Въведение в VHDL

Последователни Оператори

Последователни оператори

Предназначени са за поведенческо описание.

Използват се само в **process**.

Изпълняват се в реда, в който са записани – т.е. последователно.

Последователни оператори

<= – присвояване (сигнал)

:= – присвояване (променлива)

if – условно изпълнение

case – разклонение

loop – цикъл

wait – изчакване

null – неизпълним оператор

Сигнално Присвояване "<="

Синтаксис

цел <= [transport] израз [after време];

цел – име на сигнал

Примери

A <= B;

A <= B nand C;

A <= B nand C after 0.2 ns;

"<=" Правила

Сигналното присвояване не променя веднага стойността на сигнала!

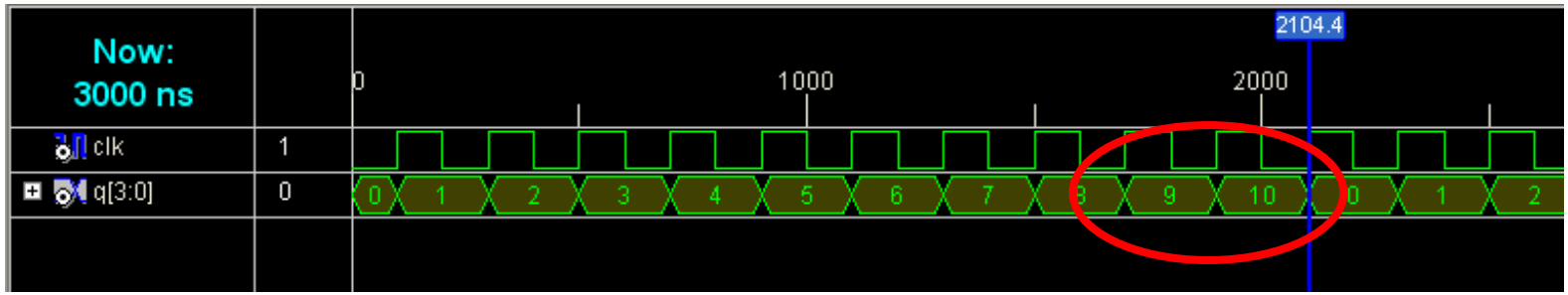
Всички закъснения са относителни спрямо момента, в който се изпълнява присвояването.

Сигнално присвояване с нулево закъснение ще предизвика събитие след една “делта”, т.е. след един симулационен цикъл.

```
entity counter is
    Port ( clk : in  std_logic;
           q : out  std_logic_vector(3 downto 0));
end counter;

architecture Proba of counter is
signal tmp: std_logic_vector(3 downto 0):="0000";
begin
process (CLK) begin
      if rising_edge(CLK) then
            TMP <= TMP + 1;
            if TMP = "1010" then
                  TMP <= "0000";
            end if;
      end if;
  end process;
  q <= tmp;
end Proba;
```

Сигналното присвояване не променя веднага стойността на сигнала!



"<=" Синтез

За целите на синтеза закъсненията се игнорират. Ако е необходиме, използвайте възможностите на програмата за синтез за да укажете изисквания желаните закъснения в критичните пътища.

Изразът от дясната страна на оператора за присвояване се синтезира като комбинационна логика.

“Целта” в лявата страна на оператора за присвояване се синтезира като:

–Връзка – при комбинационни процеси;

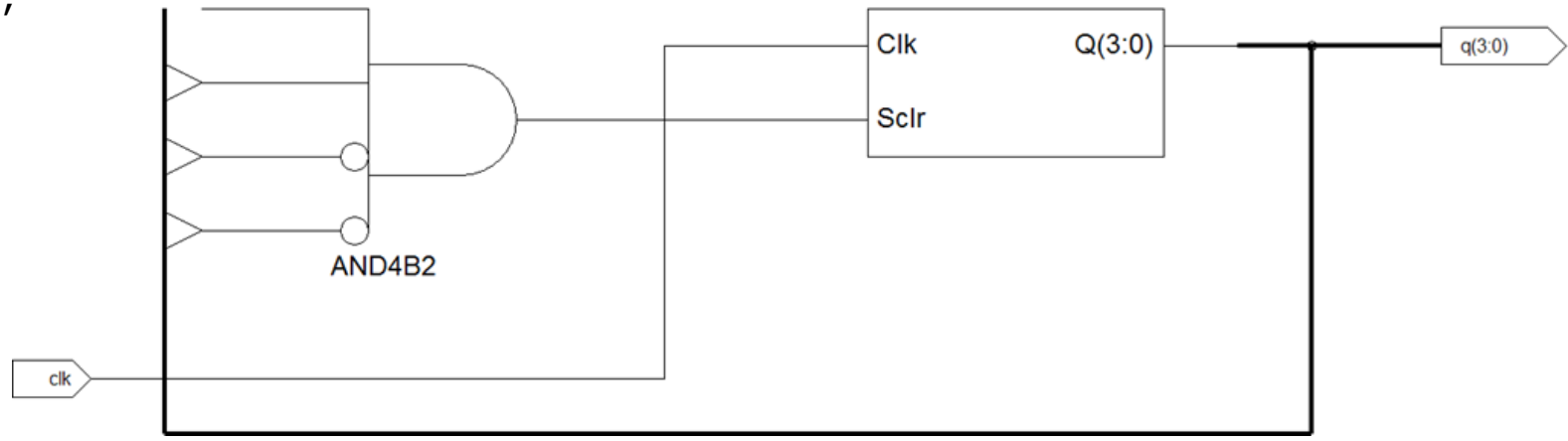
–Тригер управляван по ниво – при комбинационни процеси и непълно присвояване;

–Тригер или регистър – при тактувани процеси.

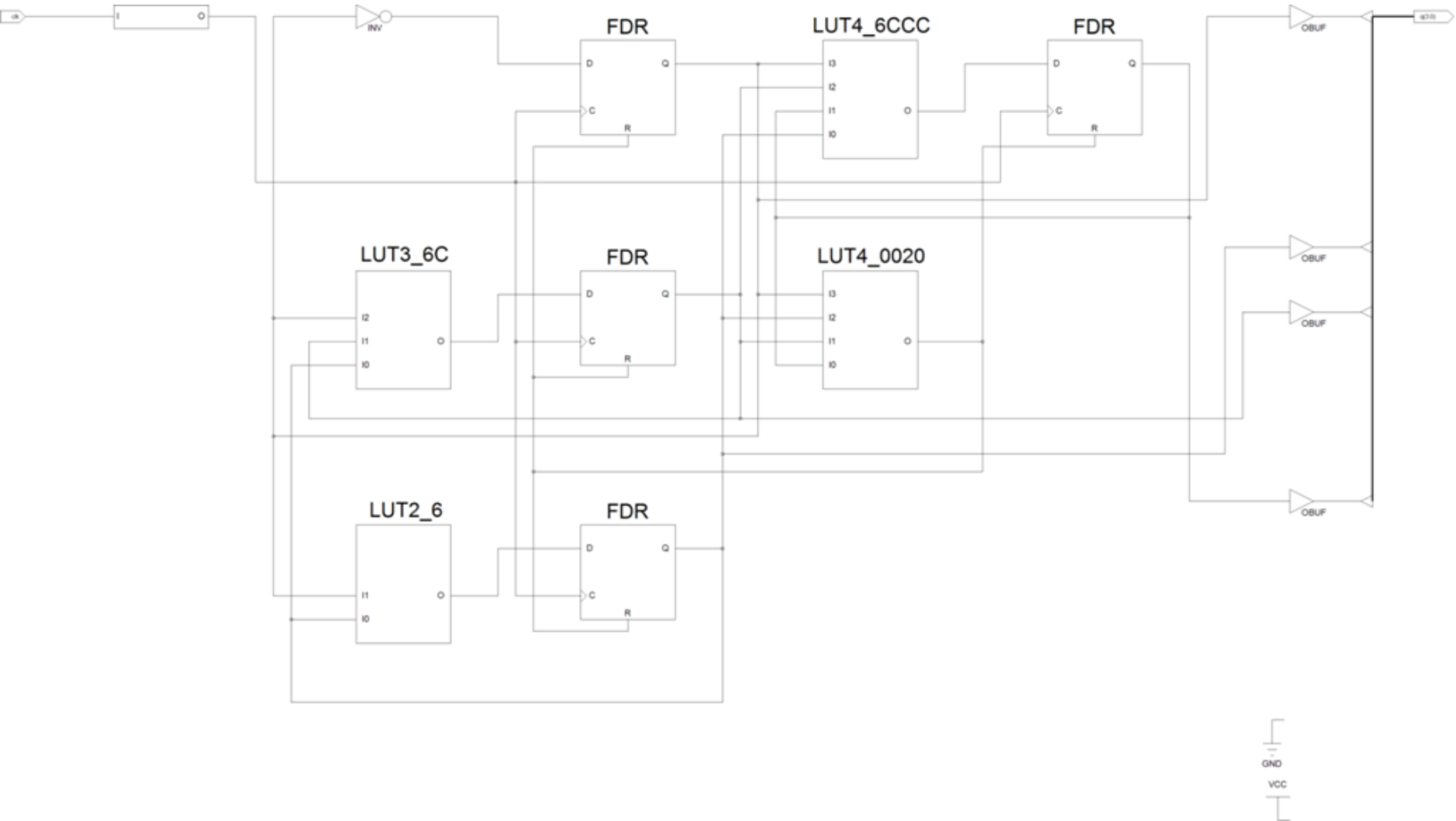
```
entity counter is
    Port ( clk : in  std_logic;
           q : out  std_logic_vector(3 downto 0));
end counter;

architecture Proba of counter is
signal tmp: std_logic_vector(3 downto 0):="0000";
begin
process (CLK) begin
       if rising_edge(CLK) then
             TMP <= TMP + 1;
             if TMP = "1010" then
                   TMP <= "0000";
             end if;
       end if;
  end process;
  q <= tmp;
end Proba;
```

Синтез
RTL схема
XST/Spartan3



Синтез
Технологична схема
XST/Spartan3



Пример

Непълно дефиниране на изхода Q1

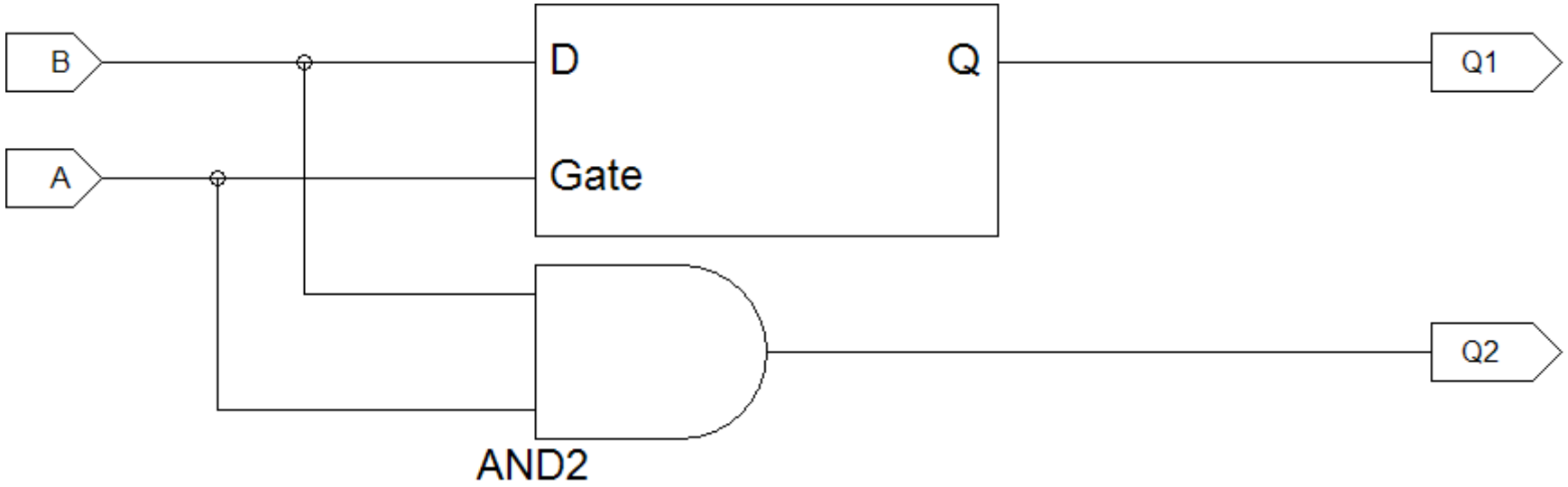
```
process (D, E) begin
    if E = '1' then
       Q1 <= D;
    end if;
  end process;
```

Пълно дефиниране на изхода Q2

```
process (D, E) begin
    if E = '1' then
       Q2 <= D;
    else
       Q2 <= '0';
    end if;
  end process;
```

Непълни присвоявания

```
architecture Test of fun is
begin
process (A,B) begin
        if A = '1' then
                Q1 <= B;
        end if;
end process;
process (A, B) begin
        if A = '1' then
                Q2 <= B;
        else
                Q2 <= '0';
        end if;
end process;
end Test;
```



Типове закъснения

Инерционно закъснение – импулси по-кратки от указаното закъснение се игнорират;

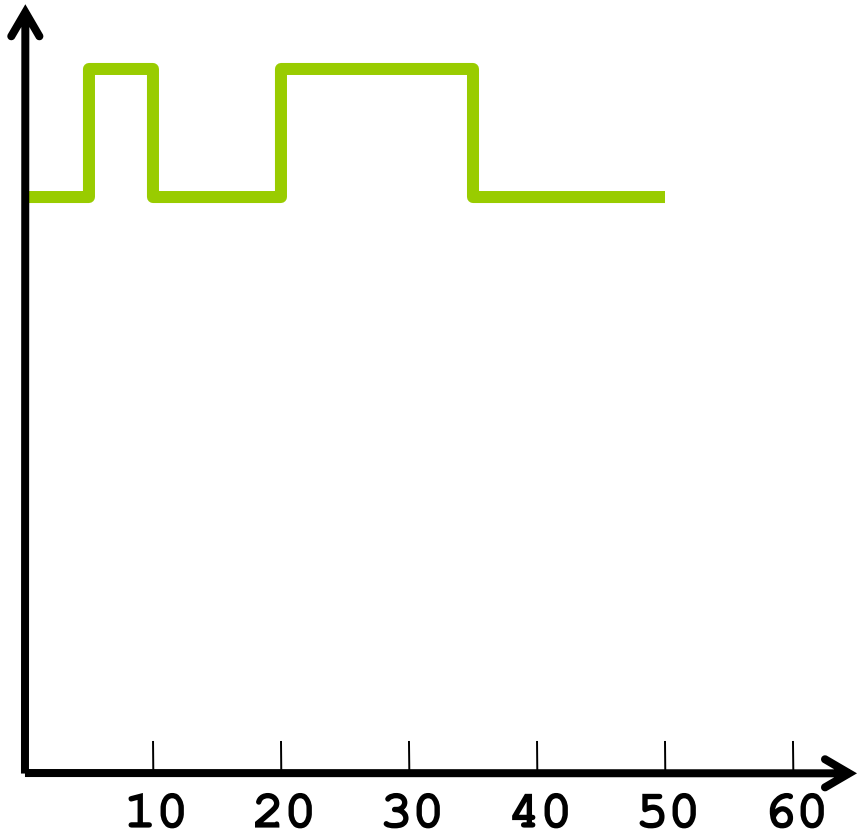
Транспортно закъснение – всички импусли се пропускат (закъснителна линия).

Типове закъснения - Пример

Входен сигнал (A)

B <= transport A after 10 ns;

C <= A after 10 ns;

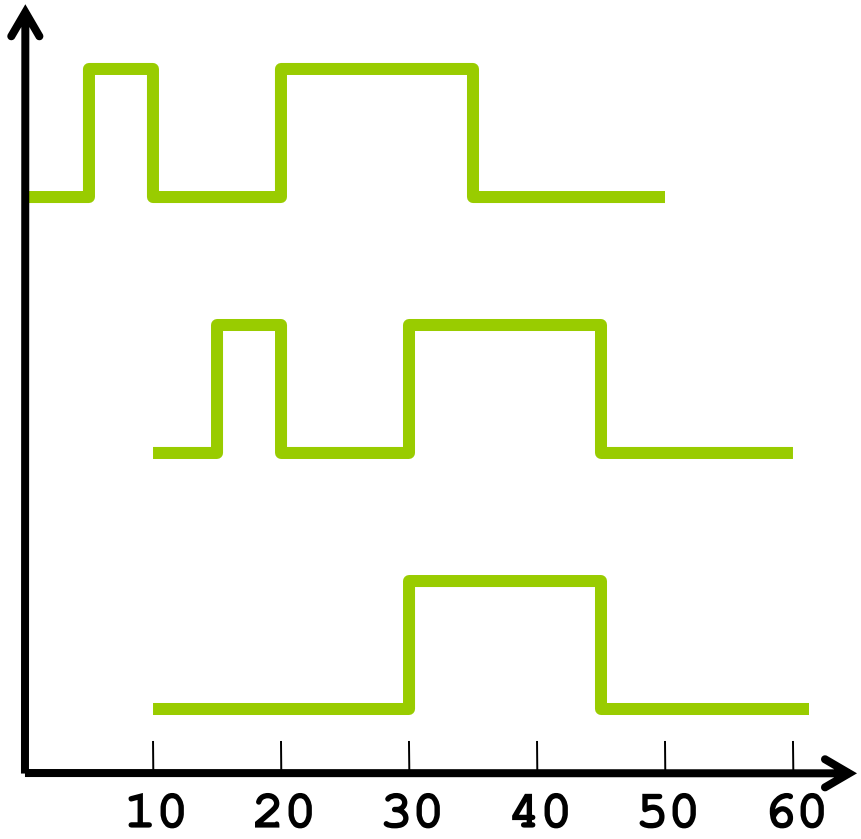


Типове закъснения - Пример

Сигнал A

B <= transport A after 10 ns;

C <= A after 10 ns;



Присвояване на Променлива ":="

NB! Промяната на стойността на променливата се извършва **незабавно**.

Синтаксис

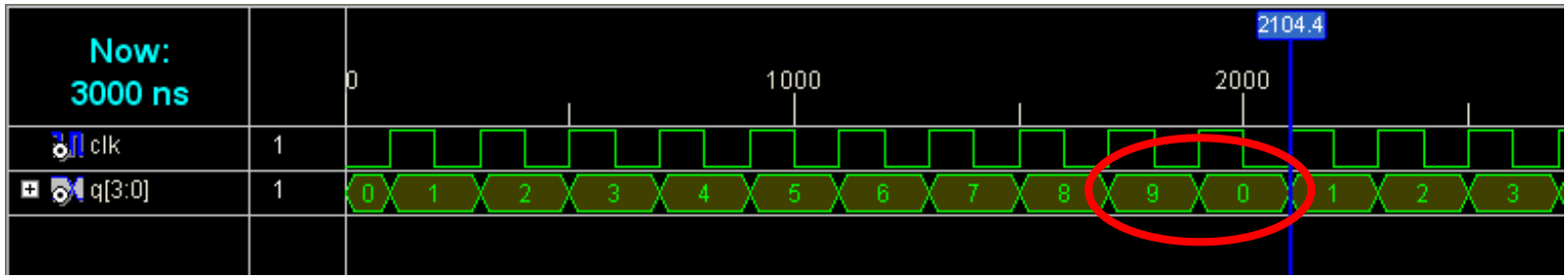
цел := израз;

цел – име на променлива

```
entity counter is
    Port ( clk : in  std_logic;
           q : out  std_logic_vector(3 downto 0));
end counter;

architecture Proba of counter is
begin
process (CLK)
      variable tmp: std_logic_vector(3 downto 0):="0000";
      begin
      if rising_edge(CLK) then
            tmp := tmp + 1;
            if tmp = "1010" then
                  tmp := "0000";
            end if;
      end if;
      q <= tmp;
  end process;
end Proba;
```

Промяната на стойността на променливата се извършва **незабавно**.



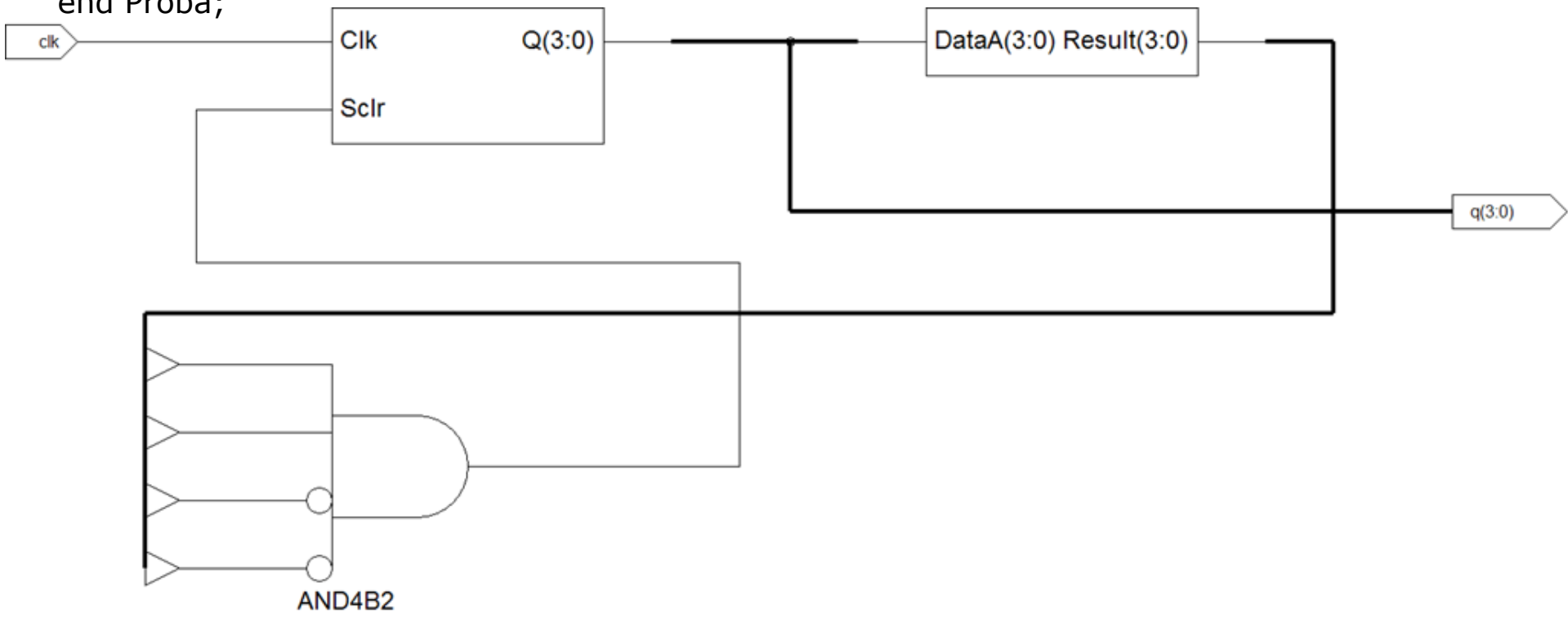
":=" Синтез

Изразът от дясната страна на оператора за присвояване се синтезира като комбинационна логика.

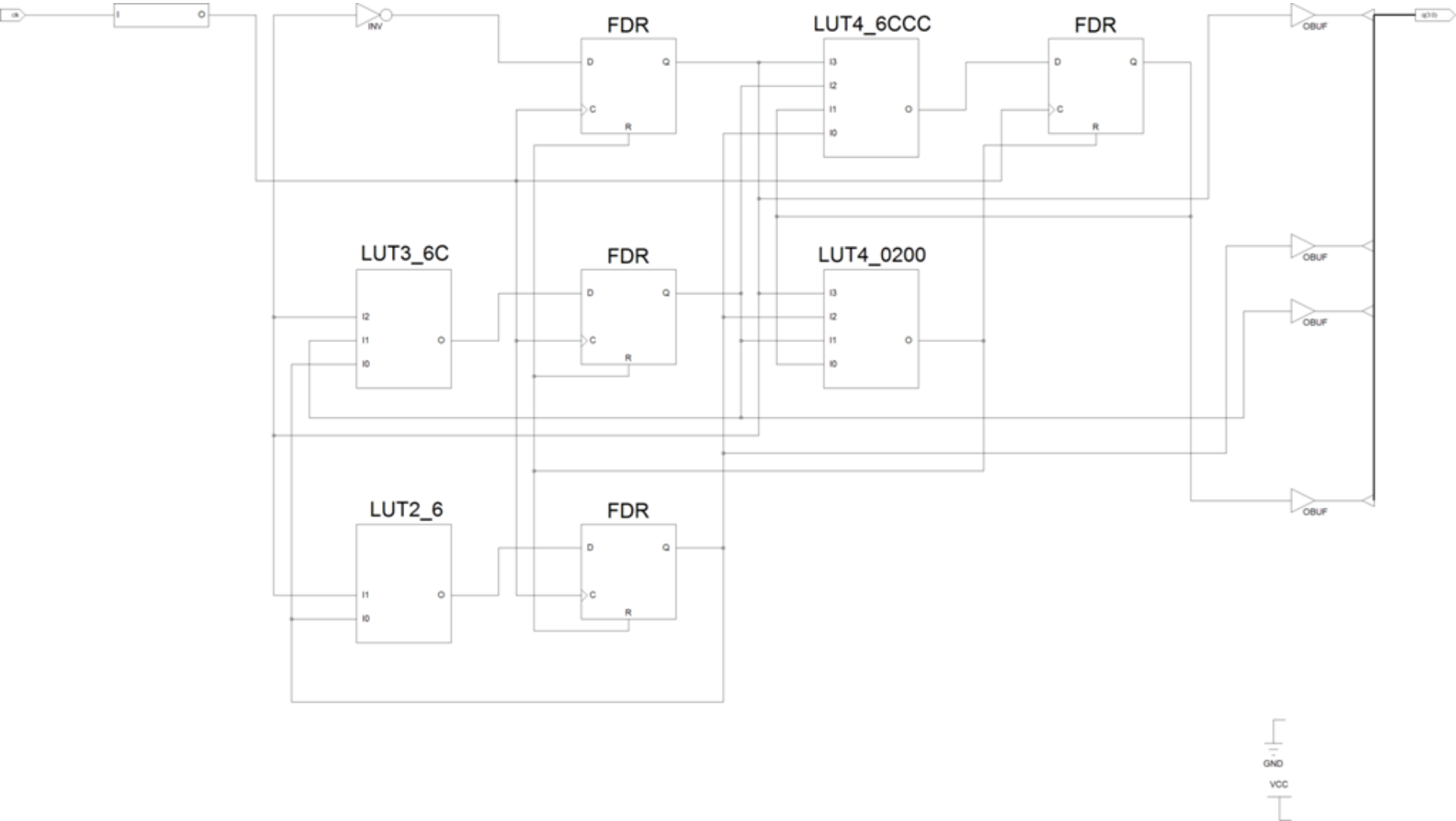
“Целта” в лявата страна на оператора за присвояване се синтезира като връзка или тригер в зависимост от това дали стойността на променливата се запазва между симулационните цикли.

```
architecture Proba of counter is
begin
process (CLK)
        variable tmp: std_logic_vector(3 downto 0):="0000";
        begin
        if rising_edge(CLK) then
                tmp := tmp + 1;
                if tmp = "1010" then
                        tmp := "0000";
                end if;
        end if;
        q <= tmp;
  end process;
end Proba;
```

Синтез
RTL схема
XST/Spartan3



Синтез
Технологична схема
XST/Spartan3



Условен Оператор - "if"

Синтаксис

if *условие* then

Последователни оператори ...

[elsif *условие* then

Последователни оператори …]

... {други elsif секции}

[else

Последователни оператори …]

end if;

"if" - Пример

```
architecture simple of BCNT is
     signal TMP: unsigned(3 downto 0) := (others=>'0');
begin
  increment: process (CLK) begin
    if rising_edge(CLK) then
      TMP <= TMP + 1;
    end if;
  end process;
  COUNT <= std_logic_vector(TMP);
end simple;
```

"if" - Правила

Реда, в който са записани условията в if и elsif секциите определя приоритета на тези условия и респективно реда, в който те ще бъдат тествани.

За да се извърши декодиране без да се дава приоритет на някои условия се използва оператора case вместо if.

"if" - Синтез

Обикновенно присвояванията в оператора if се синтезират като мултиплексори.

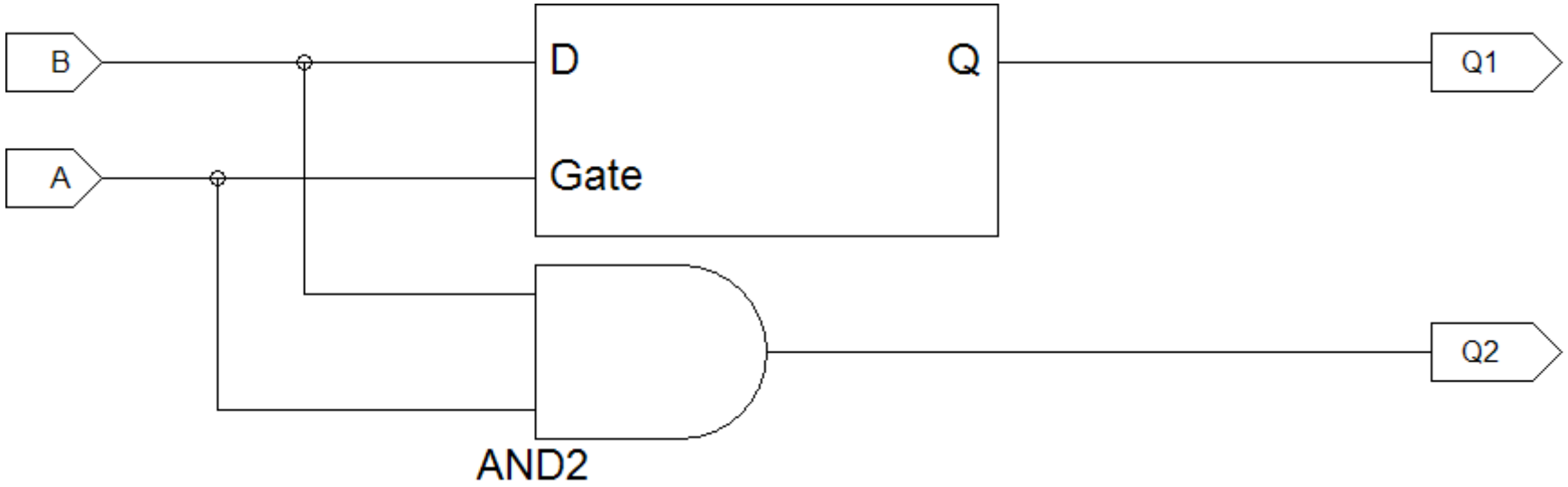
Непълните присвоявания (т.е. тогава когато изходите остават непроменени за някои входни комбинации) се синтезират като:

–Тригери управлявани по ниво – за нетактуваните (комбинационни) процеси;

–Обратни връзки обхващащи тригерите – за тактуваните процеси.

Непълни присвоявания (повторение)

```
architecture Test of fun is
begin
process (A,B) begin
        if A = '1' then
                Q1 <= B;
        end if;
end process;
process (A, B) begin
        if A = '1' then
                Q2 <= B;
        else
                Q2 <= '0';
        end if;
end process;
end Test;
```



Разклонение - "case"

Синтаксис

case *Израз* is

when *Избори* =>

Последователни оператори …

when *Избори* =>

Последователни оператори …

... {други when секции}

end case;

"case" - Пример

```
case ADDRESS is
when 0 => -- единична стойност
    A <= '1';
when 1 to 15 => -- диапазон
    C <= '1';
when 16 | 20 | 24 => -- множество стойности
    B <= '1';     C <= '1';   D <= '1';
when others => -- всички останали стойности
    null;
end case;
```

"case" - Правила

Типа на управляващият *Израз* трябва да бъде изброим, физически, integer или едномерен масив.

Всяка възможна стойност на *Израз*-а трябва да попада точно в един от *Избор*-ите.

Използвайте when others за да покриете всички останали стойности.

"case"- Пример

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

entity decoder is
    Port ( a : in  STD_LOGIC_VECTOR (3 downto 0);
           led : out  STD_LOGIC_VECTOR (7 downto 1));
end decoder;

architecture Behavioral of decoder is
function BIN2LED (BIN: STD_LOGIC_VECTOR (3 downto 0)) return STD_LOGIC_VECTOR is
      begin          -- 7-segment display decoder
            case BIN is
                  -- SEGMENTS: abcdefg
                  when "0000" => return "0000001";
                  when "0001" => return "1001111";
                  when "0010" => return "0010010";
                  when "0011" => return "0000110";
                  when "0100" => return "1001100";
                  when "0101" => return "0100100";
                  when "0110" => return "0100000";
                  when "0111" => return "0001101";
                  when "1000" => return "0000000";
                  when "1001" => return "0000100";
                  when others => return "1111111";
      end case;
      end;
begin
led <= BIN2LED( a );
end Behavioral;
```

"case"- Синтез

Обикновенно присвояванията в оператора case се синтезират като мултиплексори.

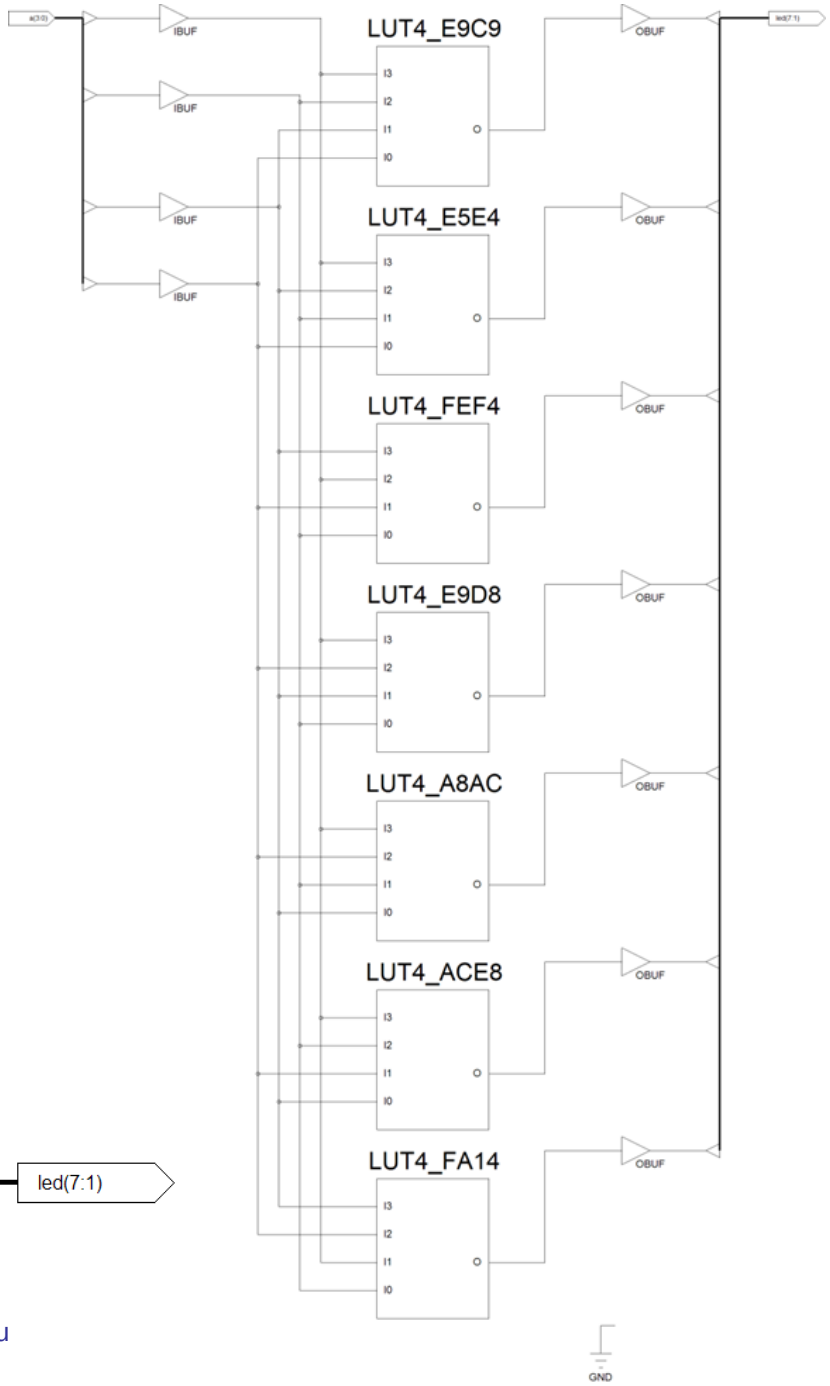
Непълните присвоявания, когато изходите остават непроменени за някои входни комбинации, се синтезират като:

–Тригери управлявани по ниво – за нетактуваните (комбинационни) процеси;

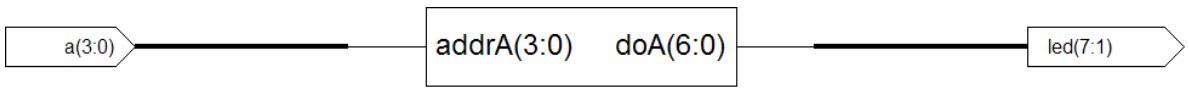
–Обратни връзки обхващащи тригерите – за тактуваните процеси.

"case"- Пример за синтез XST/Spartan3

```
 case BIN is
when "0000" => return "0000001";
        when "0001" => return "1001111";
        when "0010" => return "0010010";
        when "0011" => return "0000110";
        when "0100" => return "1001100";
        when "0101" => return "0100100";
        when "0110" => return "0100000";
        when "0111" => return "0001101";
        when "1000" => return "0000000";
        when "1001" => return "0000100";
        when others => return "1111111";
end case;
```



ROM



Цикъл - "for"

Циклично изпълнение на група последователни оператори. При всяко повторение на цикъла Параметъра получава последователни стойности от Диапазона.

Синтаксис

for *Параметър* in *Диапазон* loop

Последователни оператори …

end loop;

"for" - Пример

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

entity parity is
    Port ( D : in  STD_LOGIC_VECTOR (7 downto 0);
           ODD : out  STD_LOGIC);
end parity;

architecture Behavioral of parity is
begin
process (D)
    variable tmp: std_logic;
  begin
    tmp := '0';
    for i in 0 to 7 loop
       tmp := tmp xor D( i ) ;
    end loop ;
    ODD <= tmp;
  end process;
end Behavioral;
```

"for" – Правила

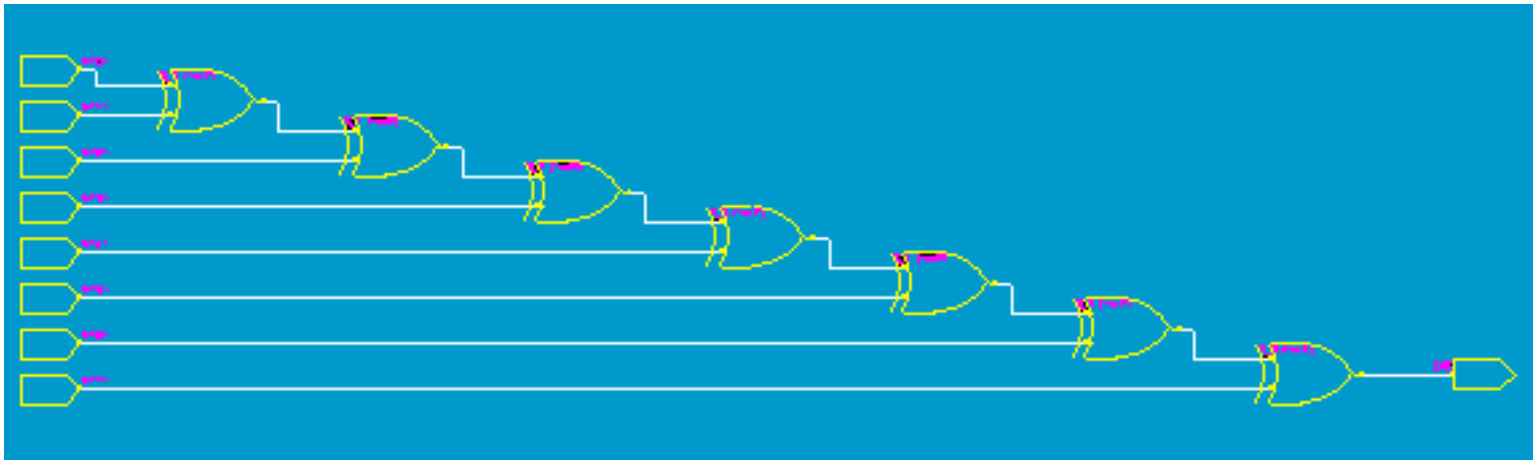
Параметърът на цикъла е деклариран само в рамките на цикъла.

Параметърът на цикъла е константна величина.

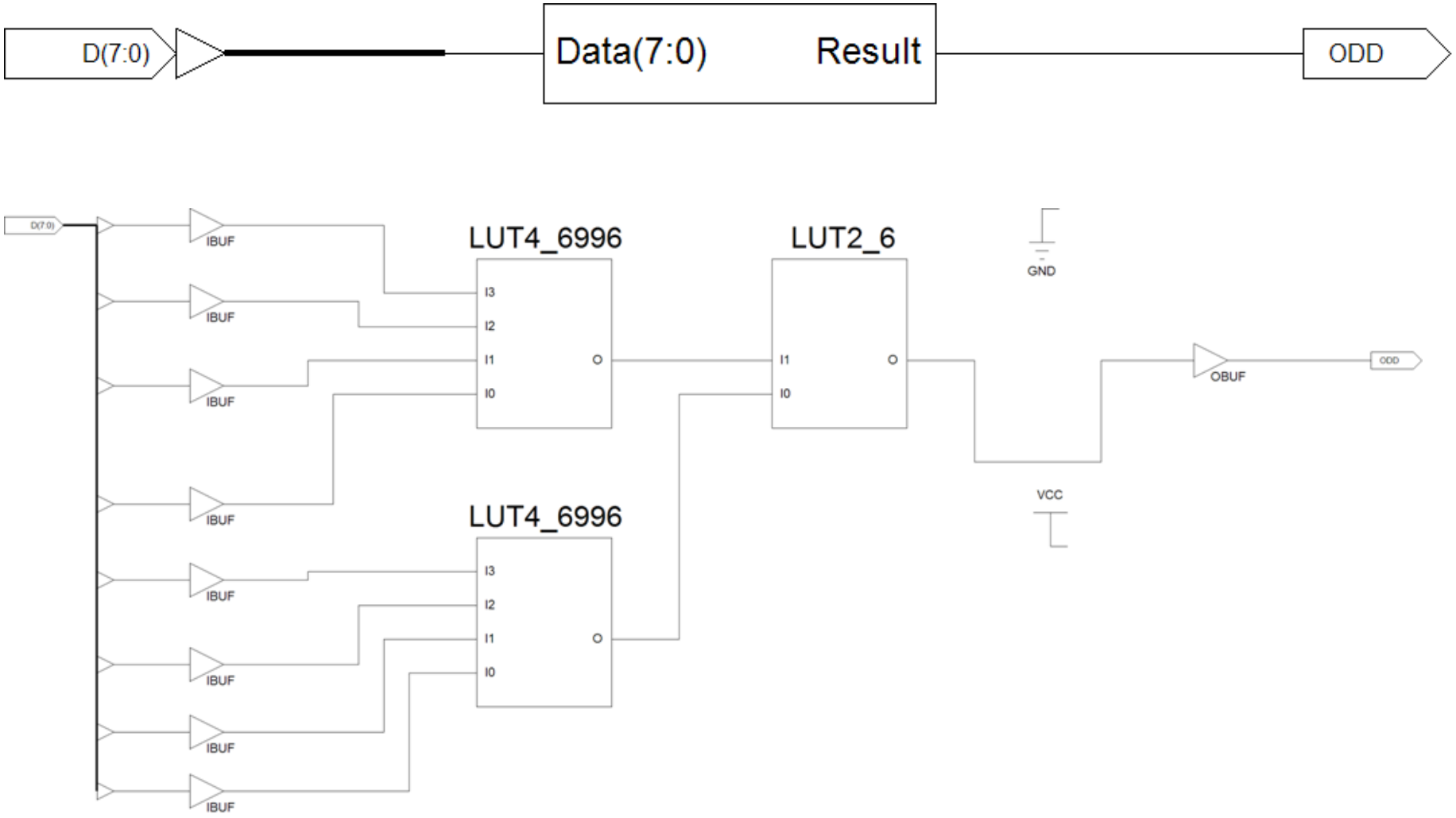
"for" – Синтез

При синтеза се създава по едно копие на логиката описана в тялото на цикъла за всяко завъртане на цикъла.

Диапазонът трябва да е статичен за да може цикъла да е синтезируем..



"for" – Пример за синтез XST/Spartan3



Оператор NULL

Оператор, който не извърша никакво действие.

Използва се в if и case за да се укаже в явен вид, че при определени входни комбинации не трябва де се извършват никакви действия.

Оператор WAIT

- Оператор който чака едно от следните събития:
  - Промяна на сигнал
  - Изпълнение на условие
  - Изтичане на период от време
- Синтаксис

  wait [on Сигнали] [until Условие] [for Време];
- Примери
  - wait on clk;
  - wait until clk = '1';
  - wait for 100 ns;
  - wait;

Оператор WAIT - Синтез

☹ не подлежат на синтез:

–**wait fo**r

–**wait on**

–**wait**

☺ wait until е синтезируем само когато се използва за синхронизиране на процес към фронт на такт. Пример:

wait until clock = '1';

WAIT - Пример

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

entity counter_wait is
    Port ( CLK : in  STD_LOGIC;
           Q1 : out  STD_LOGIC_VECTOR (3 downto 0);
           Q2 : out  STD_LOGIC_VECTOR (3 downto 0));
end counter_wait;

architecture Behavioral of counter_wait is

begin
process (CLK)
        variable tmp: std_logic_vector(3 downto 0):="0000";
begin
        if rising_edge(CLK) then
                tmp := tmp + 1;
        end if;
        Q1 <= tmp;
end process;
process
        variable tmp: std_logic_vector(3 downto 0):="0000";
begin
        wait until CLK = '1';
        tmp := tmp + 1;
        Q2 <= tmp;
  end process;
end Behavioral;
```