Национальный исследовательский ядерный университет «МИФИ»

Факультет Кибернетики и информационной безопасности

Кафедра «Компьютерные системы и технологии»

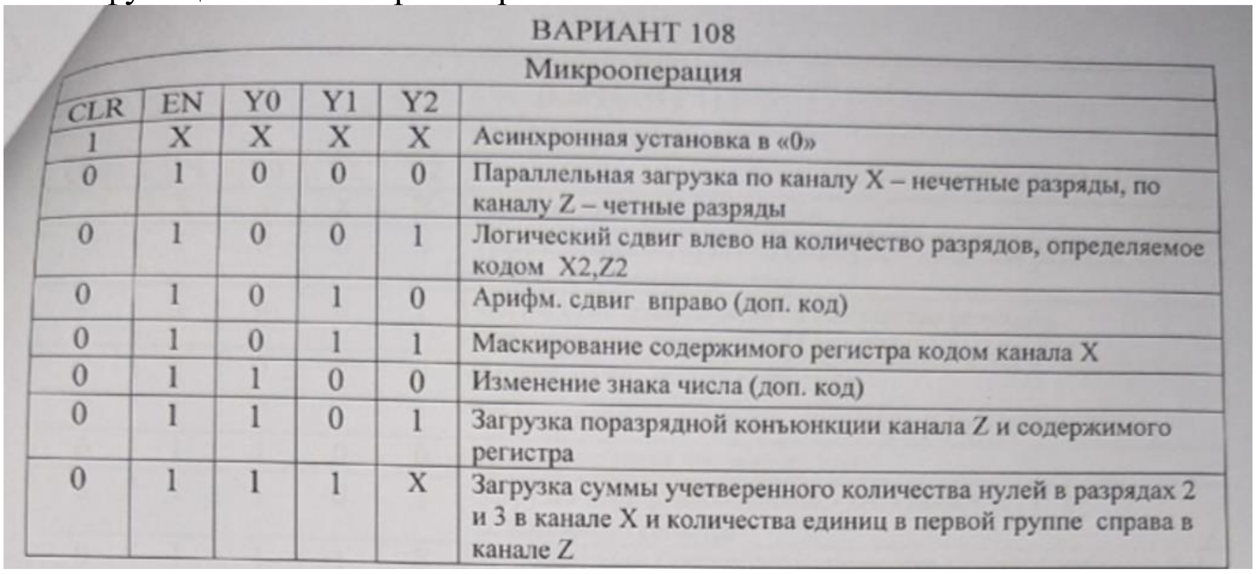
ОТЧЕТ
по выполнению лабораторного практикума
по курсу «Функционально-логическое моделирование и автоматизация проектирования»

Студент гр. Б19-503	________________	/Новиков Т.И. /
Руководитель	________________	/ Ёхин М.Н. /

Москва 2022

1. Задание

На рисунке 1.1 представлено задание для разработки многофункционального регистра.



ВАРИАНТ 108

Микрооперация					
CLR	EN	Y0	Y1	Y2	
1	X	X	X	X	Асинхронная установка в «0»
0	1	0	0	0	Параллельная загрузка по каналу X – нечетные разряды, по каналу Z – четные разряды
0	1	0	0	1	Логический сдвиг влево на количество разрядов, определяемое кодом X2,Z2
0	1	0	1	0	Арифм. сдвиг вправо (доп. код)
0	1	0	1	1	Маскирование содержимого регистра кодом канала X
0	1	1	0	0	Изменение знака числа (доп. код)
0	1	1	0	1	Загрузка поразрядной конъюнкции канала Z и содержимого регистра
0	1	1	1	X	Загрузка суммы учетверенного количества нулей в разрядах 2 и 3 в канале X и количества единиц в первой группе справа в канале Z

Рисунок 1.1 – Задание

2. Графическое представление МФР

На рисунке 1.2 представлено графическое представление МФР на уровне функциональных модулей.

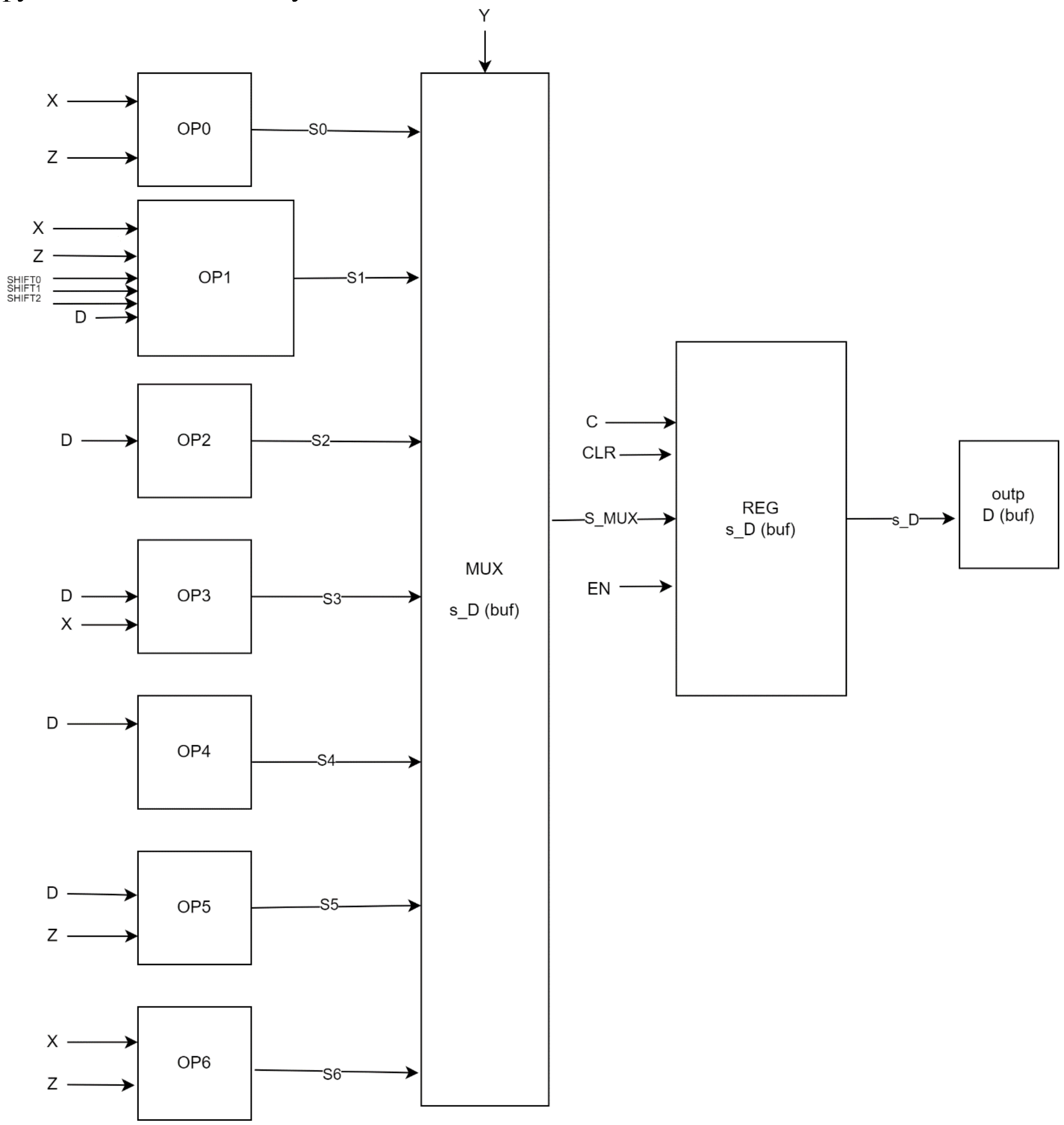


Рисунок 1.2 – Графическое представление МФР

3. Код программы

Ниже представлен код программы, написанный на VHDL.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity  global_reg is
    port (
        -- Inputs
        CLR:    in std_logic;
        EN:     in std_logic;
        C:      in std_logic;
        Y:      in std_logic_vector(2 downto 0);
        X:      in std_logic_vector(3 downto 0);
        Z:      in std_logic_vector(3 downto 0);
        -- For logic compliment
        SHIFT0: in std_logic;
        SHIFT1: in std_logic;
        SHIFT2: in std_logic;

        -- Outputs
        D:      buffer std_logic_vector(3 downto 0)

    );
end global_reg;

architecture global_reg_arch of global_reg is
    signal s0: std_logic_vector(3 downto 0);
    signal s1: std_logic_vector(3 downto 0);
    signal s2: std_logic_vector(3 downto 0);
    signal s3: std_logic_vector(3 downto 0);
    signal s4: std_logic_vector(3 downto 0);
    signal s5: std_logic_vector(3 downto 0);
    signal s6: std_logic_vector(3 downto 0);
    signal s_mux: std_logic_vector(3 downto 0);
    signal s_D: std_logic_vector(3 downto 0);

    component mux
        port(
            s0: in std_logic_vector(3 downto 0);
            s1: in std_logic_vector(3 downto 0);
            s2: in std_logic_vector(3 downto 0);
            s3: in std_logic_vector(3 downto 0);
            s4: in std_logic_vector(3 downto 0);
            s5: in std_logic_vector(3 downto 0);
            s6: in std_logic_vector(3 downto 0);
            Y:  in std_logic_vector(2 downto 0);
            s_mux: out std_logic_vector(3 downto 0)
        );
    end component;

    component reg
        port(
            s_mux:      in std_logic_vector(3 downto 0);
            CLR:        in std_logic;
            C:          in std_logic;
            EN:         in std_logic;
            s_D:        buffer std_logic_vector(3 downto 0)
        );
    end component;

    component outp
        port(
            s_D:       in std_logic_vector(3 downto 0);
            D:         buffer std_logic_vector(3 downto 0)
        );
    end component;

-- Parallel load from X and Z
    component OP0
        port(
            X:      in std_logic_vector(3 downto 0);
            Z:      in std_logic_vector(3 downto 0);
            s0:     out std_logic_vector(3 downto 0)
        );
    end component;

-- Logic shift to left (compliments with 3 inputs)
    component OP1
        port(
            X:      in std_logic_vector(3 downto 0);
            Z:      in std_logic_vector(3 downto 0);
            SHIFT0: in std_logic;
            SHIFT1: in std_logic;
            SHIFT2: in std_logic;
            D:      in std_logic_vector(3 downto 0);
            s1:     out std_logic_vector(3 downto 0)
        );
    end component;

-- Arithmetic shift to right (compliments with sign)
    component OP2
        port(
            D:      in std_logic_vector(3 downto 0);
            s2:     out std_logic_vector(3 downto 0)
        );
    end component;

-- Masking D with chanel X
    component OP3
        port(
            X:      in std_logic_vector(3 downto 0);
            D:      in std_logic_vector(3 downto 0);
            s3:     out std_logic_vector(3 downto 0)
        );
    end component;

-- Change sign of number
    component OP4
        port(
            D:      in std_logic_vector(3 downto 0);
            s4:     out std_logic_vector(3 downto 0)
        );
    end component;

-- Logic and between D and Z
    component OP5
        port(
            D:      in std_logic_vector(3 downto 0);
            Z:      in std_logic_vector(3 downto 0);
            s5:     out std_logic_vector(3 downto 0)
        );
    end component;
```

```vhdl
-- Operation with zeros and ones
    component OP6
        port(
            X:      in std_logic_vector(3 downto 0);
            Z:      in std_logic_vector(3 downto 0);
            s6:     out std_logic_vector(3 downto 0)
        );
    end component;
begin
    F0: OP0 port map(
        X => X,
        Z => Z,
        s0 => s0
    );
    F1: OP1 port map(
        X => X,
        Z => Z,
        SHIFT0 => SHIFT0,
        SHIFT1 => SHIFT1,
        SHIFT2 => SHIFT2,
        D => D,
        s1 => s1
    );
    F2: OP2 port map(
        D => D,
        s2 => s2
    );
    F3: OP3 port map(
        X => X,
        D => D,
        s3 => s3
    );
    F4: OP4 port map(
        D => D,
        s4 => s4
    );

    F5: OP5 port map(
        D => D,
        Z => Z,
        s5 => s5
    );
    F6: OP6 port map(
        X => X,
        Z => Z,
        s6 => s6
    );
    RG: reg port map(
        s_mux => s_mux,
        CLR => CLR,
        C => C,
        EN => EN,
        s_D => s_D
    );

    M: mux port map(
        s0 => s0,
        s1 => s1,
        s2 => s2,
        s3 => s3,
        s4 => s4,
        s5 => s5,
        s6 => s6,
        Y  => Y,
        s_mux => s_mux
    );

    OU: outp port map(
        s_D => s_D,
        D => D
    );

end global_reg_arch;


-- Parallel load from X and Z
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP0 is
    port(
        X:      in std_logic_vector(3 downto 0);
        Z:      in std_logic_vector(3 downto 0);
        s0:     out std_logic_vector(3 downto 0)
    );
end OP0;

architecture OP0_arch of OP0 is
begin
    process (X, Z) is
    begin
        s0(3) <= X(3);
        s0(2) <= Z(2);
        s0(1) <= X(1);
        s0(0) <= Z(0);
    end process;
end OP0_arch;

-- Logic shift to left (compliments with 3 inputs)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP1 is
    port(
        X:      in std_logic_vector(3 downto 0);
        Z:      in std_logic_vector(3 downto 0);
        SHIFT0: in std_logic;
        SHIFT1: in std_logic;
        SHIFT2: in std_logic;
        D:      in std_logic_vector(3 downto 0);
        s1:     out std_logic_vector(3 downto 0)
    );
end OP1;

architecture OP1_arch of OP1 is
begin
    process (X, Z, SHIFT0, SHIFT1, SHIFT2, D) is
    begin
        if (X(2) = '0' and Z(2) = '0') then
            s1 <= D;
        elsif (X(2) = '0' and Z(2) = '1') then
            s1(3) <= D(2);
            s1(2) <= D(1);
            s1(1) <= D(0);
            s1(0) <= SHIFT0;
        elsif (X(2) = '1' and Z(2) = '0') then
            s1(3) <= D(1);
            s1(2) <= D(0);
            s1(1) <= SHIFT1;
            s1(0) <= SHIFT0;
        elsif (X(2) = '1' and Z(2) = '1') then
            s1(3) <= D(0);
            s1(2) <= SHIFT2;
            s1(1) <= SHIFT1;
            s1(0) <= SHIFT0;
        end if;
    end process;
end OP1_arch;

-- Arithmetic shift to right (compliments with sign)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP2 is
    port(
        D:      in std_logic_vector(3 downto 0);
        s2:     out std_logic_vector(3 downto 0)
    );
end OP2;

architecture OP2_arch of OP2 is
begin
    process (D) is
    begin
        s2(0) <= D(1);
        s2(1) <= D(2);
        s2(2) <= D(3);
        s2(3) <= D(3);
    end process;
end OP2_arch;

-- Masking D with chanel X
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP3 is
    port(
        X:      in std_logic_vector(3 downto 0);
        D:      in std_logic_vector(3 downto 0);
        s3:     out std_logic_vector(3 downto 0)
    );
end OP3;

architecture OP3_arch of OP3 is
begin
    process (D, X) is
    begin
        for i in 0 to 3 loop
            s3(i) <= D(i) and X(i);
        end loop;
    end process;
end OP3_arch;

-- Change sign of number
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP4 is
    port(
        D:      in std_logic_vector(3 downto 0);
        s4:     out std_logic_vector(3 downto 0)
    );
end OP4;

architecture OP4_arch of OP4 is
begin
    process (D) is
        variable buf_vector : std_logic_vector(3 downto 0);
        variable buf : std_logic;
        variable buf2 : std_logic;
    begin
        buf_vector := D;
        buf_vector(0) := not buf_vector(0);
        buf_vector(1) := not buf_vector(1);
        buf_vector(2) := not buf_vector(2);
        buf_vector(3) := not buf_vector(3);
        buf := '1';
        buf2 := buf_vector(0);
        buf_vector(0) := (buf_vector(0) and (not buf)) or ((not
buf_vector(0)) and buf);

        buf := buf2 and buf;
        buf2 := buf_vector(1);
        buf_vector(1) := (buf_vector(1) and (not buf)) or ((not
buf_vector(1)) and buf);

        buf := buf2 and buf;
        buf2 := buf_vector(2);
        buf_vector(2) := (buf_vector(2) and (not buf)) or ((not
buf_vector(2)) and buf);

        buf := buf2 and buf;
        buf_vector(3) := (buf_vector(3) and (not buf)) or ((not
buf_vector(3)) and buf);
        s4 <= buf_vector;
    end process;
end OP4_arch;

-- Logic and between D and Z (CHANGE)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP5 is
    port(
        D:      in std_logic_vector(3 downto 0);
        Z:      in std_logic_vector(3 downto 0);
        s5:     out std_logic_vector(3 downto 0)
    );
end OP5;

architecture OP5_arch of OP5 is
begin
    process (D, Z) is
    begin
        for i in 0 to 3 loop
            s5(i) <= D(i) and Z(i);
        end loop;
    end process;
end OP5_arch;
```

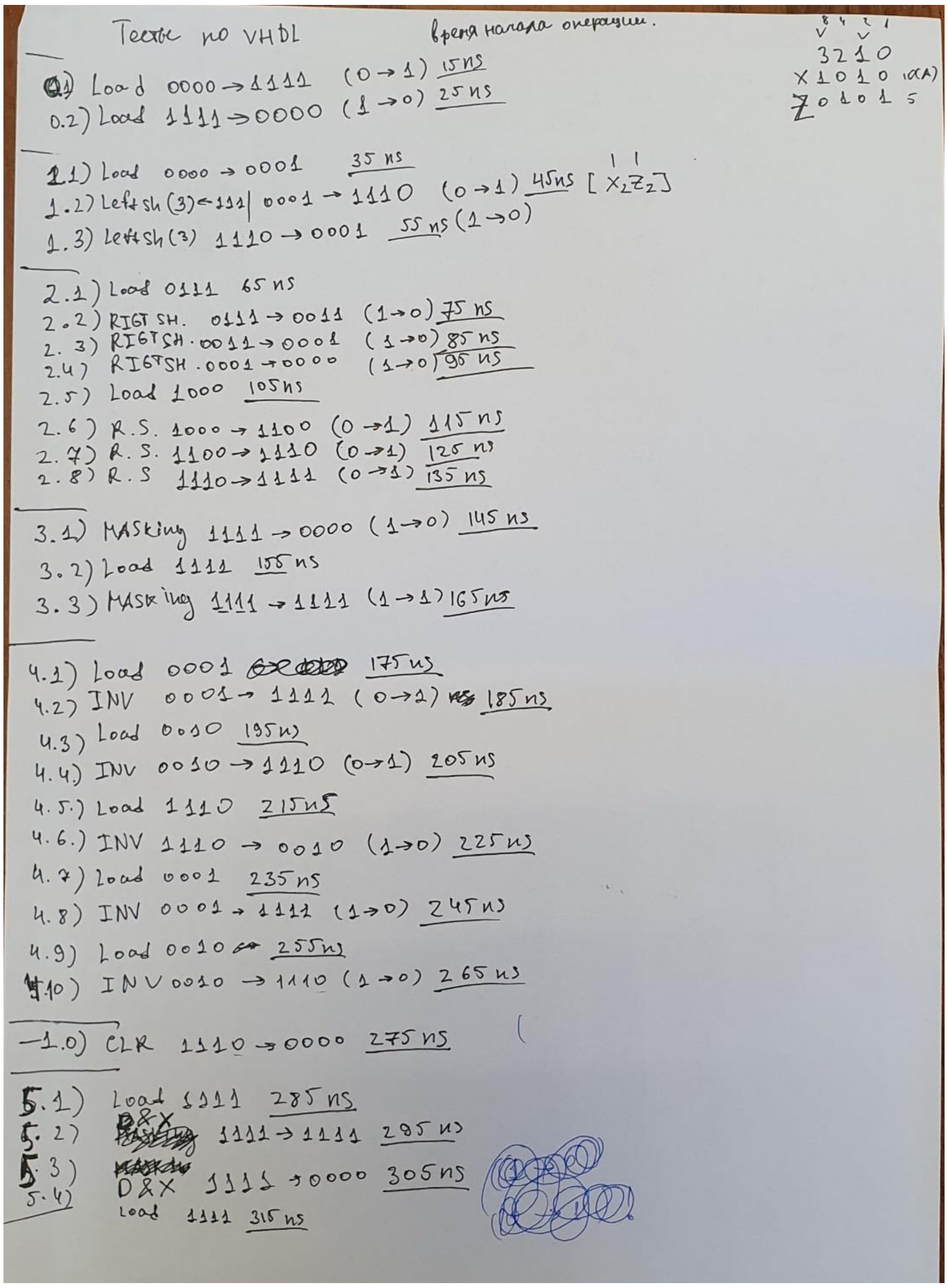
```vhdl
-- Operation with zeros and ones (CHANGE)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity OP6 is
    port(
        X:      in std_logic_vector(3 downto 0);
        Z:      in std_logic_vector(3 downto 0);
        s6:     out std_logic_vector(3 downto 0)
    );
end OP6;

architecture OP6_arch of OP6 is
begin
    process (X, Z) is
        variable zerosQuantity : integer := 0;
        variable firstGroupOnesQnt : integer := 0;
    begin
        if (X(3) = '1' and X(2) = '1') then
            zerosQuantity := 0;
        elsif ((X(3) = '0' and X(2) = '1') or (X(3) = '1' and
X(2) = '0')) then
            zerosQuantity := 1;
            zerosQuantity := zerosQuantity * 4;
        elsif (X(3) = '0' and X(2) = '0') then
            zerosQuantity := 2;
            zerosQuantity := zerosQuantity * 4;
        end if;
        if (Z = "0000") then
            firstGroupOnesQnt := 0;
        elsif (Z = "0001" or Z = "0010" or Z = "0100" or Z =
"0101" or Z = "1000"
             or Z = "1001" or Z = "1010" or Z = "1101") then
            firstGroupOnesQnt := 1;
        elsif (Z = "0011" or Z = "0110" or Z = "1011" or Z =
"1100") then
            firstGroupOnesQnt := 2;
        elsif (Z = "0111" or Z = "1110") then
            firstGroupOnesQnt := 3;
        elsif (Z = "1111") then
            firstGroupOnesQnt := 4;
        end if;
        zerosQuantity := zerosQuantity + firstGroupOnesQnt;
        s6 <= std_logic_vector(to_unsigned(zerosQuantity, 4));
    end process;
end OP6_arch;


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity mux is
    port(
        s0: in std_logic_vector(3 downto 0);
        s1: in std_logic_vector(3 downto 0);
        s2: in std_logic_vector(3 downto 0);
        s3: in std_logic_vector(3 downto 0);
        s4: in std_logic_vector(3 downto 0);
        s5: in std_logic_vector(3 downto 0);
        s6: in std_logic_vector(3 downto 0);
        Y:  in std_logic_vector(2 downto 0);
        s_mux: out std_logic_vector(3 downto 0)
    );
end mux;

architecture mux_arch of mux is
begin
    process (Y, s0, s1, s2, s3, s4, s5, s6) is
    begin
        if (Y = o"0") then
            s_mux <= s0;
        elsif (Y = o"1") then
            s_mux <= s1;
        elsif (Y = o"2") then
            s_mux <= s2;
        elsif (Y = o"3") then
            s_mux <= s3;
        elsif (Y = o"4") then
            s_mux <= s4;
        elsif (Y = o"5") then
            s_mux <= s5;
        elsif (Y = o"6") then
            s_mux <= s6;
        elsif (Y = o"7") then
```

```vhdl
            s_mux <= s6;
        end if;
    end process;
end mux_arch;


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity reg is
    port(
        s_mux:      in std_logic_vector(3 downto 0);
        CLR:        in std_logic;
        C:          in std_logic;
        EN:         in std_logic;
        s_D:        out std_logic_vector(3 downto 0)
    );
end reg;

architecture reg_arch of reg is
begin
    process (s_mux, CLR, C, EN) is
    begin
        if (CLR = '1') then
            s_D <= x"0";
        elsif (C'event and C = '1') then
            if (EN = '1') then
                s_D <= s_mux;
            end if;
        end if;
    end process;
end reg_arch;


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.Numeric_Std.all;
entity outp is
    port(
        s_D:      in std_logic_vector(3 downto 0);
        D:        buffer std_logic_vector(3 downto 0)
    );
```

```
end outp;

architecture outp_arch of outp is
begin
    process (s_D) is
    begin
        D <= s_D;
    end process;
end outp_arch;
```

4. Таблица значений для последней микрооперации

На данных рисунках показан план тестирования макроэлемента.



Тесты по VHDL   Время начала операции.

8 4 2 1
3 2 1 0
X 1 0 1 0 10(A)
Z 0 1 0 1 5

0.1) Load 0000 → 1111 (0 → 1) 15 ns
0.2) Load 1111 → 0000 (1 → 0) 25 ns

1.1) Load 0000 → 0001 35 ns
1.2) Left sh (3) ← 111| 0001 → 1110 (0 → 1) 45ns [X_2Z_2]
1.3) Left sh (3) 1110 → 0001 55 ns (1 → 0)

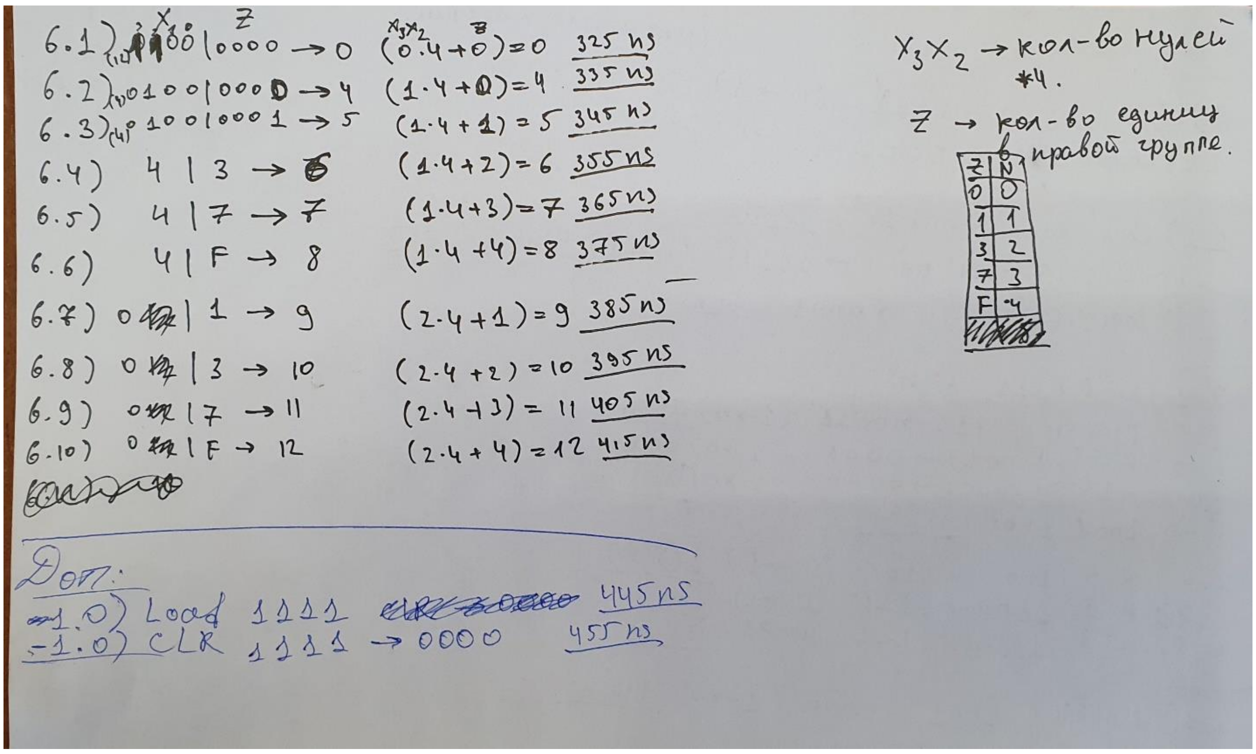
2.1) Load 0111 65 ns
2.2) RIGT SH. 0111 → 0011 (1 → 0) 75 ns
2.3) RIGT SH. 0011 → 0001 (1 → 0) 85 ns
2.4) RIGT SH. 0001 → 0000 (1 → 0) 95 ns
2.5) Load 1000 105 ns
2.6) R.S. 1000 → 1100 (0 → 1) 115 ns
2.7) R.S. 1100 → 1110 (0 → 1) 125 ns
2.8) R.S 1110 → 1111 (0 → 1) 135 ns

3.1) MASKing 1111 → 0000 (1 → 0) 145 ns
3.2) Load 1111 155 ns
3.3) MASKing 1111 → 1111 (1 → 1) 165 ns

4.1) Load 0001 [illegible] 175 ns
4.2) INV 0001 → 1111 (0 → 1) 185 ns
4.3) Load 0010 195 ns
4.4) INV 0010 → 1110 (0 → 1) 205 ns
4.5.) Load 1110 215 ns
4.6.) INV 1110 → 0010 (1 → 0) 225 ns
4.7) Load 0001 235 ns
4.8) INV 0001 → 1111 (1 → 0) 245 ns
4.9) Load 0010 255 ns
4.10) INV 0010 → 1110 (1 → 0) 265 ns

1.0) CLR 1110 → 0000 275 ns

5.1) Load 1111 285 ns
5.2) D&X ~~MASKing~~ 1111 → 1111 295 ns
5.3) ~~MASKing~~ D&X 1111 → 0000 305 ns
5.4) Load 1111 315 ns



6.1) (12) [illegible]00|0000 → 0 $(0\cdot4+0)=0$ 325 ns

6.2) (8) 0100|0000 → 4 $(1\cdot4+0)=4$ 335 ns

6.3) (4) 0100|0001 → 5 $(1\cdot4+1)=5$ 345 ns

6.4) 4 | 3 → 6 $(1\cdot4+2)=6$ 355 ns

6.5) 4 | 7 → 7 $(1\cdot4+3)=7$ 365 ns

6.6) 4 | F → 8 $(1\cdot4+4)=8$ 375 ns

6.7) 0[illegible] | 1 → 9 $(2\cdot4+1)=9$ 385 ns

6.8) 0[illegible] | 3 → 10 $(2\cdot4+2)=10$ 395 ns

6.9) 0[illegible] | 7 → 11 $(2\cdot4+3)=11$ 405 ns

6.10) 0[illegible] | F → 12 $(2\cdot4+4)=12$ 415 ns

X_3X_2 → кол-во нулей *4.

Z → кол-во единиц в правой группе.

Z	N
0	0
1	1
3	2
7	3
F	4
▨	▨

Доп:

1.0) Load 1111 ~~CLR → 0000~~ 445 ns

1.0) CLR 1111 → 0000 455 ns

5. Функционирование тестовой программы

Функционирование тестовой программы изображено на рисунках 5.1, 5.2 и 5.3

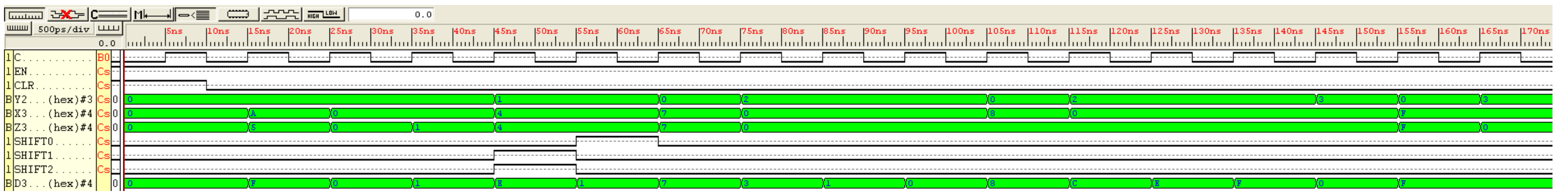


Рисунок – 5.1

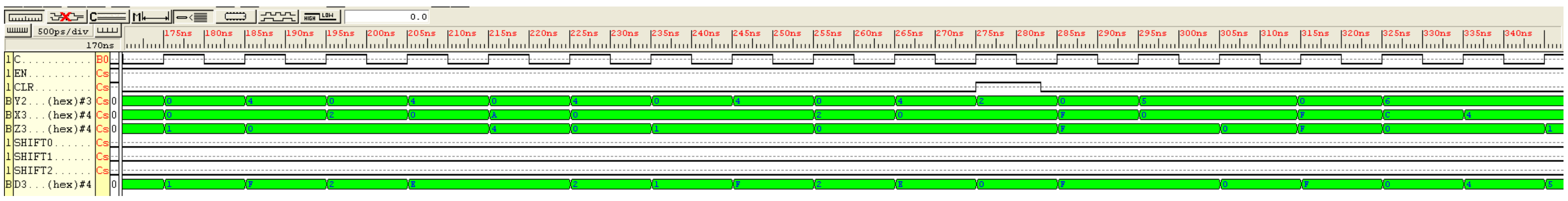


Рисунок – 5.2

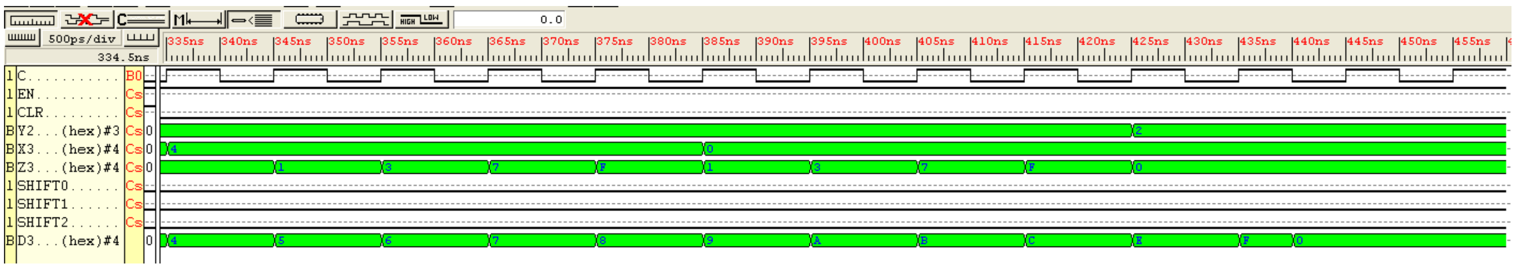


Рисунок – 5.3