Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

**Национальный исследовательский университет «МЭИ»**
**Институт информационных и вычислительных технологий**

**Кафедра Вычислительных технологий**

# Арифметико-логическое устройство.

## Моделирование

## курсовой работы по курсу «Функциональные узлы и процессоры»

**Выполнил:**

Студент: Архипов Д. Г.
Группа: А-06-21
Вариант 1

**Проверил:**

Ключников А. М.

**Москва 2024**

# 1. Задание на курсовую работу

Цель: разработать схему арифметико-логического устройства (АЛУ), осуществляющую:

1) алгебраическое сложение двух многоразрядных чисел A и B в заданном коде (МДК или МОК);
2) операцию * над суммой чисел A и B.

Числа A и B поступают в АЛУ в нормализованном виде параллельным модифицированным прямым кодом на входы многоразрядных регистров RGA и RGB.

Результат алгебраического сложения передается в регистр RGS.

Перед выполнением операции алгебраического сложения числа A и B, заданные в модифицированном прямом коде, должны быть предварительно преобразованы в заданный код.

В сумматоре МДК входной перенос поступает извне, а выходной перенос не используется.

Исходные данные:

- Серия элементов: 155
- Вид операции *:

  Выделение признака: необходима нормализация вправо
- Тип сумматора: МДК
- Разрядность SM: 12
- $T_{\max зад}$ SM: 148 нс

Провести расчеты:

1) Задержку самого длинного тракта SM;

2) Расчет тактовой частоты.

## 2. Разработка структурной схемы АЛУ

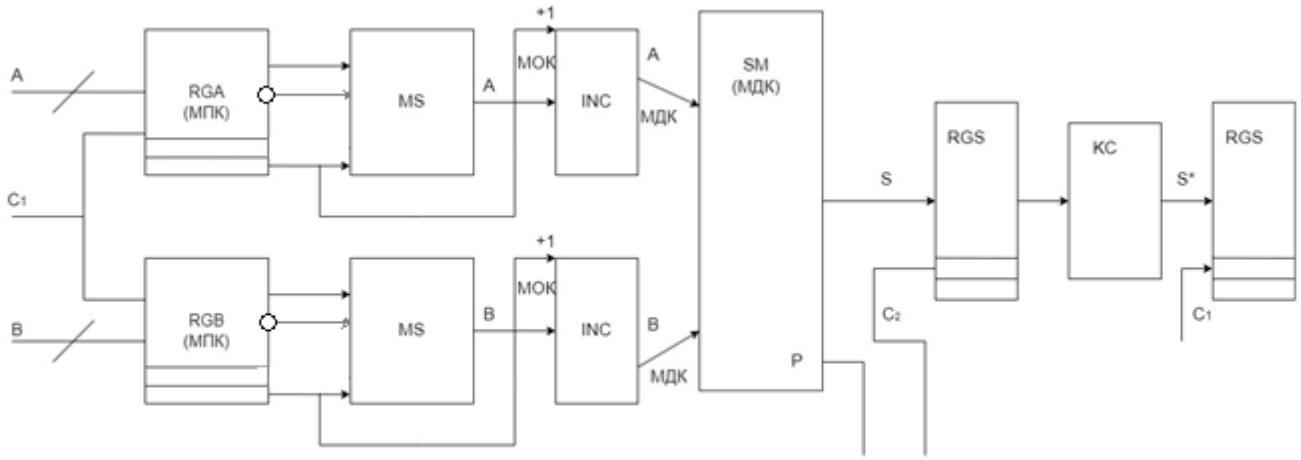


Рисунок 1. Структурная схема АЛУ

Обозначения на схеме:

A, B – многоразрядные числа,

RGA – регистр числа A,

RGB – регистр числа B,

MS – мультиплексор,

SM – сумматор,

INC – инкрементор,

RGS – регистр результата сложения A и B и поиска старшего разряда,

КС – комбинационная схема,

RGS* – регистр конечного результата,

$C_1$ и $C_2$ – синхросигналы,

S, S* - значения суммы,

МПК – модифицированный прямой код,

МДК – модифицированный дополнительный код.

Принцип работы: на входы регистров RGA и RGB подаются двенадцатиразрядные сигналы A и B в прямом модифицированном коде. С управляющего устройства подаются синхросигналы $C_1$ и $C_2$ . Мультиплексор

MS и инкрементор INC преобразуют модифицированный прямой код в модифицированный дополнительный код. Далее числа складываются в сумматоре SM, где проводится операция выделения признака: нормализация вправо. Результат операции записывается в RGS.

## 3. Разработка регистра

Для создания регистра будут использованы D-триггеры типа «защелка». Для реализации 12-разрядных чисел понадобится 3 набора триггеров К155ТМ7.

Функциональная схема регистра представлена на рисунке 2.

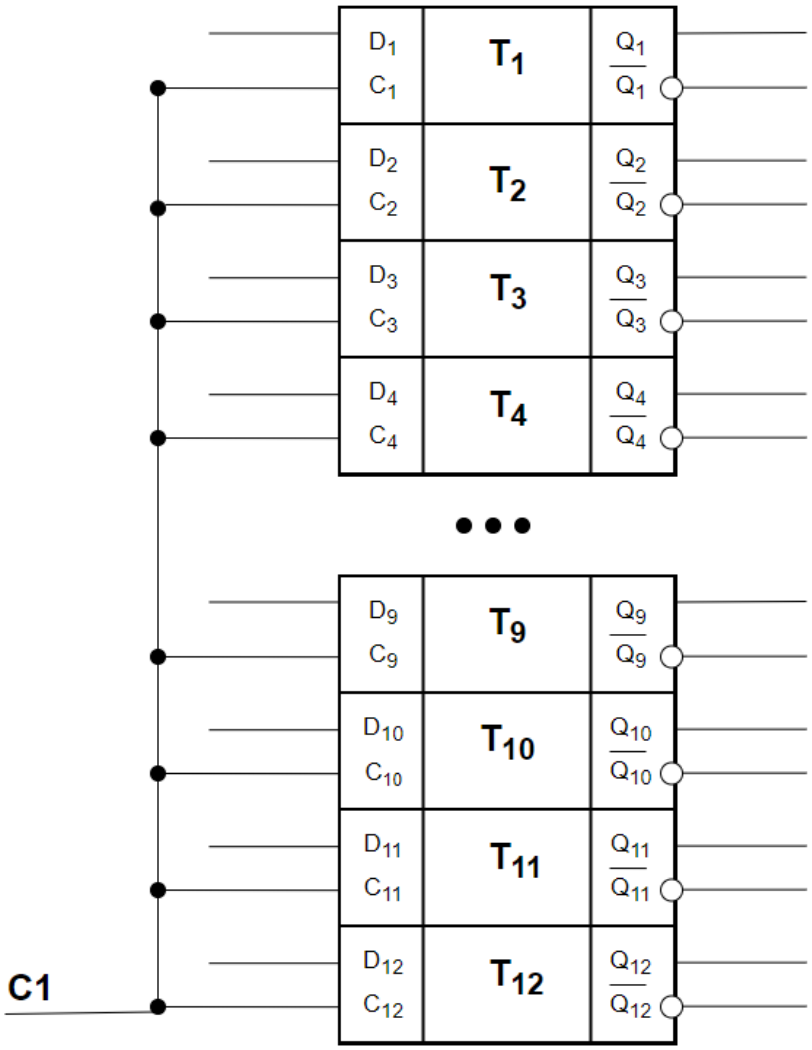


Рисунок 2. Функциональная схема регистра

Задержка регистра: $T_{reg} = 40$ нс

Реализация регистра и микросхемы К155ТМ7 на VHDL представлена ниже.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity K155TM7 is
  port(C: in std_logic; D: in std_logic_vector(3 downto 0); Q: out std_logic_vector(3 downto 0));
end K155TM7;

Architecture arch_K155TM7 of K155TM7 is
  begin process(C, D(3 downto 0))
    begin
      if (C='1') then
            Q(3 downto 0) <= D(3 downto 0) after 40 ns;
    end if;
  end process;
end arch_K155TM7;

#-------------------------------------------------------------------------#

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity K155TM7_scheme_12digits is
  port(C: in std_logic; D: in std_logic_vector(11 downto 0); Q: out std_logic_vector(11 downto 0));
end K155TM7_scheme_12digits;

Architecture arch_K155TM7_scheme_12digits of K155TM7_scheme_12digits is
  component K155TM7 is
    port(C: in std_logic; D: in std_logic_vector(3 downto 0); Q: out std_logic_vector(3 downto 0));
  end component K155TM7;

  begin
    RG1: K155TM7 port map (C, D(3 downto 0), Q(3 downto 0));
    RG2: K155TM7 port map (C, D(7 downto 4), Q(7 downto 4));
    RG3: K155TM7 port map (C, D(11 downto 8), Q(11 downto 8));

end arch_K155TM7_scheme_12digits;
```

## 4. Синтез элементов россыпи

Ниже представлена реализация элементов 1И-НЕ, 2И-НЕ, 3И-НЕ, 4И-НЕ, 5И-НЕ, 6И-НЕ, 7И-НЕ, 2И-2ИЛИ-НЕ, а также вспомогательных функций PI (прозрачности) и GEN (генерации).

Задержки всех элементов типа И-НЕ, И-ИЛИ-НЕ составляют 17.5 нс.

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

Entity NAND1 is
        Generic (T: time:=17.5 ns);
        Port(x1: in std_logic; y: out std_logic);
End NAND1;

Architecture arch_NAND1 of NAND1 is
Begin
        y<=NOT x1 after T;
end arch_NAND1;

#-------------------------------------------------------------------------#

Entity NAND2 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2: in std_logic; y: out std_logic);
End NAND2;

Architecture arch_NAND2 of NAND2 is
Begin
        y<=NOT(x1 AND x2) after T;
end arch_NAND2;

#-------------------------------------------------------------------------#

Entity NAND3 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2,x3: in std_logic; y: out std_logic);
End NAND3;

Architecture arch_NAND3 of NAND3 is
Begin
        y<=NOT(x1 AND x2 AND x3) after T;
end arch_NAND3;

#-------------------------------------------------------------------------#

Entity NAND4 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2,x3,x4: in std_logic; y: out std_logic);
End NAND4;

Architecture arch_NAND4 of NAND4 is
Begin
        y<=NOT(x1 AND x2 AND x3 AND x4) after T;
end arch_NAND4;

#-------------------------------------------------------------------------#

Entity NAND5 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2,x3,x4,x5: in std_logic; y: out std_logic);
End NAND5;
```

```
Architecture arch_NAND5 of NAND5 is
Begin
        y<=NOT(x1 AND x2 AND x3 AND x4 AND x5) after T;
end arch_NAND5;

#-------------------------------------------------------------------------#

Entity NAND6 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2,x3,x4,x5,x6: in std_logic; y: out std_logic);
End NAND6;

Architecture arch_NAND6 of NAND6 is
Begin
        y<=NOT(x1 AND x2 AND x3 AND x4 AND x5 AND x6) after T;
end arch_NAND6;

#-------------------------------------------------------------------------#

Entity NAND7 is
        Generic (T: time:=17.5 ns);
        Port(x1,x2,x3,x4,x5,x6,x7: in std_logic; y: out std_logic);
End NAND7;

Architecture arch_NAND7 of NAND7 is
Begin
        y<=NOT(x1 AND x2 AND x3 AND x4 AND x5 AND x6 AND x7) after T;
end arch_NAND7;

#-------------------------------------------------------------------------#

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

Entity PI is
        Generic (T: time:=35 ns);
        Port(x1,x2: in std_logic; y: out std_logic);
End PI;

Architecture arch_PI of PI is

Begin
        y<= (x1 OR x2) after T;
end arch_PI;

#-------------------------------------------------------------------------#

Entity GEN is
        Generic (T: time:=35 ns);
        Port(x1,x2: in std_logic; y: out std_logic);
End GEN;

Architecture arch_GEN of GEN is
```

```
Begin
        y<= (x1 AND x2) after T;
end arch_GEN;
```

# 5. Сумматор

## 5.1 II ярус сумматора

Разобъем разряды на 2 группы: 2 группы по 6 разрядов.

Используются 2 функции:

1) $\text{г}_i = a_i \cdot b_i$ 2) $\pi_j = a_i \vee b_i$

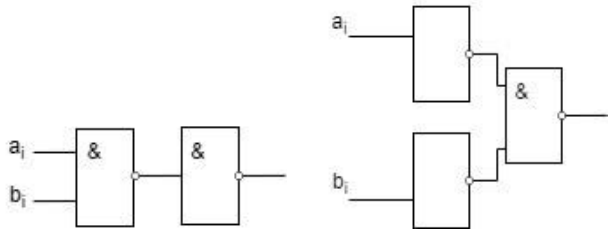


Рисунок 3. Схемы блоков генерации и прозрачности

На VHDL для удобства Г и П рассчитываются заранее и используются на 1х и 2х ярусах.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity PI_and_GEN is
  port(A, B: in std_logic_vector(11 downto 0); PI_out, GEN_out: out std_logic_vector(11 downto 0));
end PI_and_GEN;

architecture arch_PI_and_GEN of PI_and_GEN is

  component PI is
    port(x1,x2: in std_logic; y: out std_logic);
  end component PI;

  component GEN is
    port(x1,x2: in std_logic; y: out std_logic);
  end component GEN;

  begin

    PI1: PI port map(A(0), B(0), PI_out(0));
    PI2: PI port map(A(1), B(1), PI_out(1));
    PI3: PI port map(A(2), B(2), PI_out(2));
    PI4: PI port map(A(3), B(3), PI_out(3));
    PI5: PI port map(A(4), B(4), PI_out(4));
```

```
    PI6: PI port map(A(5), B(5), PI_out(5));
    PI7: PI port map(A(6), B(6), PI_out(6));
    PI8: PI port map(A(7), B(7), PI_out(7));
    PI9: PI port map(A(8), B(8), PI_out(8));
    PI10: PI port map(A(9), B(9), PI_out(9));
    PI11: PI port map(A(10), B(10), PI_out(10));
    PI12: PI port map(A(11), B(11), PI_out(11));

    GEN1: GEN port map(A(0), B(0), GEN_out(0));
    GEN2: GEN port map(A(1), B(1), GEN_out(1));
    GEN3: GEN port map(A(2), B(2), GEN_out(2));
    GEN4: GEN port map(A(3), B(3), GEN_out(3));
    GEN5: GEN port map(A(4), B(4), GEN_out(4));
    GEN6: GEN port map(A(5), B(5), GEN_out(5));
    GEN7: GEN port map(A(6), B(6), GEN_out(6));
    GEN8: GEN port map(A(7), B(7), GEN_out(7));
    GEN9: GEN port map(A(8), B(8), GEN_out(8));
    GEN10: GEN port map(A(9), B(9), GEN_out(9));
    GEN11: GEN port map(A(10), B(10), GEN_out(10));
    GEN12: GEN port map(A(11), B(11), GEN_out(11));

end arch_PI_and_GEN;
```

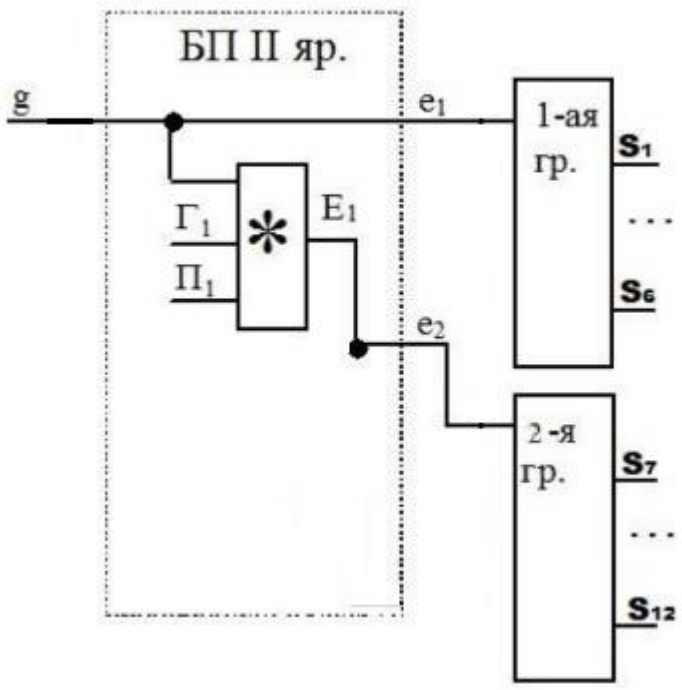


Рисунок 4. II ярус двухъярусного сумматора

Выходной перенос из j-ой группы равен:

$$E_j = e_{j+1} = \Gamma_j \vee e_j \Pi_j$$

Система уравнений:

$$\begin{cases} e_1 = g \\ E_1 = e_2 = \Gamma_1 \vee e_1\Pi_1 = \text{г}_6 \vee \text{г}_5\pi_6 \vee \text{г}_4\pi_6\pi_5 \vee \ldots \vee \text{г}_1\pi_6\pi_5\pi_4\pi_3\pi_2 \vee g\pi_6 \ldots \pi_1 \end{cases}$$

*где* $\text{г}_i = a_i b_i, \pi_i = a_i \vee b_i$

С помощью формул де Моргана преобразуем систему уравнений:

$$\begin{cases} e_1 = g \\ E_1 = e_2 = \overline{\overline{\Gamma_1} * \overline{e_1\Pi_1}} = \overline{\overline{\text{г}_6} \cdot \overline{\text{г}_5\pi_6} \cdot \overline{\text{г}_4\pi_6\pi_5} \cdot \ldots \cdot \overline{\text{г}_1\pi_6\pi_5\pi_4\pi_3\pi_2} \cdot \overline{g\pi_6 \ldots \pi_1}} \end{cases}$$

Снизу представлена схема блока переноса 1 группы.

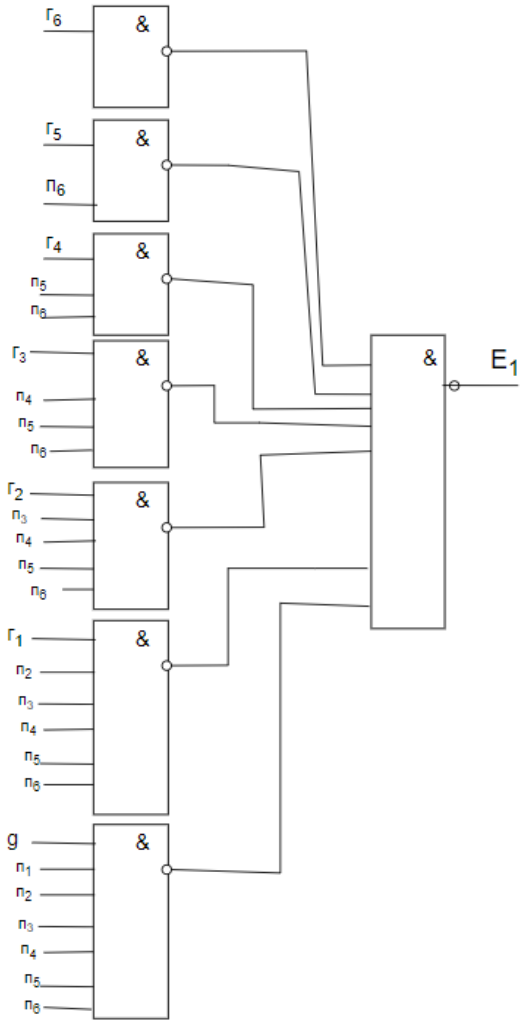


Рисунок 5. Схема блока переноса

Задержка построенного блока переноса II-го яруса равна $T_{II} = 4 \cdot \tau$

Код для моделирования II-го яруса на VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity Block_carry_II is
  port(g: in std_logic; PI, GEN: in std_logic_vector(5 downto 0); E: out std_logic);
end Block_carry_II;

architecture arch_Block_carry_II of Block_carry_II is
  component NAND1 is
    port(x1: in std_logic; y: out std_logic);
  end component NAND1;

  component NAND2 is
    port(x1, x2: in std_logic; y: out std_logic);
  end component NAND2;

  component NAND3 is
    port(x1, x2, x3: in std_logic; y: out std_logic);
  end component NAND3;

  component NAND4 is
    port(x1, x2, x3, x4: in std_logic; y: out std_logic);
  end component NAND4;

  component NAND5 is
    port(x1, x2, x3, x4, x5: in std_logic; y: out std_logic);
  end component NAND5;

  component NAND6 is
    port(x1, x2, x3, x4, x5, x6: in std_logic; y: out std_logic);
  end component NAND6;

  component NAND7 is
    port(x1, x2, x3, x4, x5, x6, x7: in std_logic; y: out std_logic);
  end component NAND7;

  signal tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7: std_logic;

  begin
    COMP1: NAND1 port map (GEN(5), tmp1);
    COMP2: NAND2 port map (GEN(4), PI(5), tmp2);
    COMP3: NAND3 port map (GEN(3), PI(4), PI(5), tmp3);
    COMP4: NAND4 port map (GEN(2), PI(3), PI(4), PI(5), tmp4);
    COMP5: NAND5 port map (GEN(1), PI(2), PI(3), PI(4), PI(5), tmp5);
    COMP6: NAND6 port map (GEN(0), PI(1), PI(2), PI(3), PI(4), PI(5), tmp6);
    COMP7: NAND7 port map (g, PI(0), PI(1), PI(2), PI(3), PI(4), PI(5), tmp7);
    COMP8: NAND7 port map (tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, E);

end arch_Block_carry_II;
```

## 5.2 I ярус сумматора

Составим логические уравнения для реализации параллельного переноса на I ярусе.

$$\begin{cases} r_1 = e \\ r_2 = \text{г}_1 \vee e\pi_1 \\ r_3 = \text{г}_2 \vee \text{г}_1\pi_2 \vee e\pi_1\pi_2 \\ r_4 = \text{г}_3 \vee \text{г}_2\pi_3 \vee \text{г}_1\pi_2\pi_3 \vee e\pi_1\pi_2\pi_3 \\ r_5 = \text{г}_4 \vee \text{г}_3\pi_4 \vee \text{г}_2\pi_3\pi_4 \vee \text{г}_1\pi_2\pi_3\pi_4 \vee e\pi_1\pi_2\pi_3\pi_4 \\ r_6 = \text{г}_5 \vee \text{г}_4\pi_5 \vee \text{г}_3\pi_4\pi_5 \vee \text{г}_2\pi_3\pi_4\pi_5 \vee \text{г}_1\pi_2\pi_3\pi_4\pi_5 \vee e\pi_1\pi_2\pi_3\pi_4\pi_5 \end{cases}$$

Преобразуем эти логические уравнения по закону де Моргана. В

$$\begin{cases} r_1 = e \\ r_2 = \overline{\overline{\text{г}_1} \cdot \overline{e\pi_1}} \\ r_3 = \overline{\overline{\text{г}_2} \cdot \overline{\text{г}_1\pi_2} \cdot \overline{e\pi_1\pi_2}} \\ r_4 = \overline{\overline{\text{г}_3} \cdot \overline{\text{г}_2\pi_3} \cdot \overline{\text{г}_1\pi_2\pi_3} \cdot \overline{e\pi_1\pi_2\pi_3}} \\ r_5 = \overline{\overline{\text{г}_4} \cdot \overline{\text{г}_3\pi_4} \cdot \overline{\text{г}_2\pi_3\pi_4} \cdot \overline{\text{г}_1\pi_2\pi_3\pi_4} \cdot \overline{e\pi_1\pi_2\pi_3\pi_4}} \\ r_6 = \overline{\overline{\text{г}_5} \cdot \overline{\text{г}_4\pi_5} \cdot \overline{\text{г}_3\pi_4\pi_5} \cdot \overline{\text{г}_2\pi_3\pi_4\pi_5} \cdot \overline{\text{г}_1\pi_2\pi_3\pi_4\pi_5} \cdot \overline{e\pi_1\pi_2\pi_3\pi_4\pi_5}} \end{cases}$$

Изобразим на рисунке 6 схему блока переноса I яруса

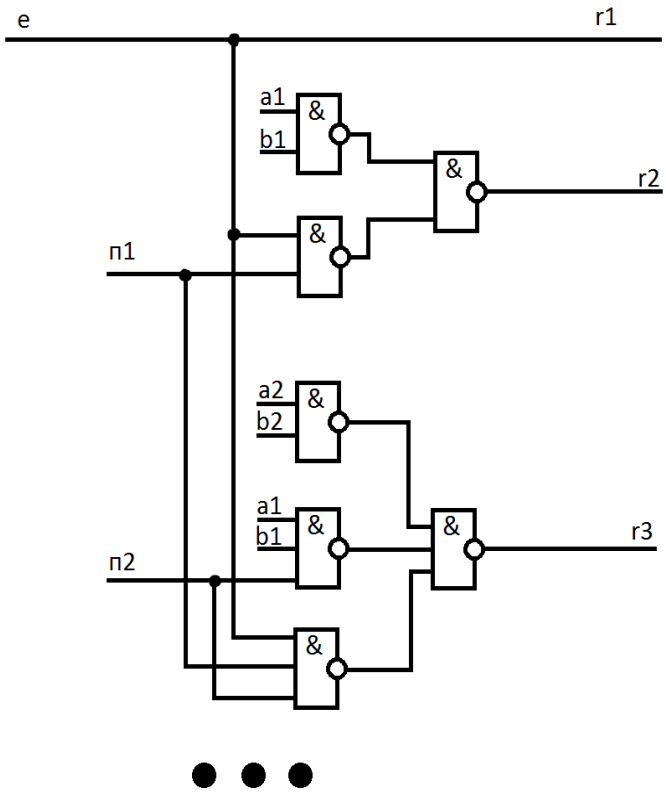


Рисунок 6. Схема блока переноса I яруса

Задержка построенного блока переноса I-го яруса равна $T_I = 2 \cdot \tau$

Код моделирования I-го яруса переноса:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity Block_carry_I is
  port(e: in std_logic; PI, GEN: in std_logic_vector(5 downto 0); r: out std_logic_vector(5 downto 1));
end Block_carry_I;

architecture arch_Block_carry_I of Block_carry_I is
  component NAND1 is
    port(x1: in std_logic; y: out std_logic);
  end component NAND1;

  component NAND2 is
    port(x1, x2: in std_logic; y: out std_logic);
  end component NAND2;

  component NAND3 is
    port(x1, x2, x3: in std_logic; y: out std_logic);
  end component NAND3;

  component NAND4 is
    port(x1, x2, x3, x4: in std_logic; y: out std_logic);
  end component NAND4;

  component NAND5 is
    port(x1, x2, x3, x4, x5: in std_logic; y: out std_logic);
  end component NAND5;

  component NAND6 is
    port(x1, x2, x3, x4, x5, x6: in std_logic; y: out std_logic);
  end component NAND6;

  signal r2_1, r2_2: std_logic;
  signal r3_1, r3_2, r3_3: std_logic;
  signal r4_1, r4_2, r4_3, r4_4: std_logic;
  signal r5_1, r5_2, r5_3, r5_4, r5_5: std_logic;
  signal r6_1, r6_2, r6_3, r6_4, r6_5, r6_6: std_logic;

  begin

    RR2_1: NAND1 port map (GEN(0), r2_1);
    RR2_2: NAND2 port map (e, PI(0), r2_2);
    RR2_3: NAND2 port map (r2_1, r2_2, r(1));--

    RR3_1: NAND1 port map (GEN(1), r3_1);
    RR3_2: NAND2 port map (GEN(0), PI(1), r3_2);
    RR3_3: NAND3 port map (e, PI(0), PI(1), r3_3);
    RR3_4: NAND3 port map (r3_1, r3_2, r3_3, r(2));--

    RR4_1: NAND1 port map (GEN(2), r4_1);
    RR4_2: NAND2 port map (GEN(1), PI(2), r4_2);
    RR4_3: NAND3 port map (GEN(0), PI(1), PI(2), r4_3);
    RR4_4: NAND4 port map (e, PI(0), PI(1), PI(2), r4_4);
    RR4_5: NAND4 port map (r4_1, r4_2, r4_3, r4_4, r(3));--

    RR5_1: NAND1 port map (GEN(3), r5_1);
    RR5_2: NAND2 port map (GEN(2), PI(3), r5_2);
    RR5_3: NAND3 port map (GEN(1), PI(2), PI(3), r5_3);
    RR5_4: NAND4 port map (GEN(0), PI(1), PI(2), PI(3), r5_4);
    RR5_5: NAND5 port map (e, PI(0), PI(1), PI(2), PI(3), r5_5);
```

```
    RR5_6: NAND5 port map (r5_1, r5_2, r5_3, r5_4, r5_5, r(4));--

    RR6_1: NAND1 port map (GEN(4), r6_1);
    RR6_2: NAND2 port map (GEN(3), PI(4), r6_2);
    RR6_3: NAND3 port map (GEN(2), PI(3), PI(4), r6_3);
    RR6_4: NAND4 port map (GEN(1), PI(2), PI(3), PI(4), r6_4);
    RR6_5: NAND5 port map (GEN(0), PI(1), PI(2), PI(3), PI(4), r6_5);
    RR6_6: NAND6 port map (e, PI(0), PI(1), PI(2), PI(3), PI(4), r6_6);
    RR6_7: NAND6 port map (r6_1, r6_2, r6_3, r6_4, r6_5, r6_6, r(5));--

end arch_Block_carry_I;
```

Построим блок суммы:

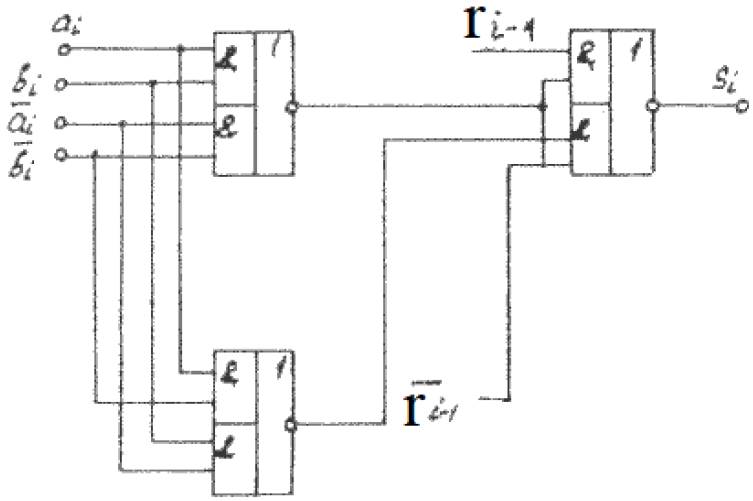

Рисунок 7. Схема блока суммы

Функция:

$$S_i = \overline{(a_i b_i \vee \bar{a}_i \bar{b}_i)}\, r_{i-1} \vee \overline{(a_i \bar{b}_i \vee \bar{a}_i b_i)}\, \bar{r}_{i-1}$$

Задержка сумматора равна $T_{\text{одн.}sm} = 2 \cdot \tau$

Код моделирования блока суммы:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Basic_adder is
  port(r, a, b: in std_logic; s: out std_logic);
end Basic_adder;

architecture arch_Basic_adder of Basic_adder is

  component NAND1 is
```

```
  port(x1: in std_logic; y: out std_logic);
 end component NAND1;

 component AND2_OR2_NOT is
  port(x1, x2, x3, x4: in std_logic; y: out std_logic);
 end component AND2_OR2_NOT;

 signal tmp1, tmp2, not_r, not_a, not_b: std_logic;

 begin
  -- 'NOT a' and 'NOT b'  are without delays (we counted it yearlier)
  not_a <= not a;
  not_b <= not b;
  NOTR: NAND1 port map (r, not_r);
  COMP1: AND2_OR2_NOT port map (a, b, not_a, not_b, tmp1);
  COMP2: AND2_OR2_NOT port map (a, not_b, b, not_a, tmp2);
  COMP3: AND2_OR2_NOT port map (r, tmp1, tmp2, not_r, s);

end arch_Basic_adder;
```

## 6. Синтез многоразрядного сумматора и получение схемы для тестирования

Применим полученные блоки переносов и одноразрядные сумматоры для создания 12-ти разрядного сумматора.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Adder_12digits is
 port(A,B: in std_logic_vector(11 downto 0);
      S_out: out std_logic_vector(11 downto 0));
end Adder_12digits;

architecture arch_Adder_12digits of Adder_12digits is

 component PI_and_GEN is
  port(A, B: in std_logic_vector(11 downto 0); PI_out, GEN_out: out std_logic_vector(11 downto 0));
 end component PI_and_GEN;

 component Block_Carry_II is
  port(g: in std_logic; PI, GEN: in std_logic_vector(5 downto 0); E: out std_logic);
 end component Block_carry_II;

 component Block_Carry_I is
  port(e: in std_logic; PI, GEN: in std_logic_vector(5 downto 0); r: out std_logic_vector(5 downto 1));
 end component Block_carry_I;

 component Basic_adder is
  port(R: in std_logic; A, B: in std_logic; S: out std_logic);
```

```
  end component Basic_adder;

  signal PI: std_logic_vector(11 downto 0);
  signal GEN: std_logic_vector(11 downto 0);
  signal E1: std_logic;
  signal R: std_logic_vector(11 downto 0);

  begin

    COMP_PI_AND_GEN: PI_and_GEN port map (A(11 downto 0), B(11 downto 0), PI(11 downto 0),
GEN(11 downto 0));

    COMP_CARRY_II: Block_Carry_II port map ('0', PI(5 downto 0), GEN(5 downto 0), E1);

    R(0) <= '0';
    COMP1_CARRY_I: Block_Carry_I port map ('0', PI(5 downto 0), GEN(5 downto 0), R(5 downto 1));
    R(6) <= E1;
    COMP2_CARRY_I: Block_Carry_I port map (E1, PI(11 downto 6), GEN(11 downto 6), R(11 downto
7));

    COMP0_ADDER: Basic_adder port map ('0', A(0), B(0), S_out(0));
    COMP1_ADDER: Basic_adder port map (R(1), A(1), B(1), S_out(1));
    COMP2_ADDER: Basic_adder port map (R(2), A(2), B(2), S_out(2));
    COMP3_ADDER: Basic_adder port map (R(3), A(3), B(3), S_out(3));
    COMP4_ADDER: Basic_adder port map (R(4), A(4), B(4), S_out(4));
    COMP5_ADDER: Basic_adder port map (R(5), A(5), B(5), S_out(5));
    COMP6_ADDER: Basic_adder port map (R(6), A(6), B(6), S_out(6));
    COMP7_ADDER: Basic_adder port map (R(7), A(7), B(7), S_out(7));
    COMP8_ADDER: Basic_adder port map (R(8), A(8), B(8), S_out(8));
    COMP9_ADDER: Basic_adder port map (R(9), A(9), B(9), S_out(9));
    COMP10_ADDER: Basic_adder port map (R(10), A(10), B(10), S_out(10));
    COMP11_ADDER: Basic_adder port map (R(11), A(11), B(11), S_out(11));

end arch_Adder_12digits;
```

Добавим к этой сущности реализацию регистров A и B, чтобы получить финальную версию устройства Adder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Adder is
   port(clk: in std_logic;
        cin: in std_logic;
            a,b: in std_logic_vector(11 downto 0);
            s: out std_logic_vector(11 downto 0);
            cout: out std_logic);
end Adder;

architecture arch_Adder of Adder is

  component K155TM7_scheme_12digits is
    port(C: in std_logic; D: in std_logic_vector(11 downto 0); Q: out std_logic_vector(11 downto 0));
```

```
  end component K155TM7_scheme_12digits;

  component Adder_12digits is
    port(A,B: in std_logic_vector(11 downto 0);
         S_out: out std_logic_vector(11 downto 0));
  end component Adder_12digits;

  signal A_reg, B_reg: std_logic_vector(11 downto 0);

  begin
    REGISTER_A: K155TM7_scheme_12digits port map (clk, a(11 downto 0), A_reg(11 downto 0));
    REGISTER_B: K155TM7_scheme_12digits port map (clk, b(11 downto 0), B_reg(11 downto 0));
    ADDER_AB: Adder_12digits port map (A_reg(11 downto 0), B_reg(11 downto 0), S(11 downto 0));

end arch_Adder;
```

# 7. Тестирование суммирующего устройства

Проведем несколько заранее заготовленных тестов для оценки правильности работы полученного сумматора.

Время подачи сигнала CLK в 40 нс обусловлено задержкой загрузки информации в регистры. 148 нс – максимальное время работы сумматора по условию.

Тест 1 – тест сумматора без переносов.

Тест 2 – тест сумматора без переносов при одном из нулевых чисел.

Тест 3 – проверка правильности переноса через все разряды.

Тест 4 – проверка правильности переноса до середины числа.

Тест 5, 6 – проверка работы сумматора на случайно заданных числах.

Код программы тестирования:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

use std.textio.all;
use IEEE.numeric_std.all;

library osvvm;
use osvvm.RandomPkg.all;

entity Test_Adder is
end Test_Adder;

architecture arch_Test_Adder of Test_Adder is
```

```
  component Adder is
    port(clk: in std_logic;
         cin: in std_logic;
              a,b: in std_logic_vector(11 downto 0);
              s: out std_logic_vector(11 downto 0);
              cout: out std_logic);
  end component Adder;

  signal clk, cin: std_logic;
  signal a, b, s: std_logic_vector(11 downto 0);
  signal cout: std_logic;

  begin
    DUT: Adder port map (clk, cin, a, b, s, cout);
    process
          begin
-- Test 1
          clk <= '1';
       A <= "011011001001";
          B <= "100100110110";
       wait for 40 ns;
          clk <= '0';
          wait for 148 ns;

-- Test 2
          clk <= '1';
       A <= "000000000000";
          B <= "111111111111";
       wait for 40 ns;
          clk <= '0';
          wait for 148 ns;

-- Test 3
          clk <= '1';
       A <= "011111111111";
          B <= "000000000001";
       wait for 40 ns;
          clk <= '0';
          wait for 148 ns;

-- Test 4
          clk <= '1';
       A <= "011111011111";
          B <= "000000000001";
       wait for 40 ns;
          clk <= '0';
          wait for 148 ns;

-- Test 5 - random digits
          clk <= '1';
       A <= "011010101111";
          B <= "000111100110";
       --expected: 100010010101
       wait for 40 ns;
          clk <= '0';
          wait for 148 ns;
```

```
-- Test 6 - random digits
            clk <= '1';
        A <= "000110101010";
            B <= "001011000111";
        --expected: 10001110001
        wait for 40 ns;
            clk <= '0';
            wait for 148 ns;

    end process;
end arch_Test_Adder;
```

Диаграмма тестов №1-3 представлена на рисунке 8.

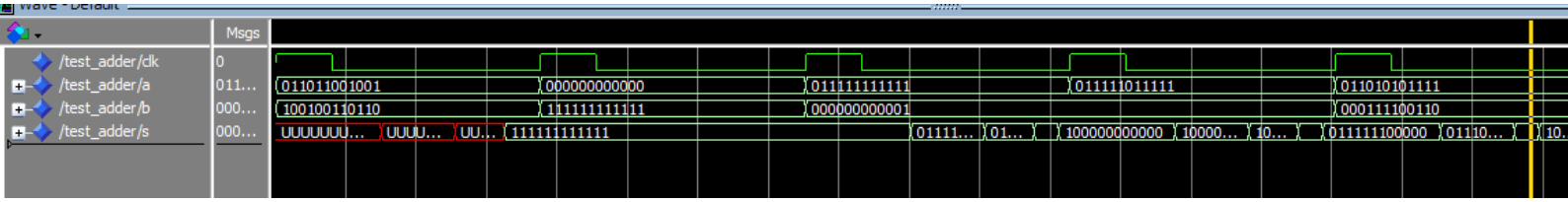


Рисунок 8. Тесты №1-3

Диаграмма тестов №4-6 представлена на рисунке 9.

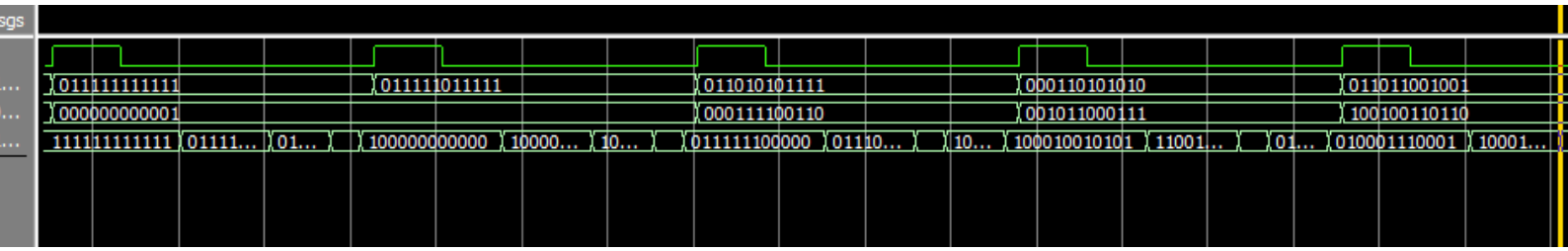


Рисунок 9. Тесты №4-6

Тестирование устройства с помощью случайных чисел и автоматической проверки.

Код программы:

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

use std.textio.all;
use IEEE.numeric_std.all;

library osvvm;
use osvvm.RandomPkg.all;

entity Test_Adder is
```

```
end Test_Adder;

architecture arch_Test_Adder of Test_Adder is

  component Adder is
    port(clk: in std_logic;
         cin: in std_logic;
            a,b: in std_logic_vector(11 downto 0);
            s: out std_logic_vector(11 downto 0);
            cout: out std_logic);
  end component Adder;

  signal testa, testb, testResult, result: std_logic_vector(11 downto 0);
  signal c: std_logic;
  signal cin_empty, cout_empty: std_logic;

  begin
    dut: adder port map(c, cin_empty, testA, testB, result, cout_empty);
    gen_rand: process
      variable rA, rB: RandomPType;
      variable randA, randB, b: integer;
      variable L: line;

      begin
        randA := rA.RandInt(0, 4095);
        randB := rB.RandInt(0, 4095);
        testA <= std_logic_vector(to_unsigned(randA, 12));
        testB <= std_logic_vector(to_unsigned(randB, 12));

        testResult <= std_logic_vector(to_unsigned(randA+randB, testResult'length));
        c <= '1';
        wait for 40 ns;
        c <= '0';
        wait for 148 ns;
        if (testResult = result) then
          write(L, string'("Success"));
        else
          write(L, string'("Error"));
        end if;
    end process;
end arch_Test_Adder;
```

Результат моделирования на рисунке :

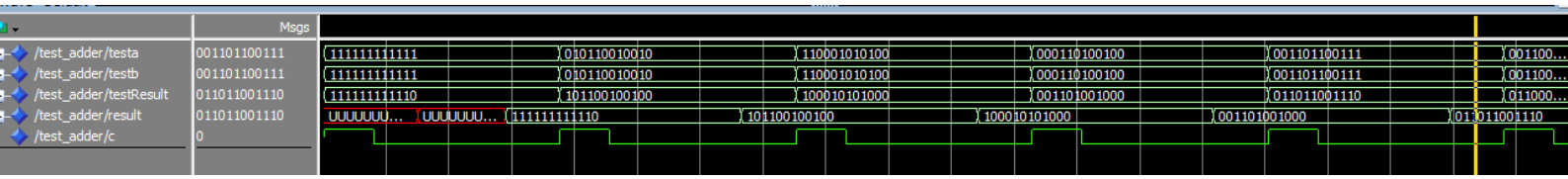


Рисунок 10. Тестирование с автоматической проверкой

# 8. Нахождение критического пути сумматора

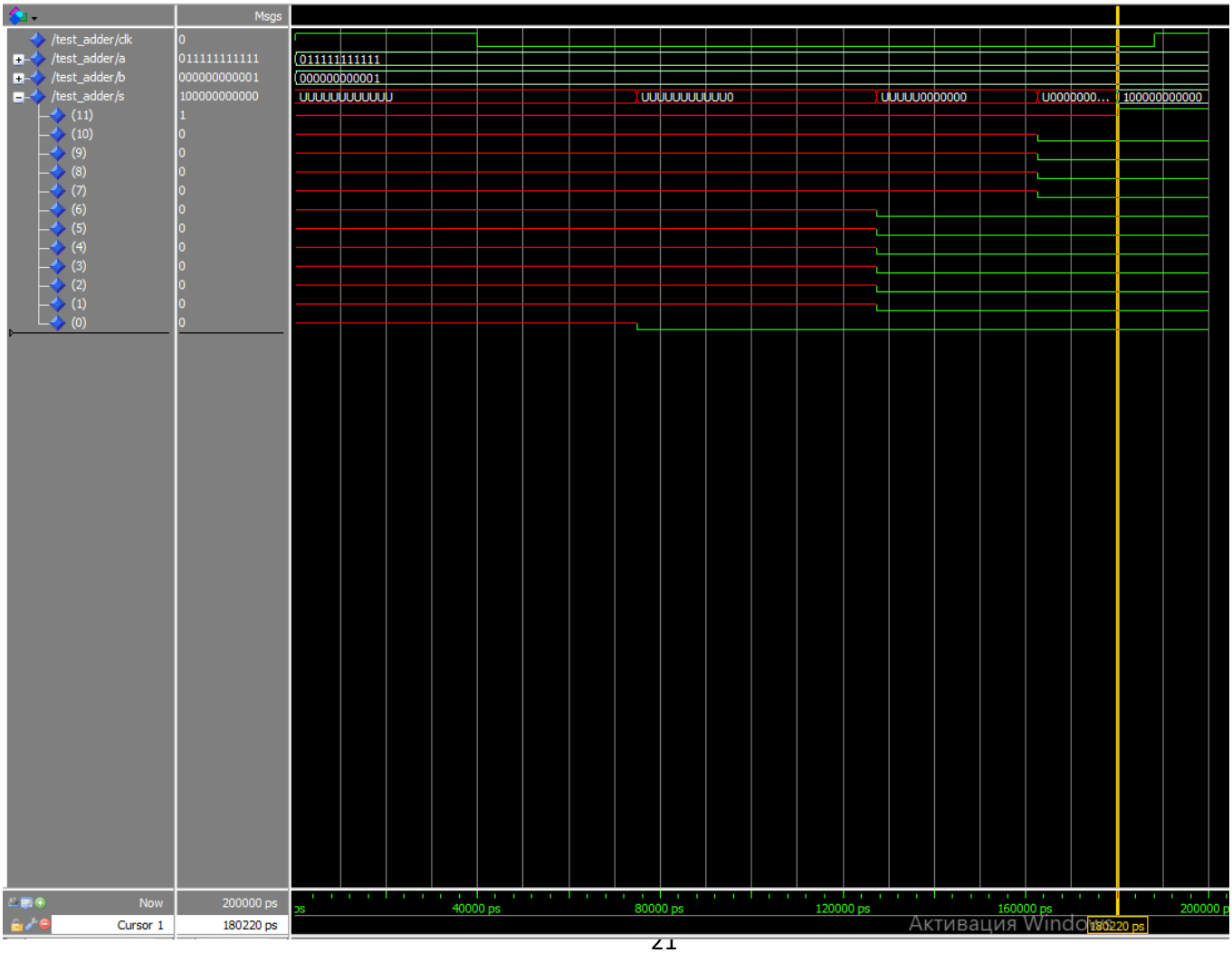

Рисунок 11. Нахождение критического пути

Для поиска используем тест переноса через все разряды числа. Задержка критического пути составила 140 нс (40 нс – задержка регистров) < 148 нс. Условие задания выполнено.

## 9. Вывод

В ходе выполнения работы было смоделировано многоразрядное суммирующие устройство, которое прошло необходимые тесты и прошло по условию максимальной задержки.