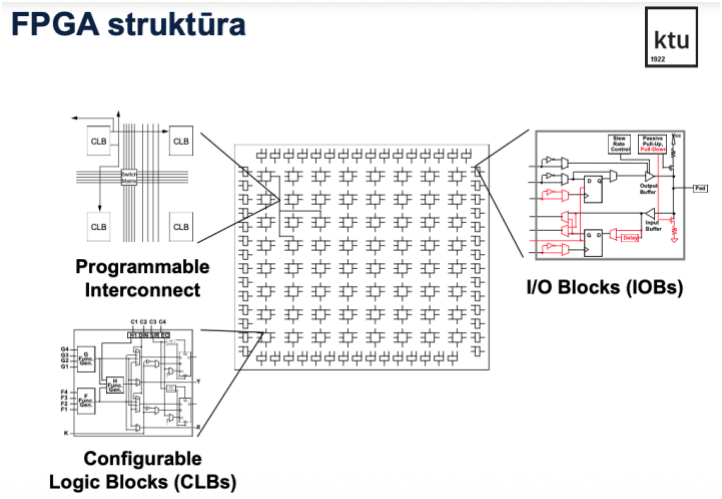
1. Bazinė FPGA sandara ir ypatybės

1.1. Pagrindiniai FPGA struktūros elementai.



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 (Multipiers)
- Sinchro valdiklio blokas (DCM)
- Kita
- DSP blokai

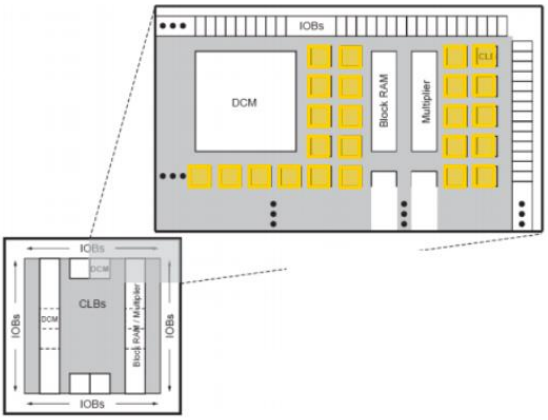
1.2. Kokius žinote FPGA struktūros blokus.

CLB (Configurable Logic Blocks)
I/O blokai (I/O Blocks)
Atminties blokai (RAM Blocks)
Daugintuvų blokai (Multipiers)
Sinchro valdiklio blokas (DCM)
DSP BLOCK

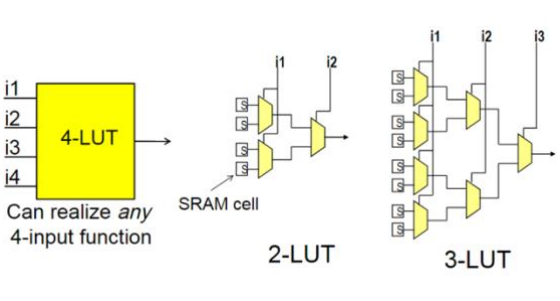
1.3. Kas yra konfigūruojamas loginis blokas (CLB), kas jį sudaro?

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.

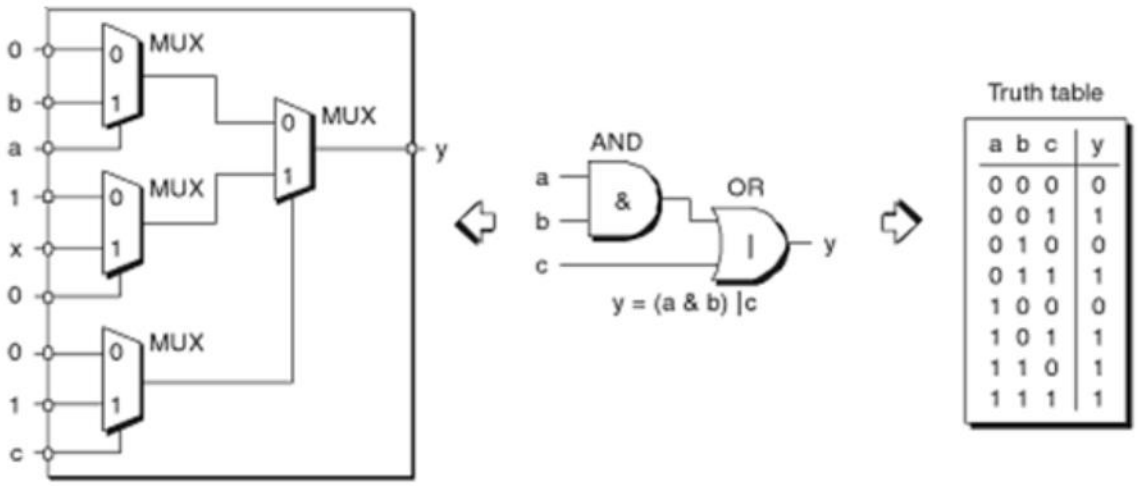
Atliekamos funkcijos:
- loginės operacijos;
- signalo užlaikymo (flip-flop);
- postumio registro;
- paskirstytos atminties;



1.4. Loginių funkcijų realizavimas programuojamais grandynais.
Loginės funkcijos realizuojamos loginiuose blokuose (Altera-LC, Xilinx – CLB). Dažniausiai naudijami 2, 3 ar 4 įėjimų (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.



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



1.5. Kas yra LUT?

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

- 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.

1.6. Taktinių impulsų valdiklio paskirtis.

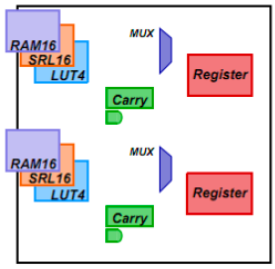
DCM (Digital Clock Manager)
TIV sudedmosios dalys:
- Fiksuoto vėlinimo grandynai
(DLL - Delay-Locked Loop)
- Skaitmeniniai dažnio sintezatoriai
(DFS - Digital Frequency Synthesizer)
- Skaitmeninai fazės sukikliai
(DPS - Digital Phase Shifter)
- Fiksuotos fazės grandynai
(PLL - Phase-Locked Loop)

Dažnio sintezė: DCM generuoja projektui reikiamo dažnio taktinius impulsus.
• **Fazės valdymas:** Valdiklis leidžia atlikti **fazės sukimą** (angl. *phase shifting*).
• **Signalo stabilizavimas ir kokybė:** DCM **mažina taktinių impulsų frontų virpėjimą** (angl. *jitter reduction*).
• **Vėlinimo kompensavimas:** Naudojant fiksuoto vėlinimo grandines (**DLL** – *Delay-Locked Loop*), valdiklis pašalina taktinio signalo sklaidą (angl. *skew*) tarp įėjimo kojelės ir vidinių loginių elementų

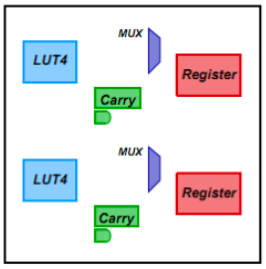
1.7. Xilinx FPGA loginės celės (slice) sandara. Kas tai yra SLICEM ir SLICEL?

Spartan™ -3 kristale yra keturios loginės celės (slice)

- Celės grupuojamos poromis:
  - SLICEM (Memory)
    - LUT
      - Loginės funkcijos
      - Paskirstytoji atmintis
      - Postūmio registras SRL16
    - Multipeksoriai
    - Registrai (užlaikymo, flip-flop)
  - SLICEL (Logic)
    - LUT
      - Loginės funkcijos
    - Multipleksoriai
    - Registrai (užlaikymo, flip-flop)



SLICEM



SLICEL / SLICEX (SLICEX omits carry logic)

VHDL kalba. Struktūra ir pagrindinės konstrukcijos

2.1. Kas VHDL kalboje vadinama objektu (entity), sąsaja (interface) ir architektūra (architecture)?

- **Objektas (entity):** Tai VHDL kalba aprašyta loginė grandinė, kuri gali būti labai paprasta (pvz., loginis ventilis IR) arba labai sudėtinga (pvz., mikroprocesorius). Objektas yra pagrindinis sistemos elementas, kurio veikimas ir jungtys aprašomi programiškai.
- **Sąsaja (interface):** Tai objekto aprašo dalis, kuri deklaruoja jo įėjimus ir išėjimus (prievadus)
- **Architektūra (architecture):** Tai aprašas, kuris specifikuoja objekto vidinę sandarą arba elgseną

2.2. Pateikite loginio elemento IR-NE, 3ARBA, 3IR-NE ar kito sąsajos ir architektūros aprašą.

Loginis elementas IR-NE (2-iejų įėjimų NAND) - atlieka loginę sandaugą su inversija

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

-- Sąsajos aprašas
entity nand2_gate is
    Port ( A : in  STD_LOGIC; -- Įėjimas A
           B : in  STD_LOGIC; -- Įėjimas B
           Y : out STD_LOGIC); -- Išėjimas Y
end nand2_gate;

-- Architektūros aprašas
architecture Behavioral of nand2_gate is
begin
    Y <= A nand B; -- Loginė operacija IR-NE
end Behavioral;
```

Loginis elementas 3ARBA (3-jų įėjimų OR) išėjimas yra '1', jei bent vienas įėjimas yra '1'

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

entity or3_gate is
    Port ( IN1 : in  STD_LOGIC;
           IN2 : in  STD_LOGIC;
           IN3 : in  STD_LOGIC;
           OUT_VAL : out STD_LOGIC);
end or3_gate;

architecture Behavioral of or3_gate is
begin
    -- Loginė operacija ARBA trims įėjimams
    OUT_VAL <= IN1 or IN2 or IN3;
end Behavioral;
```

Loginis elementas 3IR-NE (3-jų įėjimų NAND)

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

entity nand3_gate is
    Port ( A, B, C : in  STD_LOGIC;
```

```
        Y : out STD_LOGIC);
end nand3_gate;

architecture Algorithmic of nand3_gate is
begin
   -- Elgsenos aprašas naudojant sąlyginį priskyrimą
   Y <= '0' when (A='1' and B='1' and C='1') else '1';
end Algorithmic;
```

2.3. Kuo skiriasi objekto architektūros algoritminis ir struktūrinis aprašai?

- **Algoritminis (elgsenos) aprašas** - Naudojamas pradiniuose projekto etapuose vykdant modeliavimą, siekiant patikrinti, ar įrenginio logika veikia teisingai. Aprašymui naudojamos aukšto abstrakcijos lygmens konstrukcijos, tokios kaip ciklai, sąlyginiai sakiniai (if-else, case) ir būsenų automatai. Aprašo ką įrenginys daro (elgseną), o ne iš ko jis sudarytas.
- **Struktūrinis aprašas** Specifikuoja įrenginį kaip sistemą, sudarytą iš konkrečių paprastesnių komponentų. Naudojamas **objektų egzempliorių kūrimas** (angl. *instantiation*) ir jų sujungimas. Pagrindinė konstrukcija yra `port map`, kuri naudojama susieti išorinius signalus su komponentų vidiniais prievadais

2.4. Kokie duomenų tipai naudojami VHDL kalboje?

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 skirtingų tipų
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;
```

2.5. Kas yra VHDL kalbos tipo atributai ir kam jie naudojami?

VHDL tipo **atributai** yra specialios savybės, kurios nurodomos po apostrofo ir suteikia papildomą informaciją apie duomenų objektą arba jo tipą. Jie dažniausiai naudojami norint programiškai sužinoti masyvų **ilgį** (`'length`), **indeksų diapazoną** (`'range`) ar konkrečias kraštines indeksų reikšmes. Šie atributai yra labai naudingi automatizuojant užduotis, pavyzdžiui, **skenuojant masyvo elementus** cikluose arba lanksčiai aprašant logines operacijas nepriklausomai nuo kintamojo dydžio PVZ:

```
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 pirmąjį 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;
```

2.6. Kuo skiriasi neapibrėžti ir apibrėžti tipai, pvz., IEEE 1164 std_ulogic ir std_logic?

Pagrindinis skirtumas yra lygiagrečių signalų verčių (angl. drivers) valdymas: neapibrėžtiems tipams (pvz., std_ulogic) priskiriant kelias reikšmes vienu metu generuojama klaida, o apibrėžtiems (pvz., std_logic) galutinė vertė nustatoma pagal išsprendimo lentelę. Ši funkcija leidžia modeliuoti realias grandines, kuriose prie vienos magistralės jungiami keli siųstuvai,

pavyzdžiui, derinant loginį nulį su aukšto impedanso (Z) būsena. std_logic yra laikomas pramoniniu standartu, nes tai yra std_ulogic subtipas su įdiegta konfliktų sprendimo logika

```
YPE 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;
```

2.7. Kokie duomenų objektai naudojami VHDL kalboje?

VHDL kalboje naudojamos trys pagrindinės duomenų objektų klasės: konstantos, kintamieji ir signalai. Konstantų reikšmės nustatomos kompiliavimo metu ir nekinta, o kintamieji ir signalai leidžia saugoti bei keisti duomenis vykdant programą. Svarbu tai, kad signalams apibūdinti reikalinga laiko dimensija, o visi objektai turi būti deklaruoti prieš juos naudojant

2.8. Kuo skiriasi VHDL kalbos kintamieji ir signalai?

Kintamieji keičia reikšmę akimirksniu naudojant „:=“ operatorių ir yra deklaruojami tik procesų ar paprogramių viduje. Signalai reikšmę keičia su laiko užlaika naudojant „<=“ operatorių ir aprašo fizines duomenų linijas tarp įrenginio dalių. Svarbiausias skirtumas yra tas, kad kintamieji yra lokalūs algoritminiai objektai, o signalams būdinga laiko dimensija ir jie negali būti deklaruojami procesuose

2.9. Kokios priskyrimo konstrukcijos naudojamos VHDL kalboje?

VHDL kalboje naudojamos dvi pagrindinės priskyrimo konstrukcijos: kintamojo priskyrimas (naudojant operatorių :=) ir signalo priskyrimas (naudojant operatorių <=),. Kintamųjų reikšmės keičiasi momentiškai (blokuojantis priskyrimas), o signalams būdinga laiko dimensija, todėl jų vertės kinta su vėlinimu,. Taip pat plačiai naudojamos lygiagrečiosios sąlyginės konstrukcijos, tokios kaip when-else ir with-select-when

```
-- 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);
```

2.10. Apibūdinkite VHDL modulio aprašo struktūrą (iš ko ji sudaryta).

- VHDL modulis, vadinamas objektu (entity), susideda iš dviejų pagrindinių dalių: sąsajos (interface) ir architektūros (architecture). Sąsajos aprašas deklaruoja įrenginio išorinius prievadus (įėjimus ir išėjimus), o architektūra aprašo vidinę elgseną arba loginę struktūrą. Be to, kodo pradžioje paprastai deklaruojamos naudojamos bibliotekos ir jų paketai, apibrėžiantys duomenų tipus bei operacijas.
- Analogija: VHDL modulį galime palyginti su namu: sąsaja yra durys, langai ir pašto dėžutė (tai, per kur perduodama informacija ar energija į vidų ir išorę), o architektūra – namo vidaus planas ir elektros grandinės (tai, kaip viskas viduje sujungta ir kaip jis funkcionuoja).

Kombinacinės logikos aprašymas VHDL kalba

3.1. Aprašykite multipleksorių iš 4 į 1 VHDL kalba.

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

```
entity my_mux is
   port (
      SEL : in STD_LOGIC_VECTOR(0 to 1);   -- Valdymo signalas (2 bitai)
      S0, S1, S2, S3 : in STD_LOGIC;      -- Duomenų įėjimai
      MUX_OUT : out STD_LOGIC             -- Multipleksoriaus išėjimas
   );
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;  -- Pasirenkamas S0
         when "01" => MUX_OUT <= S1;  -- Pasirenkamas S1
         when "10" => MUX_OUT <= S2;  -- Pasirenkamas S2
         when "11" => MUX_OUT <= S3;  -- Pasirenkamas S3
         when others => MUX_OUT <= 'X'; -- Neapibrėžta būsena
      end case;
   end process;
end architecture my_mux_arch;
```

3.2. Aprašykite septynių segmentų indikatoriaus dekoderį VHDL kalba.

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

entity Bin2LED is
   port (
      Data : in std_logic_vector(3 downto 0);   -- 4 bitų binarinis kodas
      Output : out std_logic_vector(6 downto 0) -- 7 segmentų valdymo signalai
   );
end Bin2LED;

architecture RTL of Bin2LED is
begin
   process (Data)
   begin
      case Data is
         when "0000" => Output <= "1111110"; -- Skaitmuo "0"
         when "0001" => Output <= "0110000"; -- Skaitmuo "1"
         when "0010" => Output <= "1101101"; -- Skaitmuo "2"
         when "0011" => Output <= "1111001"; -- Skaitmuo "3"
         when "0100" => Output <= "0110011"; -- Skaitmuo "4"
         when "0101" => Output <= "1011011"; -- Skaitmuo "5"
         when "0110" => Output <= "0011111"; -- Skaitmuo "6"
         when "0111" => Output <= "1110000"; -- Skaitmuo "7"
         when "1000" => Output <= "1111111"; -- Skaitmuo "8"
         when "1001" => Output <= "1111011"; -- Skaitmuo "9"
         when others => Output <= "0000000"; -- Neleistina kombinacija (išjungta)
      end case;
   end process;
end architecture RTL;
```

4. Nuosekliosios logikos aprašymas VHDL kalba

4.1. Kuo pasižymi nuoseklioji logika?

Nuoseklioji logika pasižymi tuo, kad įrenginiai turi atmintį, kurioje saugoma vidinė grandinės būsena. Skirtingai nei kombinacinėje logikoje, čia išėjimo reikšmė priklauso ne tik nuo esamų įėjimo signalų, bet ir nuo ankstesnės vidinės būsenos. Dažniausiai šios grandinės yra sinchroninės, o jų atminties elementai (pavyzdžiui, trigeriai) valdomi globaliu taktiniu signalu.

4.2. Aprašykite D trigerį (flip-flop) VHDL kalba.

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

entity d_ff is
   port(
```

```
        clk : in std_logic; -- Taktinis signalas
        D   : in std_logic; -- Duomenų įėjimas
        Q   : out std_logic -- Išėjimas
    );
end d_ff;

architecture d_arch of d_ff is
begin
    process (clk)
    begin
        -- Reaguojama tik į taktinio signalo kylantį frontą
        if (rising_edge(clk)) then
            Q <= D; -- Reikšmės priskyrimas [cite: 2277-2280]
        end if;
    end process;
end d_arch;
```

4.3. Aprašykite 4 bitų postūmio registrą VHDL kalba.

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

entity shift_reg is
    generic(N: integer := 4); -- Registro ilgis nustatomas parametru [cite: 2340]
    port(
        clk, reset : in std_logic;
        d_in       : in std_logic;  -- Nuoseklus įėjimas
        d_out      : out std_logic -- Nuoseklus išėjimas
    );
end shift_reg;

architecture arch4 of shift_reg is
    signal r_reg: std_logic_vector(N-1 downto 0) := "0000"; -- Vidiniai registro bitai [cite: 2407]
begin
    process (clk, reset)
    begin
        if (reset = '1') then
            r_reg <= (others => '0'); -- Asinchroninis nunulinimas [cite: 2411, 2412]
        elsif (rising_edge(clk)) then
            -- Bitų postūmis: naujas bitas d_in įrašomas į 0 poziciją, kiti paslenkami
            r_reg <= r_reg(N-2 downto 0) & d_in; -- Konkatenacijos naudojimas [cite: 2399]
        end if;
    end process;

    d_out <= r_reg(N-1); -- Paskutinis bitas perduodamas į išėjimą [cite: 2426]
end arch4;
```

4.4. Aprašykite dvejetainį 8 bitų skaitiklį VHDL kalba.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all; -- Reikalinga aritmetinėms operacijoms [cite: 2927, 2929]

entity counter_8bit is
    port (
        CLK   : in STD_LOGIC; -- Taktinis signalas
        RESET : in STD_LOGIC; -- Asinchroninis numetimas
        COUNT : out STD_LOGIC_VECTOR (7 downto 0) -- 8 bitų išėjimas [cite: 2947]
    );
end counter_8bit;

architecture arch_counter of counter_8bit is
    signal COUNT_INT: STD_LOGIC_VECTOR (7 downto 0) := X"00"; -- Vidinis signalas [cite: 2951]
begin
    process (CLK, RESET)
    begin
        if RESET = '1' then
            COUNT_INT <= (others => '0'); -- Nunulinimas [cite: 2954, 2955]
```

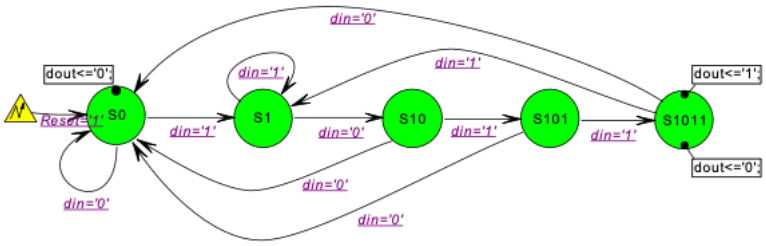
```
    elsif rising_edge(CLK) then
        COUNT_INT <= COUNT_INT + 1; -- Inkrementavimas (didinimas vienetu) [cite: 2966]
    end if;
  end process;

  COUNT <= COUNT_INT; -- Vidinės reikšmės priskyrimas išėjimui [cite: 2980]
end arch_counter;
```

5. Įrenginių projektavimas, panaudojant būsenų diagramas

5.1. Kas yra baigtinių būsenų automatas (angl. FSM)?

Baigtinių būsenų automatas (FSM) yra sinchroninis įrenginys, sudarytas iš nuosekliosios ir kombinacinės logikų, kurio būsena ir išėjimai kinta pagal apibrėžtą seką, reaguojant į taktinius impulsus bei įėjimo signalus. Pagal veikimo principą šie automatai skirstomi į Muro (Moore) ir Milio (Mealy) tipus. Formaliai FSM aprašomas parametrų rinkiniu, kurį sudaro baigtinės būsenų, įėjimų bei išėjimų aibės, pradinė būsena bei perėjimo ir išėjimo funkcijos.



5.2. Pateikite skaitmeninių įranginių pavyzdžių, kurie gali būti aprašomi baigtinių būsenų automatais.

Baigtinių būsenų automatais (FSM) aprašomi tokie skaitmeniniai įrenginiai kaip bitų sekų aptikliai, pavyzdžiui, skirti specifinei kombinacijai „1011“ atpažinti. Šiuo principu taip pat veikia įvairūs skaitikliai (binariniai, dešimtainiai ar žiediniai), atsitiktinių skaičių generatoriai bei postūmio registrai. FSM modelis yra pagrindas bet kuriai sinchroninei sistemai su atmintimi, įskaitant trigerius ar multivibratorius, kurios veikimas priklauso nuo ankstesnės būsenos

6. Baigtinių būsenų automatų aprašymas VHDL kalba

6.1. Kaip baigtinių būsenų automatai realizuojami VHDL kalba?

- Baigtinių būsenų automatai (FSM) VHDL kalba realizuojami kaip sinchroniniai įrenginiai, apjungiantys nuosekliąją (atminties) ir kombinacinę logikas. Realizavimo procesas susideda iš kelių pagrindinių etapų:
- Būsenų apibrėžimas: Pirmiausia sukuriamas specialus sąrašo tipo duomenų tipas (angl. enumerated type), kuriame išvardijamos visos galimos automato būsenos (pavyzdžiui, S0, S1, S2).
- Būsenos registro deklaravimas: Deklaruojamas signalas, kurio duomenų tipas yra anksčiau apibrėžtas būsenų sąrašas; šis signalas saugo einamąją įrenginio būseną.
- Logikos aprašymas procese: Automato elgsena aprašoma naudojant process konstrukciją, kuri yra jautri taktiniam signalui (clk) ir paprastai asinchroniniam numetimo signalui (reset).
- Taktinio fronto aptikimas: Proceso viduje naudojamas if-elsif sakinys, skirtas kylančiam taktinio signalo frontui nustatyti, naudojant tokias konstrukcijas kaip rising_edge(clk) arba clk'event and clk='1'.
- Būsenų perėjimų valdymas: Perėjimai tarp būsenų ir išėjimo signalų formavimas realizuojamas per case sakinį, kuriame pagal esamą būseną ir įėjimo signalų reikšmes nustatoma kita būsena.
- Automatų tipai: Priklausomai nuo to, kaip formuojami išėjimai, realizuojamas Muro (išėjimai priklauso tik nuo būsenos) arba Milio (išėjimai priklauso nuo būsenos ir įėjimų) automatas.
- Sintezės optimizavimas: Sintezės įrankiai automatiškai atpažįsta tokias kodo struktūras (atliekama FSM extraction) ir parenka optimalų būsenų kodavimo algoritmą konkrečiai FPGA architektūrai

7. Sintezė ir simuliavimas (testbench)

7.1. Kas yra ir kam naudojami virtualieji bandymo stendai (testbeches)?

7.2. Kokių tipų virtualiuosius stendus žinote?

Virtualusis bandymo stendas yra aukščiausio hierarchijos lygmens VHDL objektas, naudojamas sukurtų skaitmeninių įrenginių modelių verifikavimui ir funkciniam modeliavimui atlikti,. Jis skirtas testuojamai sistemai žadinti vartotojo aprašytais stimulais bei gautų reakcijų analizei, siekiant įsitikinti sistemos funkcionavimo teisingumu,. Pilnieji stendai

leidžia ne tik formuoti stimulus, bet ir automatiškai lyginti įrenginio išėjimus su laukiamais rezultatais, taip nustatant projekto klaidas,.

7.3. Pateikite paprasto kombinacinio loginio elemento IR-NE, 3ARBA, 3IR-NE virtualiojo bandymų stendo aprašą VHDL kalba.

Ir ne dvieju iejimu nand

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

entity tb_nand2 is
-- Tuščia sąsaja
end tb_nand2;

architecture sim of tb_nand2 is
   -- Komponento deklaravimas (turi sutapti su jūsų NAND elementu)
   component nand2_gate
      port ( A, B : in std_logic; Y : out std_logic );
   end component;

   -- Vidiniai signalai jungimui
   signal s_a, s_b : std_logic := '0';
   signal s_y      : std_logic;

begin
   -- Įrenginio prijungimas
   UUT: nand2_gate port map ( A => s_a, B => s_b, Y => s_y );

   -- Stimulų generavimas
   process
   begin
      s_a <= '0'; s_b <= '0'; wait for 10 ns;
      s_a <= '0'; s_b <= '1'; wait for 10 ns;
      s_a <= '1'; s_b <= '0'; wait for 10 ns;
      s_a <= '1'; s_b <= '1'; wait for 10 ns;
      wait; -- Simuliacijos pabaiga
   end process;
end sim;
```

3 arba

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

entity tb_or3 is
end tb_or3;

architecture sim of tb_or3 is
   component or3_gate
      port ( IN1, IN2, IN3 : in std_logic; OUT_VAL : out std_logic );
   end component;

   signal s_1, s_2, s_3 : std_logic := '0';
   signal s_res         : std_logic;

begin
   UUT: or3_gate port map ( IN1 => s_1, IN2 => s_2, IN3 => s_3, OUT_VAL => s_res );

   process
   begin
      -- Logiškai pereiname per visas būsenas binarine seka
      s_1 <= '0'; s_2 <= '0'; s_3 <= '0'; wait for 10 ns;
      s_1 <= '0'; s_2 <= '0'; s_3 <= '1'; wait for 10 ns;
      s_1 <= '0'; s_2 <= '1'; s_3 <= '0'; wait for 10 ns;
      s_1 <= '0'; s_2 <= '1'; s_3 <= '1'; wait for 10 ns;
      s_1 <= '1'; s_2 <= '0'; s_3 <= '0'; wait for 10 ns;
      s_1 <= '1'; s_2 <= '0'; s_3 <= '1'; wait for 10 ns;
      s_1 <= '1'; s_2 <= '1'; s_3 <= '0'; wait for 10 ns;
      s_1 <= '1'; s_2 <= '1'; s_3 <= '1'; wait for 10 ns;
```

```
        wait;
    end process;
end sim;
```

3ir ne nand

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

entity tb_nand3 is
end tb_nand3;

architecture sim of tb_nand3 is
    component nand3_gate
        port ( A, B, C : in std_logic; Y : out std_logic );
    end component;

    signal sa, sb, sc : std_logic := '0';
    signal sy         : std_logic;

begin
    UUT: nand3_gate port map ( A => sa, B => sb, C => sc, Y => sy );

    process
    begin
        -- Stimulai: nuo 000 iki 111
        sa <= '0'; sb <= '0'; sc <= '0'; wait for 10 ns;
        sa <= '0'; sb <= '0'; sc <= '1'; wait for 10 ns;
        sa <= '0'; sb <= '1'; sc <= '0'; wait for 10 ns;
        sa <= '0'; sb <= '1'; sc <= '1'; wait for 10 ns;
        sa <= '1'; sb <= '0'; sc <= '0'; wait for 10 ns;
        sa <= '1'; sb <= '0'; sc <= '1'; wait for 10 ns;
        sa <= '1'; sb <= '1'; sc <= '0'; wait for 10 ns;
        sa <= '1'; sb <= '1'; sc <= '1'; wait for 10 ns;
        wait;
    end process;
end sim;
```

8. FPGA konfigūravimo būdai

8.1. Konfigūruojamieji FPGA elementai.

FPGA programavimas iš esmės yra jos ventilių masyvo elementų būsenų nustatymas, kur kiekvieno elemento būsenai nusakyti reikalingas vienas bitas atminties. Pagrindiniai konfigūruojami elementai yra konfigūruojamo loginio bloko funkcija (LUT lentelės) bei komutacinės matricos sujungimai. Taip pat konfigūruojami I/O blokai, RAM blokai bei specializuoti aritmetiniai ar sinchronizavimo moduliai

8.2. Kokios yra FPGA konfigūravimo technologijos? Privalumai, trūkumai.

• SRAM: Privalumai – neribotas perrašymų skaičius ir paprasta konstrukcija, todėl puikiai tinka prototipams. Trūkumai – energetinė priklausomybė (reikia perprogramuoti kaskart įjungus maitinimą), dideli celių gabaritai ir nuotėkio srovės sukeliamas didelis galios suvartojimas.
• EEPROM (FLASH): Privalumai – energetiškai nepriklausoma (paruošta darbui tik padavus maitinimą), mažesnė nuotėkio srovė ir fiziškai mažesnės celės už SRAM. Trūkumai – sudėtinga gamyba, didesnis vėlinimas ir ilgesnis rašymo laikas.
• Antisaugikliai (Anti-fuse): Privalumai – maži gabaritai, mažas energijos suvartojimas, nereikia išorinės atminties ir atsparumas radiacijai. Trūkumas – negalima perprogramuoti.

8.3. Kam naudojami antisaugikliai?

Antisaugikliai naudojami kaip vienkartinio programavimo jungtys, kurios prieš programavimą yra atjungtos (didelė varža), o programavimo metu jų dielektrinė struktūra suardoma ir varža sumažėja. Jie leidžia sukurti mažus, energetiškai efektyvius ir aplinkos poveikiui (pvz., radiacijai) atsparius įrenginius, kuriems nereikia papildomų komponentų programos saugojimui.

8.4. Kokio tipo FPGA konfigūravimą naudotumėte norėdami apsaugoti savo programos kodą?

Siekiant apsaugoti programos kodą, rekomenduojama naudoti Flash FPGA arba Antifuse FPGA, nes jos priskiriamos aukštam 4-am saugumo lygiui (stipri apsauga nuo atakų). Įprastos SRAM FPGA pasižymi labai prastu saugumu (0 lygis), nors situaciją galima pagerinti naudojant SRAM FPGA su bitų srauto šifravimu (3 saugumo lygis)

8.5. Kuo skiriasi aktyvusis ir pasyvusis FPGA konfigūravimas?

- **Aktyvusis konfigūravimas:** Konfigūruojama FPGA pati generuoja visus valdymo bei sinchronizavimo signalus, naudodama nuoseklius konfigūravimo įrenginius.
- **Pasyvusis konfigūravimas:** Naudojamas išorinis konfigūravimo įrenginys (pvz., mikroprocesorius), kuris visiškai valdo procesą ir siunčia duomenis iš laikmenos į FPGA.

9. Programuojamų loginių įrenginių (FPGA) projektavimo procesas

9.1. Xilinx firmos projektavimo įrankiai ir jų tarpusavio ryšiai

Pagrindinės Xilinx projektavimo aplinkos yra ISE Design Suite ir ją pakeitusi moderni Vivado Design Suite . Jose integruotas XST įrankis atlieka HDL aprašų sintezę, o funkcionalumo patikrai naudojami vidiniai simuliatoriai – ISim arba XSim . Šie įrankiai glaudžiai sąveikauja, nukreipdami projektą nuo aprašo įvedimo iki galutinio realizavimo ir konfigūravimo failo generavimo

9.2. FPGA projekto aprašo įvedimo variantai.

Projekto aprašas gali būti įvedamas naudojant HDL kalbas (VHDL arba Verilog) bei elektrines schemas, nors pastarasis būdas palaikomas tik ISE aplinkoje . Taip pat galima įkelti paruoštus EDIF arba NGC/NGO failus, jei pirminė sintezė buvo atlikta ne Xilinx projektavimo terpėje . Šiuolaikiniame projektavimo procese HDL modeliavimas yra dominuojantis metodas dėl savo lankstumo ir suderinamumo

9.3. Tipiniai FPGA projekto etapai

Tipinis projektavimas prasideda nuo planavimo ir kodo kūrimo, po kurio seka pirminis HDL RTL modeliavimas ir sintezė grandinės aprašui gauti . Toliau vykdomas realizavimas (apimantis transliavimą, išdėstymą bei trasavimą) ir galutinis laikinių parametrų tikrinimas . Paskutiniame etape generuojamas konfigūracinis (BIT) failas, kuriuo suprogramuojamas fizinis FPGA lustas .

9.4. Kokius ribojimai naudojami sintezuojant ir realizuojant FPGA projektus?

Projektavimo metu taikomi trijų tipų ribojimai (angl. constraints): laikiniai (greitaveikai nustatyti), talpinimo (I/O prievadams ir blokų vietai nurodyti) bei paties sintezės proceso ribojimai . Šie reikalavimai nurodomi specialiuose UCF (ISE aplinkoje) arba XDC (Vivado aplinkoje) failuose, kurie valdo įrankių optimizavimo strategijas . Teisingas ribojimų įvedimas yra būtinas norint pasiekti sėkmingą projekto „laikinį užbaigimą“ (angl. timing closure) .

9.5. Kokie žingsniai sudaro FPGA projektavimo proceso realizavimo (angl. implementation) etapą?

Realizavimo etapą sudaro transliavimas (netlistų ir ribojimų apjungimas), išdėstymas (pritaikymas prie tikslinio įrenginio resursų) bei talpinimas ir trasavimas . Šio etapo metu loginiai ryšiai paverčiami fiziniais sujungimais kristale, griežtai laikantis vartotojo nustatytų laikinių ribojimų . Galiausiai sugeneruojamas programavimo failas (angl. bitstream), paruoštas siuntimui į fizinį įrenginį .

9.6. Projektavimas naudojant intelektinės nuosavybės (IP) šerdis.

Šiuolaikinis projektavimas remiasi IP šerdimis – paruoštais ir patikrintais loginiais moduliais (pvz., DSP blokais ar procesoriais), esančiais gamintojo kataloguose . Naudojant Vivado IP katalogą, projektuotojai gali greitai pasirinkti ir sukonfigūruoti sudėtingas funkcijas, taip žymiai pagreitindami sistemų kūrimą . Tai leidžia inžinieriams sutelkti dėmesį į specifinę projekto logiką, naudojant jau optimizuotus standartinius komponentus .

10. Diskretinių schemų sintezė

10.1. Kas yra loginė sintezė?

Loginė sintezė yra procesas, kurio metu aukštesnio abstrakcijos lygmens RTL aprašas (pavyzdžiui, parašytas VHDL kalba) transformuojamas į žemesnio lygmens loginių ventilių grandinę. Šio etapo metu sukuriamas loginis tinklas pagal projekte nurodytas lygtis arba teisingumo lentelę. Sintezė paprastai yra daugiapakopė, apimanti pirminį lygčių tinklo kūrimą, technologijos nepriklausomą optimizavimą ir galutinį pritaikymą (angl. mapping) konkretiems techninės įrangos resursams.

10.2. Kas yra mintermas ir maxtermas?

Mintermas yra loginės sandaugos narys, kuriame sudauginami visi galimi funkcijos kintamieji tiesioginiu arba invertuotu pavidalu. Maxtermas apibrėžiamas kaip sumos narys, kuriame visi kintamieji arba jų inversijos yra apjungiami loginės sumos operacija. Kiekvienas toks narys atitinka vieną konkrečią teisingumo lentelės eilutę ir yra naudojamas sudarant standartines logines išraiškas.

10.3. Kas yra sumų sandauga ir sandaugų suma?

Sandaugų suma (SOP) yra loginė forma, sudaryta iš sudėtų (OR) mintermų, kurių kiekvienas aprašo teisingumo lentelės eilutę, kurioje išėjimo reikšmė lygi vienetui. Sumų sandauga (POS) yra išraiška, kurioje logiškai sudauginami (AND) maxtermai, sudaryti remiantis eilutėmis, kuriose išėjimo reikšmė lygi nuliui. Šios formos yra pagrindas optimizuojant skaitmenines schemas naudojant Karno diagramas arba tokius algoritmus kaip „Espresso“, siekiant minimizuoti reikiamą ventilių skaičių.

10.4. Užrašykite loginę funkciją duotą teisingumo lentele sandaugų sumos forma.

Sandaugų sumos forma (angl. *Sum-of-Products*, SOP) sudaroma analizuojant teisingumo lentelę ir išrenkant tas eilutes, kuriose funkcijos reikšmė lygi **1** (vadinamieji mintermai).
SOP formos sudarymo žingsniai:

1. Suraskite visas teisingumo lentelės eilutes, kur išėjimas **Y = 1**.
2. Kiekvienai tokiai eilutei užrašykite **sandaugos narį** (mintermą):
    - Jei kintamojo reikšmė eilutėje yra **1**, rašomas pats kintamasis (pvz., A).
    - Jei kintamojo reikšmė eilutėje yra **0**, rašomas invertuotas kintamasis (pvz., A^- arba !A).
3. Visus gautus sandaugos narius sujunkite loginės **sumos (ARBA)** operacija.

Pavyzdys iš skaidrių: Jei turime 3 įėjimus (a,b,c) ir išėjimas lygus 1, kai įėjimai yra `001`, `011`, `100`, `101`, funkcija užrašoma taip:
$y=(a^-\cdot b^-\cdot c)+(a^-\cdot b\cdot c)+(a\cdot b^-\cdot c^-)+(a\cdot b^-\cdot c)$

10.5. Susintezuokite loginį ventilį XOR iš bazinių ventilių. (UŽDAVINYS)

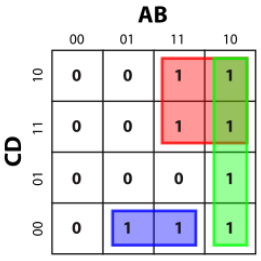
Loginis elementas XOR (suma moduliu 2) išduoda „1“ tik tada, kai jo įėjimai yra skirtingi. Norint jį susintetinti iš bazinių ventilių (IR, ARBA, NE), pirmiausia užrašoma jo SOP išraiška.
1. XOR teisingumo lentelė: | A | B | Y (XOR) | | :--- | :--- | :--- | | 0 | 0 | 0 | | 0 | 1 | 1 | | 1 | 0 | 1 | | 1 | 1 | 0 |
2. SOP išraiška: Iš lentelės matome, kad Y=1, kai (A=0,B=1) arba (A=1,B=0).
$Y=(A^-\cdot B)+(A\cdot B^-)$
3. Realizacija baziniais ventiliais: Norint gauti šią funkciją, mums reikia:

- Dviejų NE (NOT) ventilių (įėjimams A ir B invertuoti).
- Dviejų IR (AND) ventilių (sandaugoms $A^-\cdot B$ ir $A\cdot B^-$ gauti).
- Vieno ARBA (OR) ventilio (gautoms sandaugoms sudėti).

Ši struktūra yra standartinis XOR realizavimo būdas skaitmeninėje logikoje.

10.6. Kam naudojamos Karno diagramos?

Karno diagramos naudojamos siekiant optimaliai generuoti ir supaprastinti logines funkcijas remiantis pateiktomis teisingumo lentelėmis. Tai yra grafinis įrankis, leidžiantis rasti minimalią sandaugų sumos (SOP) arba sumų sandaugos (POS) formą, apjungiant gretimus langelius kintamiesiems eliminuoti. Šis metodas dažniausiai taikomas rankiniam skaitmeninių schemų optimizavimui, kai įėjimo kintamųjų skaičius yra nedidelis (iki 6).



10.7. Sudarykite teisingumo lentele pateiktos loginės funkcijos schemą, naudodami bazinius ventilius (panaudokite Karno diagramų metodą). (UŽDAVINYS)

Tarkime, turime 3 įėjimų (a,b,c) funkciją, kurios išėjimas y yra lygus „1“ šioms kombinacijoms: `001`, `011`, `100`, `101`.

a	b	c	y
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

Sumazinus lentele gaunam

a \ bc	00	01	11	10
0	0	1	1	0
1	1	1	0	0

Išskiriame kontūrus (grupes), kuriuos sudaro 2, 4 arba 8 gretimi vienetai:

1. Grupė 1 (horizontali pora viršuje): apima `001` ir `011`. Čia a=0 ir c=1 nesikeičia, o b kinta. Gauname dėmenį: $a\bar{c}$.
2. Grupė 2 (horizontali pora apačioje): apima `100` ir `101`. Čia a=1 ir b=0 nesikeičia, o c kinta. Gauname dėmenį: $ab\bar{}$.

Minimali SOP forma:
$y=a\bar{c}+ab\bar{}$

10.8. Sudarykite multipleksoriaus 2->1 (du įėjimai, viena adreso linija, vienas išėjimas) loginę funkciją, panaudodami Karno diagramų metodą. (UŽDAVINYS)

Multipleksorius 2->1 turi:

- Du duomenų įėjimus: D0 ir D1.
- Vieną valdymo (adreso) liniją: S (Select).
- Vieną išėjimą: Y.

Veikimo principas: jei S=0, išėjimas Y atkartoja D0; jei S=1, išėjimas Y atkartoja D1.

S	D0	D1	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

S \ D0D1	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Išskiriame dvi grupes po du vienetus (kontūrus):

1. Grupė 1 (viršutinė eilutė, S=0): Apima langelius, kur S=0 ir D0=1. Kintamasis D1 čia keičiasi, todėl jis išnyksta. Gauname narį: $S^{-}\cdot D0$.
2. Grupė 2 (apatinė eilutė, S=1): Apima langelius, kur S=1 ir D1=1. Kintamasis D0 čia keičiasi, todėl jis išnyksta. Gauname narį: $S\cdot D1$.

5. Galutinė loginė funkcija

Sujungiame gautus narius į sandaugų sumos (SOP) formą:
$Y=S^{-}\cdot D0+S\cdot D1$

10.9. Kokie yra Karno diagramų metodo trūkumai?

Karno diagramos dažniausiai naudojamos rankiniam optimizavimui, nes šis grafinis metodas sunkiai pasiduoda kompiuterizavimui. Dirbant rankiniu būdu, ypač didėjant kintamųjų skaičiui, galima labai greitai pridaryti klaidų. Metodas yra efektyvus tik esant nedideliam įėjimų skaičiui – teoriškai iki 6, tačiau praktiškai jis patogus naudoti tik iki 4 kintamųjų.

10.10. Kas yra nereikšminga loginių kintamųjų kombinacija?

Nereikšminga kombinacija (angl. don't care) nurodo tokią įėjimo būseną, kurios išėjimo reikšmė neturi įtakos sistemos veikimui, todėl ji gali būti laikoma tiek nuliu, tiek vienetu. Karno diagramose tokios būsenos žymimos „X“ simboliu ir gali būti apjungiamos į kontūrus kartu su vienetais, siekiant gauti paprastesnę loginę lygtį. Šiuolaikinėse projektavimo aplinkose tokie „nesvarbūs“ kintamieji tekste dažnai žymimi brūkšneliu „-“,.

10.11. Kam skirtas ir kuo pasižymi Espresso algoritmas?

Espresso yra heuristinis loginis minimizatorius, sukurtas kaip kompiuterinė programa sudėtingoms loginėms funkcijoms optimizuoti. Jis tapo de-facto standartu, nes pasižymi mažesnėmis skaičiavimų apimtimis ir praktiškai neriboja įėjimų bei išėjimų skaičiaus (gali apdoroti dešimtis kintamųjų),. Algoritmas yra itin tinkamas CPLD ir FPGA sintezei, nes geba pritaikyti lygtis prie konkrečių gamintojo siūlomų bazinių ventilių