МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ

«МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ»



Институт Интеллектуальных Кибернетических Систем

Кафедра «Компьютерные системы и технологии»

Отчёт о лабораторной работе

Разработка многофункционального   
регистра на языке VHDL

Студент группы Б20-503 Коломенский В. Г. / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Москва 2023

**Оглавление**

1. Введение1

2. Структурная схема2

3. Реализация на VHDL4

4. Функциональное тестирование11

5. Список литературы и ссылки14

1. Введение

Цель лабораторной работы: изучить технологию проектирования многофункционального регистра (далее – МФР) при помощи языка описания аппаратуры интегральных схем – VHDL.

Задача – разработать МФР на языке VHDL, реализующий микрооперации, заданные вариантом. Перечень микроопераций варианта 106 приведён на рисунке 1.

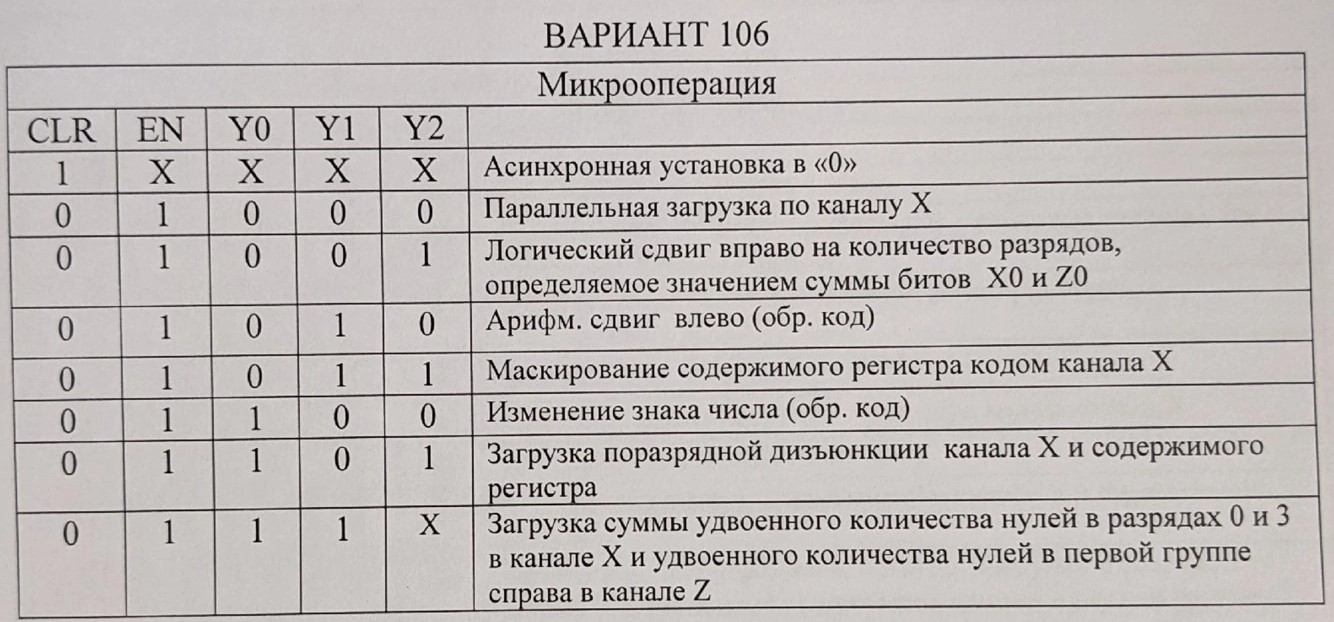


Рисунок 1. – Микрооперации МФР

2. Структурная схема

На рисунке 2 приведена структурная схема мультифункционального регистра. Описание компонент структурной схемы приведено в таблице 1.

Таблица 1. – Описание компонент МФР

|  |  |
| --- | --- |
| CoZ1  (Count of Zeroes 1) | Если Y0 = ‘0’, то на выходе D0; иначе на выходе удвоенное количество нулей на входах D0 и D1. |
| CoZ2  (Count of Zeroes 2) | Если Y0 = ‘0’, то на выходе D0; иначе на выходе удвоенное количество в первой группе справа на входе D0-D3. |
| SM | На выходе A + B. |
| RG | Многофункциональной 4-х разрядный регистр с динамической синхронизацией по фронту (0/1). |

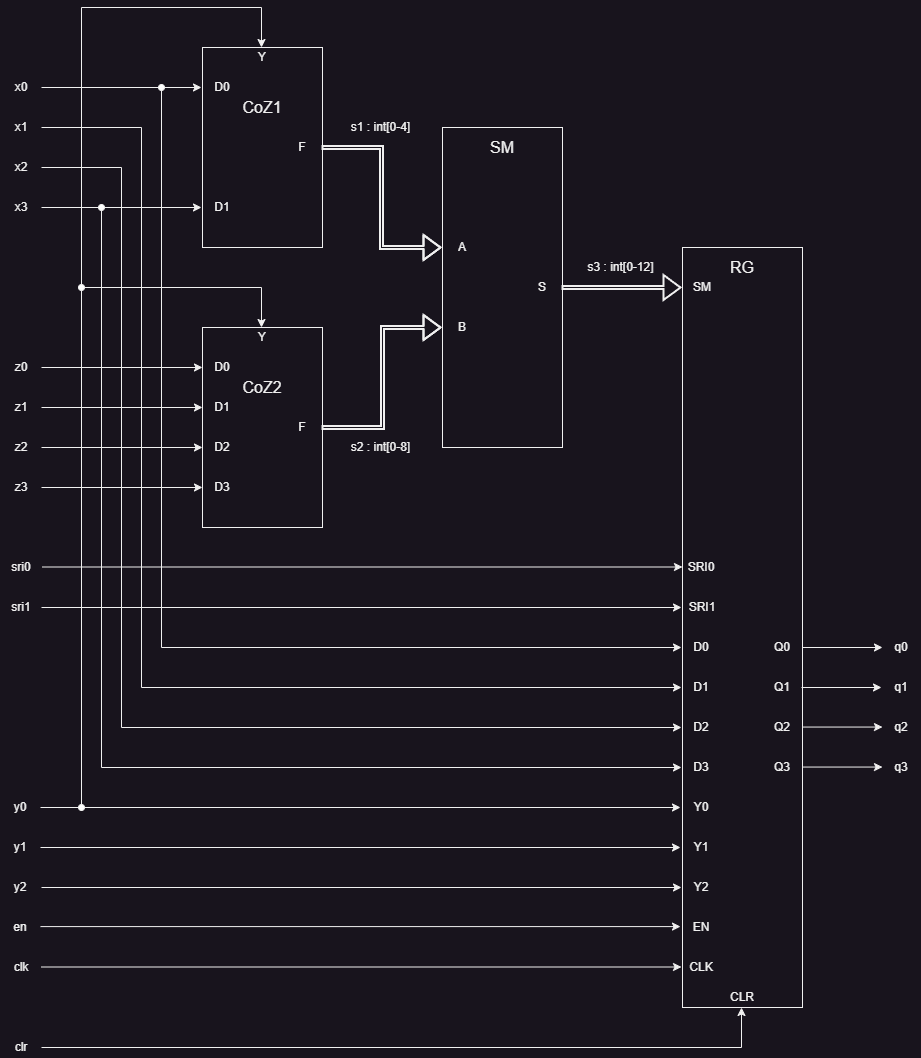


Рисунок 2. – Структурная схема МФР

3. Реализация на VHDL

Реализация МФР на языке VHDL приведена в приложении 1.

Приложение 1. – реализация МФР на VHDL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332 | library IEEE;  use IEEE.std\_logic\_1164.all;  entity MFReg is  port (  x0: in BIT;  x1: in BIT;  x2: in BIT;  x3: in BIT;  z0: in BIT;  z1: in BIT;  z2: in BIT;  z3: in BIT;  sri0: in BIT;  sri1: in BIT;  y0: in BIT;  y1: in BIT;  y2: in BIT;  en: in BIT;  clk: in BIT;  clr: in BIT;  q0: buffer BIT;  q1: buffer BIT;  q2: buffer BIT;  q3: buffer BIT  );  end MFReg;  architecture MFReg\_arch of MFReg is  signal s1: INTEGER range 0 to 4;  signal s2: INTEGER range 0 to 8;  signal s3: INTEGER range 0 to 12;  component CoZ1  port (  D0: in BIT;  D1: in BIT;  Y: in BIT;  F: out INTEGER range 0 to 4  );  end component;  component CoZ2  port (  D0: in BIT;  D1: in BIT;  D2: in BIT;  D3: in BIT;  Y: in BIT;  F: out INTEGER range 0 to 8  );  end component;  component SM  port (  A: in INTEGER range 0 to 4;  B: in INTEGER range 0 to 8;  S: out INTEGER range 0 to 12  );  end component;  component RG  port (  SM: in INTEGER range 0 to 12;  SRI0: in BIT;  SRI1: in BIT;  D0: in BIT;  D1: in BIT;  D2: in BIT;  D3: in BIT;  Y0: in BIT;  Y1: in BIT;  Y2: in BIT;  EN: in BIT;  CLK: in BIT;  CLR: in BIT;  Q0: buffer BIT;  Q1: buffer BIT;  Q2: buffer BIT;  Q3: buffer BIT  );  end component;  begin  OCoZ1: CoZ1  port map (  D0 =>x0,  D1 =>x3,  Y => y0,  F => s1  );  OCoZ2: CoZ2  port map (  D0 =>z0,  D1 =>z1,  D2 =>z2,  D3 =>z3,  Y => y0,  F => s2  );  OSM: SM  port map (  A => s1,  B=> s2,  S => s3  );  ORG: RG  port map (  SM => s3,  SRI0 => sri0,  SRI1 => sri1,  D0 => x0,  D1 => x1,  D2 => x2,  D3 => x3,  Y0 => y0,  Y1 => y1,  Y2 => y2,  EN => en,  CLK => clk,  CLR => clr,  Q0 => q0,  Q1 => q1,  Q2 => q2,  Q3 => q3  );  end MFReg\_arch;  entity CoZ1 is  port (  D0: in BIT;  D1: in BIT;  Y: in BIT;  F: out INTEGER range 0 to 4  );  end CoZ1;  architecture CoZ1\_arch of CoZ1 is  begin  process (D0, D1, Y)  variable D: BIT\_VECTOR (1 downto 0);  begin  if Y='1' then  D:= D1 & D0;  case D is  when "00" => F<= 4;  when "01" => F<= 2;  when "10" => F<= 2;  when "11" => F<= 0;  end case;  else  case D0 is  when '0' => F <= 0;  when '1' => F <= 1;  end case;  end if;  end process;  end CoZ1\_arch;  entity CoZ2 is  port (  D0: in BIT;  D1: in BIT;  D2: in BIT;  D3: in BIT;  Y: in BIT;  F: out INTEGER range 0 to 8  );  end CoZ2;  architecture CoZ2\_arch of CoZ2 is  begin  process (D0, D1, D2, D3, Y)  variable D: BIT\_VECTOR (3 downto 0);  begin  if Y='1' then  D := D3 & D2 & D1 & D0;  case D is  when "0000" => F<= 8;  when "0001" => F<= 6;  when "0010" => F<= 2;  when "0011" => F<= 4;  when "0100" => F<= 4;  when "0101" => F<= 2;  when "0110" => F<= 2;  when "0111" => F<= 2;  when "1000" => F<= 6;  when "1001" => F<= 4;  when "1010" => F<= 2;  when "1011" => F<= 2;  when "1100" => F<= 4;  when "1101" => F<= 2;  when "1110" => F<= 2;  when "1111" => F<= 0;  end case;  else  case D0 is  when '0' => F <= 0;  when '1' => F <= 1;  end case;  end if;  end process;  end CoZ2\_arch;  entity SM is  port (  A: in INTEGER range 0 to 4;  B: in INTEGER range 0 to 8;  S: out INTEGER range 0 to 12  );  end SM;  architecture SM\_arch of SM is  begin  S <= A + B;  end SM\_arch;  entity RG is  port (  SM: in INTEGER range 0 to 12;  SRI0: in BIT;  SRI1: in BIT;  D0: in BIT;  D1: in BIT;  D2: in BIT;  D3: in BIT;  Y0: in BIT;  Y1: in BIT;  Y2: in BIT;  EN: in BIT;  CLK: in BIT;  CLR: in BIT;  Q0: buffer BIT;  Q1: buffer BIT;  Q2: buffer BIT;  Q3: buffer BIT  );  end RG;  architecture RG\_arch of RG is  begin  process (CLR, EN, CLK)  variable Y: BIT\_VECTOR (2 downto 0);  variable Q: BIT\_VECTOR (3 downto 0);  begin  if CLR='1' then  Q0 <= '0';  Q1 <= '0';  Q2 <= '0';  Q3 <= '0';  elsif EN = '0' then  null;  elsif CLK'event and CLK='1' then  Y := Y0 & Y1 & Y2;  if Y = "000" then  Q0 <= D0;  Q1 <= D1;  Q2 <= D2;  Q3 <= D3;  elsif Y = "001" then  if SM = 1 then  Q0 <= Q1;  Q1 <= Q2;  Q2 <= Q3;  Q3 <= SRI0;  elsif SM = 2 then  Q0 <= Q2;  Q1 <= Q3;  Q2 <= SRI0;  Q3 <= SRI1;  end if;  elsif Y = "010" then  Q0 <= Q3;  Q1 <= Q0;  Q2 <= Q1;  elsif Y = "011" then  Q0 <= Q0 and D0;  Q1 <= Q1 and D1;  Q2 <= Q2 and D2;  Q3 <= Q3 and D3;  elsif Y = "100" then  Q := Q0 & Q1 & Q2 & Q3;  if Q = "0000" then  Q0 <= '0';  Q1 <= '0';  Q2 <= '0';  Q3 <= '0';  else  Q0 <= not Q0;  Q1 <= not Q1;  Q2 <= not Q2;  Q3 <= not Q3;  end if;  elsif Y = "101" then  Q0 <= Q0 or D0;  Q1 <= Q1 or D1;  Q2 <= Q2 or D2;  Q3 <= Q3 or D3;  elsif Y = "110" or Y = "111" then  Q0 <= '0';  case SM is  when 2 | 6 | 10 => Q1 <= '1';  when others => Q1 <= '0';  end case;  case SM is  when 4 | 6 | 12 => Q2 <= '1';  when others => Q2 <= '0';  end case;  case SM is  when 8 | 10 | 12 => Q3 <= '1';  when others => Q3 <= '0';  end case;  end if;  end if;  end process;  end RG\_arch; |

4. Функциональное тестирование

На рисунке 3.1 и 3.2 представлены тесты, на которых проверялся многофункциональный регистр.

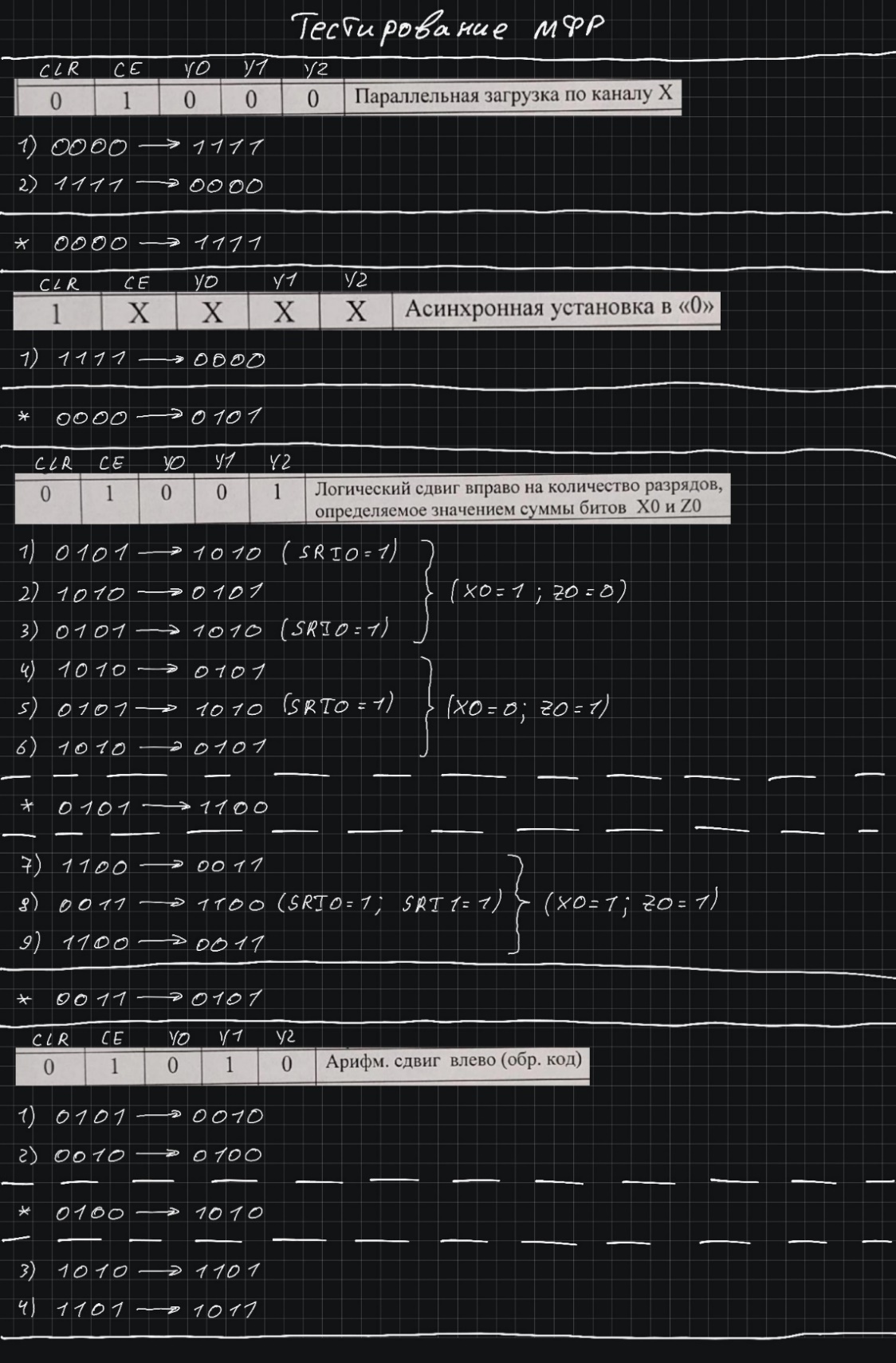


Рисунок 3.1. – Набор тестов

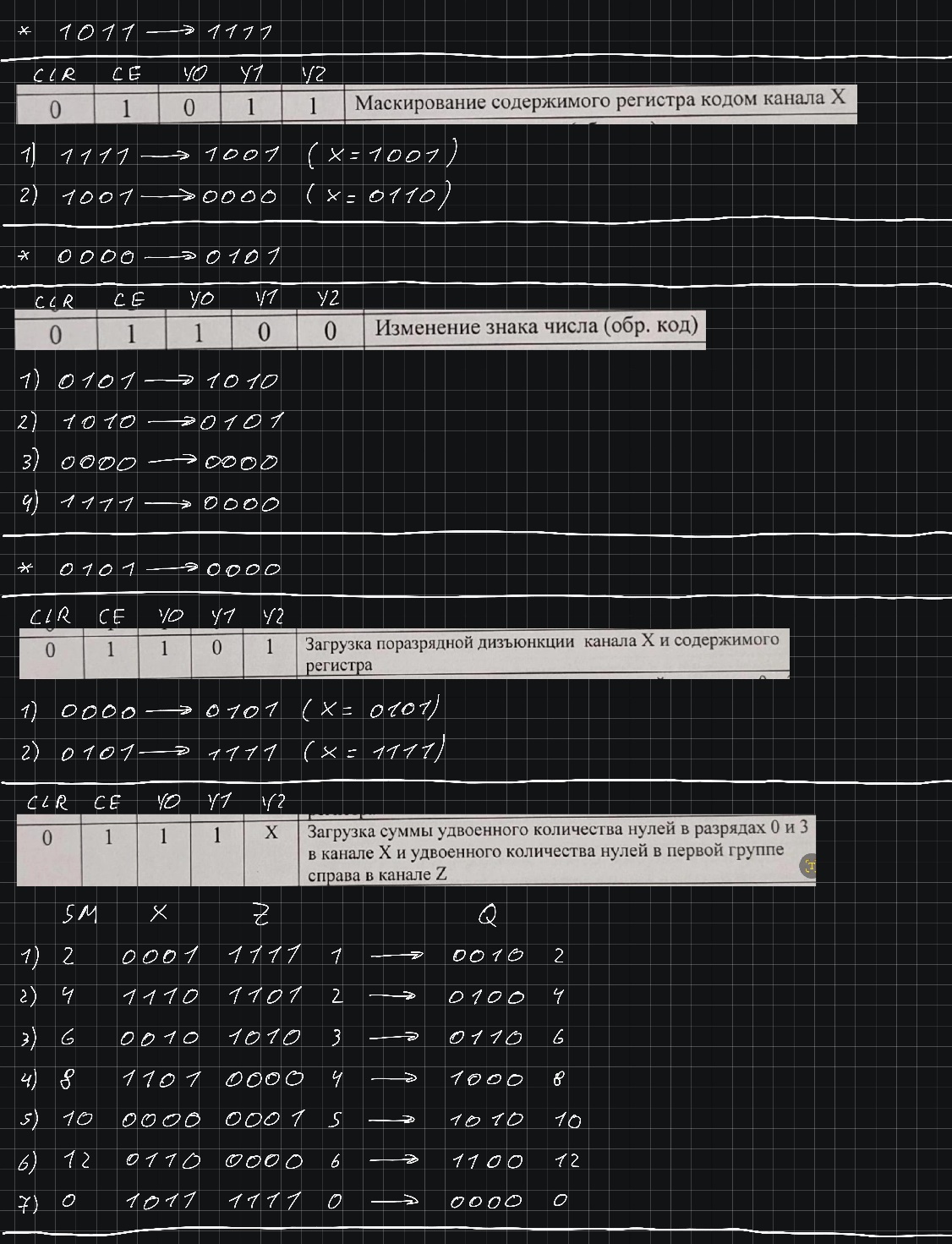


Рисунок 3.2. – Набор тестов

Результаты тестирования МФР представлены на рисунках 4.1, 4.2 и 4.3.

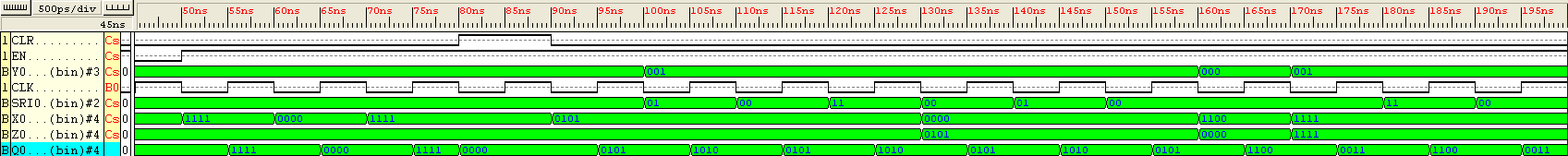


Рисунок 4.1 – Результаты тестирования

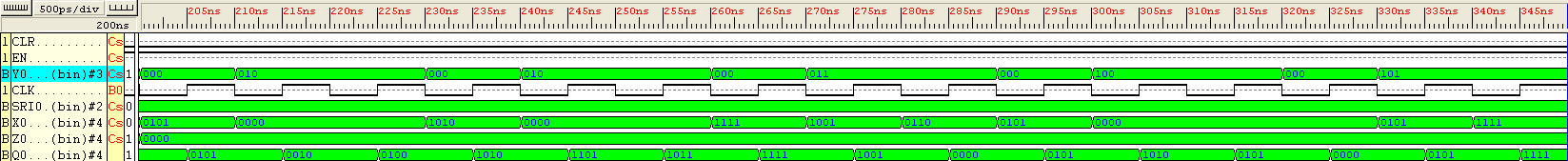


Рисунок 4.2 – Результаты тестирования

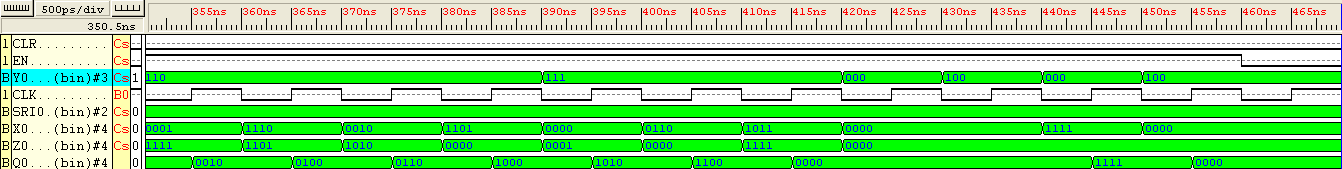


Рисунок 4.3 – Результаты тестирования

5. Список литературы и ссылки

1. Ковригин Б. Н. Введение в инструментальные средства проектирования и отладки цифровых устройств на ПЛИС: Учебно-методическое пособие. М.: МИФИ, 2006.
2. Исходный код МФР на языке VHDL [Электронный ресурс] // <https://github.com/Hypex146/FLMnDA-Lab>.