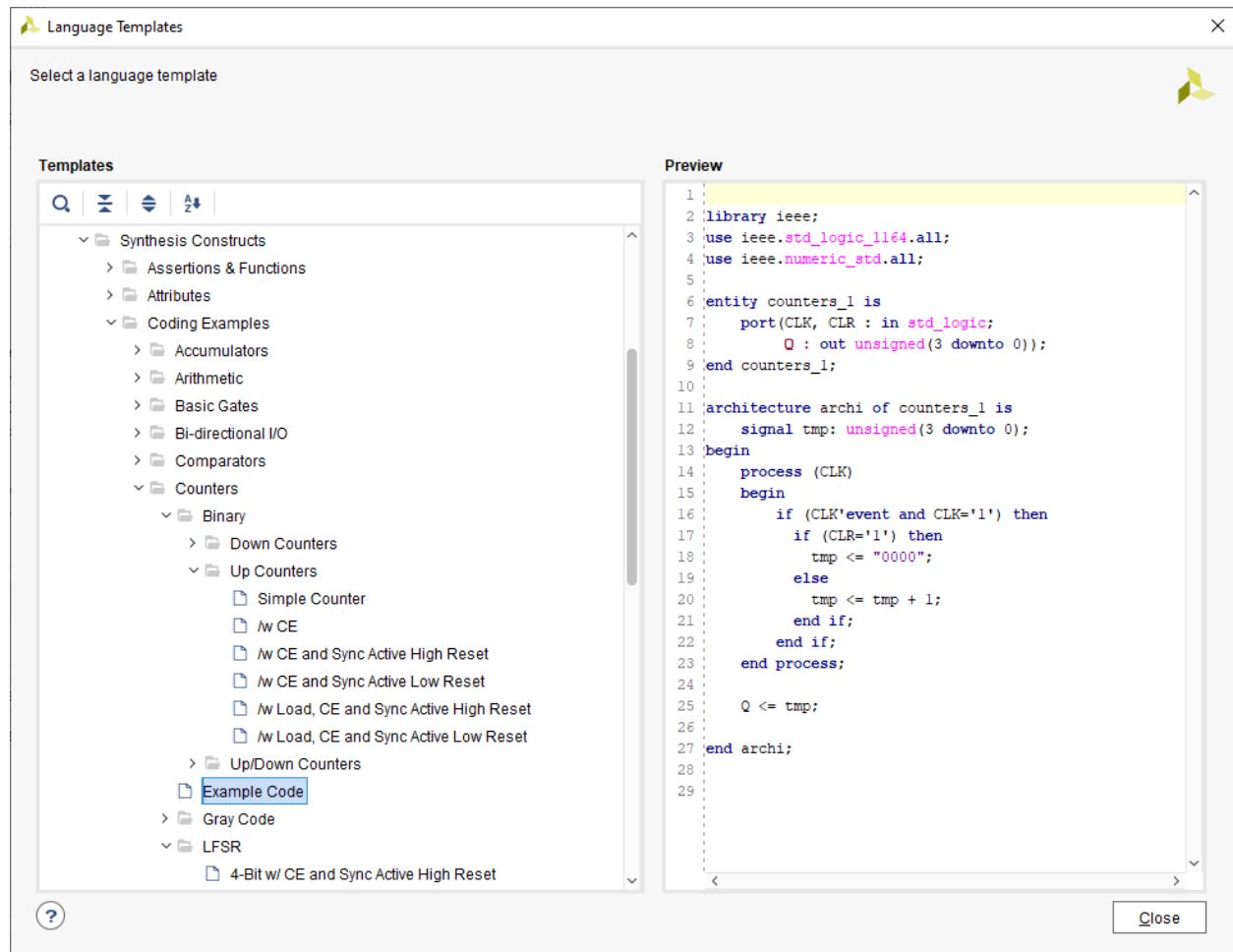
**Xilinx ISE ISim/ Xilinx Vivado XSim**

Intel (Altera) Quartus II

**Altium Designer**

## 7 Sintezuojamos logikos aprašai

Modeliavimo būdu įsitikinus projektuojamu skaitmeninio įrenginio funkcionavimo teisingumu (atlikus verifikavimą) pereinama prie realaus fizinio įrenginio, aprašyto VHDL modeliu, kūrimo. Šis procesas vadinamas sinteze. Reikia pastebėti, kad ne visos VHDL kalbos konstrukcijos yra naudojamos sintezei atliskti, o tik jų dalis vadinama sintezei skirtu poaibiu.



### 7.1 Kombinacinės logikos aprašai

Kombinacinei logikai yra priskiriami: sumatoriai (angl. adders), skirtumo skaičiuokliai (angl. subtractors), komparatoriai (angl. comparator), multipleksoriai (angl. multiplexors arba MUX), demultipleksoriai (angl. demultiplexors arba DEMUX), enkoderiai (šifratoriai) (angl. encoder), dešifratoriai (angl. decoder), kodo keitikliai (angl. code converters). Kombinacinės logikos įrenginiai aprašyti [https://www.electronics-tutorials.ws/combination/comb\\_1.html](https://www.electronics-tutorials.ws/combination/comb_1.html). Galima panagrinėti įvairius sintezei skirtus kombinacinės logikos (angl. *combinational logic*) skaitmeninių įrenginių aprašus VHDL kalba.

#### 7.1.1 Loginiai ventiliai

##### 23 progr. Keturių jėjimų loginės sandaugos (4IR) aprašas

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity my_4_and is
    port (

```

```

A0: in std_logic;
A1: in std_logic;
A2: in std_logic;
A3: in std_logic;
Y: out std_logic);
end my_4_and;

architecture my_4_and of my_4_and is
begin
    Y <= A0 and A1 and A2 and A3;
end my_4_and;

```

### 7.1.2 Multipleksorius

#### 24 progr. Multipleksoriaus iš 4 į 1 aprašas

```

library ieee;
use ieee.std_logic_1164.all;

entity my_mux is
    port (
        -- 4 -> 1 multipleksorius
        SEL : in STD_LOGIC_VECTOR(0 to 1);
        S0, S1, S2, S3 :in STD_LOGIC;
        MUX_OUT : out STD_LOGIC
    );
end my_mux;

architecture my_mux_arch of my_mux is
begin
    process (SEL, S0, S1, S2, S3)
begin
    case SEL is
        when "00" => MUX_OUT <= S0;
        when "01" => MUX_OUT <= S1;
        when "10" => MUX_OUT <= S2;
        when "11" => MUX_OUT <= S3;
        when others => MUX_OUT <= 'X';
    end case;
end process;
end architecture my_mux_arch;

```

### 7.1.3 Prioritetinis enkoderis

#### 4 lent. Prioritetinio enkoderio teisingumo lentelė

Iejimas X(4:1)	Išėjimas out_code(2:0)	Išėjimas out_code(2:0) dešimtainis
1xxx	100	4
01xx	011	3
001x	010	2
0001	001	1
0000	000	0

#### 25 progr. Prioritetinio enkoderio aprašas [1]

```

library ieee;
use ieee.std_logic_1164.all;
entity prio_encoder is
    port(
        x: in std_logic_vector(4 downto 1);
        out_code: out std_logic_vector(2 downto 0)
    );

```

```

-- Teisingumo lentelė
-----
-- | X(4) X(3) X(2) X(1) | out_code |
-- | 1      x      x      x | 100      |
-- | 0      1      x      x | 011      |
-- | 0      0      1      x | 010      |
-- | 0      0      0      x | 001      |
-- | 0      0      0      0 | 000      |
-- x - bet kuri loginė reikšmė (0 arba 1)

end prio_encoder;

architecture cond_arch of prio_encoder is
begin
    out_code <= "100" when (x(4)='1') else
        "011" when (x(3)='1') else
        "010" when (x(2)='1') else
        "001" when (x(1)='1') else
        "000";
end cond_arch;

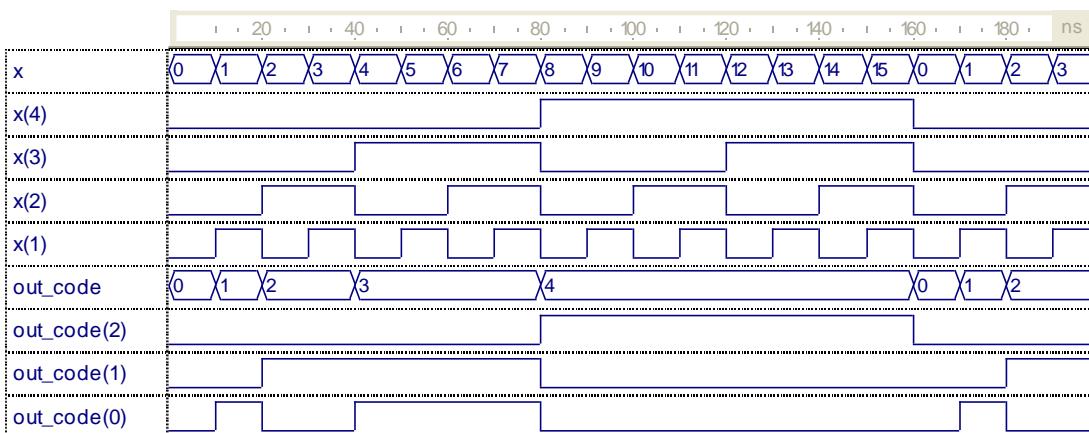
-- architektūra su select sakiniu
architecture sel_arch of prio_encoder is
begin
    with x select
        out_code <= "100" when "1000"|"1001"|"1010"|"1011"|
                            "1100"|"1101"|"1110"|"1111",
                    "011" when "0100"|"0101"|"0110"|"0111",
                    "010" when "0010"|"0011",
                    "001" when "0001",
                    "000" when others;    -- r="0000"
end sel_arch;

-- architektūra su if sakiniu
architecture if_arch of prio_encoder is
begin
    process(x)
    begin
        if (x(4)='1') then
            out_code <= "100";
        elsif (x(3)='1') then
            out_code <= "011";
        elsif (x(2)='1') then
            out_code <= "010";
        elsif (x(1)='1') then
            out_code <= "001";
        else
            out_code <= "000";
        end if;
    end process;
end if_arch;

-- architektūra su case sakiniu
architecture case_arch of prio_encoder is
begin
    process(x)
    begin
        case x is
            when "1000"|"1001"|"1010"|"1011"|
                "1100"|"1101"|"1110"|"1111" =>
                out_code <= "100";
            when "0100"|"0101"|"0110"|"0111" =>
                out_code <= "011";
            when "0010"|"0011" =>
                out_code <= "010";
            when "0001" =>
                out_code <= "001";
            when others =>
                out_code <= "000";
        end case;
    end process;
end;

```

```
end case_arch;
```



16 pav. Prioritetinio enkoderio modeliavimo laikinės diagramos

### 7.1.4 7 segmentų indikatoriaus dekoderis

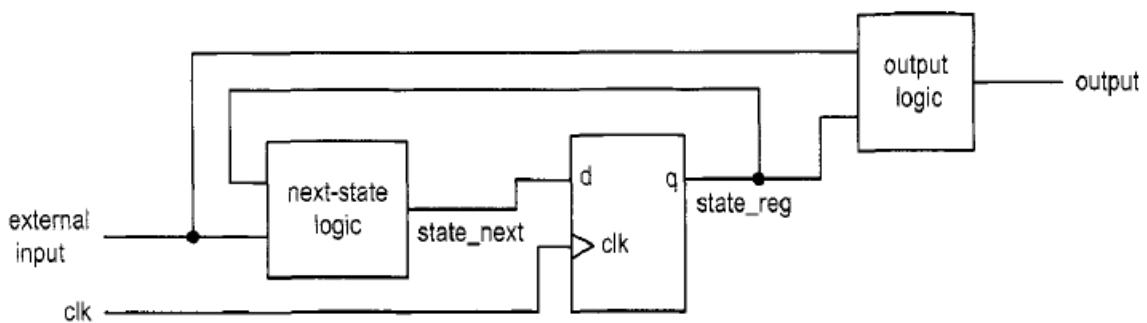
Formuoja 7 segmentų LED indikatorius skaitmenų simbolių atvaizdus, atitinkančius dekoderio iėjime veikiantį binarinį skaitmens kodą. Dekoderio išėjimo linijos jungiamos prie LED indikatoriaus segmentų.

```
library ieee;
use ieee.std_logic_1164.all;
entity Bin2LED is
    port (Data:in std_logic_vector(3 downto 0);
          Output:out std_logic_vector(6 downto 0));
end Bin2LED;
architecture RTL of Bin2LED is
begin
    process (Data)
    begin
        case Data is
            when "0000"=>Output<="1111110"; -- "0"
            when "0001"=>Output<="0110000"; -- "1"
            when "0010"=>Output<="1101101"; -- "2"
            when "0011"=>Output<="1111001"; -- "3"
            when "0100"=>Output<="0110011"; -- "4"
            when "0101"=>Output<="1011011"; -- "5"
            when "0110"=>Output<="X011111"; -- "6"
            when "0111"=>Output<="1110000"; -- "7"
            when "1000"=>Output<="1111111"; -- "8"
            when "1001"=>Output<="111X011"; -- "9"
            when others=>Output<="XXXXXXX"; -- neleistina iėjimo kombinacija
        end case;
    end process;
end RTL;
```

## 7.2 Nuosekliosios logikos aprašai

Nuosekliosios logikos (angl. *sequential logic*) skaitmeniniai įrenginiai vadinami įrenginiai su atmintimi [1]. Atminties elemente saugoma vidinė grandinės būsena. Skirtingai nuo kombinacinės logikos įrenginių, kuriuose išėjimas yra tik iėjimų funkcija, nuosekliosios logikos įrenginių išėjimas priklauso ne tik nuo iėjimų, bet ir nuo vidinės įrenginio būsenos. Projektuojant nuosekliuosius įrenginius dažniausiai naudojamos synchroninės grandinės. Visi atminties elementai valdomi (synchronizuojami) globaliu taktiniu impulsu signalu (angl. *clock*), o duomenys priimami ir

išsaugojami ties kylančiu arba krentančiu šio taktinio signalo frontu. Tai leidžia projektuotojui atskirti atminties elementus nuo likusios grandinės dalies, taip palengvinant projektavimą.



17 pav. Sinchroninės sistemos pavyzdys

Nuosekliajai logikai yra priskiriami: ([https://www.electronics-tutorials.ws/sequential/seq\\_1.html](https://www.electronics-tutorials.ws/sequential/seq_1.html)) trigeriai (angl. triggers), multivibratoriai (angl. multivibrators), skaitikliai (angl. counters; binariniai, dešimtaininiai, žiediniai ir kt.), postūmio registrai (angl. shift registers).

### 7.2.1 Bistabilieji elementai

Pats elementariausias atminties elementas, kuris gali saugoti minimalų atminties kiekį bitą (gali įgyti reikšmes 0 ir 1) yra realizuojamas D trigeriu (angl. D Flip-Flop). Žemiau pateikti trys analogiški D trigerio aprašai.

#### 26 progr. Bistabilaus elemento D trigerio (flip-flop) aprašas

```

library ieee;
use ieee.std_logic_1164.all;

entity d_ff is
  port(
    clk: in std_logic;
    D: in std_logic;
    Q: out std_logic
  );
end d_ff;

architecture d_arch of d_ff is
begin
  process(clk)
  begin
    if (clk'event and clk='1') then -- naudojamas atributas `event
      Q <= D;
    end if;
  end process;
end d_arch;

architecture d2_arch of d_ff is
begin
  process
  begin
    wait until (clk'event and clk='1') - alternatyvi aprašo forma
    Q <= D;
  end process;
end d2_arch;

architecture d_arch3 of d_ff is
begin
  process(clk)
  begin

```

```

if (rising_edge(clk)) then -- naudojamas rising_edge užrašas
    Q <= D;
end if;
end process;
end d_arch3;

```

### 27 progr. Fiksatoriaus (angl. flip-flop) aprašas

```

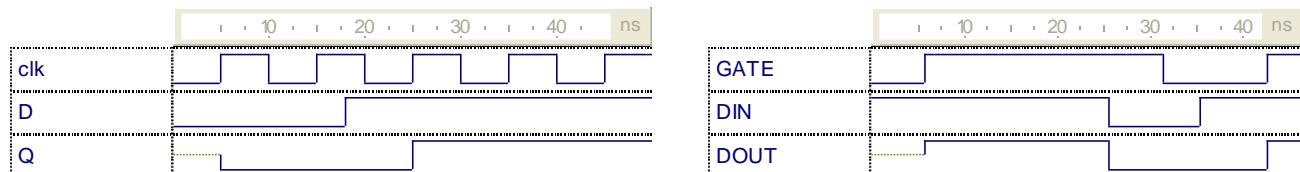
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity latch is -- Latch (fiksatorius)
port (
    GATE: in STD_LOGIC;
    DIN: in STD_LOGIC;
    DOUT: out STD_LOGIC);
end latch;

architecture latch of latch is
begin

process (GATE, DIN)
begin
    if GATE='1' then --GATE aktyvus aukštasis lygis
        DOUT <= DIN;
    end if;
end process;
end latch;

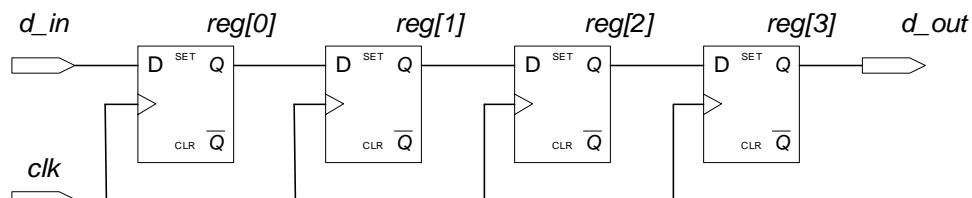
```



18 pav. D trigerio (kairėje) ir fiksatoriaus (dešinėje) laikinės diagramos

### 7.2.2 Postūmio registrai

Postūmio registras yra viena iš plačiausiai naudojamų diskretinės elektronikos struktūrų. Šie postūmio registrai naudojami nuoseklaus kodo konvertavimui į lygiagretę ir atvirkšciai, o papildyti grižtamaisiais ryšiais tarp skilčių naudojami atsitiktinių skaičių generatorių, skaitiklių ir t.t. realizavimui. Paties paprasčiausio sinchroninio (taktojamas signalu *clk*) postūmio registro schema parodyta 19 pav. Jį sudaro nuosekliai sujungti D trigeriai. Iejimo duomenys srautas paduodamas į *reg[0]* trigerio įėjimą D, o išėjimo duomenys gaunami *reg[3]* trigerio išėjime Q. Be to, postūmio registro visus D trigerius nuskaičius kaip daugiabitį signalą gautume nuoseklaus bitų srauto konvertavimą į lygiagretę formatą.



19 pav. Keturių bitų postūmio registras

28 programe pateiki 4 bitų postūmio registro (objekto *shift\_reg*) keturi aprašai (*arch1*, *arch2*, *arch3* ir *arch4* architektūros). Aprašai *arch1* ir *arch4* skiriasi tuo, kad priskyrimas išėjimo signalui *d\_out* atliekamas skirtingose vietose. Iš 20 pav. pateiktų modeliavimo laikinių diagramų galime matyti,

kad dėl to veikimas bus šiek tiek skirtinas – *arch4* atveju *d\_out* įgyja *reg[3]* reikšmę iš karto, o *arch1* atveju – tik atėjus dar vienam papildomam taktiniam impulsui.

Aprašas *arch2* demonstruoja ciklo panaudojimą, o *arch3* konkatenacijos panaudojimą.

#### 28 progr. Postūmio registro realizavimo architektūros (E:\Works\Vivado\PLI\2023\ShiftReg)

```

library ieee;
use ieee.std_logic_1164.all;

entity shift_reg is
    generic(N: integer := 4); -- parametrizavimui
    port(
        clk, reset: in std_logic;
        d_in: in std_logic;
        d_out: out std_logic
    );
end shift_reg;

-- pirmoji architektura (daugelio priskyrimu panaudojimas)
architecture arch1 of shift_reg is
    signal r_reg: std_logic_vector(N-1 downto 0) := x"0";
begin
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
            d_out <= '0';
        elsif (clk'event and clk='1') then
            r_reg(0) <= d_in;
            r_reg(1) <= r_reg(0);
            r_reg(2) <= r_reg(1);
            r_reg(3) <= r_reg(2);

            d_out <= r_reg(3); -- output assignment
        end if;
    end process;
end arch1;

-- trečioji architektura (for ciklo panaudojimas)
architecture arch2 of shift_reg is
    signal r_reg: std_logic_vector(N-1 downto 0) := x"0";
begin
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
            d_out <= '0';
        elsif (clk'event and clk='1') then

            r_reg(0) <= d_in;

            for i in 0 to N-2 loop
                r_reg(i+1) <= r_reg(i);
            end loop;

            d_out <= r_reg(3); -- output assignment
        end if;
    end process;
end arch2;

-- ketvirtoji architektura (konkatenacijos panaudojimas)
architecture arch3 of shift_reg is
    signal r_reg: std_logic_vector(N-1 downto 0) := x"0";
begin
    process(clk,reset)
    begin
        if (reset='1') then
            r_reg <= (others=>'0');
        elsif (clk'event and clk='1') then
            d_out <= r_reg(3);
            r_reg <= r_reg(N-2 downto 0) & d_in;
        end if;
    end process;
end arch3;

-- antroji architektura (daugelio priskyrimu panaudojimas)
architecture arch4 of shift_reg is

```

```

    signal r_reg: std_logic_vector(N-1 downto 0) := X"0";
begin
  process(clk,reset)
  begin
    if (reset='1') then
      r_reg <= (others=>'0');
      --d_out <= '0'; -- negali buti daugiau negu viename process sakinyje
    elsif (clk'event and clk='1') then
      r_reg(0) <= d_in;
      r_reg(1) <= r_reg(0);
      r_reg(2) <= r_reg(1);
      r_reg(3) <= r_reg(2);

    end if;
  end process;

  d_out <= r_reg(3); -- priskyrimas lygiagrečiai process sakiniui
end arch4;

```

**29 progr. Postūmio registro modeliavimo bandymų stendas**

```

library ieee;
use ieee.std_logic_1164.all;

entity tb_shift_reg is
end tb_shift_reg;

architecture tb of tb_shift_reg is

  component shift_reg
    port (clk : in std_logic;
          reset : in std_logic;
          d_in : in std_logic;
          d_out : out std_logic);
  end component;

  for all: shift_reg use entity work.shift_reg(arch1);

  signal clk    : std_logic;
  signal reset : std_logic;
  signal d_in   : std_logic;
  signal d_out  : std_logic;

  constant TbPeriod : time := 10 ns;
  signal TbClock : std_logic := '0';
  signal TbSimEnded : std_logic := '0';

begin
  begin
    dut : shift_reg
    port map (clk    => clk,
              reset  => reset,
              d_in   => d_in,
              d_out  => d_out);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
    clk <= TbClock;

    stimuli : process
    begin
      reset <= '0';

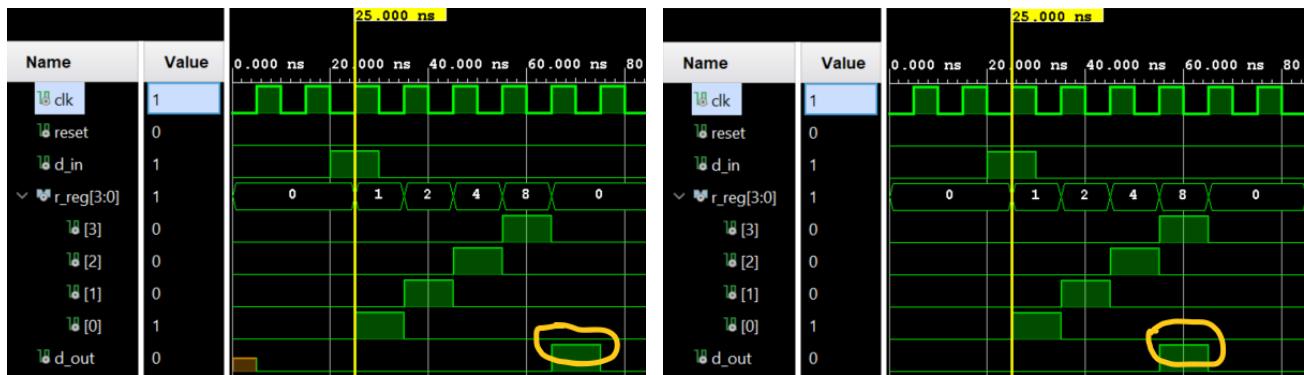
      d_in <= '0';
      wait for 20 ns;
      d_in <= '1';
      wait for 10 ns;
      d_in <= '0';

      wait for 15 * TbPeriod;

      -- Stop the clock and hence terminate the simulation
      TbSimEnded <= '1';
      wait;
    end process;
  end;

```

```
end tb;
```

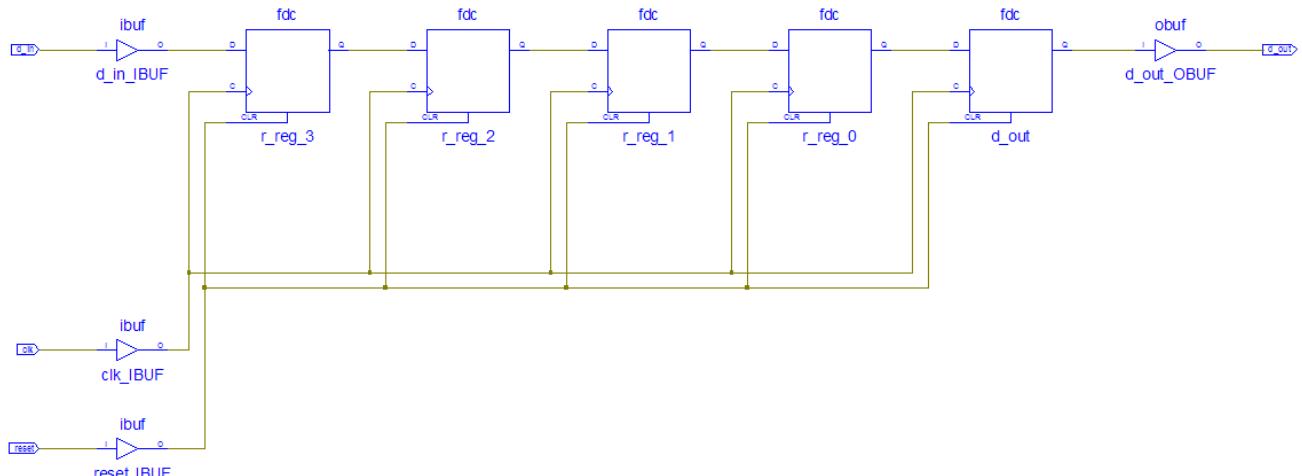


20 pav. Kairėje arch1, arch2, arch3, o dešinėje arch4 architektūros postūmio registro funkcinio modeliavimo laikinės diagramos (atkreipkite dėmesį į skirtumus dėl d\_out priskyrimo)

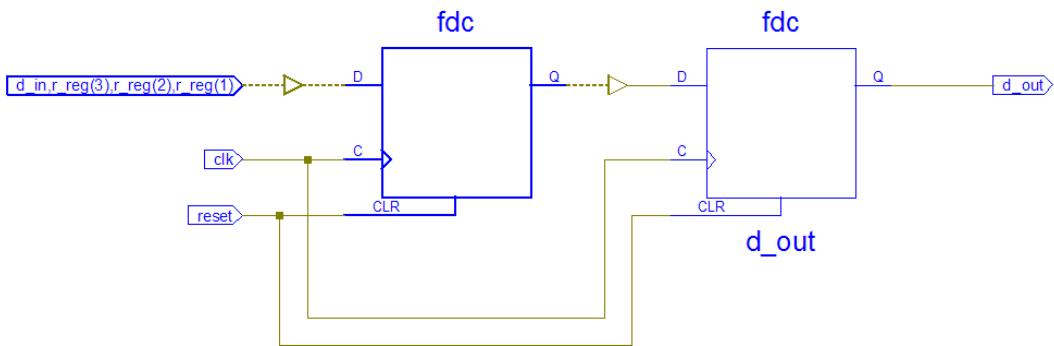
Galima panagrinėti kaip sintezuoojamas architektūros *arch1* postūmio registras *shift\_reg* bei kokie vidiniai FPGA resursai tam panaudojami, priklausomai nuo konkrečios FPGA komponentų šeimos. Technologinio lygmens schemas, kurias atlikę sintezę gali pavaizduoti projektavimo paketai tokie kaip Xilinx ISE (reikia pasirinkti meniu punktą **Tools/Schematics Viewer/Technology...**) arba Altera Quartus (reikia pasirinkti meniu punktą **Tools/Netlist Viewers/Technology Map Viewer...**) yra parodyti toliau, o jų sąrašas su paaiškinimais pateiktas 5 lentelėje. Galima matyti, kad daugeliu atveju sukuriama nuosekliai sujungtų keturių registrų, taktuojamų tuo pačiu taktinių impulsų signalu *clk*, grandinėlė, kuri ir realizuos postūmio registrą. Tačiau 24 pav. parodytoje schemaje jau sunkiau būtų atsekti klasikinio postūmio registro struktūrą. Taip yra todėl, kad pastūmio registrui realiuozti Xilinx Spartan-3 šeimos komponente buvo panaudota peržiūros lentelė (angl. LUT).

##### 5 lent. Technologinio lygmens grandinės, atlikus postūmio registro sintezę

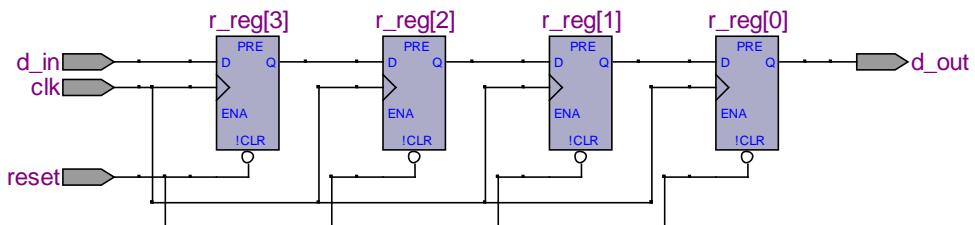
Pav. nr.	FPGA gamintojas	FPGA komponentas	Sintezės priemonės
21 pav.	Xilinx	Cool Runner XC2C128	Xilinx ISE 11.1 WebPack
23 pav.	Altera	Cyclone II EP2C5A	Quartus II 9.0 Web Edition
24 pav.	Xilinx	Spartan 3E XA3S500E	Xilinx ISE 11.1 WebPack



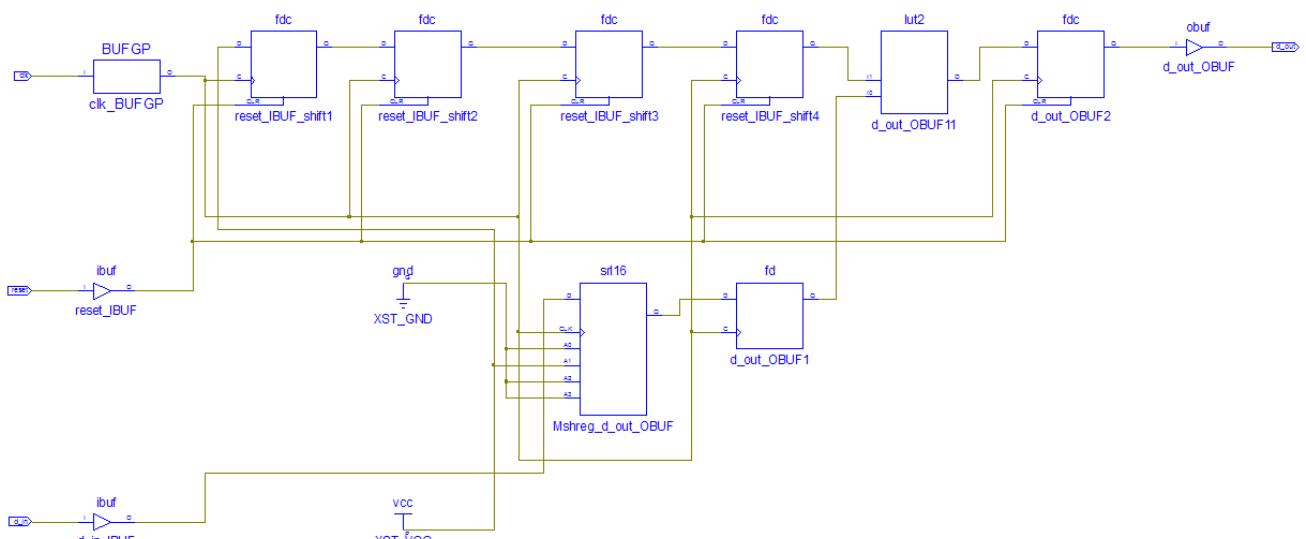
21 pav. Postūmio registro shift\_reg (28 progr.) architektūros arch1 technologinė grandinė Technology Schematics (komponetas Xilinx Cool Runner XC2C128)



22 pav. arch1 tarpregistrinių perdavimų lygmens schema (RTL schematics) (komponetas Xilinx CoolRunner2 CPLD XC2C128)

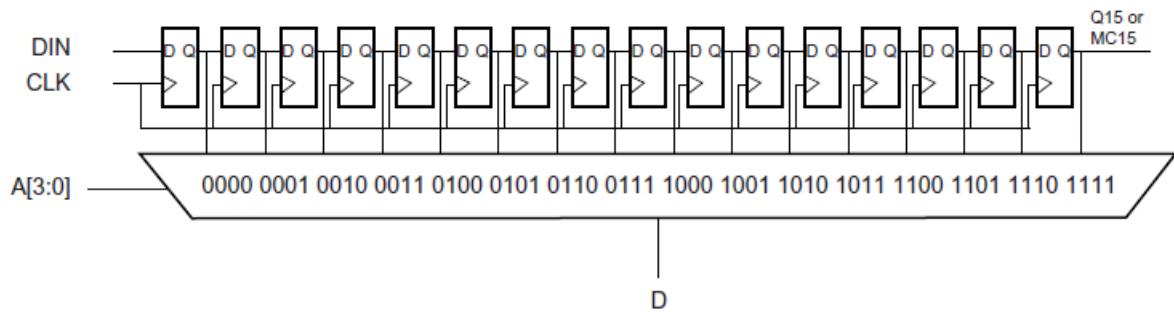


23 pav. arch1 Technology Schematics (komponetas Altera Cyclone II EP2C5A)

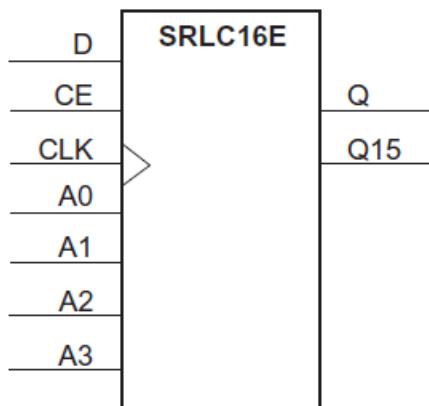


24 pav. arch1 Technology Schematics (komponetas Xilinx Spartan 3E XA3S500E)

Peržiūros lentelės (angl. LUT – *Look Up Table*) yra vienas iš fundamentaliai struktūriniai komponentai, kurie naudojami praktiškai visų FPGA gamintojų. Xilinx Spartan-3 šeimos FPGA peržiūros lentelės gali būti konfigūruojamos darbui postūmio registro režimu (žr. 25 pav.). Maksimalus vieno tokio postūmio registro ilgis yra 16, tačiau panaudojant adreso linijas ilgis gali būti parenkamas [8]. Postūmio operacija yra atliekama sinchroniškai taktinio signalo impulsams. Xilinx primityvas SRL16 (Shift Register LUT 16-bit) yra parodytas 26 pav. Jis kaip tik ir galime matyti technologinio lygmens grandinėje parodytoje 24 pav.

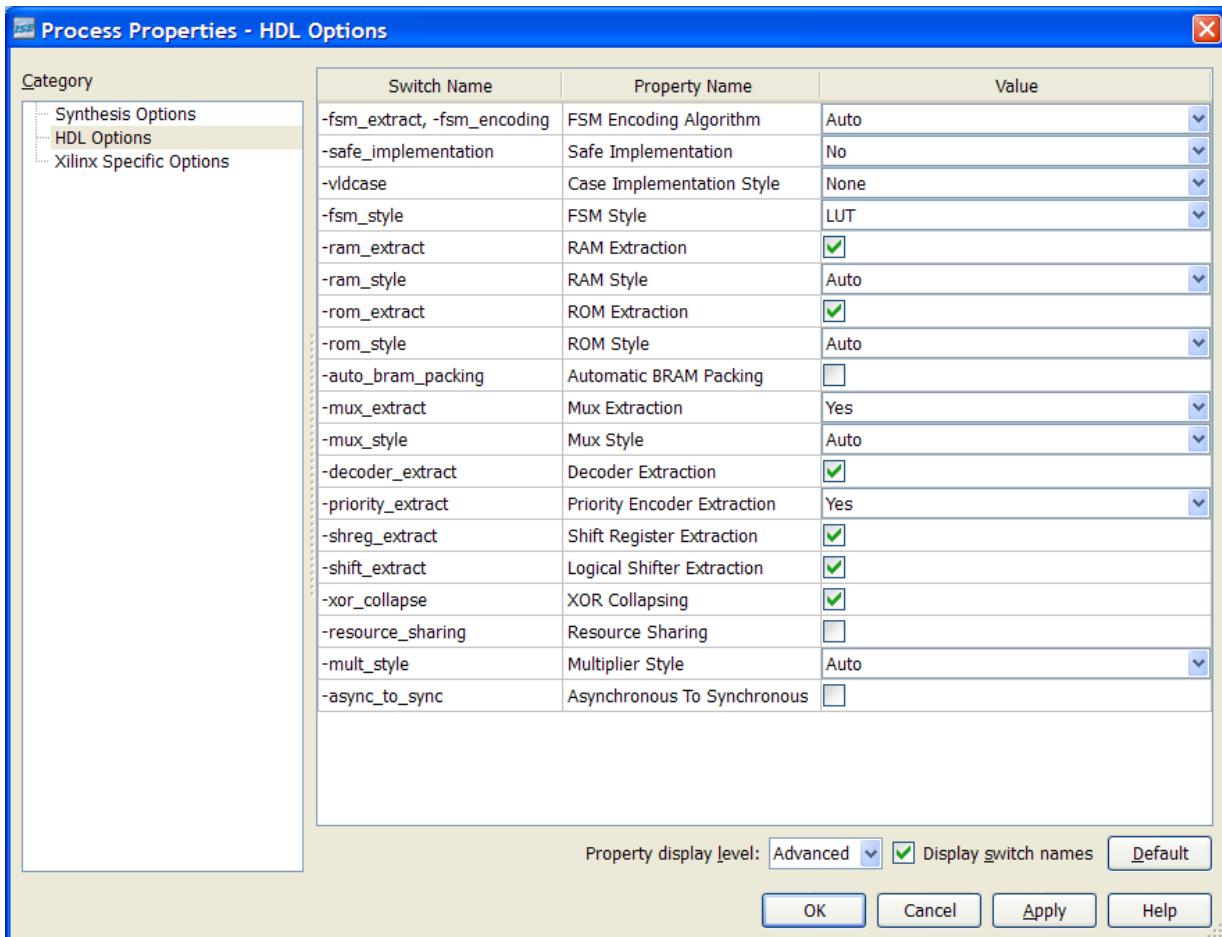


25 pav. Xilinx Spartan 3E šeimos FPGA peržiūros lentelė (LUT) konfigūruota kaip adresuojamas postūmio registras [8]

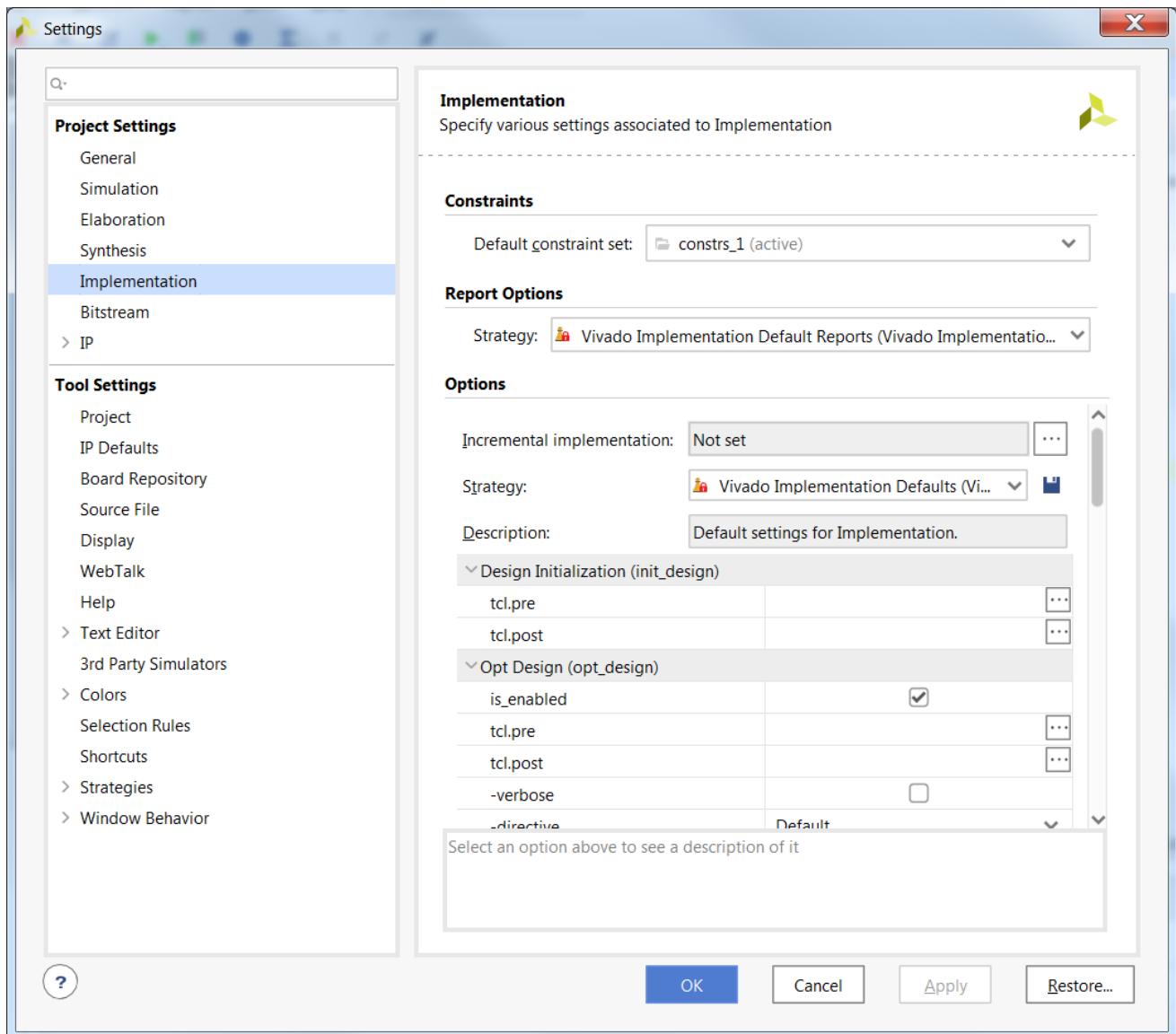


26 pav. Xilinx bibliotekų primityvo SRL16 komponentas [8]

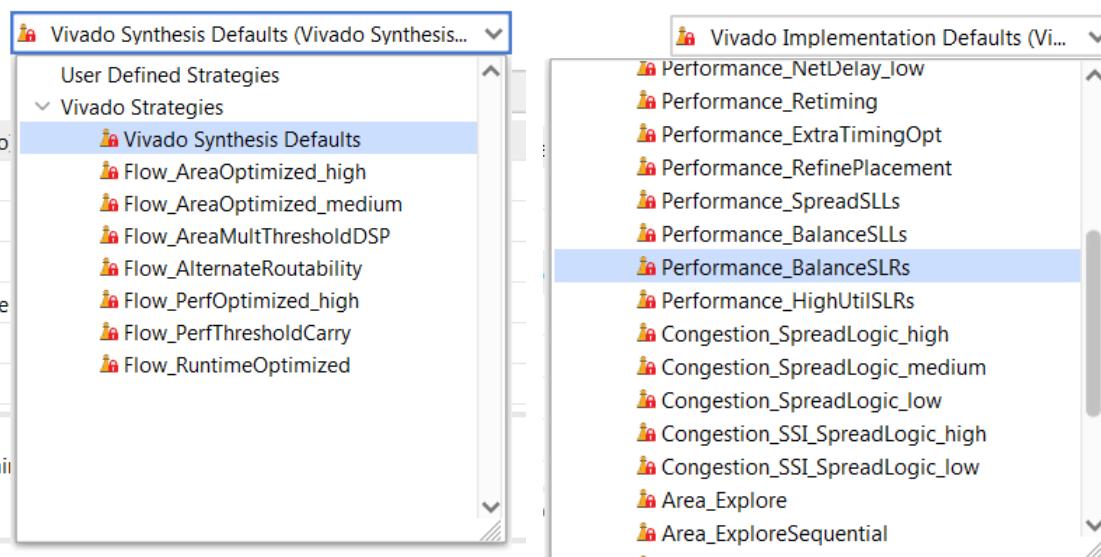
Generuojama technologinio lygmens grandinė ir joje naudojami struktūriniai FPGA vidiniai resursai taip pat priklauso nuo sintezei naudojamų opcijų, panašiai kaip kompiliuojant mikroprocesoriaus programą visą eilę įvairių opcijų nurodome komplifikatoriai. Iš tikrujų, pašalinus opciją **Shift Register Extraction** (žr. 26 pav.) sintezę atliekančiame Xilinx ISE pakete (**Synthesize/Process Properties.../HDL Options** meniu punktas), generuojama postūmio registro schema sutampa su parodyta 21 pav.



27 pav. Xilinx ISE projektavimo paketo sintezės opcijų nustatymo dialogo langas



28 pav. Xilinx Vivado projektavimo paketo sintezės opcijų nustatymo dialogo langas



29 pav. Xilinx Vivado sintezės ir realizavimo optimizavimo strategijos

### 7.2.3 Skaitikliai

Paprasto keturių bitų dvejetainio skaitiklio aprašas VHDL kalba yra pateiktas 30 programe, o jo funkcinio modeliavimo laikinės diagramos ir technologinio lygmens grandinė atitinkamai 30 ir 31 pav. Kaip matyti iš technologinio lygmens grandinės skaitklė sudaro keturi D trigeriai (komponentai fdc) taktuojami taktinių impulsų signalu *Clock*. Trigerių jėjimų (schemoje jėjimai D) signalai yra formuojami iš to paties trigerio išėjimo žemesnių skilčių trigerių išėjimų (schemoje išėjimai Q) per grįžtamojo ryšio kombinacines schemas, kurios realizuojamos panaudojant peržiūros lenteles. Vienos peržiūros lentelės *lut4* Karno žemėlapis ir teisingumo lentelė yra parodyti 32 pav.

#### 30 progr. Dvejetainio skaitiklio aprašas

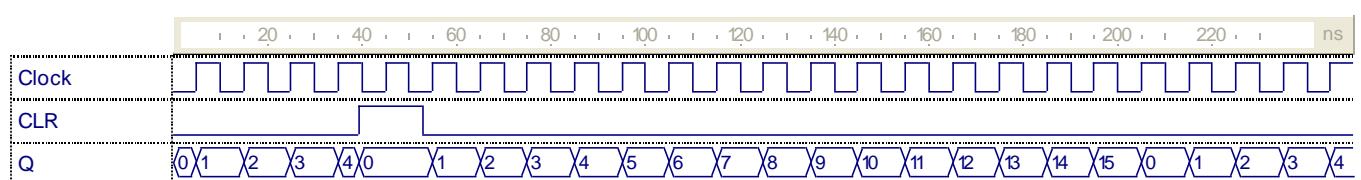
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- bus naudojamas tipas unsigned

entity simple_counter is
    port(Clock, CLR : in std_logic;
        Q : out std_logic_vector(3 downto 0));
end simple_counter;

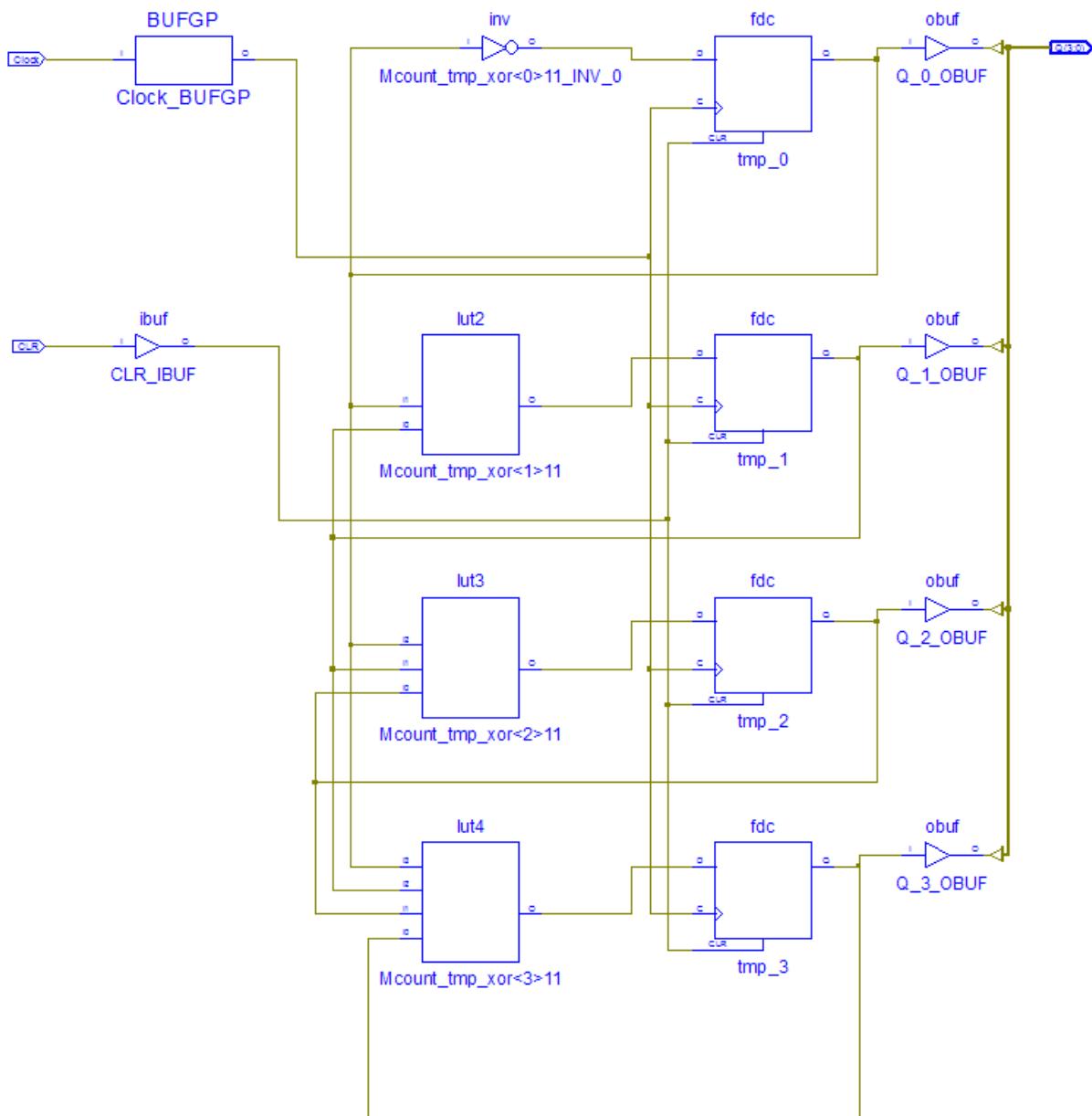
architecture arch_cnt of simple_counter is
    signal tmp: unsigned (3 downto 0) := X"0";
begin
    process (Clock, CLR)
    begin
        if (CLR='1') then
            tmp <= "0000";
        elsif (Clock'event and Clock='1') then
            tmp <= tmp + 1;
        end if;
    end process;

    Q <= std_logic_vector(tmp); -- tipo konvertavimas

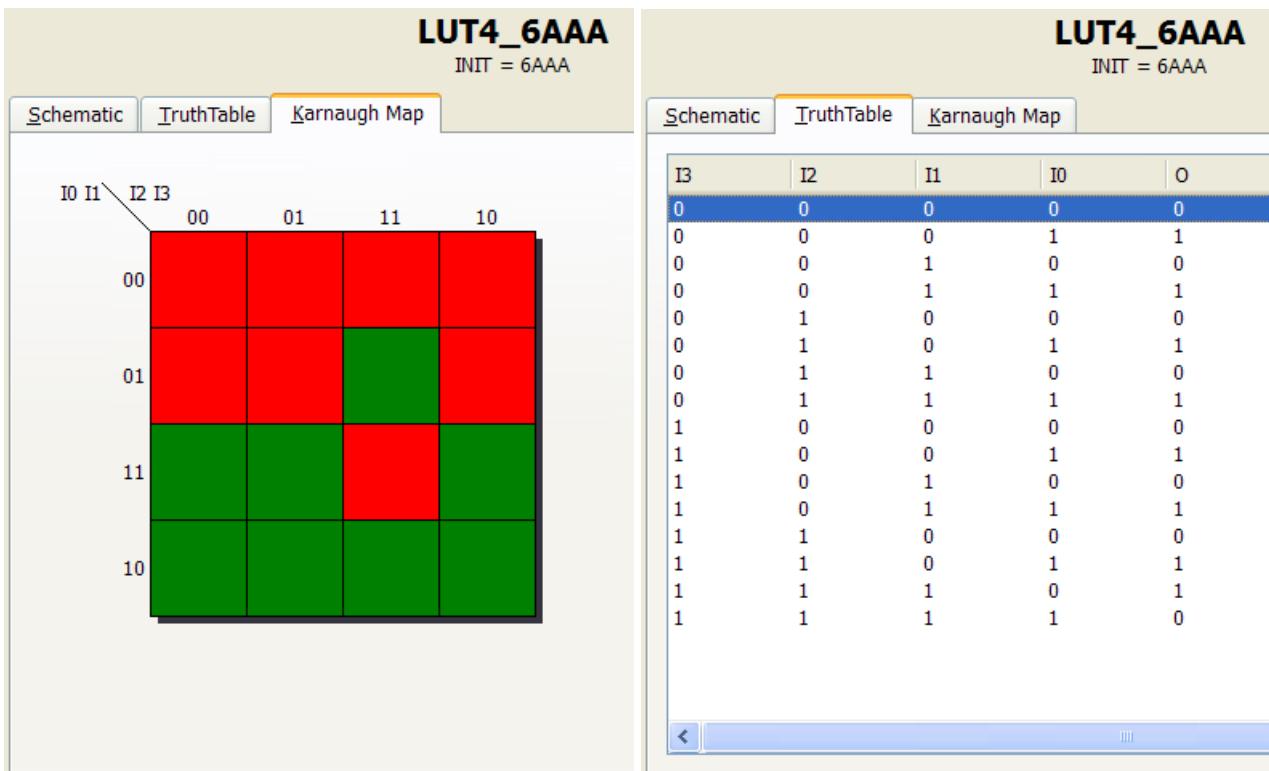
end arch_cnt;
```



30 pav. Dvejetainio skaitiklio *simple\_counter(arch\_cnt)* funkcinio modeliavimo laikinės diagramos



31 pav. Dvejetainio skaitiklio *simple\_counter(arch\_cnt)* technologinė grandinė po sintezės į Xilinx Spartan 3E XC3S500E FPGA komponentą, nenaudojant peržiūros lentelių (LUT) konfigūravimo postūmio registro režimu



32 pav. Peržiūros lentelės lut4 iš 31 pav. Karko žemėlapis ir teisingumo lentelė

Dažnai tenka naudoti skaitiklius, kurie persipildo pasiekę tam tikrą užduotą reikšmę. 31 programoje pateiktas aprašas sakitiklio, inkrementavimo režime skaičiuojančio nuo 0 iki 9, o dekrementavimo režime, nuo 9 iki 0, kaip parodyta 33 pav.

### 31 progr. Išplėstu galimybių skaitiklio aprašas

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; -- reikia dėl aritmetinių operacijų
                                -- su STD_LOGIC_VECTOR tipo objektais

entity counter is
    generic(MAX: STD_LOGIC_VECTOR(7 downto 0):= X"09"); -- parametrizavimui, skaičiuoja
    from 0 iki 9
    port (
        CLK: in STD_LOGIC; -- taktiniai impulsai
        RESET: in STD_LOGIC; -- asinchroninis numetimas
        CE: in STD_LOGIC; -- leidimas skaičiuoti
        LOAD: in STD_LOGIC; -- pradinė reikšmės užkrovimo signalas
        DIR: in STD_LOGIC; -- krypties valdymas (inkrementuojantis/dekrementuojantis skaitikl).
        DIN: in STD_LOGIC_VECTOR (7 downto 0); -- pradinė į skaitiklį užkraunama reikšmė
        COUNT: out STD_LOGIC_VECTOR (7 downto 0) -- einamoji skaitiklio reikšmė
    );
end counter;

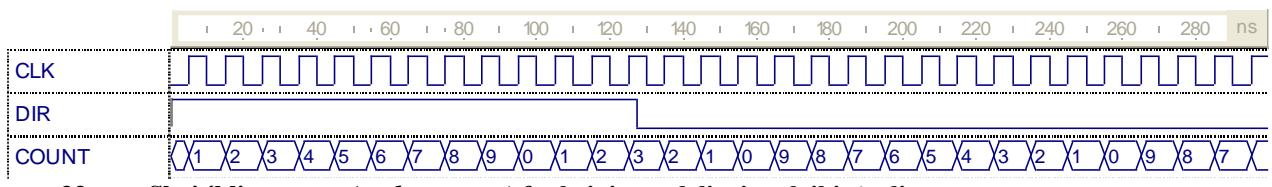
-- Synchroninis dvejetainis dvikryptis (up/down) skaitiklis su
-- asinchroniniu numetimu (reset) ir synchroniniu užkrovimu
architecture arch_counter of counter is
    signal COUNT_INT: STD_LOGIC_VECTOR (7 downto 0):=X"00";
begin
process (CLK, RESET)
begin
    if RESET = '1' then
        COUNT_INT <= (others => '0');
    elsif CLK'event and CLK='1' then
        if LOAD = '1' then
            COUNT_INT <= DIN;
        else
            if CE = '1' then
                COUNT_INT
            end if;
        end if;
    end if;
end process;
end architecture;

```

```

if DIR = '1' then -- inkrementavimas
    if COUNT_INT=MAX then
        COUNT_INT <= X"00";
    else
        COUNT_INT <= COUNT_INT + 1;
    end if;
else                      -- dekrementavimas
    if COUNT_INT=0 then
        COUNT_INT <= MAX;
    else
        COUNT_INT <= COUNT_INT - 1;
    end if;
end if;
end if;
end process;
COUNT <= COUNT_INT;
end architecture arch_counter;

```



33 pav. Skaitiklio counter(arch\_counter) funkcinio modeliavimo laikinės diagramos

## 8 Programavimo stiliai arba aprašų lygmenys

Kuriant skaitmeninio įrenginio aprašą skirtinguose projekto fazėse yra naudojami įvairūs programavimo stiliai arba aprašų lygmenys (abstrakcijos lygmenys) tam pačiam objektui aprašyti (žr. 6 lent.).

6 lent. Aprašų lygmenys (programavimo stiliai [13])

Lygmens pavadinimas	Naudojamos VHDL konstrukcijos
Elgsenos (angl. behavioral)	Tik <i>process</i> sakiniai
Duomenų srautų (angl. datapath)	Tik lygiagretaus priskyrimo sakiniai (įskaitant sąlyginius), procedūrų iškvietimai
Struktūrinis	Tik objektų egzempliorių kūrimas
Mišrus	Naudojamos įvairios

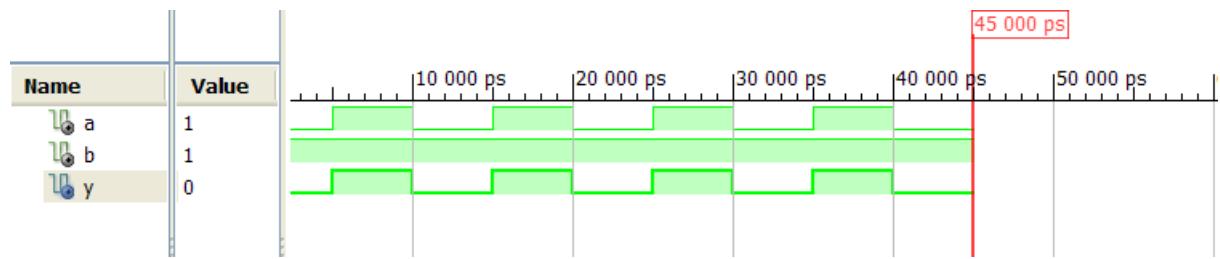
Aprašuose skirtuose sintezei, registrai ir kiti įrenginiai su atmintimi turi būti aprašomi tik panaudojant *process* sakinį arba sukuriant objektų su atmintimi egzempliorius. Tokiu būdu iš duomenų srautų lygmenyje aprašytų architektūrų bus sintezuojami tik kombinacinio (be atminties) tipo įrenginiai.

Geru programavimo stiliumi laikomas tokis, kuriamo hierarchiniuose projektuose aukščiausio lygmens objektas aprašomas struktūriniu lygmeniu [13, p.88].

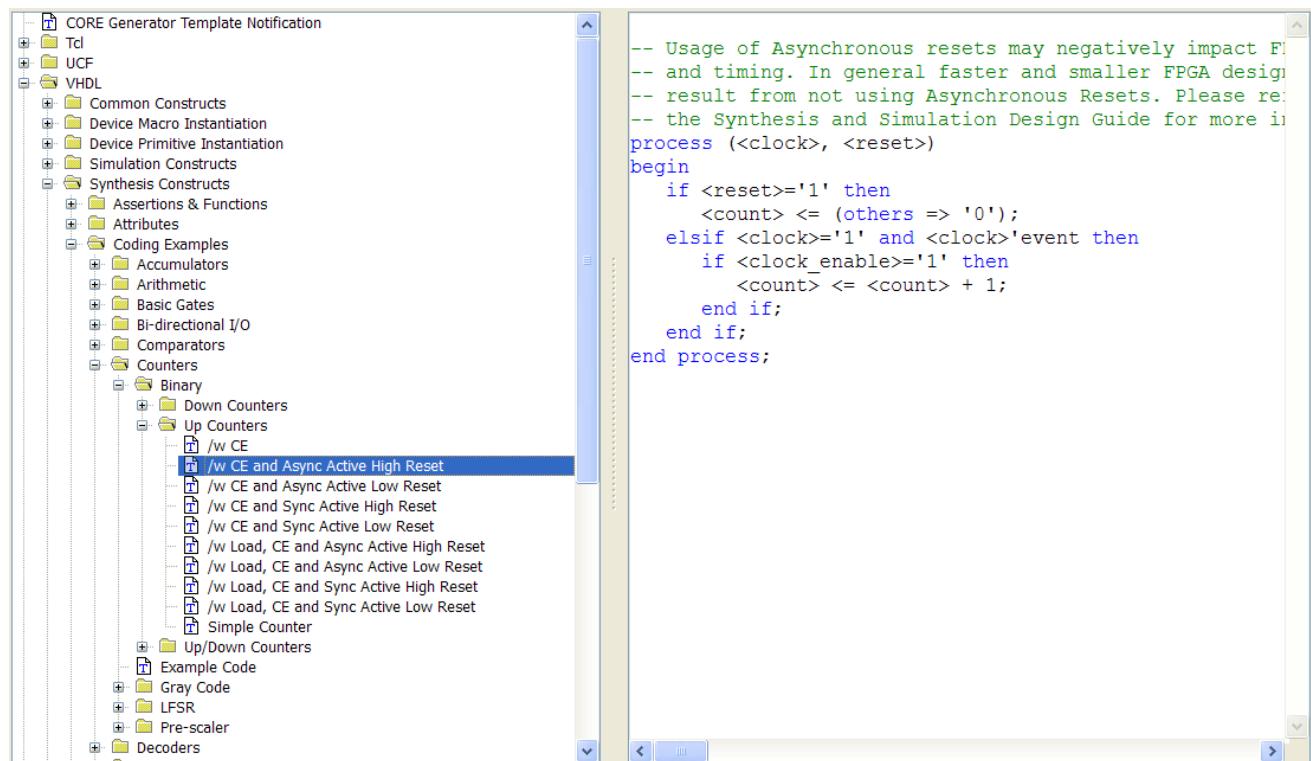
## 9 VHDL kalba projektavimo paketuose

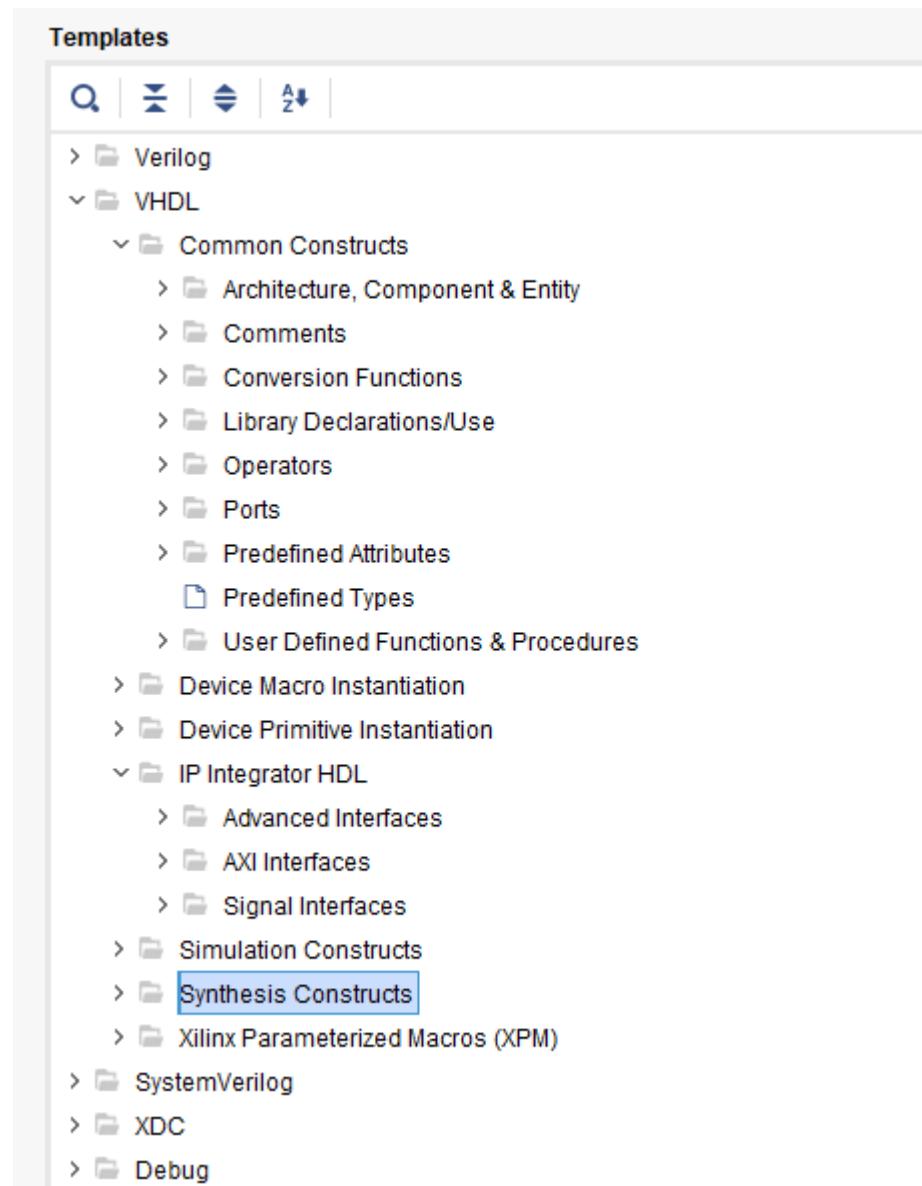
### 9.1 Xilinx projektavimo priemonėse

#### 9.1.1 Kodo šablonai (templates)



VHDL teksto įvedimui yra patogu naudotis Xilinx ISE paketo šablonų archyvu, kadangi nereikia rankiniu būdu įvedinėti standartinių konstrukcijų. Įvedimo procesas ne tik pagreitėja, bet tame išvengiama ir paprastų spausdinimo klaidų. Šablonų archyvą (žr. 34 pav.) galima iškvesti pasirinkus meniu punktą **Edit/Language Templates ...**.





34 pav. Xilinx ISE ir Vivado paketo VHDL kalbos šablonų (template) archyvo fragmentas

Analogiški HDL tekstu įvesties pagalbininkai yra realizuojami ir kitų firmų projektavimo paketuose, pvz., Aldec Active-HDL aplinkoje pasirinkus meniu punktą **Tools/Language Assitant**, galima išsikvesti pasirinktos kalbos (VHDL, Verilog ar kitos) įvesties vedlį (žr. **Error! Reference source not found.** pav.)

### 9.1.2 IP šerdžių katalogas

(Xilinx Vivado IP Catalog)

IP Catalog

Cores | Interfaces

Name AXI4 Status License

- > AXI Infrastructure
- > AXIS Infrastructure
- > BaseIP
- > Basic Elements
- > Communication & Networking
- > Debug & Verification
- > Digital Signal Processing
  - > Building Blocks
  - > Filters
  - > Modulation
  - > Transforms
  - > Trig Functions
    - CORDIC** AXI4-Stream Production Included
    - DDS Compiler** AXI4-Stream Production Included
  - > Waveform Synthesis
    - DDS Compiler** AXI4-Stream Production Included
- > Embedded Processing
- > FPGA Features and Design

### Details

Name: **DDS Compiler**

Version: 6.0 (Rev. 15)

Interfaces: AXI4-Stream

Description: The Xilinx DDS Compiler LogiCORE provides Direct Digital Synthesizers (DDS) and optionally either Phase Generator or Sine, cosine or quadrature outputs with 3 to 26-bit output sample precision. The core supports up to 16 channels by time-slots if multiple channels are needed. Phase Dithering and Taylor Series Correction options provide high dynamic range signals and phase noise capability, providing support for multiple synthesizers with precisely controlled phase differences.

## 9.2 EDA Playground aplinka

### 9.2.1 Grafinė aplinka

EDA Playground (<https://www.edaplayground.com>) – tai integruota projektavimo aplinka naudojama per naršyklę ir skirta modeliuoti VHDL, Verilog, SystemVerilog, ir kitomis HDL kalbomis aprašytus skaitmeninius įrenginius.

The screenshot shows the EDA playground interface. On the left, there's a sidebar with options like 'Start a new playground', 'Languages & Libraries' (set to VHDL), 'Top entity' (set to testbench), 'Tools & Simulators' (set to GHDL 0.35), and 'Compile & Run Options'. The main area has two code editors: 'testbench.vhd' and 'design.vhd'. The 'testbench.vhd' code defines a testbench for an OR gate, using IEEE std\_logic\_1164.all library. The 'design.vhd' code defines a logic entity 'LogicEntity' with a port map for X (std\_logic\_vector(3 downto 0)) and Y (std\_logic). An architecture 'MyAnd1' is shown with a begin block containing a connect statement. Below the code editors is a sharing section with social media icons and a link: 'VHDL - Mano bandymas'. At the bottom, there's a note: 'Pirmasis VHDL programos pavyzdys.'

Naudojimosi aplinka dokumentaciją galima iškvesti paspaudus klaustuko simbolį viršutinėje įrankių liniuotėje arba tiesiogiai pagal nuorodą <https://eda-playground.readthedocs.io/en/latest/>.

### 9.2.2 Nestandardinių bibliotekų naudojimas

Kai kurios VHDL bibliotekos nors ir realizuotos IEEE bibliotekoje, bet neaprašytos IEEE standarte, pvz., IEEE.STD\_LOGIC\_UNSIGNED.ALL yra nepalaikomos kai kurių simulatorių, pvz. GHDL (<https://readthedocs.org/projects/ghdl-devfork/downloads/pdf/latest/>). Norint naudoti šias bibliotekas galima persijungti į kitą simulatorių, pvz., Aldec Riviera Pro 2022.04.

The screenshot shows the 'Tools & Simulators' configuration. It includes a dropdown menu set to 'Aldec Riviera Pro 2022.04', a 'Compile Options' dropdown set to '-2019 -o', and a note: 'Compiles existing standards'.

Apie nestandardinių bibliotekų atsiradimą ir pakeitimą galima perskaityti „What are deprecated IEEE libraries?“ (<https://insights.sigasi.com/tech/deprecated-ieee-libraries/#what-are-deprecated-ieee-libraries>) arba „VHDL-2008: Incorporates existing standards“ [https://www.doulos.com/knowhow/vhdl\\_designers\\_guide/vhdl\\_2008/vhdl\\_200x\\_merged/](https://www.doulos.com/knowhow/vhdl_designers_guide/vhdl_2008/vhdl_200x_merged/)

### 9.2.3 Laikinių diagramų atvaizdavimas

Atlikus modeliavimą VHDL kalba aprašyto bandymų stendo rezultatus galima atvaizduoti laikinių diagramų lange. Tam reikia (taip pat žiūrėkite langų atvaizdus žemiau):

- Pažymėti **Open EPWave after run.**
- Nurodyti **Top entity** bandymų stendo objekto pavadinimą (nesumaišykite su failo pavadinimu).
- Paspausti mygtuką **Run**. Modeliavimui pasibaigus laikinės diagramos bus atvaizduotos EPWave lange (Naršyklėje turi būti leistas langų iššokimas (Pop-ups enabled))
- EPWave lange paspaudus mygtuką **Get Signals** galima pasirinkti norimus atvaizduoti signalus.

Modeliavimo rezultatų atvaizdavimą laikinių diagramų pavidalu galime išbandyti įkélé į **VHDL Design** langą žemiau pateikto projektuojamio įrenginio aprašą, o į **VHDL Testbench** langą – bandymų stendo aprašą.

#### Projektuojamas įrenginys (skaitiklis)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use ieee.numeric_std.all;

entity UP_COUNTER is
    Port ( clk: in std_logic; -- clock input
            reset: in std_logic; -- reset input
            enable: in std_logic;
            counter: out std_logic_vector(5 downto 0) -- output 6-bit counter
        );
end UP_COUNTER;

architecture Behavioral of UP_COUNTER is
signal counter_up: std_logic_vector(5 downto 0);
begin
process(clk,reset)
begin
if(rising_edge(clk)) then
    if(reset='1') then
        counter_up <= B"000000";
    else
        if(enable='1') then
            counter_up <= counter_up + x"1";
        end if;
    end if;
end if;
end process;
counter <= counter_up;
end Behavioral;
```

#### Bandymų standas

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_counters is
end tb_counters;

architecture Behavioral of tb_counters is

component UP_COUNTER
    Port ( clk: in std_logic; -- clock input
            reset: in std_logic; -- reset input
            enable: in std_logic;
            counter: out std_logic_vector(5 downto 0) -- output 6-bit counter
```

```

    );
end component;
signal reset,clk: std_logic;
signal counter:std_logic_vector(5 downto 0);
signal enable: std_logic;

begin
dut: UP_COUNTER port map (clk => clk, reset=>reset, counter => counter, enable=>enable);
-- Clock process definitions
clock_process :process
begin
    clk <= '1';
    wait for 1 ns;
    clk <= '0';
    wait for 1 ns;
end process;

stim_proc: process
begin
    reset <= '1';
    wait for 3 ns;
    reset <= '0';
    enable <= '1';
    wait for 5 ns;
    enable <= '0';
    reset <= '0';
    wait for 5 ns;
    reset <= '1';
    wait for 5 ns;
    enable <= '1';
    reset <= '0';
    wait for 5 ns;
    enable <= '0';
    wait;
end process;
end Behavioral;

```

Patikrinkite visus aplinkos nustatymus kairėje panelėje (Aldec Riviera Pro 2022.04, [Run Time](#)  
**20 ns**, pažymėta Open EPWave after run) ir paspauskite mygtuką  . Rezultatus turite pamatyti EPWave lange.

The screenshot shows the EDA Playground interface with two VHDL files open:

- testbench.vhd** (VHDL Testbench):
 

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_counters is
end tb_counters;

architecture Behavioral of tb_counters is

component UP_COUNTER
  Port ( clk: in std_logic; -- clock
         reset: in std_logic; -- reset
         enable: in std_logic;
         counter: out std_logic_vector(5 downto 0) -- output 6-bit counter
      );
end component;
signal reset,clk: std_logic;
signal counter:std_logic_vector(5
  downto 0);
      
```
- design.vhd** (VHDL Design):
 

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--use ieee.numeric_std.all;

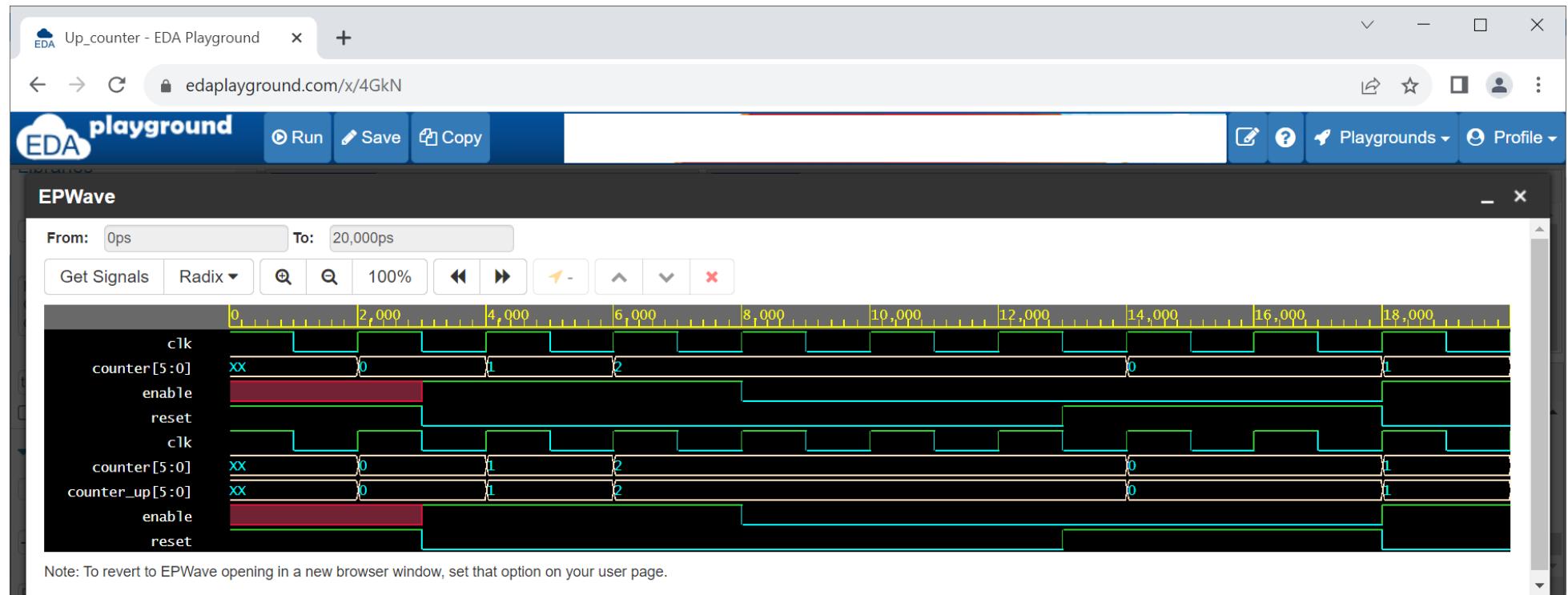
entity UP_COUNTER is
  Port ( clk: in std_logic; -- clock input
         reset: in std_logic; -- reset input
         enable: in std_logic;
         counter: out std_logic_vector(5 downto 0) -- output 6-bit counter
      );
end UP_COUNTER;

architecture Behavioral of UP_COUNTER is
signal counter_up: std_logic_vector(5 downto 0);
begin
process(clk,reset)
begin
if(rising_edge(clk)) then
  if(reset='1') then
    counter_up <= B"000000";
    
```

The left sidebar shows the project structure and run options. The "Log" tab at the bottom shows the simulation results:

```

# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO/bin/libsysf.so'
# KERNEL: stopped at time: 20 ns
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2023-10-15 19:54:02 UTC] Opening EPWave...
Done
      
```



## 10 Pagrindiniai literatūros šaltiniai

1. Pong P. Chu, FPGA prototyping by VHDL examples, 2008 by John Wiley & Sons, Inc.
2. V.Jusas, E.Bareiša, R.Šeinauskas, Skaitmeninių sistemų projektavimas VHDL kalba, Kaunas: Technologija, 1997, 196 p.
3. J.R. Armstrong, F. Gail Gray, VHDL Design Representation and Synthesis, 2 nd. ed.
4. D.L.Perry, VHDL Programming by Example, 4 th ed. McGraw-Hill, 2002, 476 p.
5. VHDL Tutorial, Jan Van der Speigel, (Interneto prieiga:  
[http://www.seas.upenn.edu/~ese171/vhdl/vhdl\\_primer.html](http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html)), Žiūrėta 2016-10-04 ).
6. VHDL Tutorial: Learn by Example by Weijun Zhang, July 2001(Interneto prieiga:  
<http://esd.cs.ucr.edu/labs/tutorial/>), Žiūrėta 2016-10-04)
7. Archive of VDLANDE.com VHDL Reference , (Interneto prieiga:  
<http://www.ics.uci.edu/~jmoorkan/vhdlref/>), Žiūrėta 2016-10-04).
8. Spartan-3 Generation FPGA User Guide Extended Spartan-3A, Spartan-3E, and Spartan-3 FPGA Families UG331 (v1.7) August 19, 2010 (Interneto prieiga:  
[http://www.xilinx.com/support/documentation/user\\_guides/ug331.pdf](http://www.xilinx.com/support/documentation/user_guides/ug331.pdf)), Žiūrėta 2016-10-04).
9. VHDL coding tips and tricks (Interneto prieiga:< <http://vhdlguru.blogspot.lt/> > Žiūrėta 2016-11-07).

## 11 Papildomi literatūros šaltiniai

10. R.Dueck, Digital Design with CPLD Applications and VHDL. Delmar Cengage Learning; 2nd ed., 2004, 896 p.
11. Enoch O. Hwang, Microprocessor Design. Principles and Practice with VHDL. Brooks / Cole, 2004, 341 p.
12. Sequential and concurrent statements in the VHDL language, (Interneto prieiga: <  
<http://ftp.utcluj.ro/pub/users/calceng/SSC/Ssc06/SSC06-e.pdf>>), Žiūrėta 2014-05-15).
13. А.М.Сергиенко, VHDL для проектирования вычислительных устройств – ООО ТИД ДС. 2003, 208 стр.
14. П.Н. Бибило, Основы языка VHDL, 2nd ed. Москва: СОЛООН-Р, 2002, 218 p.