**Московский государственный технический университет
имени Н. Э. Баумана**

**Факультет «Информатика и системы управления»
Кафедра «Проектирование и технология производства
электронной аппаратуры» (ИУ-4)**

Д. А. Аминев

Цифровая схемотехника

*Учебно-методический комплекс практических заданий по дисциплине
«Цифровая схемотехника»
по направлению 11.11.03 «Конструирование и технология
электронных средств»*

Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2020

СОДЕРЖАНИЕ

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ И ТЕРМИНОВ

ЭА	– Электронная аппаратура
ЭС	– Электронные средства
ПККТР	– Программного комплекса конструкторско- технологических расчетов
КТР	– Конструкторско-технологические расчеты

ВВЕДЕНИЕ

Предлагаемое учебное пособие содержит основы оформления документации для конструирования изделий электронной аппаратуры (ЭА), а также теоретические и практические вопросы по автоматизированному проведению конструкторских расчетов с учетом требований технического задания.

Содержание данного пособия базируется на лекционном материале дисциплин «Теоретические основы конструирования и надежности», «Автоматизированное проектирование ЭС» и «Конструкторское проектирование».

В пособии содержатся методические рекомендации по организации комплексного подхода к проектированию изделий ЭА, разработаны принципы построения программного комплекса конструкторско-технологических расчетов (ПККТР), сформулированы задачи, решаемые отдельными модулями, и представлено необходимое информационное обеспечение.

Предлагаемый материал имеет важное значение для подготовки высококвалифицированных специалистов по направлениям «Конструирование и технология электронных средств» и «Информатика и вычислительная техника».

При проектировании функциональных узлов ЭА актуальной проблемой является автоматизация процесса проведения конструкторско-технологических расчетов.

Целью работы является объединение отдельных расчетов, реализуемых программными модулями, в комплекс с единым пользовательским интерфейсом, что позволит ускорить обучение работе с программами расчетов и сократить сроки их проведения.

В процессе выполнения КТР возникает необходимость большого числа вычислений, обращений к стандартным алгоритмам решения типовых задач, проверки правильности результатов.

Представленный в настоящем учебном пособии материал призван упростить процедуру выполнения курсовой работы, а также улучшить усвоение материала по вышеперечисленным курсам.

1 ТРИГГЕРЫ

1.1 D-триггер

D-триггером называется триггер с одним информационным входом, работающий так, что сигнал на выходе после переключения равен сигналу на входе D до переключения, т. е. Qn+1=Dn Основное назначение D-триггеров - задержка сигнала, поданного на вход D.

В таблицах 1.1.1 и 1.1.2 показан листинг реализации D-триггера на языках VHDLи Verilog.

Таблица 1.1.1 – Листинг реализации D – триггера на языке VHDL

```
entitytrig_vhdlis
port(CLK,DIN: in bit; DOUT: out bit);
end;
architecture Behavioral of trig_vhdl is
begin
process(CLK)
begin
     if(CLK'event and CLK='1') then
          DOUT <= DIN;
     end if;
end process;
end;
```

Таблица 1.1.2 – Листинг реализации D – триггера на языке Verilog

```
module trig(d,clk,q);
input d,clk;
output q;
                                reg q;
always @(posedgeclk)
     q <= d;
endmodule
```

В таблицах 1.1.3 и 1.1.4 показан листинг проверкиD-триггера на языках VHDLи Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.1.3 – Листинг проверкиD – триггера на языке VHDL

```
module trig_vhdl_tb;

     // Inputs
       reg CLK;
       reg DIN;

     // Outputs

       wire DOUT;


     // Instantiate the Unit Under Test (UUT)

     trig_vhdluut (
                  .CLK(CLK),
                  .DIN(DIN),
                  .DOUT(DOUT)
                  );

     initial begin
```

Продолжение таблицы1.1.3

```
            // Initialize Inputs
            CLK = 0;
            DIN = 0;

            // Wait 100 ns for global reset to finish
            #100;
            // Add stimulus here

        end
        always begin #20 DIN=1; #30 DIN=0;end
            always begin #5 CLK = ~CLK;end
    endmodule
```

Таблица 1.1.4 – Листинг проверкиD – триггера на языке Verilog

```
    module trig_tb;

        // Inputs
        reg d;
        regclk;

        // Outputs
        wire q;

        // Instantiate the Unit Under Test (UUT)
        trig uut (
                .d(d),
                .clk(clk),
                .q(q)
                );

        initial begin
            // Initialize Inputs
            d = 0;
            clk = 0;

            // Wait 100 ns for global reset to finish
            #100;

            // Add stimulus here

        end
            always begin #20 d=1; #30 d=0;end
            always begin #5 clk = ~clk;end
    endmodule
```

В рисунке 1.1.1 показана временная диаграммаработы D-триггера.

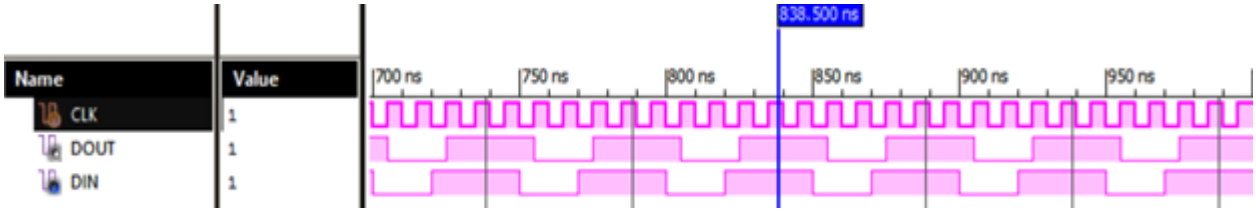


Рисунок 1.1.1 Временная диаграмма работы D-триггер

На временной диаграмме видно, что он имеет информационный вход DIN (вход данных) и вход синхронизации CLK. Вход синхронизации CLK может быть статическим

(потенциальным) и динамическим. У триггеров со статическим входом С информация записывается в течение времени, при котором уровень сигнала CLK=1. В триггерах с динамическим входом CLK информация записывается только в течение перепада напряжения на входе С. Динамический вход изображают на схемах треугольником. Если вершина треугольника обращена в сторону микросхемы (прямой динамический вход), то триггер срабатывает по фронту входного импульса, если от нее (инверсный динамический вход) - по срезу импульса. В таком триггере информация на выходе может быть задержана на один такт по отношению к входной информации.

1.2 D-триггер RST

Триггеры, у которых состояние меняется только путем поступления тактовых импульсов, называют синхронными триггерами.

К ним относятся RST-триггер, имеющий времязадающий вход С (clock). В промежутках между тактовыми импульсами изменения сигналов на входах не вызывают переключения триггера.

В таблицах 1.2.1 и 1.2.2 показан листинг реализации D-триггера RST на языках VHDLи Verilog.

Таблица 1.2.1 – Листинг реализации D – триггераRST на языке VHDL

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

entity dffr is
port (CLK,DIN,RST_N: in STD_LOGIC;
            DOUT: out STD_LOGIC);
end;

architecture Behavioral of dffr is
     begin
     process (CLK, RST_N) begin
          if RST_N = '0' then
               DOUT <='0' after 6ns;
          elsif (CLK'event and CLK = '1') then
               DOUT <= DIN after 6ns;
          end if;
     end process;
end Behavioral;
```

Таблица 1.2.2 – Листинг реализации D – триггераRST на языке Verilog

```
moduledffr
(input wire clk, din, RST_N,
outputregdout);


always @(posedgeclk,

     begin
         if(!RST_N)
         dout<= 0;
else
```

В таблицах 1.2.3 и 1.2.4 показан листинг реализации D-триггераRST на языках VHDLи Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.2.3 – Листинг проверкиD – триггераRST на языке VHDL

```
    always begin #10 CLK = ~CLK; end
    always begin #20 DIN = ~DIN; end
    always begin #200 RST_N = ~RST_N; end
```

Таблица 1.2.4 – Листинг проверкиD – триггераRST на языке Verilog

```
                    moduletrig_tb;

 // Inputs
 regclk;
 reg din;
 reg RST_N;

 // Outputs
 wiredout;

 // Instantiate the Unit Under Test (UUT)
                      dffruut (
                            .clk(clk),
                            .din(din),
                            .RST_N(RST_N),
                            .dout(dout)
                    );

                    initial begin
                       // Initialize Inputs
                       clk = 1;
                       din = 0;
                       RST_N = 0;

                       // Wait 100 ns for global reset to finish
                       #100;

      // Add stimulus here

                    end

                          always begin #20 din=1; #30 din=0; end
                          always begin #5 clk = ~clk; end
                          always begin #50 RST_N=0; #50 RST_N=1; end
                    endmodule
```

Рисунок 1.2.1Временная диаграмма работы D-триггер RST

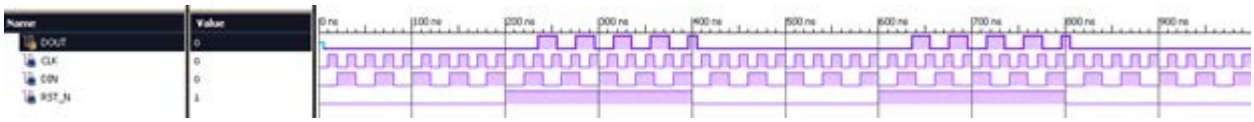


Рисунок 1.2.1Временная диаграмма работы D-триггер RST

На временной диаграмме видно, что он имеет информационный вход DIN (вход данных) и вход синхронизации CLK. Вход синхронизации CLK может быть статическим (потенциальным) и динамическим. У триггеров со статическим входом С информация записывается в течение времени, при котором уровень сигнала CLK=1. В триггерах с

динамическим входом CLK информация записывается только в течение перепада напряжения на входе С. Динамический вход изображают на схемах треугольником. Если вершина треугольника обращена в сторону микросхемы (прямой динамический вход), то триггер срабатывает по фронту входного импульса, если от нее (инверсный динамический вход) - по срезу импульса. В таком триггере информация на выходе может быть задержана на один такт по отношению к входной информации.

1.3 D-триггер ~& Verilog

Триггеры предназначены для запоминания двоичной информации. Использование триггеров позволяет реализовывать устройства оперативной памяти (то есть памяти, информация в которой хранится только на время вычислений). Данный элемент реализован на элементах “НЕ-И”.

В таблицах 1.3.1 показан листинг реализации D-триггера ~&на языкеVerilog.

Таблица 1.3.1 – Листинг реализации D – триггера~&на языке Verilog

```
`timescale 1ns / 1ps
module trig(d,clk,q);

     input d,clk;
     output q;
     wire a,b,qinv;
          nand GATE00(a,d,clk);
          nand GATE01(b,a,clk);
          nand GATE02(q,a,qinv);
          nand GATE03(qinv,q,b);
endmodule
```

В таблице 1.3.2 показан листинг проверкиD-триггера ~&на языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.3.2– Листинг проверкиD – триггера~& на языке Verilog

```
`timescale 1ns / 1ps
module trig_tb;
     // Inputs
     reg d;
     reg clk;
     // Outputs
      wire q;
     // Instantiate the Unit Under Test (UUT)
          trig uut (
                   .d(d),
                   .clk(clk),
                   .q(q)
                   );
     initial begin
          // Initialize Inputs
          d = 0;
          clk = 0;
          #100;
// Add stimulus here
end
          always begin #20 d=1; #30 d=0;end
```

```
                always begin #5 clk = ~clk;end
    endmodule
```

В рисунке 1.3.1 показана временная диаграммаработы D-триггера~&Verilog.

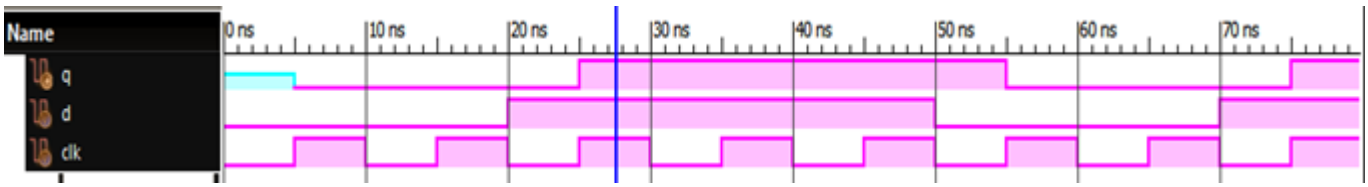


Рисунок 1.3.1Временная диаграмма работы D-триггер ~&Verilog

На временной диаграмме видно, что он имеет информационный вход DIN (вход данных) и вход синхронизации CLK. Вход синхронизации CLK может быть статическим (потенциальным) и динамическим. У триггеров со статическим входом С информация записывается в течение времени, при котором уровень сигнала CLK=1. В триггерах с динамическим входом CLK информация записывается только в течение перепада напряжения на входе С. Динамический вход изображают на схемах треугольником. Если вершина треугольника обращена в сторону микросхемы (прямой динамический вход), то триггер срабатывает по фронту входного импульса, если от нее (инверсный динамический вход) - по срезу импульса. В таком триггере информация на выходе может быть задержана на один такт по отношению к входной информации.

1.4 RS-триггер Verilog

RS-триггер - асинхронный триггер, который сохраняет своё предыдущее состояние при неактивном состоянии обоих входов и изменяет своё состояние при подаче на один из его входов активного уровня.

В таблице 1.4.1 показан листинг реализации RS-триггер на языкахVerilog.

Таблица 1.4.1 – Листинг реализации RS-триггер на языке Verilog

```
module rs(R, S, Q);
   input R, S;
   output Q;
   reg Q;
   reg Q;
   always @(R or S) begin
         if (((R==0) && (S==0)))
               $display("RS-vozmozhni gonki");
         if (((R==1) && (S==0)))
               Q <= #17 1;
         else if (((R==0) && (S==1)))
               Q <= #13 0;
   end
endmodule
```

Представленный выше модуль при изменении значений входов R или S согласно таблице истинности изменяет состояние выхода в соответствии с таблицей истинности, добавляя задержку на выходной сигнал.

В таблице 1.4.2 показан листинг задержекRS-триггер на языках Verilog.

Таблица 1.4.2 – Листинг задержекRS-триггер на языке Verilog

```
                    Timescale 1ns/100 ps
                              module RST(S,R,Q)
                      input S,R
                      outpot Q; reg Q;
                     always @(R or S) begin

                      if ( ((R==0) && (S==0)))
                      $display("RS - возможны гонки");


                      if ( ((R==1) && (S==0)))
                                          Q<=#17 1;
                                    else if((R==0) && (S==1))
                    Q<=#13 0;


                          end
    endmodule
```

Код проверки представляет из себя попеременное включение портов set и reset, которые захватывают три основных режима работы триггера, исключая режим неопределенного состояния.В 1.4.3 показан листинг проверкиRS-триггер на языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.4.3 – Листинг проверкиRS-триггер на языке Verilog

```
                    always begin #20 S = 0; #40 S = 1; end
                    always begin #40 R = 1; #20 R = 0; end
```

На рисунке 1.4.1 представлена временная диаграмма работы модуля в соответствии с кодом отладки, приведенным в разделе 1.4.1.

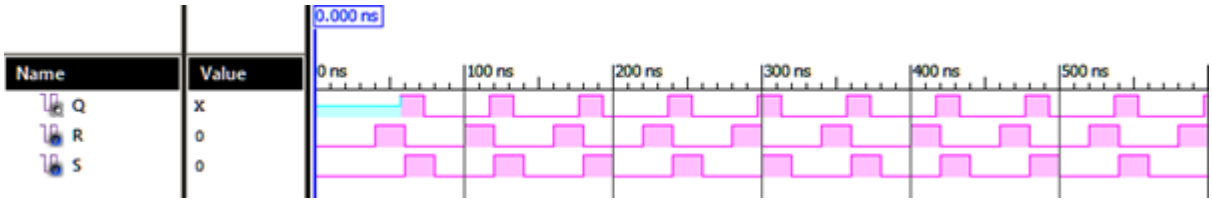


Рисунок 1.4.1 Временная диаграмма работы RS-триггер Verilog

RS-триггер запоминает значения поданные на S или R вход, только при наличии единицы на C (Clock) сигнале – синхронизирующий или тактовый. Он позволяет избежать переходных процессов в схемах, а если быть точнее, переходных состязаний, когда один сигнал на вход может поступить раньше другого, и схема будет работать неправильно.

Именно для этого предусмотрен синхронизирующий сигнал, который позволяет «включать» триггер в нужный нам момент времени.

1.5 RS триггер OR VHDL

RS триггер получил название по названию своих входов. Вход S (Set — установить англ.) позволяет устанавливать выход триггера Q в единичное состояние (записывать единицу). Вход R (Reset — сбросить англ.) позволяет сбрасывать выход триггера Q (Quit — выход англ.) в нулевое состояние (записывать ноль). Данный RS триггер реализован на элементах “ИЛИ”.

В таблице 1.5.1показан листинг реализации RS триггер ORна языке VHDL.

Таблица 1.5.1 – Листинг реализации RS триггер ORна языке VHDL

```
MAIN PROGRAM (1 file)

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RS_trigg1 is
     port(
          R : in STD_LOGIC;
          S : in STD_LOGIC;
          Q : out STD_LOGIC;
                NQ : out STD_LOGIC
          );
end RS_trigg1;

architecture Behavioral of RS_trigg1 is
   component OR_Not is
         port(
                     A: in std_logic;
                     B: in std_logic;
                     C: out std_logic
         );
   end component;

   signal q_temp, nq_temp : std_logic;
begin

   u1: OR_Not
         port map(
               A => R,
               B => nq_temp,
               C => q_temp
         );

   u2: OR_Not
         port map(

               A => S,
               B => q_temp,
               C => nq_temp
         );
   Q <= q_temp;

end Behavioral;
```

В таблице 1.5.2 показан листинг проверкиRS триггер ORна языках VHDL. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.5.2 – Листинг проверкиRS триггер ORна языке VHDL

```
MAIN COMPONENT (2 file)
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity OR_Not is
   port(
                    A: in std_logic;
                    B: in std_logic;
                    C: out std_logic
        );
end OR_Not;

architecture Behavioral of OR_Not is

begin
   C <= NOT (A or B);

end Behavioral;
                   TEST BENCH(3 file)

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity RS_trigg1_tb is
end RS_trigg1_tb;

architecture Behavioral of RS_trigg1_tb is
   component RS_trigg1 is
     port(
                    R : in STD_LOGIC;
                    S : in STD_LOGIC;
                    Q : out STD_LOGIC;
                    NQ : out STD_LOGIC
               );
   end component;
   signal r: std_logic := '0';
   signal s: std_logic := '0';
   signal q,nq: std_logic;
begin

   changing: RS_trigg1
   port map(
                           R => r,
                           S => s,
                           Q => q,
                           NQ => NQ
                    );
   change_r: process(r)
         begin
               r <= not r after 2.5 ns;
         end process;

   change_s: process(s)
         begin
               s <= not s after 5 ns;
         end process;
```

```
end Behavioral;
```

В рисунке 1.5.1 показана временная диаграммаработы RS триггер OR.

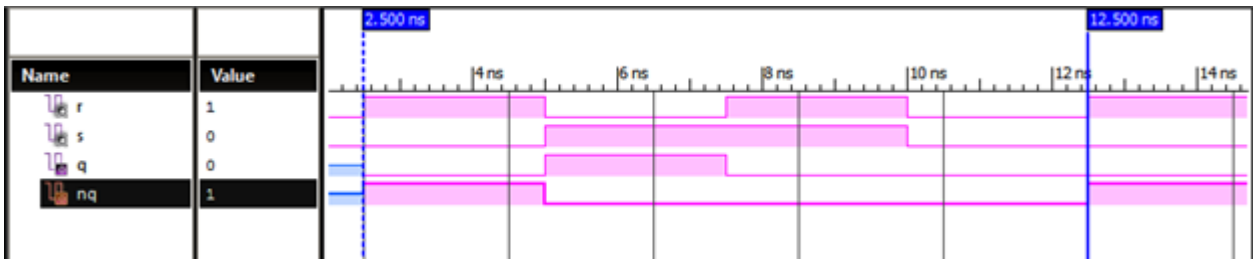


Рисунок 1.5.1 Временная диаграмма работы RS триггер ORVHDL

RS-триггер запоминает значения поданные на S или R вход, только при наличии единицы на C (Clock) сигнале – синхронизирующий или тактовый. Он позволяет избежать переходных процессов в схемах, а если быть точнее, переходных состязаний, когда один сигнал на вход может поступить раньше другого, и схема будет работать неправильно. Именно для этого предусмотрен синхронизирующий сигнал, который позволяет «включать» триггер в нужный нам момент времени.

1.6 Т-триггер

Т-триггер — это счетный триггер. У данного триггера имеется только один вход. Принцип работы Т-триггера заключается в следующем. После поступления на вход Т импульса, состояние триггера меняется на прямо противоположное. Счётным он называется потому, что Т триггер как бы подсчитывает количество импульсов, поступивших на его вход. Жаль только, что считать этот триггер умеет только до одного. При поступлении второго импульса Т-триггер снова сбрасывается в исходное состояние.

В таблицах 1.6.1и 1.6.2 показан листинг реализацииТ-триггер на языках VHDLи Verilog.

Таблица 1.6.1 – Листинг реализации Т-триггера на языке VHDL

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

entity ttrig is
port(
        CLK: in STD_LOGIC;
        Tt: out STD_LOGIC
        );
end ttrig;

architecture Behavioral of ttrig is
   signal Tt_out: STD_LOGIC := '0';
begin
   Tt <= Tt_out;
   process(CLK)
   begin
        if(CLK'event and CLK='1') then
```

```
                        Tt_out <= not (Tt_out);
                end if;
        end process;
end Behavioral;
```

Таблица 1.6.2 – Листинг реализации Т-триггера на языке Verilog

```
modulet_trigger(d,clk,q);
inputd,clk;
output q;
reg q;
initial q=0;
always @(posedgeclk)
   q = ~q;
endmodule
```

В таблицах 1.6.3 и1.6.4 показан листинг проверки Т-триггера на языках VHDL и Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.6.3 – Листинг проверки Т-триггера на языке VHDL

```
`timescale 1ns / 1ps

module ttrig_TB3;

   // Inputs
   reg CLK;

   // Outputs
   wire Tt;

   // Instantiate the Unit Under Test (UUT)
   ttrig uut (
         .CLK(CLK),
         .Tt(Tt)
   );
   initial begin
         // Initialize Inputs
         CLK = 0;

         // Wait 100 ns for global reset to finish
         #100;

         // Add stimulus here

   end
    always #5 CLK=~CLK;

endmodule
```

Таблица 1.6.4 – Листинг проверки Т-триггера на языке Verilog

```
modulet_trigger_tb;

   // Inputs
   reg d;
   regclk;

   // Outputs
   wire q;

   // Instantiate the Unit Under Test (UUT)
   t_triggeruut (
         .d(d),
```

```
                                .clk(clk),
                                .q(q)
                        );

                        initial begin
```

Продолжение таблицы 1.6.4

```
        // Initialize Inputs
d = 0;
                clk = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

        end
                always begin #20 d=1; #30 d=0;end
                always begin #5 clk = ~clk;end
endmodule
```

В рисунке 1.6.1 показана временная диаграммаработы Т-триггера.

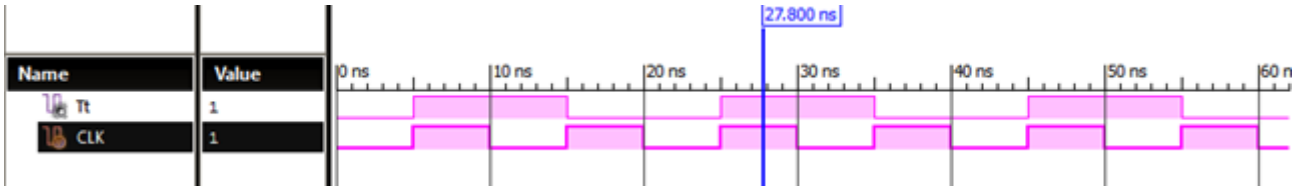


Рисунок 1.6.1 Временная диаграмма работы Т-триггер VHDL

У данного триггера имеется только один вход. Принцип работы Т-триггера заключается в следующем. После поступления на вход Т импульса, состояние триггера меняется на прямо противоположное. Счётным он называется потому, что Т триггер как бы подсчитывает количество импульсов, поступивших на его вход. Жаль только, что считать этот триггер умеет только до одного. При поступлении второго импульса Т-триггер снова сбрасывается в исходное состояние.

1.7 Т-триггер ~& Verilog

Т-триггер — это счетный триггер. У данного триггера имеется только один вход. Принцип работы Т-триггера заключается в следующем. После поступления на вход Т импульса, состояние триггера меняется на прямо противоположное. Счётным он называется потому, что Т триггер как бы подсчитывает количество импульсов, поступивших на его вход. Жаль только, что считать этот триггер умеет только до одного. При поступлении второго импульса Т-триггер снова сбрасывается в исходное состояние.Данный Т-Триггер реализован на элементах “НЕ-И”.

В таблице 1.7.1показан листинг реализации Т-триггер ~&на языках Verilog.

Таблица 1.7.1 – Листинг реализации Т-триггера ~&на языке Verilog

```
                `timescale 1ns / 1ps
```

```
module Tflipflop(clk, t);
output t;
wire a,b,c,d,e,f;
input clk;
```

Продолжение таблицы 1.7.1

```
nand GATE01(f,e,t);
nand GATE02(clk,c,f);
nand GATE03(d,f,b);

Endmodule
```

В таблице 1.7.2 показан листинг проверки Т-триггер ~&на языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.7.3 – Листинг проверкиТ-триггер ~&на языке Verilog

```
`timescale 1ns / 1ps
module tf_tb;

   // Inputs
   reg t;
   reg clk;

   // Outputs
   //wire t;

   // Instantiate the Unit Under Test (UUT)
   Tflipflop uut (
         .clk(clk),
         .t(t)
   );

   initial begin
         // Initialize Inputs
         t = 0;
         clk = 0;

         // Wait 100 ns for global reset to finish
         #100;

         // Add stimulus here

   end
   always begin #5 clk = ~clk; end
endmodule
```

В рисунке 1.7.1 показана временная диаграммаработы Т-триггер ~&.

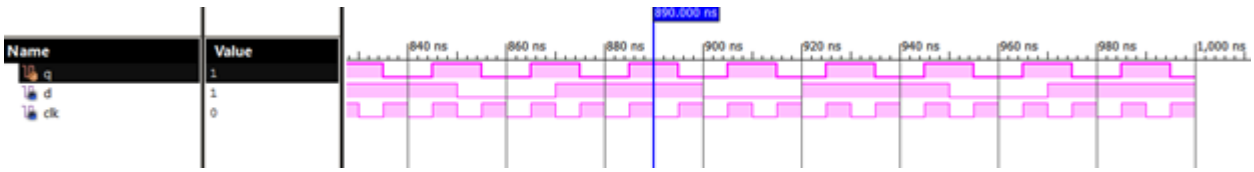


Рисунок 1.7.1 Временная диаграмма работы Т-триггер ~&Verilog

У данного триггера имеется только один вход. Принцип работы Т-триггера заключается в следующем. После поступления на вход Т импульса, состояние триггера

меняется на прямо противоположное. Счётным он называется потому, что Т триггер как бы подсчитывает количество импульсов, поступивших на его вход. Жаль только, что считать этот триггер умеет только до одного. При поступлении второго импульса Т-триггер снова сбрасывается в исходное состояние.

1.8 JK триггер Verilog

JK-триггерработает так же как RS-триггер, с одним лишь исключением: при подаче логической единицы на оба входа J и K состояние выхода триггера изменяется на противоположное, то есть выполняется операция инверсии (чем он отличается от RS-триггеров с доопределённым состоянием, которые строго переходят в логический ноль или единицу, независимо от предыдущего состояния).

В таблице 1.8.1показан листинг реализации JK-триггера на языках Verilog.

Таблица 1.8.1 – Листинг реализации JK-триггерна языке Verilog

```
`timescale 1ns / 1ps

module JK(j,k,clk,rst, q,qbar);
    input j,k,clk,rst;
    output q,qbar;
reg q;
always @ (posedge clk)
begin
if (rst) q<=1'b0;
else begin
            case ({j,k})
                 2'b00: q<=q;
                 2'b01: q<=1'b0;
                 2'b10: q<=1'b1;
                 2'b11: q<=~q;
            endcase
    end
end
assign qbar=~q;
endmodule
```

В таблицах 1.8.2 показан листинг проверки JK-триггера на языкахVerilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 1.8.2 – Листинг проверкиJK-триггера на языке Verilog

```
`timescale 1ns / 1ps
module JKtb;
   // Inputs
   reg j;
   reg k;
   reg clk;
   reg rst;
   // Outputs
   wire q;
   wire qbar;
   // Instantiate the Unit Under Test (UUT)
   JK uut (
         .j(j),
         .k(k),
         .clk(clk),
```

```
                .rst(rst),
                .q(q),
                .qbar(qbar)
        );
```

Продолжение таблицы 1.8.2

```
        initial begin
        clk=1'b1;
        forever #5 clk=~clk;
        end

        initial begin
                // Initialize Inputs
                j = 0;
                k = 0;
                rst = 1;#20
                j = 1;




                k = 0;
                rst = 0;#20
                j = 0;
                k = 1;
                rst = 0;#20
                j = 1;
                k = 1;
                rst = 0;#20
                j = 0;
                k = 0;
                rst = 0;#20

                // Wait 100 ns for global reset to finish
                $stop;

                // Add stimulus here

        end

endmodule
```

В рисунке 1.8.1 показана временная диаграммаработы JK-триггера.

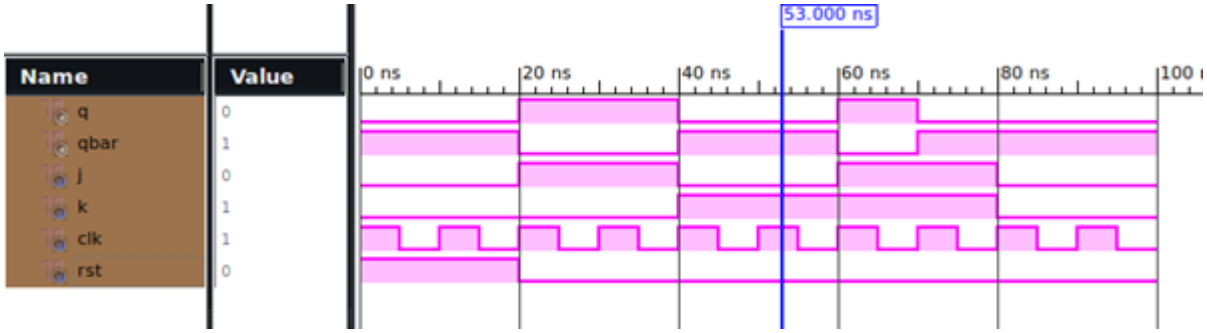


Рисунок 1.8.1 Временная диаграмма работы JK триггер Verilog

Вход J аналогичен входу S у RS-триггера. Вход K аналогичен входу R у RS-триггера. При подаче единицы на вход J и нуля на вход K выходное состояние триггера становится равным логической единице. А при подаче единицы на вход K и нуля на вход J выходное состояние триггера становится равным логическому нулю. JK-триггер в

отличие от RS-триггера не имеет запрещённых состояний на основных входах, однако это никак не помогает при нарушении правил разработки логических схем. На практике применяются только синхронные JK-триггеры, то есть состояния основных входов J и K учитываются только в момент тактирования, например по положительному фронту импульса на входе синхронизации, поскольку понятие «одновременности» для асинхронных сигналов уже само по себе, в самом определении, содержит неопределённость поведения по типу гонки состояний.

2 ГЕНЕРАТОРЫ, СЧЕТЧИКИ

2.1 ГенераторVHDL

При работе цифровых схем часто возникает задача синхронизации моментов изменения или записи сигналов. Для этого можно воспользоваться любым известным генератором периодических сигналов.Генератор сигналов — это устройство, позволяющее получать сигнал определённой природы (электрический, акустический и т. д.), имеющий заданные характеристики (форму, энергетические или статистические характеристики и т. д.). Генераторы широко используются для преобразования сигналов, для измерений и в других областях. Состоит из источника (устройства с самовозбуждением, например, усилителя, охваченного цепью положительной обратной связи) и формирователя (например, электрического фильтра).

В таблице 2.1.1 показан листинг реализации Генераторана языках VHDL.

Таблица 2.1.1 – Листинг реализации Генераторана языке VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Gen is
         port(
               Y : in STD_LOGIC
         );
end Gen;
architecture Behavioral of Gen is
begin
end Behavioral;
```

В таблице2.1.2 показан листинг проверкиГенераторана языках VHDL. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 2.1.2 – Листинг проверкиГенераторана языке VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY gen_tb IS
END gen_tb;
ARCHITECTURE behavior OF gen_tb IS
    COMPONENT Gen
    PORT(
         Y : IN  std_logic
        );
    END COMPONENT;
   signal q : std_logic := '0';
BEGIN
   changing: Gen
   PORT MAP (
          Y => q
        );
   stim_proc: process(q)
begin
   q <= not q after 10 ns;
   end process;

END Behavior;
```

В рисунке 2.1.1 показана временная диаграммаработы Генератора VHDL.

Рисунок 2.1.1 Временная диаграмма работы Генератора VHDL

Из диаграммы видно, что генератор генерирует тактовый сигнал 50МГц.

2.2 Счетчик Verilog

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на двухступенчатыхD-триггерах, T-триггерах и JK-триггерах.

Счётчики используются для построения схем таймеров или для выборки инструкций из ПЗУ в микропроцессорах.

В таблице 2.2.1 показан листинг реализации Счетчика на языках Verilog.

Таблица 2.2.1 – Листинг реализации Счетчика на языке Verilog

```
module schetchik(reset, clk, count);
parameter n=7;
input reset, clk;
output [n-1:0] count;
reg [n-1:0] count;
always @(posedge reset or posedge clk)
begin
        if (reset==1) begin
              count=0;
        end
        else begin
              count=count+1;
        end
   end
endmodule
```

В таблице 2.2.2 показан листинг проверки Счетчика на языках Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 2.2.2 – Листинг проверкиСчетчика на языке Verilog

```
module schetchik_tb;
   // Inputs
   reg reset;
   reg clk;
   // Outputs
   wire [6:0] count;
   // Instantiate the Unit Under Test (UUT)
   schetchik uut (
         .reset(reset),
         .clk(clk),
         .count(count)
   );
   initial begin
```

Продолжение таблицы 2.2.2

```
                        // Initialize Inputs
                        reset = 1;
                        clk = 0;

                        // Wait 100 ns for global reset to finish
                        #100;
                        // Add stimulus here
                    end
                            always begin #5 clk = ~clk; end
                            always begin #10 reset=0; #1000 reset = 1;
            end
                endmodule
```

В рисунке 2.2.1 показана временная диаграммаработы Счетчика.

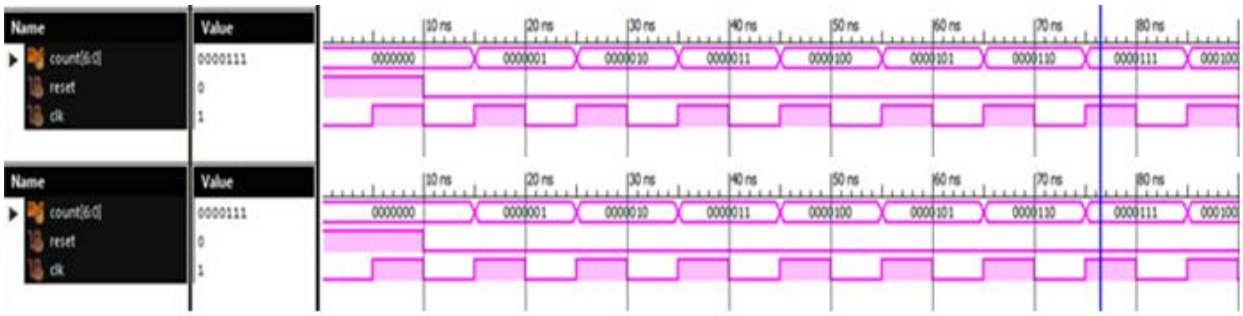


Рисунок 2.2.1 Временные диаграммы работы Счетчика Verilog

При поступлении первого счетного импульса по его спаду первый триггер переходит в состояние Q1 = 1, т.е. в счетчике записан цифровой код 0001. По окончании второго счетного импульса первый триггер переходит в состояние «0», а второй переключается в состояние «1». В счетчике записывается число 2 с кодом 0010. Так далее счетчик идет до модуля счета Счетчика.

2.3 Счетчик 100 Verilog

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на двухступенчатыхD-триггерах, Т-триггерах и JK-триггерах. Данный счетчик имеет модуль счета 100.

В таблице 2.3.1 показан листинг реализации Счетчика 100 на языках Verilog.

Таблица 2.3.1 – Листинг реализации Счетчика 100 на языке Verilog

```
                `timescale 1ns / 1ps

                module counterto100(clk, count);
                input clk;
                output [6:0] count;
                reg [6:0] count;
                initial count=0;
                always @(posedge clk)

                begin
                        count=count+1;
```

Продолжение таблицы 2.3.1

```
            if (count==7'b1100011) begin
            count=0;
      end
      end
endmodule
```

В таблице 2.3.2 показан листинг проверки Счетчика 100 на языках Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 2.3.2 – Листинг проверкиСчетчика 100 на языке Verilog

```
`timescale 1ns / 1ps

module counterto100tb;

   // Inputs
   reg clk;

   // Outputs
   wire [6:0] count;

   // Instantiate the Unit Under Test (UUT)
         counterto100 uut (
         .clk(clk),
         .count(count)
   );

   initial begin
         // Initialize Inputs
         clk = 0;

         // Wait 100 ns for global reset to finish
         #100;

         // Add stimulus here

   end
    always#5 clk=~clk;
endmodule
```

В рисунке 2.3.1 показана временная диаграммаработы Счетчика.

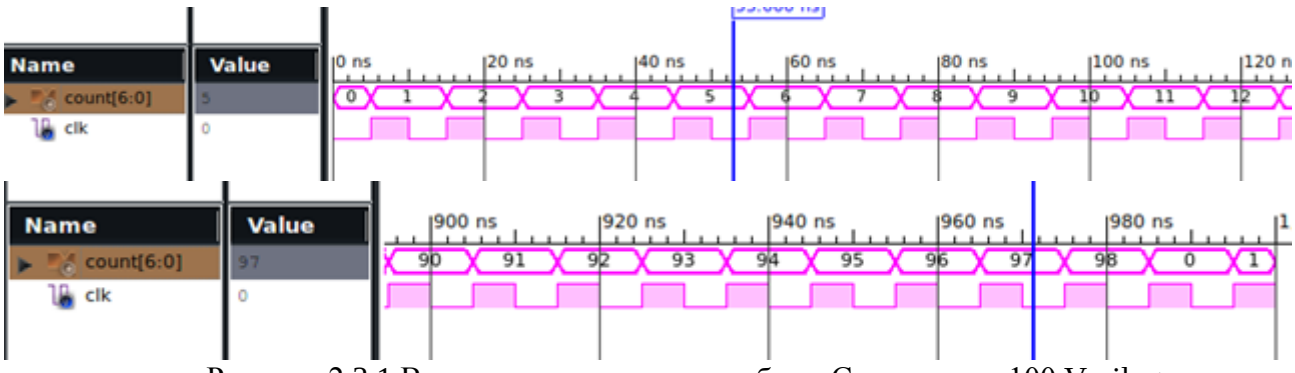


Рисунок 2.3.1 Временные диаграммы работы Счетчика до 100 Verilog

При поступлении первого счетного импульса по его спаду первый триггер переходит в состояние Q1 = 1, т.е. в счетчике записан цифровой код 0001. По окончании второго счетного импульса первый триггер переходит в состояние «0», а второй

переключается в состояние «1». В счетчике записывается число 2 с кодом 0010. Так далее счетчик идет до модуля счета 100. Сигнал останавливается на 99, потому что счет идет с 0.

2.4 Счетчик 128 VHDL

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на двухступенчатыхD-триггерах, T-триггерах и JK-триггерах. Данный счетчик имеет модуль счета 128.

В таблице 2.4.1 показан листинг реализации Счетчика 128на языкеVHDL.

Таблица 2.4.1 – Листинг реализации Счетчика 128на языкеVHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity cnter is

port (
          clk: in std_logic;
          cnt: out std_logic_vector(7 downto 0)
          );

end cnter;

architecture Behavioral of cnter is
          signal  cnt_signal:  std_logic_vector(7  downto  0)  :=
(others => '0');
begin

    cnt <= cnt_signal;

    process(clk) is
begin
          if rising_edge(clk) then

                if (cnt_signal = b"11111111") then


                      cnt_signal <= (others => '0');
                else
                      cnt_signal <= cnt_signal + '1';
                end if;
          end if;
    end process;
end Behavioral;
`timescale 1ns / 1ps
module TB_cnter;
```

Продолжение таблицы 2.4.1

```
                // Inputs
                reg clk;

                // Outputs
                wire [7:0] cnt;

                // Instantiate the Unit Under Test (UUT)
                cnter uut (
                      .clk(clk),
                      .cnt(cnt)
                );

                initial begin
                      // Initialize Inputs
                      clk = 0;

                      // Wait 100 ns for global reset to finish
                      #100;

                      // Add stimulus here

                end

                always
                begin
                      #5 clk=~clk;
                end;

          endmodule
```

В рисунке 2.4.1 показана временная диаграммаработы Счетчика 128 VHDL.

Рисунок 2.4.1 Временная диаграмма работы Счетчика 128 VHDL

При поступлении первого счетного импульса по его спаду первый триггер переходит в состояние Q1 = 1, т.е. в счетчике записан цифровой код 0001. По окончании второго счетного импульса первый триггер переходит в состояние «0», а второй переключается в состояние «1». В счетчике записывается число 2 с кодом 0010. Так далее счетчик идет до модуля счета 128. Сигнал останавливается на 127, потому что счет идет с 0.

2.5 Счетчик 70 Verilog

Счётчик числа импульсов — устройство, на выходах которого получается двоичный (двоично-десятичный) код, определяемый числом поступивших импульсов. Счётчики могут строиться на двухступенчатыхD-триггерах, Т-триггерах и JK-триггерах. Данный счетчик имеет модуль счета 70.

В таблице 2.5.1 показан листинг реализации Счетчика 70 на языкеVerilog.

Таблица 2.5.1 – Листинг реализации Счетчика 70 на языкеVerilog.

```
module Counter_70(

   input clk;
   output reg out_put,
output reg [6:0] count = 0
);


always @(posedge clk)*

begin
count = count + 1;
if (count == 70) begin out_put = 1; count = 0;end
else begin out_put = 0; end
end
endmodule
```

В рисунке 2.5.1 показана временная диаграммаработы Счетчика 70 Verilog

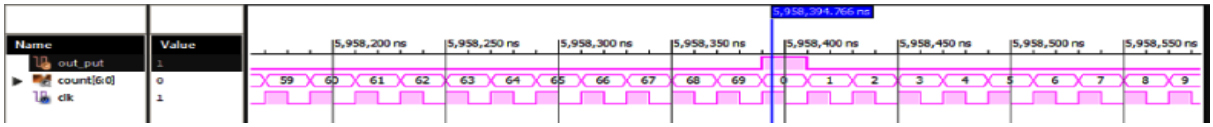


Рисунок 2.5.1 Временная диаграмма работыСчетчика70 Verilog

При поступлении первого счетного импульса по его спаду первый триггер переходит в состояние Q1 = 1, т.е. в счетчике записан цифровой код 0001. По окончании второго счетного импульса первый триггер переходит в состояние «0», а второй переключается в состояние «1». В счетчике записывается число 2 с кодом 0010. Так далее счетчик идет до модуля счета 70. Сигнал останавливается на 69, потому что счет идет с 0.

3 РЕГИСТРЫ

3.1 Параллельный регистр

Регистром называется последовательное или параллельное соединение триггеров. Они обычно строятся на основе D триггеров. При этом для построения регистров могут использоваться как динамические, так и статические D-триггеры.

Параллельный регистр служит для запоминания многоразрядного двоичного (или недвоичного) слова. Количество триггеров, входящее в состав параллельного регистра, определяет его разрядность.

В таблицах 3.1.1 и 3.1.2 показан листинг реализации Параллельного регистрана языках VHDLи Verilog.

Таблица 3.1.1 – Листинг реализации Параллельного регистрана языках VHDL

```
entityprim_f is
   generic (size: integer:=32);
   port (rout:outstd_logic_vector
               (size-1 downto 0);
               rin:instd_logic_vector
               (size-1 downto 0);
               en:instd_logic);
endprim_f;

architecture Behavioral of prim_f is

begin

   process (en,rin)
         begin
               if en='1' then
                     rout<= rin after 1 ns;
               end if;
         end process;
end Behavioral;
```

Таблица 3.1.2 – Листинг реализации Параллельного регистрана языках Verilog

```
module Ex4(r_out, r_in, en);
parameter size = 32;
input en;
input [size-1:0] r_in;
output [size-1:0] r_out;
reg [size-1:0] r_out;

always@(en or r_in)
begin
if (en)
r_out<= #1 r_in;
end
endmodule
```

В таблицах 3.1.3 и 3.1.4 показан листинг проверки Параллельного регистрана языках VHDLи Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 3.1.3 – Листинг проверкиПараллельного регистрана языке VHDL

```
module Ex5_tb;

   // Inputs
   reg [31:0] rin;
   reg en;

   // Outputs
   wire [31:0] rout;

   // Instantiate the Unit Under Test (UUT)
   Ex5 uut (
         .rout(rout),
         .rin(rin),
         .en(en)
   );

   initial begin
         // Initialize Inputs
         rin = 0;
         en = 0;

         // Wait 100 ns for global reset to finish
         #100;

         // Add stimulus here

   end
         always begin #10 rin = 1; #10 rin = 0; end;
         always begin #5 en = 1; #5 en = 0; end;
endmodule
```

Таблица 3.1.4 – Листинг проверкиПараллельного регистрана языке Verilog

```
module Ex5_tb;

   // Inputs
   reg [31:0] r_in;
   regen;

   // Outputs
   wire [31:0] r_out;

   // Instantiate the Unit Under Test (UUT)
   Ex5 uut (
         .r_out(r_out),
         .r_in(r_in),
         .en(en)
   );

   initial begin
         // Initialize Inputs
         r_in = 0;
         en = 0;

         // Wait 100 ns for global reset to finish
         #100;

         // Add stimulus here

   end
         always begin #10 r_in = 1; #10 r_in = 0; end;
         always begin #5 en = 1; #5 en = 0; end;
```

```
endmodule
```

В рисунке 3.1.1 показана временная диаграммаработы Параллельного регистра.

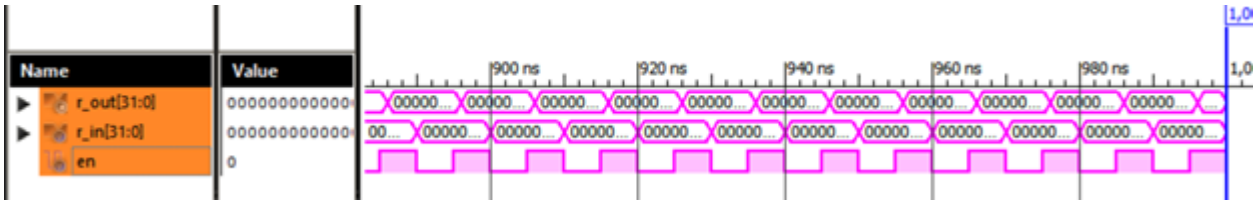


Рисунок 3.1.1 Временная диаграмма работы Параллельного регистра

При записи информации в параллельный регистр все биты (двоичные разряды) должны быть записаны одновременно. Поэтому все тактовые входы триггеров, входящих в состав регистра, объединяются параллельно.

3.2 Сдвиговый параллельно-параллельный регистр VHDL

Параллельный сдвиговый регистр аналогичен регистру сдвига последовательного ввода, последовательного вывода, в котором он перемещает данные во внутренние элементы хранения и сдвигает данные на выводе последовательного вывода, вывода данных, вывода.

В таблице 3.2.1 показан листинг реализации Сдвиговый параллельно-параллельный регистрна языке VHDL.

Таблица 3.2.1 – Листинг реализации Сдвигово параллельно-параллельногорегистрана языкеVHDL.

```
                    library IEEE;
                                use IEEE.STD_LOGIC_1164.ALL;
                                entity sdvig is
                                  generic (size:positive:=8);
                                  port  (rout:  out  std_logic_vector  (size-1
          downto 0);
                                                rin:  in  std_logic_vector  (size-1
          downto 0);
                                                en, clr_n, clk: in std_logic);
                                end sdvig;
    architecture Behavioral of sdvig is
    begin
    process(clk, clr_n) begin
          if(clr_n = '0') then
                rout<= (others=>'0');
          else if (clk'event and clk='1') then
                              if (en='1') then
                                    rout<= rin after 1 ns;
                              end if;
                        end if;
          end if;
    end process;
    end Behavioral;
```

В таблице 3.2.2 показан листинг проверки Сдвигово параллельно-параллельного регистра на языке VHDL. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 3.2.2 – Листинг проверкиСдвигово параллельно-параллельного регистра на языке VHDL

```
module sdvig_tb;
   // Inputs   reg [7:0] rin;
   reg en;
   reg clr_n;
   reg clk;
   // Outputs
   wire [7:0] rout;
   // Instantiate the Unit Under Test (UUT)
   sdvig uut (
         .rout(rout),
         .rin(rin),
         .en(en),
         .clr_n(clr_n),
         .clk(clk)
   );
   initial begin
         // Initialize Inputs
         rin = 0;
         en = 0;
         clr_n = 0;
         clk = 0;
   // Wait 100 ns for global reset to finish
         #100;
         // Add stimulus here
end
   always begin #10 clk=~clk; end
   always begin #20 rin=rin+1; end
   always begin #200 en=0; #200 en=1; end
   always begin #100 clr_n=0; #100 clr_n=1; end
endmodule
```

В рисунке 3.2.1 показана временная диаграммаработы Сдвигового параллельно-параллельный регистр VHDL.

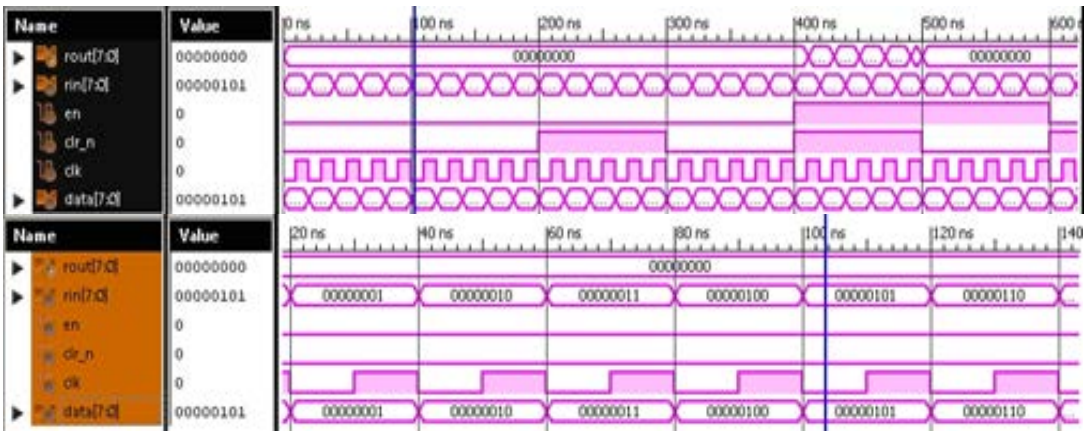


Рисунок 3.2.1 Временная диаграмма работы Сдвигового параллельно-параллельный регистр VHDL

Из диаграммы видно, что элемент работает как параллельно-параллельный регистр. Представленный выше модуль имеет четыре входа и один выход. Порт rin предназначен для ввода данных, en – вход разрешения работы, clk – вход сигнала тактирования, clr_n - вход сброса, выход rout предназначен для вывода данных.

И с каждым спадом такта CLKчисло, записываемое в память увеличивается на 1 бит.

3.3 Сдвиговый регистр Verilog

Регистр называется сдвиговым, потому что при добавлении каждого нового бита в него, мы как бы сдвигаем все остальные в сторону. Вспомним, что один бит позволяет нам хранить ноль или единицу, истину или ложь.

В таблице 3.3.1 показан листинг реализации Сдвиговый регистрна языке Verilog.

Таблица 3.3.1 – Листинг реализации Сдвиговый регистрана языке Verilog.

```
module shiftreg(rout, rin, en, clr_n, clk);
   parameter size = 8;
   output [size-1:0] rout;
   reg [size-1:0] rout;
   input [size-1:0] rin;
   input en, clr_n, clk;

   always @(posedge clk or negedge clr_n)

   begin
         if(!clr_n)
               rout <= 0;
         else
               if(en)
                     rout <= #(1) rin;
   end
endmodule
```

В таблице 3.3.2 показан листинг проверки Сдвиговый регистрна языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 3.3.2 – Листинг проверкиСдвиговый регистрна языке Verilog

```
module shiftreg_tb;
   // Inputs
   reg [7:0] rin;
   reg en;
   reg clr_n;
   reg clk;
   reg [7:0] data;
   // Outputs
   wire [7:0] rout;

   // Instantiate the Unit Under Test (UUT)
   shiftreg uut (
         .rout(rout),
```

```
                .rin(rin),
                .en(en),
                .clr_n(clr_n),
                .clk(clk)
        );
        initial begin
                // Initialize Inputs
```

Продолжение таблицы 3.3.2

```
                rin = 0;
                en = 0;
                clr_n = 0;
                clk = 0;

                data = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

        end

        always begin #10 clk=~clk; end
        always begin #20 data=data+1; end

        always begin #20 rin = data; end
        always begin #200 en=0; #200 en=1; end
        always begin #100 clr_n=0; #100 clr_n=1; end
endmodule
```

В рисунке 3.3.1 показана временная диаграммаработы Сдвиговый регистрна языке Verilog.

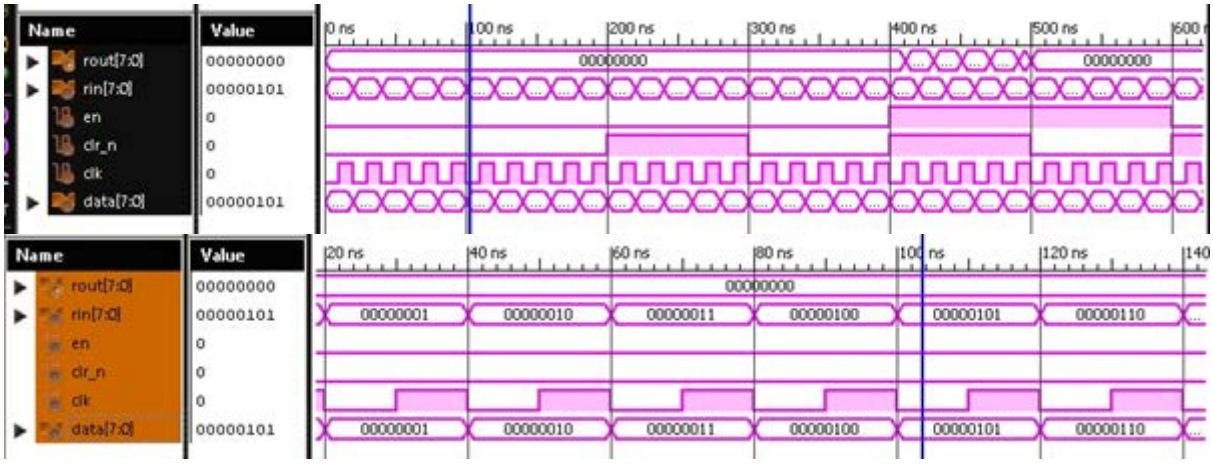


Рисунок 3.3.1 Временная диаграмма работы Сдвигового регистра Verilog

Представленный выше модуль имеет четыре входа и один выход. Порт rin предназначен для ввода данных, en – вход разрешения работы, clk – вход сигнала тактирования, clr_n - вход сброса, выход rout предназначен для вывода данных.

И с каждым спадом такта CLKчисло, записываемое в память увеличивается на 1 бит.

3.4 Двухвходовый сдвиговый регистр с разрешающим входом Verilog

Регистр называется сдвиговым, потому что при добавлении каждого нового бита в него, мы как бы сдвигаем все остальные в сторону. Вспомним, что один бит позволяет нам хранить ноль или единицу, истину или ложь. Данный элемент имеет два входа и разрешающий счет вход. Параметр size в элементе предназначен для масштабирования модуля до больших размеров.

В таблице 3.4.1 показан листинг реализации Двухвходового сдвигового регистра с разрешающим входом на языке Verilog.

Таблица 3.4.1 – Листинг реализации Двухвходового сдвигового регистра с разрешающим входом на языке Verilog.

```
module regenn(rout, rin, en, clr_n, clk);
   parameter size = 8;
   output [size-1:0] rout;
   reg [size-1:0] rout;
   input [size-1:0] rin;
   input en, clr_n, clk;

   always @(posedge clk or negedge clr_n)

   begin
         if(!clr_n)
               rout <= 0;
         else
               if(en)
                     rout <= #(1) rin;
   end
endmodule
```

В ходе работы устройства происходит включение и выключение сигнала на порту тактирования. Для проверки работоспособности используется дополнительно созданный в тестовой программе регистр данных размером 8 бит, значения которого инкрементируются каждый период.

На вход rin поступает значение от регистра данных. Входы разрешения работы и сброса включаются и выключаются после длительного промежутка для проверки работоспособности.

В таблице 3.4.2 показан листинг проверки Двухвходового сдвигового регистра с разрешающим входом на языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 3.4.2 – Листинг проверкиДвухвходового сдвигового регистра с разрешающим входом на языке Verilog.

```
reg [7:0] data;

always begin #10 clk=~clk; end
always begin #20 data=data+1; end

always begin #20 rin = data; end
always begin #200 en=0; #200 en=1; end
always begin #100 clr_n=0; #100 clr_n=1; end
```

В рисунке 3.4.1 показана временная диаграммаработы Двухвходового сдвигового регистра с разрешающим входом на языке Verilog.

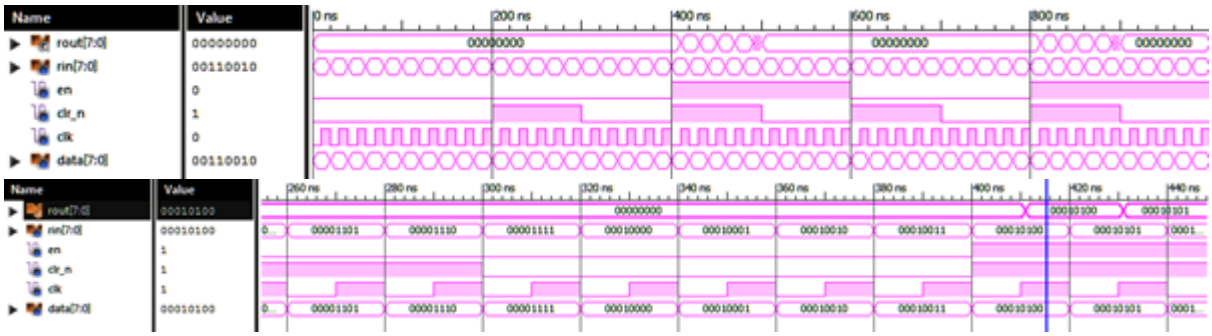


Рисунок 3.4.1 Временная диаграмма работы Двухвходового сдвигового регистра с разрешающим входом Verilog

Представленный выше модуль имеет четыре входа и один выход. Порт rin предназначен для ввода данных, en – вход разрешения работы, clk – вход сигнала тактирования, clr_n - вход сброса, выход rout предназначен для вывода данных.

И с каждным спадом такта CLK число, записываемое в память увеличивается на 1 бит.

3.5 Сдвиговый последовательно-последовательный регистр VHDL

Последовательный регистр (регистр сдвига или сдвиговый регистр) обычно служит для преобразования последовательного кода в параллельный и наоборот. Применение последовательного кода связано с необходимостью передачи большого количества двоичной информации по ограниченному количеству соединительных линий.

В таблице 3.5.1 показан листинг реализации Сдвигового последовательно-последовательного регистрана языке Verilog.

Таблица 3.5.1 – Листинг реализации Сдвигового последовательно-последовательного регистрана языке Verilog.

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

```
                -- Uncomment the following library declaration if using
                -- arithmetic functions with Signed or Unsigned values
                --use IEEE.NUMERIC_STD.ALL;

                --   Uncomment   the   following   library   declaration   if
            instantiating
                -- any Xilinx primitives in this code.
                --library UNISIM;
                --use UNISIM.VComponents.all;

                entity shift_final is
                    port(
                                        C, SI : in std_logic;
```

Продолжение таблицы 3.5.1

```
                                        SO : out std_logic
                                );
                end shift_final;

                architecture archi of shift_final is
                    signal tmp: std_logic_vector(7 downto 0);
                begin
                    process (C)
                        begin
                            if rising_edge (C) then
                                for i in 0 to 6 loop
                                    tmp(i+1) <= tmp(i);
                                end loop;
                                tmp(0) <= SI;
                            end if;
                    end process;
                    SO <= tmp(7);

                end archi;
```

В таблице 3.5.2 показан листинг проверкиСдвигового последовательно-последовательного регистрана языке Verilog. Он необходим для проверки корректности работы смоделированной схемы.

Таблица 3.5.2 – Листинг проверкиСдвигового последовательно-последовательного регистра на языке Verilog.

```
                module shift_final_tb(
                    );

                   // Inputs
                   reg C;
                   reg SI;

                   // Outputs
                   wire SO;

                   // Instantiate the Unit Under Test (UUT)
                   shift_final uut (
                         .C(C),
                         .SI(SI),
```

```
                .SO(SO)
        );
        initial begin
                // Initialize Inputs
                C = 0;
                SI = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here
end
```

Продолжение таблицы 3.5.2

```
        end

        always
                begin
                        #20 C=~C;
                end

        always
                begin
                        #40 SI=1;
                        #40 SI=0;
                        #40 SI=0;
                        #40 SI=1;
                end
endmodule
```

В рисунке 3.5.1 показана временная диаграммаработыСдвигового последовательно-последовательного регистра на языке Verilog.

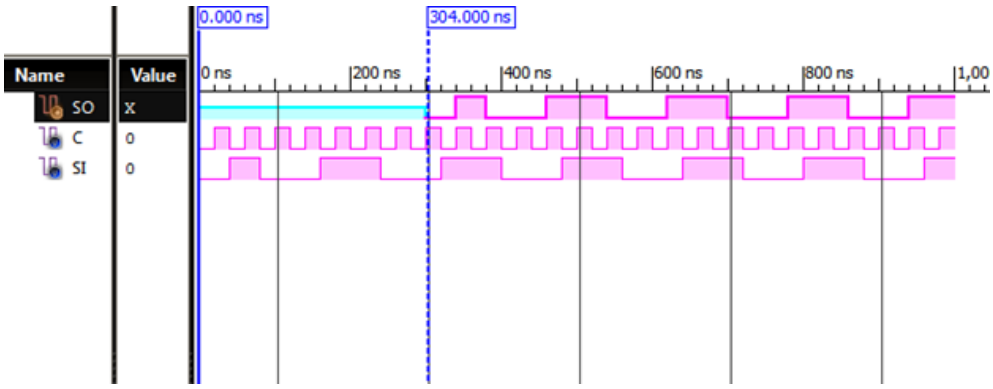


Рисунок 3.5.1Временная диаграмма работы Сдвигового последовательно-последовательного регистра VHDL

В моментпоявление синхронизирующего импульса на входе C вызывает сдвиг информации в регистре на один разряд вправо. Если до этого момента в регистре было число 0000, то в результате сдвига в первом, втором, третьем разрядах сохранится значение 0, в четвертый разряд будет со входа принято значение 1. Таким образом, в регистре возникает число 1000. В следующий момент появления следующего синхронизирующего

импульса процессы сдвига и приема очередного разряда вводимого числа приводят регистр в состояние 11002. Далее,в регистре образуется число 0110 и, наконец, в следующий момент число 1011. Поданное на вход число оказывается зафиксированным в регистре.

4 МУЛЬТИПЛЕКСОРЫ

4.1 Мультиплексор 2-1

Представленный ниже модуль имеет три входа и один выход. Порты a и b - информационные и предназначены для ввода информации в устройство, порт y используется для вывода информации от устройства, а порт c необходим для определения используемого в данный момент информационного входа. При поступлении единицы на выход передается значение с порта a, при поступлении нуля - значение с порта c.

В таблице 4.1.1 показан листинг реализации Мультиплексор 2-1 на языке VHDL.

Таблица 4.1.1 – Листинг реализации Мультиплексор 2-1 на языке VHDL.

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

entity mux2x1 is
   port(a,b:std_logic; c:in std_logic; y:out std_logic);
end mux2x1;

architecture Behavioral of mux2x1 is
   constant one: std_logic:='1';
begin
   process (a,b,c)
   begin
         if c = one then
               y <= a;
         else
               y<=b;
         end if;
   end process;
end Behavioral;

`timescale 1ns / 1ps
module mux2x1tb;

   // Inputs
   reg a;
   reg b;
   reg c;

   // Outputs
   wire y;

   // Instantiate the Unit Under Test (UUT)
   mux2x1 uut (
         .a(a),
         .b(b),
         .c(c),
         .y(y)
   );
   initial begin
```

Продолжение таблицы 4.1.1

```
                    // Initialize Inputs
                    a = 0;
                    b = 1;
                    c = 0;
                    // Wait 100 ns for global reset to finish
                    #100;
                    // Add stimulus here
                end
always begin #10 c=~c; end
always begin #30 a=~a; end
always begin #50 b=~b; end
endmodule
```

В таблице 4.1.2 показан листинг реализации Мультиплексор 2-1 на языке Verilog.

Таблица 4.1.2 – Листинг реализации Мультиплексор 2-1 на языке Verilog.

```
module primer2(a,b,c,y);//mux_2_1_S
input c;
parameter n=1;
input [n-1:0] a,b;
output [n-1:0] y;

assign y=(c==1'b1)? a:b;

endmodule
```

В таблице 4.1.3 показан листинг проверки Мультиплексор 2-1 на языке Verilog.

Таблица 4.1.3 – Листинг проверким Мультиплексор 2-1 на языке Verilog.

```
moduleprimer2_t;

    // Inputs
    reg [0:0] a;
    reg [0:0] b;
    reg c;

    // Outputs
    wire [0:0] y;

    // Instantiate the Unit Under Test (UUT)
    primer2uut (
        .a(a),
        .b(b),
        .c(c),
        .y(y)
    );

    initial begin
        // Initialize Inputs
        a = 0;
        b = 0;
        c = 0;

        // Wait 100 ns for global reset to finish
        #100;
```

Продолжение таблицы 4.1.2

```
        // Add stimulus here

    end
always begin #10 a=1; #10 a=0 ; end
always begin #20 b=1; #20 b=0 ; end
always begin #5 c = ~c; end

endmodule
```

В рисунке 4.1.1 показана временная диаграммаработыМультиплексора 2-1.

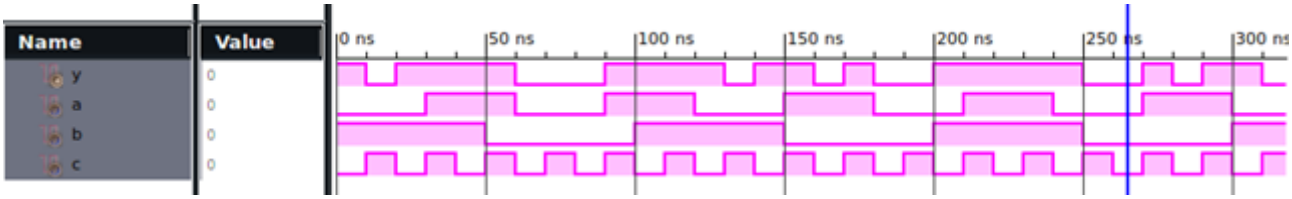


Рисунок 4.1.1 Временная диаграмма работы Мультиплексора 2-1

4.2 Мультиплексор 2-1 с поразрядным кодированием селектора на языке Verilog

Мультиплексором (от английского слова multiplex - многократный) называется комбинационный узел, способный коммутировать (передавать) информацию с нескольких входов на один выход. С помощью мультиплексора осуществляется временное разделение информации.

В таблице 4.2.1 показан листинг реализации Мультиплексор 2-1 с поразрядным кодированием селектора на языке Verilog.

Таблица 4.2.1 – Листинг реализации Мультиплексор 2-1 с поразрядным кодированием селектора на языке Verilog

```
`timescale 1ns / 1ps
module mux_2_1(a, b, c, y);
input [1:0] a,b;
    input c;
    output [1:0] y;
    reg [1:0] y;
always @(a or b or c)
    begin
        if (c==1'b1)
            y<=a;
        else
            y<=b;
    end//always
endmodule//MUX_2_1


module mux_tb;
```

```
                    // Inputs
                    reg [1:0] a;
```

Продолжение таблицы 4.2.1

```
                    reg [1:0] b;
                    reg c;

                    // Outputs
                    wire [1:0] y;
                    // Instantiate the Unit Under Test (UUT)
                          mux_2_1 uut (
                          .a(a),
                          .b(b),
                          .c(c),
                          .y(y)
                          );
                    initial begin
                          // Initialize Inputs
                          a = 0;
                          b = 0;
                          c = 0;
                          // Wait 100 ns for global reset to finish
                          #100;
                          // Add stimulus here
                    end
                          always begin #40 a=1; #40 a=0; end
                          always begin #80 b=1; #80 b=0; end
                          always begin #160 c=1; #160 c=0; end
               endmodule
```

В рисунке 4.2.1 показана временная диаграммаработы Мультиплексор 2-1 с поразрядным кодированием селектора на языке Verilog

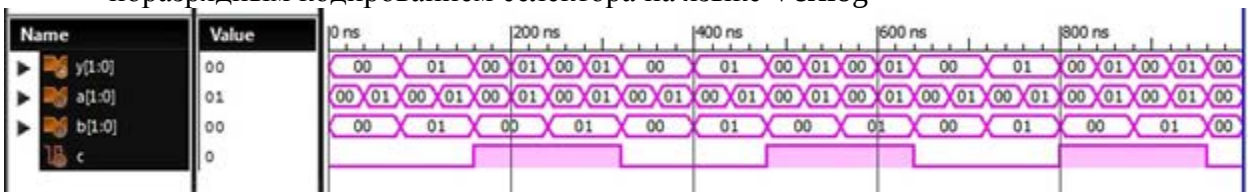


Рисунок 4.2.1 Временная диаграмма работы Мультиплексора 2-1 с поразрядным кодированием селектора (onehot) на языке Verilog

Выводы: по временным диаграммам можно сделать вывод, что симуляционная программа была написана верно и подтверждает правильность исходного кода, так как полностью отображает режимы работы мультиплексора 2-1 с поразрядным кодированием селектора.

4.3 Мультиплексор 3-1 Verilog

Данный вид мультиплексоров выбирает один сигнал из 3 входящий. В схеме такого мультиплексора обычно 3 входа и один выход. А также присутствуют два управляющих сигнала, чтобы правильно выбрать что будет поступать на выход.

В таблице 4.3.1 показан листинг реализации Мультиплексор 3-1 на языке Verilog.

Таблица 4.3.1 – Листинг реализации Мультиплексор 3-1 на языке Verilog

```
`timescale 1ns / 1ps
module mux3x1(d0,d1,d2,sel,out);
   input d0,d1,d2;
   input [1:0] sel;
   output reg out;
   always@(sel)
   begin
         case(sel)
               3'b000:out=d0;
               3'b001:out=d1;
               3'b010:out=d2;

         endcase
   end
endmodule

`timescale 1ns / 1ps
module mux3x1test;
   // Inputs
   reg d0;
   reg d1;
   reg d2;
   reg [1:0] sel;
   // Outputs
   wire out;
   // Instantiate the Unit Under Test (UUT)
   mux3x1 uut (
         .d0(d0),
         .d1(d1),
         .d2(d2),
         .sel(sel),
         .out(out)
   );

   initial begin
         // Initialize Inputs
         d0 = 1;
         d1 = 0;
         d2 = 1;
         sel =0;
         // Wait 100 ns for global reset to finish
         #100;
         // Add stimulus here
   end
always begin #50 sel=sel+1; if (sel==3) begin sel=0; end
   end
   endmodule
```

В рисунке 4.3.1 показана временная диаграммаработы Мультиплексора 3-1 на языке Verilog

Рисунок 4.3.1 Временная диаграмма работы Мультиплексора 3-1 на языкеVerilog

4.4 Мультиплексор 3-1 с поразрядным кодированием селекторана языке Verilog

Данный вид мультиплексоров выбирает один сигнал из 4 входящий. В схеме такого мультиплексора обычно 4 входа и один выход. А так же присутствуют два управляющих сигнала, чтобы правильно выбрать что будет поступать на выход.

В таблице 4.4.1 показан листинг реализации Мультиплексор 3-1 с поразрядным кодированием селектора на языке Verilog

Таблица 4.4.1 – Листинг реализации Мультиплексор 3-1 с поразрядным кодированием селектора на языке Verilog

```
module MUX31(o, d0, d1, d2, s0, s1, s2);
parameter size = 1, check = 1;
output [1:0] o; reg [1:0] o;
input [1:0] d0, d1, d2;
input s0, s1, s2;
reg [1:0] tmp0;
reg [1:0] tmp1;
reg [1:0] tmp2;
always @(d0 or d1 or s0 or s1 or s2)
begin
   if(s0) tmp0 = d0;
   else tmp2 = 0;
   if(s1) tmp1 = d1;
   else tmp1 = 0;
   if(s2) tmp2 = d2;
   else tmp2 = 0;
end

wire onehotcheck;
assign onehotcheck=(s0 & s1) | (s0 & s1) | (s1 & s2);

always @(tmp0 or tmp1 or tmp2 or onehotcheck)
begin
   if (check == 1 &&onehotcheck == 1)
   o <= {size{1'bx}};
   else
   o <= tmp0 | tmp1 | tmp2;
end
endmodule
```

В таблице 4.4.2 показан листинг проверки Мультиплексор 3-1 с поразрядным кодированием селектора на языке Verilog

Таблица 4.4.2 – Листинг проверки Мультиплексор 3-1 с поразрядным кодированием селектора на языке Verilog.

```
module MUX31_tb;

    // Inputs
    reg [0:0] d0;
    reg [0:0] d1;
    reg [0:0] d2;
    reg s0;
    reg s1;
    reg s2;

    // Outputs
    wire [0:0] o;

    // Instantiate the Unit Under Test (UUT)
    MUX31 uut (
        .o(o),
        .d0(d0),
        .d1(d1),
        .d2(d2),
        .s0(s0),
        .s1(s1),
        .s2(s2)
    );

    initial begin
        // Initialize Inputs
        d0 = 0;
        d1 = 0;
        d2 = 0;
        s0 = 0;
        s1 = 0;
        s2 = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end
        always begin #20 s0=0; s1=0; s2=1; #30 s0=0; s1=1; s0=0; end;
            end;

endmodule
```

В рисунке 4.3.1 показана временная диаграммаработы Мультиплексора 3-1 с поразрядным кодированием селектора (onehot) на языке Verilog

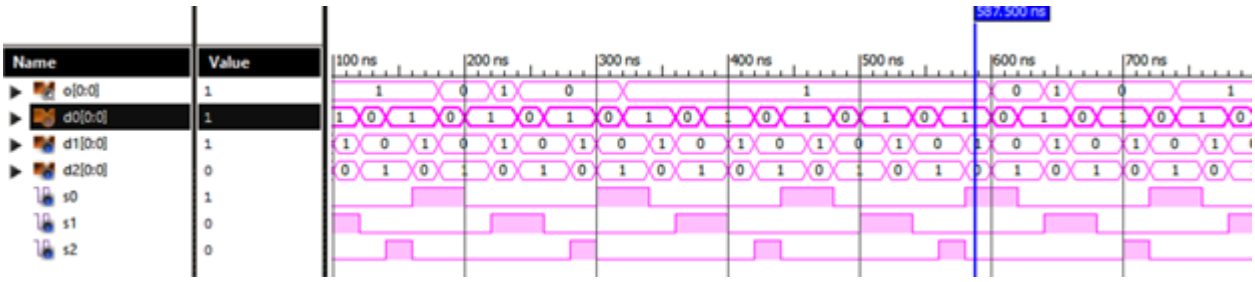


Рисунок 4.4.1 Временная диаграмма работы Мультиплексора 3-1 с поразрядным кодированием селектора (onehot) на языке Verilog

4.5 Мультиплексор 8-1

Данный вид мультиплексоров выбирает один сигнал из 3 входящий. В схеме такого мультиплексора обычно 8 входов и один выход. А так же присутствуют три управляющих сигнала, чтобы правильно выбрать что будет поступать на выход.

В таблице 4.5.1 показан листинг реализации Мультиплексор 8-1 на языке VHDL.

Таблица 4.5.1 – Листинг реализации Мультиплексор 8-1 на языке VHDL

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

entity multiplexer is
   Port( x:inSTD_LOGIC_VECTOR(7downto0);
   sel:inSTD_LOGIC_VECTOR(2downto0);
   y :outSTD_LOGIC);
end multiplexer;
architecture Behavioral of multiplexer is
  begin
         process(x,sel)
  begin
                caseselis
                 when"000"=>y<=x(0);
               when"001"=>y<=x(1);
               when"010"=>y<=x(2);
               when"011"=>y<=x(3);
               when"100"=>y<=x(4);
               when"101"=>y<=x(5);
               when"110"=>y<=x(6);
               when"111"=>y<=x(7);
               whenothers=>null;
         endcase;
         endprocess;
endBehavioral;
```

В таблице 4.5.2 показан листинг проверки Мультиплексор 8-1 на языке VHDL

Таблица 4.5.2 – Листинг проверки Мультиплексор 8-1 на языке VHDL

```
LIBRARYieee;
USE ieee.std_logic_1164.ALL;
```

```
USEieee.std_logic_unsigned.all;
USEieee.numeric_std.ALL;
ENTITYtb_multiplexer_vhdIS
ENDtb_multiplexer_vhd;
   ARCHITECTURE behavior OFtb_multiplexer_vhdIS
   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT multiplexer
   PORT(
   x:INstd_logic_vector(7downto0);
   sel:INstd_logic_vector(2downto0);
   y:OUTstd_logic
   );
ENDCOMPONENT;

--Inputs
SIGNALx :std_logic_vector(7downto0):=(others=>'0');
SIGNALsel:std_logic_vector(2downto0):=(others=>'0');

--Outputs
SIGNALy :std_logic;

BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: multiplexer PORTMAP(x => x,
   sel=>sel,
   y => y
   );
   x<="01010101"after10ns;
   sel<="001"after10ns,"010"after20ns,"011"after30ns,"100"a
f
   ter40ns,"101"after50ns,"110"after60ns,"111"after70ns;
END;
```

В рисунке 4.5.1 показана временная диаграммаработы Мультиплексора 8-1 на языке VHDL

Рисунок 4.5.1 Временная диаграмма работы Мультиплексора 8-1 VHDL

В таблице 4.5.3 показан листинг проверки Мультиплексор 8-1 на языке Verilog

Таблица 4.5.3 – Листинг проверки Мультиплексор 8-1 на языке Verilog

```
module mux_8_1(d0,d1,d2,d3,d4,d5,d6,d7,sel,out);
input d0,d1,d2,d3,d4,d5,d6,d7;
input [2:0] sel;
output reg out;
always@(sel)
begin
      case(sel)
            3'b000:out=d0;
            3'b001:out=d1;
            3'b010:out=d2;
```

```
                3'b011:out=d3;
                3'b100:out=d4;
                3'b101:out=d5;
                3'b110:out=d6;
                3'b111:out=d7;
            endcase
        end
    endmodule
```

В таблице 4.5.4 показан листинг проверки Мультиплексор 8-1 на языке Verilog

Таблица 4.5.4 – Листинг проверки Мультиплексор 8-1 на языке Verilog

```
module mux_8_1_tb;
    // Inputs
    reg d0;
    reg d1;
    reg d2;
    reg d3;
    reg d4;
    reg d5;
    reg d6;
    reg d7;
    reg [2:0] sel;
    // Outputs
    wire out;
    // Instantiate the Unit Under Test (UUT)
    mux_8_1 uut (
        .d0(d0),
        .d1(d1),
        .d2(d2),
        .d3(d3),
        .d4(d4),
        .d5(d5),
        .d6(d6),
        .d7(d7),
        .sel(sel),
        .out(out)
    );
    initial begin
        // Initialize Inputs
        d0 = 1;
        d1 = 0;
        d2 = 1;
        d3 = 0;
        d4 = 0;
        d5 = 1;
        d6 = 0;
        d7 = 0;
        sel = 0;
        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
    end
```

```
always begin #50 sel=sel+1; end
endmodule
```

В рисунке 4.5.2 показана временная диаграммаработы Мультиплексора 8-1 языке Verilog.

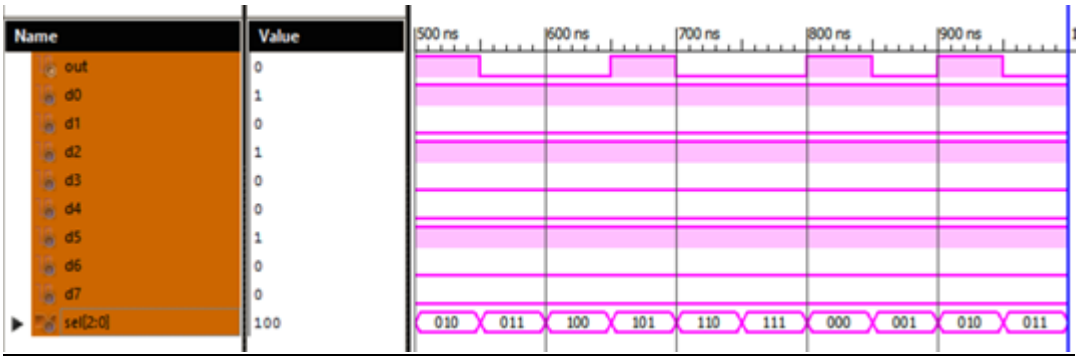


Рисунок 4.5.2 Временная диаграмма работы Мультиплексора 8-1 Verilog

5 ШИФРАТОРЫ/ДЕШИФРАТОРЫ

5.1 Дешифратор 2-4

Дешифраторы – цифровые устройства функционального назначения, предназначенные для распознавания двоичных кодов.

В таблице 5.1.1 показан листинг реализации Дешифратор 2-4 на языке Verilog

Таблица 5.1.1– Листинг реализации Дешифратор 2-4 на языке Verilog

```
moduledeshifr(en, a, sel);
   inputen;
   input [1:0]a;

   output [3:0]sel;
   reg[3:0]sel;

   always @(en or a)

begin
   case({en,a})
               3'b100: sel=4'b0001;
               3'b101: sel=4'b0100;
               3'b110: sel=4'b1000;
               3'b111: sel=4'b0000;
   endcase
end

endmodule
```

В таблице 5.1.2 показан листинг проверки Дешифратор 2-4 на языке Verilog

Таблица 5.1.2– Листинг проверки Дешифратор 2-4 на языке Verilog

```
moduledeshifr_t;

   // Inputs
   regen;
   reg [1:0] a;

   // Outputs
   wire [3:0] sel;

   // Instantiate the Unit Under Test (UUT)
   deshifruut (
         .en(en),
         .a(a),
         .sel(sel)
   );

   initial begin
         // Initialize Inputs
         en = 0;
         a = 0;

         // Wait 100 ns for global reset to finish
```

```
                                #100;

                                // Add stimulus here

                            end

                        always begin #40 en=1; #40 en=0; end
                        always begin #10 a=0; #10 a=1; #10 a=2; #10 a=3; end

                        endmodule
```

В рисунке 5.1.1 показана временная диаграммаработы Дешифратора 2-4 на языке Verilog

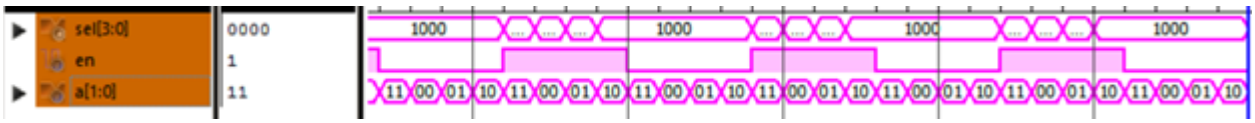


Рисунок 5.1.1 Временная диаграмма работы Дешифратора 2-4 на языке Verilog

В таблице 5.1.3 показан листинг реализации Дешифратор 2-4 на языке VHDL

Таблица 5.1.3– Листинг реализации Дешифратор 2-4 на языке VHDL

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

                        -- Uncomment the following library declaration if using
                        -- arithmetic functions with Signed or Unsigned values
                        --use IEEE.NUMERIC_STD.ALL;

                        -- Uncomment the following library declaration if
                    instantiating
                        -- any Xilinx primitives in this code.
                        --library UNISIM;
                        --use UNISIM.VComponents.all;

                        entity DC2_4 is
                           Port (
                                              i : in  STD_LOGIC_VECTOR (1 downto 0);
                                              y : out  STD_LOGIC_VECTOR (3 downto 0)
                                         );
                        end DC2_4;

                        architecture Behavioral of DC2_4 is

                           begin
                                 process(i)
                                       begin
                                              case i is
                                                    when "11" => y<="0001";
                                                    when "10" => y<="0010";
                                                    when "01" => y<="0100";
                                                    when "00" => y<="1000";
                                                    when others => null;
                                              end case;
```

```
                        end process;

                end Behavioral;

                module DC2_4_tb(
                    );

                    // Inputs
                    reg [1:0] i;

                    // Outputs
                    wire [3:0] y;

                    // Instantiate the Unit Under Test (UUT)
                    DC2_4 uut
                        (
                            .i(i),
                            .y(y)
                        );

                    initial
                        begin
                            // Initialize Inputs
                            i = 0;

                            // Wait 100 ns for global reset to finish
                            #100;

                            // Add stimulus here

                        end

                    always
                        begin
                            #10 i = i + 1;
                        end
                endmodule
```

В рисунке 5.1.2 показана временная диаграммаработы Дешифратора 2-4 на языке VHDL

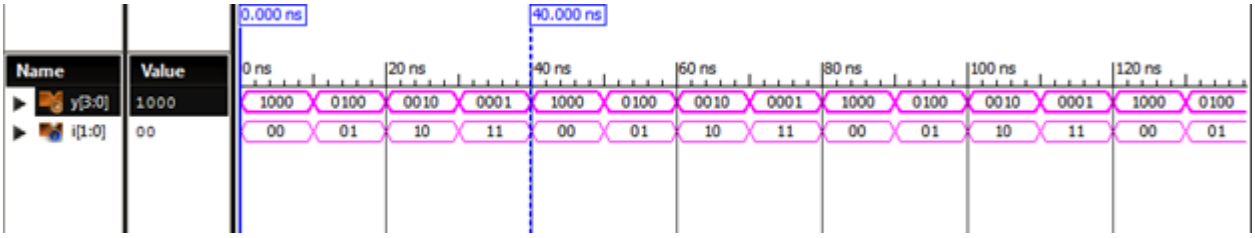


Рисунок 5.1.2 Временная диаграмма работы Дешифратора 2-4 на языке VHDL

5.2 Дешифратор 3-8

Дешифраторы обладают противолположным функционалом мультиплексорам.определенным комбинациям входных сигналов соответствует активное состояние одного из выходов. Дешифраторы преобразуют двоичный или двоично-десятичный код в унитарный код.

В таблице 5.2.1 показан листинг реализации Дешифратор 3-8 на языке Verilog

Таблица 5.2.1– Листинг реализации Дешифратор 3-8 на языке Verilog

```
module decoder3_8_tf;

 // Inputs
 reg[2:0] a;

 // Outputs
 wire[7:0] q;

 // Instantiate the Unit Under Test (UUT)
 decoder3_8 uut(
        .a(a),
        .q(q)
 );

 initial begin
        // Initialize Inputs
        a =0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

 end
     always begin #20 a=a+1;end
endmodule
```

В таблице 5.2.2 показан листинг проверки Дешифратор 3-8 на языке Verilog

Таблица 5.2.2– Листинг проверки Дешифратор 3-8 на языке Verilog

```
LIBRARYieee;
USE ieee.std_logic_1164.ALL;
USEieee.std_logic_unsigned.all;
USEieee.numeric_std.ALL;

ENTITYtb_multiplexer_vhdIS
ENDtb_multiplexer_vhd;

ARCHITECTURE behavior OFtb_multiplexer_vhdIS

  -- Component Declaration for the Unit Under Test (UUT)
  COMPONENT multiplexer
  PORT(
  x:INstd_logic_vector(7downto0);
```

```
    sel:INstd_logic_vector(2downto0);
    y:OUTstd_logic
    );
ENDCOMPONENT;

--Inputs
SIGNALx :std_logic_vector(7downto0):=(others=>'0');
SIGNALsel:std_logic_vector(2downto0):=(others=>'0');

--Outputs
SIGNALy :std_logic;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: multiplexer PORTMAP(x => x,
    sel=>sel,
    y => y
    );
    x<="01010101"after10ns;
    sel<="001"after10ns,
          "010"after20ns,
          "011"after30ns,
          "100"after40ns,
          "101"after50ns,
          "110"after60ns,
          "111"after70ns;
END;
```

В рисунке 5.2.1 показана временная диаграммаработы Дешифратора 3-8 на языке Verilog

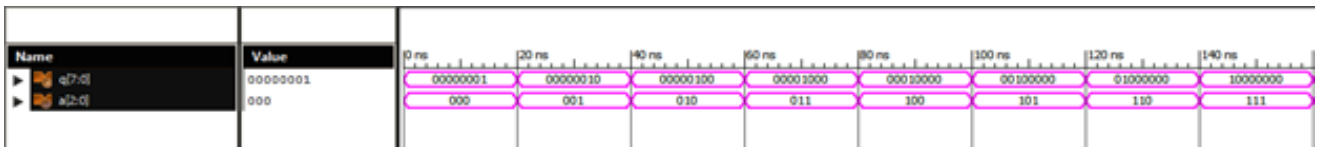


Рисунок 5.2.1 Временная диаграмма работы Дешифратора 3-8 на языке Verilog

В таблице 5.2.3 показан листинг реализации Дешифратор 3-8 на языке VHDL

Таблица 5.2.3– Листинг реализации Дешифратор3-8 на языке VHDL

```
entity DC3_8 is
Port ( i : in STD_LOGIC_VECTOR (2 downto 0);
y : out STD_LOGIC_VECTOR (7 downto 0));
end DC3_8;
architecture Behavioral of DC3_8 is
begin
process(i) begin
case i is
when "111" => y<="10000000";
when "110" => y<="01000000";
when "101" => y<="00100000";
when "100" => y<="00010000";
when "011" => y<="00001000";
```

```
when "010" => y<="00000100";
when "001" => y<="00000010";
when "000" => y<="00000001";
when others => null;
end case;
end process;
end Behavioral;

module DC3_8_tb;
  // Inputs
  reg [2:0] i;
  // Outputs
  wire [7:0] y;
  // Instantiate the Unit Under Test (UUT)
  DC3_8 uut (
         .i(i),
         .y(y)
  );
  initial begin
        // Initialize Inputs
        i = 0;
        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here
   end
   always begin #10 i = i + 1; end
endmodule
```

В рисунке 5.2.2 показана временная диаграммаработы Дешифратора3-8 на языке VHDL

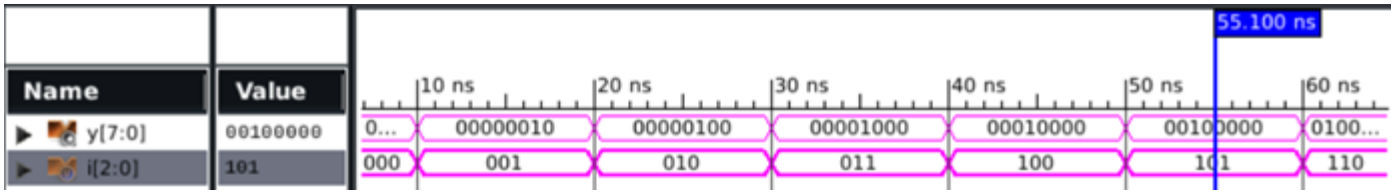


Рисунок 5.2.2 Временная диаграмма работы Дешифратора 3-8 на языке VHDL

5.3 Дешифратор 4-10 Verilog

Дешифраторы используют, когда нужно обращаться к различным цифровым устройствам, и при этом номер устройства (его адрес) представлен двоичным кодом. Входы декодера (адресные входы) часто номеруют не порядковыми номерами, а в соответствии с весами двоичных разрядов, т. е. не 1, 2, 3, 4, а 1, 2, 4, 8.

В таблице 5.3.1 показан листинг реализации Дешифратор 4-10 на языке Verilog

Таблица 5.3.1– Листинг реализации Дешифратор 4-10 на языке Verilog

```
`timescale 1ns / 1ps
module dc4_10(en, a, sel);

  input en;
  input [3:0]a;
```

```
                output [9:0]sel;
                reg[9:0]sel;

                always @(en or a)

            begin
                case({en,a})
                        4'b0000: sel=10'b0000000001;
                        4'b0001: sel=10'b0000000010;
                        4'b0010: sel=10'b0000000100;
                        4'b0011: sel=10'b0000001000;
                        4'b0100: sel=10'b0000010000;
                        4'b0101: sel=10'b0000100000;
                        4'b0110: sel=10'b0001000000;
                        4'b0111: sel=10'b0010000000;
                        4'b1000: sel=10'b0100000000;
                        4'b1001: sel=10'b1000000000;
                        4'b1010: sel=10'b0000000000;
                        4'b1011: sel=10'b0000000000;
                        4'b1100: sel=10'b0000000000;
                        4'b1101: sel=10'b0000000000;
                        4'b1110: sel=10'b0000000000;
                        4'b1111: sel=10'b0000000000;

                endcase
            end

            endmodule
```

В таблице 5.3.2 показан листинг проверки Дешифратор 4-10 на языке Verilog

Таблица 5.3.2– Листинг проверки Дешифратор 4-10 на языке Verilog

```
            `timescale 1ns / 1ps

            module TB_DC4V10;

                // Inputs
                reg en;
                reg [3:0] a;

                // Outputs
                wire [9:0] sel;

                // Instantiate the Unit Under Test (UUT)
                dc4_10 uut (
                      .en(en),
                      .a(a),
                      .sel(sel)
                );

                initial begin
                      // Initialize Inputs
                      en = 0;
```

```
                a = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here


        end

    always
            begin
                #10 a = a + 1;
            end

endmodule
```

В рисунке 5.3.1 показана временная диаграммаработы Дешифратора4-10 на языке Verilog

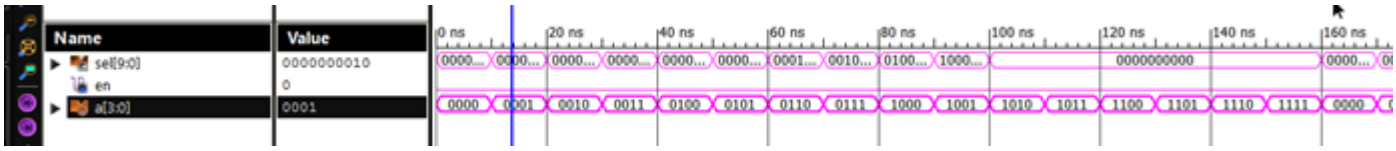


Рисунок 5.3.1 Временная диаграмма работы Дешифратора 4-10 на языке Verilog

5.4 Шифратор 10-4 Verilog

Задача шифратора – сформировать код. На вход шифратора могут подаваться различные сигналы: логический 0 или логическая 1 через контакты кнопок клавиатуры управления (кодируется состояние переключателей) или сигналы с других устройств, но во всех случаях происходит преобразование одного сигнала в n-разрядный код (преобразуется унитарный код в позиционный).

В таблице 5.4.1 показан листинг реализации Шифратор 10-4 на языке Verilog

Таблица 5.4.1– Листинг реализации Шифратор 10-4 на языке Verilog

```
module shifrat1( a, sel);

    input [9:0]a;

    output [3:0]sel;
    reg[3:0]sel;

    always @(a)

        begin
            case({a})
                10'b0000000001: sel=4'b0001;
                10'b0000000010: sel=4'b0010;
                10'b0000000100: sel=4'b0011;
                10'b0000001000: sel=4'b0100;
                10'b0000010000: sel=4'b0101;
```

```
                                                10'b0000100000: sel=4'b0110;
                                                10'b0001000000: sel=4'b0111;
                                                10'b0010000000: sel=4'b1000;
                                                10'b0100000000: sel=4'b1001;
                                                10'b1000000000: sel=4'b1010;
                                        endcase
                                end


                        endmodule
```

В таблице 5.4.2 показан листинг проверки Шифратор 10-4 на языке Verilog

Таблица 5.4.2– Листинг проверки Шифратор 10-4 на языке Verilog

```
                        module schifrattest;
                           // Inputs

                           reg [9:0] a;

                           // Outputs
                           wire [3:0] sel;

                           // Instantiate the Unit Under Test (UUT)
                           shifrat1 uut (

                                 .a(a),
                                 .sel(sel)
                           );

                           initial begin
                                 // Initialize Inputs

                                 a = 0;

                                 // Wait 100 ns for global reset to finish
                                 #100;

                                 // Add stimulus here

                           end

                            always
                           begin
                        #10 a=0; #10 a=1; #10 a=2; #10 a=4; #10 a=8; #10 a=16; #10
                    a=32; #10 a=64; #10 a=128; #10 a=256; #10 a=512; end

                        endmodule
```

В рисунке 5.4.1 показана временная диаграммаработы Дешифратора 4-10 на языке Verilog

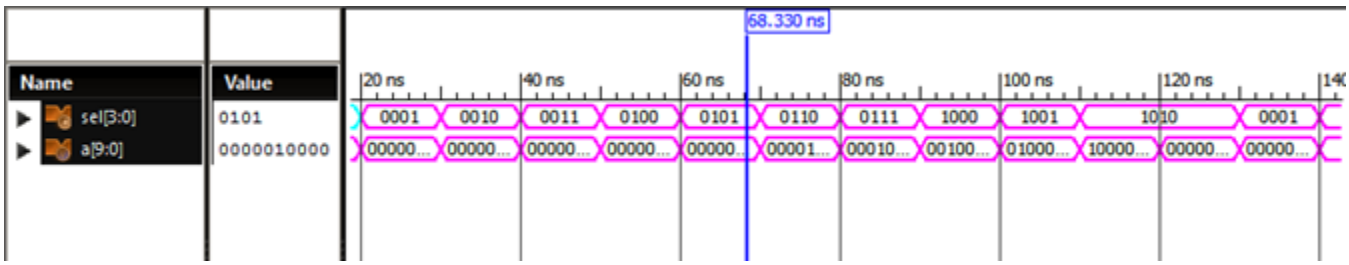


Рисунок 5.4.1 Временная диаграмма работы Шифратора 10-4 на языке Verilog

6 КРАТНЫЕ СЕРИАЛАЙЗЕРЫ И ДЕСЕРИАЛАЙЗЕРЫ

6.1 Десериалайзер 1-16

Десериалайзер 1-16 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.1.1 показан листинг реализации Десериалайзер 1-16 на языке VHDL

Таблица 6.1.1– Листинг реализации Десериалайзер 1-16 на языке VHDL

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

entity d1_16 is
port (   clk: in std_logic;
                dataIn: in bit;
                dataOut: out bit_vector(15 downto 0);
                en: out bit
                );
end d1_16;

architecture Behavioral of d1_16 is
   shared variable cnt: integer := 0;
begin
   process(clk) is
   begin
         ifrising_edge(clk) then
                en<= '0';
                casecnt is
                      when 0 =>dataOut(0) <= dataIn;
                      when 1 =>dataOut(1) <= dataIn;
                      when 2 =>dataOut(2) <= dataIn;
                      when 3 =>dataOut(3) <= dataIn;
                      when 4 =>dataOut(4) <= dataIn;
                      when 5 =>dataOut(5) <= dataIn;
                      when 6 =>dataOut(6) <= dataIn;
                      when 7 =>dataOut(7) <= dataIn;
                      when 8 =>dataOut(8) <= dataIn;
                      when 9 =>dataOut(9) <= dataIn;
                      when 10 =>dataOut(10) <= dataIn;
                      when 11 =>dataOut(11) <= dataIn;
                      when 12 =>dataOut(12) <= dataIn;
                      when 13 =>dataOut(13) <= dataIn;
                      when 14 =>dataOut(14) <= dataIn;
                      when 15 =>dataOut(15) <= dataIn;
                      when others =>cnt := cnt;
                end case;
                cnt := cnt + 1;
                ifcnt> 15 then
                      en<= '1';
                      cnt := 0;
                end if;
         end if;
   end process;
```

```
end Behavioral;
```

В таблице 6.1.2 показан листинг проверки Десериалайзер 1-16 на языке VHDL

Таблица 6.1.2– Листинг проверки Десериалайзер 1-16 на языке VHDL

```
always begin #10 clk =~ clk; end
always begin #20 dataIn = ~dataIn; end
```

В рисунке 6.1.1 показана временная диаграммаработы Десериалайзера 1-16 на языке VHDL

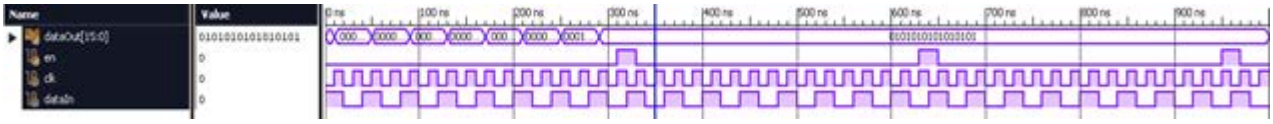


Рисунок 6.1.1 Временная диаграмма работы Десериалайзера 1-16 VHDL

Задача: Создать десериалайзер (n входов данных - m выходов данных) на языке Verilog. По варианту принять n=1, m=16. Данные Din[n-1:0] сопровождаются тактовым сигналом Cin (передний фронт). После формирования m-разрядного пакета Dout[m-1:0] должен формироваться импульс WE (WriteEnable). Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Представленный ниже модуль имеет четыре входа и один выход. Порт rin предназначен для ввода данных, en (от англ. enable) - разрешение работы, clk (от англ. clock) - ввод тактирования, clr_n (от англ. clear) - ввод сброса, выход rout предназначен для вывода данных. Параметр size предназначен для масштабирования модуля до больших размеров.

В таблице 6.1.3 показан листинг реализации Десериалайзер 1-16 на языке Verilog

Таблица 6.1.3– Листинг реализации Десериалайзер 1-16 на языке Verilog

```
`timescale 1ns / 1ps

module p1p16(clk, dataIn, dataOut, en);
   parametern = 16;

   inputwireclk;
   inputwire dataIn;
   output reg [n-1:0] dataOut;
   output reg en;

   reg [n-1:0] cnt;

   initial
   begin
         en <= 0;
         cnt <= 0;
         //assign dataOut = 0;
   end

```

```
                    always @(posedge clk)
                    begin
                          en = 0;
                          cnt <= 0;
                          //assign dataOut = 0;
                    end

                    always @(posedge clk)
                    begin
                          en = 0;
                          dataOut[cnt] <= dataIn;
                          cnt = cnt + 1;
                          if (cnt == n) begin
                                en <= 1;
                                cnt = 0;
                          end;
                    end
              endmodule
```

Для проверки работоспособности модуля на входы десериалайзера поступают два сигнала: такирование clk и данные dataIn. Тактирование происходит с частотой 1/20 Гц. На вход данных поступает случайная заранее заданная комбинация нулей и единиц, а именно 1101011011010110. Также для проверки тактирование может подаваться комбинация из чередования нулей и единиц.

В таблице 6.1.4 показан листинг проверки Десериалайзер 1-16 на языке Verilog

Таблица 6.1.4– Листинг проверки Десериалайзер 1-16 на языке Verilog

```
                    always begin #10 clk =~ clk; end
                    always begin #20 dataIn = 1; #40 dataIn = 0; #20 dataIn =
          1; #20 dataIn = 0; #20 dataIn = 1; #40 dataIn = 0; end
```

Проверка тактирования модуля, а именно поступление данных на выход при их поступлении на вход устройства.

В рисунке 6.1.2 показана временная диаграмма работы Десериалайзера 1-16 на языке Verilog

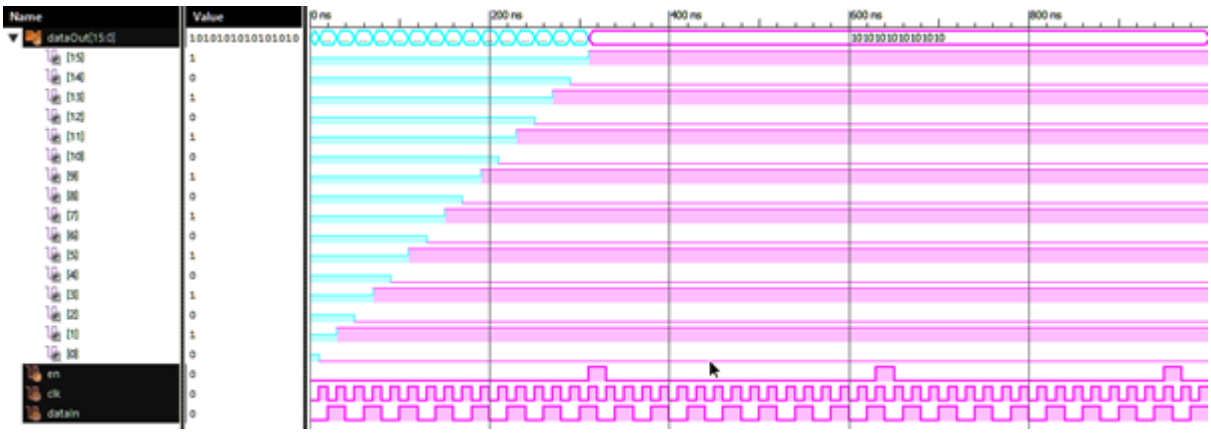


Рисунок 6.1.2 Временная диаграмма работы Десериалайзера 1-16 на языке Verilog

Временная диаграмма работы модуля в соответствии с кодом отладки, которая подтверждает, что выходная комбинация полностью соответствует входной.

6.2 Десериалайзер 2-32 Verilog

Десериализация — восстановление начального состояния структуры данных из битовой последовательности. Типы: с передачей синхросигнала, с жесткой частотой, с автоподстройкой F.

В таблице 6.2.1 показан листинг реализации Десериалайзер 2-32 на языке Verilog

Таблица 6.2.1– Листинг реализации Десериалайзер 2-32 на языке Verilog

```
module dva_tridva(Cin, Din, Dout, WE);
input Cin;
input [1:0] Din;
output [31:0] Dout;
output reg WE;
reg [31:0] Dout;
reg [4:0] cnt;

initial cnt<= 0;
initial WE <= 0;
always @(posedgeCin)
begin
WE = 0;
   case(cnt)
         4'b0000: Dout[1:0] <= Din;
         4'b0001: Dout[3:2] <= Din;
         4'b0010: Dout[5:4] <= Din;
         4'b0011: Dout[7:6] <= Din;
         4'b0100: Dout[9:8] <= Din;
         4'b0101: Dout[11:10] <= Din;
         4'b0110: Dout[13:12] <= Din;
         4'b0111: Dout[15:14] <= Din;
         4'b1000: Dout[17:16] <= Din;
         4'b1001: Dout[19:18] <= Din;
         4'b1010: Dout[21:20] <= Din;
         4'b1011: Dout[23:22] <= Din;
         4'b1100: Dout[25:24] <= Din;
         4'b1101: Dout[27:26] <= Din;
         4'b1110: Dout[29:28] <= Din;
         4'b1111: Dout[31:30] <= Din;
   endcase
   cnt = cnt + 1;
if(cnt == 5'b10000)
begin
   WE <= 1;
   cnt = 0;
end
end
endmodule
```

В таблице 6.2.2 показан листинг проверки Десериалайзер 2-32 на языке Verilog

Таблица 6.2.2– Листинг проверки Десериалайзер 2-32 на языке Verilog

```
moduledva_tridva_tb;

    // Inputs
    regCin;
    reg [1:0] Din;

    // Outputs
    wire [31:0] Dout;
    wire WE;

    // Instantiate the Unit Under Test (UUT)
    dva_tridvauut (
        .Cin(Cin),
        .Din(Din),
        .Dout(Dout),
        .WE(WE)
    );

    initial begin
        // Initialize Inputs
        Cin = 0;
        Din = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end
        always begin #5 Cin = ~Cin; end
        always begin #5 Din = Din + 1; end
endmodule
```

В рисунке 6.2.1 показана временная диаграмма работы Десериалайзера 2-32 на языке Verilog

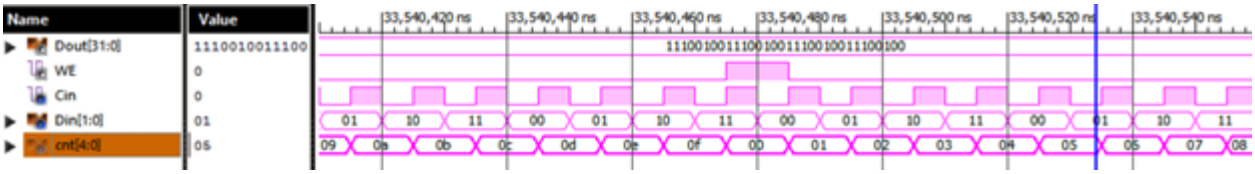


Рисунок 6.2.1 Временная диаграмма работы Десериалайзера 2-32 на языке Verilog

6.3 Десериалайзер 3-9 Verilog

Десериалайзер 3-9 используют для видео приложений при кодировании цвета. Это обеспечивает высокую скорость и при этом небольшое рассеивание энергии.

В таблице 6.3.1 показан листинг реализации Десериалайзер 3-9 на языке Verilog

Таблица 6.3.1– Листинг реализации Десериалайзер 3-9 на языке Verilog

```
`timescale 1ns / 1ps
module deserial3x9(Cin, Din, Dout, WE);
input Cin;
input [2:0] Din;
```

```
output [8:0] Dout;
output reg WE;
reg [8:0] Dout;

integer cnt;

initial cnt <= 0;
initial WE <= 0;
   always @(posedge Cin)
   begin
         WE = 0;
         case(cnt)
               0: Dout[2:0] <= Din;
               1: Dout[5:3] <= Din;
               2: Dout[8:6] <= Din;
         endcase
         cnt = cnt + 1;
   if(cnt >=4 && WE==0)
   begin
         WE <= 1;
         cnt = 0;
   end
   end
endmodule
```

В таблице 6.3.2 показан листинг проверки Десериалайзер 3-9 на языке Verilog

Таблица 6.3.2– Листинг проверки Десериалайзер 3-9 на языке Verilog

```
`timescale 1ns / 1ps

module deserial3x9tb;

   // Inputs
   reg Cin;
   reg [2:0] Din;

   // Outputs
   wire [8:0] Dout;
   wire WE;

   // Instantiate the Unit Under Test (UUT)
   deserial3x9 uut (
         .Cin(Cin),
         .Din(Din),
         .Dout(Dout),
         .WE(WE)
   );

   initial begin
         // Initialize Inputs
         Cin = 0;
         Din = 0;
         .Dout(Dout),
         .WE(WE)
```

```
        );

        initial begin
                // Initialize Inputs
                Cin = 0;
                Din = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

        end
always begin #5 Cin = ~Cin; end
always begin #10 Din = Din + 1; end
endmodule
```

В рисунке 6.3.1 показана временная диаграмма работы Десериалайзера 3-9 на языке Verilog

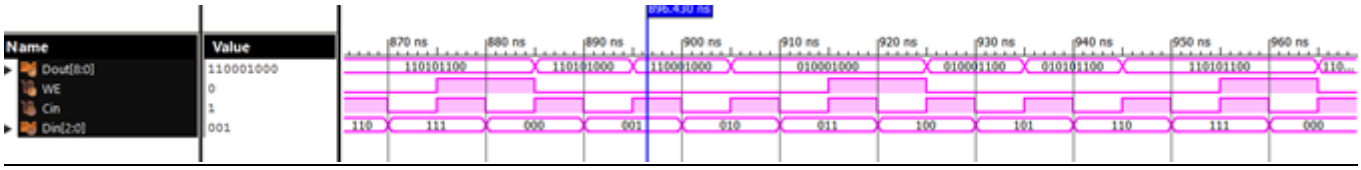


Рисунок 6.3.1 Временная диаграмма работы Десериалайзера 3-9 на языке Verilog

6.4 Десериалайзер 3-12 Verilog

Десериалайзер 3-12 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.4.1 показан листинг реализации Десериалайзер 3-12 на языке Verilog

Таблица 6.4.1– Листинг реализации Десериалайзер 3-12 на языке Verilog

```
module des_3_12(Cin, Din, Dout, WE);
input Cin;
input [2:0] Din;
output [11:0] Dout;
output reg WE;
reg [11:0] Dout;

integer cnt;

initial cnt <= 0;
initial WE <= 0;
   always @(posedge Cin)
   begin
         WE = 0;
         case(cnt)
               0: Dout[2:0] <= Din;
               1: Dout[5:3] <= Din;
               2: Dout[8:6] <= Din;
```

```
                        3: Dout[11:9] <= Din;
                endcase
                cnt = cnt + 1;
        if(cnt >=4 && WE==0)
        begin
                WE <= 1;
                cnt = 0;
        end
        end
endmodule
```

В таблице 6.4.2 показан листинг проверки Десериалайзер 3-12 на языке Verilog

Таблица 6.4.2– Листинг проверки Десериалайзер 3-12 на языке Verilog

```
module des_tb;

        // Inputs
        reg Cin;
        reg [2:0] Din;

        // Outputs
        wire [11:0] Dout;
        wire WE;

        // Instantiate the Unit Under Test (UUT)
        des_3_12 uut (
                .Cin(Cin),
                .Din(Din),
                .Dout(Dout),
                .WE(WE)
        );

        initial begin
                // Initialize Inputs
                Cin = 0;
                Din = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

        end
                always begin #5 Cin = ~Cin; end
                always begin #10 Din = Din + 1; end

endmodule
```

В рисунке 6.4.1 показана временная диаграмма работы Десериалайзера 3-12 на языке Verilog

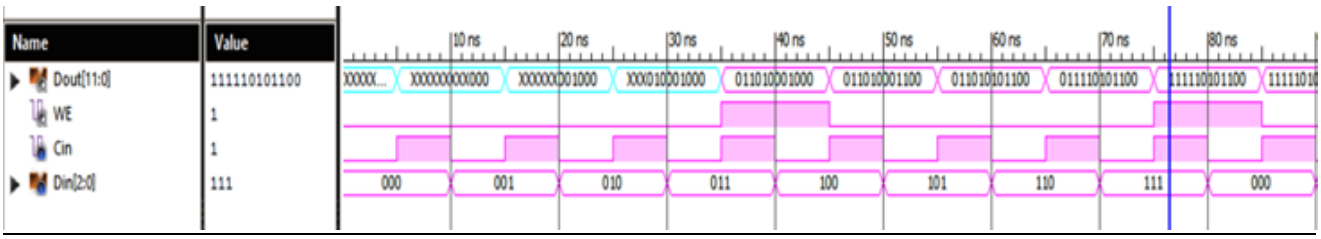


Рисунок 6.4.1 Временная диаграмма работы Десериалайзера 3-12 на языкеVerilog

6.5 Десериалайзер 4-64 Verilog

Десериалайзер 4-64 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.5.1 показан листинг реализации Десериалайзер 4-64 на языке Verilog

Таблица 6.5.1– Листинг реализации Десериалайзер 4-64 на языке Verilog

```
                    modulefour_six_for(Cin, Din, Dout, WE, cnt);
                       inputCin;
                       input [3:0] Din;
                       outputreg [63:0] Dout;

                       outputcnt;
                    // output reg [63:0] buffer;
                       outputreg WE;
                       reg [4:0] cnt;

                       initialcnt=0;
                       always @(posedgeCin) //переднийфронт
                    begin
                       WE=0;

                       case (cnt)
                             4'b0000: Dout [3:0]=Din;
                             4'b0001: Dout [7:4]=Din;
                             4'b0010: Dout [11:8]=Din;
                             4'b0011: Dout [15:12]=Din;
                             4'b0100: Dout [19:16]=Din;
                             4'b0101: Dout [23:20]=Din;
                             4'b0110: Dout [27:24]=Din;
                             4'b0111: Dout [31:28]=Din;
                             4'b1000: Dout [35:32]=Din;
                             4'b1001: Dout [39:36]=Din;
                             4'b1010: Dout [43:40]=Din;
                             4'b1100: Dout [51:48]=Din;
                             4'b1101: Dout [55:52]=Din;
                             4'b1110: Dout [59:56]=Din;
                             4'b1111: Dout [63:60]=Din;
                       endcase
                       cnt=cnt+1;
                       if (cnt == 16)
                             begin
                                   WE=1;
```

```
                    cnt=0;
                end
            end
            endmodule
```

Втаблице 6.5.2 показан листинг проверки Десериалайзер 4-64 на языке Verilog

Таблица 6.5.2– Листинг проверки Десериалайзер 4-64 на языке Verilog

```
            modulesix_four;

                // Inputs
                regCin;
                reg [3:0] Din;

                // Outputs
                wire [63:0] Dout;
                wire WE;

                // Instantiate the Unit Under Test (UUT)
                four_six_foruut (
                    .Cin(Cin),
                    .Din(Din),
                    .Dout(Dout),
                    .WE(WE)
                );

                initial begin
                    // Initialize Inputs
                    Cin = 0;
                    Din = 0;

                    // Wait 100 ns for global reset to finish
                    #100;

                    // Add stimulus here

                end

            always begin #5 Cin=1; #5 Cin=0; end
            always begin #5 Din=Din+1;  end
            endmodule
```

В рисунке 6.5.1 показана временная диаграмма работы Десериалайзера 4-64 на языке Verilog

Рисунок 6.5.1 Временная диаграмма работы Десериалайзера 4-64 на языке Verilog

6.6 Десериалайзер 12-36 Verilog

Десериалайзер 12-36 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

Данные Din[n-1:0] сопровождаются тактовым сигналом Cin (передний фронт). После формирования m-разрядного пакета Dout[m-1:0] должен формироваться импульс WE (WriteEnable).

В таблице 6.6.1 показан листинг реализации Десериалайзер 12-36 на языке Verilog

Таблица 6.6.1– Листинг реализации Десериалайзер 12-36 на языке Verilog

```
module s(Cin, Din, WE, Dout );
inputCin;
input [11:0] Din;
output [35:0] Dout;
outputreg WE;
reg [35:0] Dout;
reg [2:0] cnt;

initialcnt<= 0;
always @(posedgeCin)
begin
WE = 0;
   case (cnt)
   2'b01: Dout [11:0] <= Din;
   2'b10: Dout [23:12] <= Din;
   2'b11: Dout [35:24] <= Din;
   endcase

cnt = cnt + 1;
if (cnt == 3'b100)
   begin
         WE <= 1;
         cnt = 0;
   end

end

endmodule
```

В таблице 6.6.2 показан листинг проверки Десериалайзер 12-36 на языке Verilog

Таблица 6.6.2– Листинг проверки Десериалайзер 12-36 на языке Verilog

```
moduletestt;

   // Inputs
   regCin;
   reg [11:0] Din;

   // Outputs
   wire  WE;
   wire [35:0] Dout;
```

```
                    // Instantiate the Unit Under Test (UUT)
                    suut (
                          .Cin(Cin),
                          .Din(Din),
                          .WE(WE),
                          .Dout(Dout)
                    );

                    initial begin
                          // Initialize Inputs
                          Cin = 0;
                          Din = 0;

                          // Wait 100 ns for global reset to finish
                          #100;

                          // Add stimulus here

                    end
               always begin #5 Cin = ~ Cin; end
                          always begin #10 Din = Din + 1; end
               endmodule
```

В рисунке 6.6.1 показана временная диаграмма работы Десериалайзера 12-36 на языке Verilog

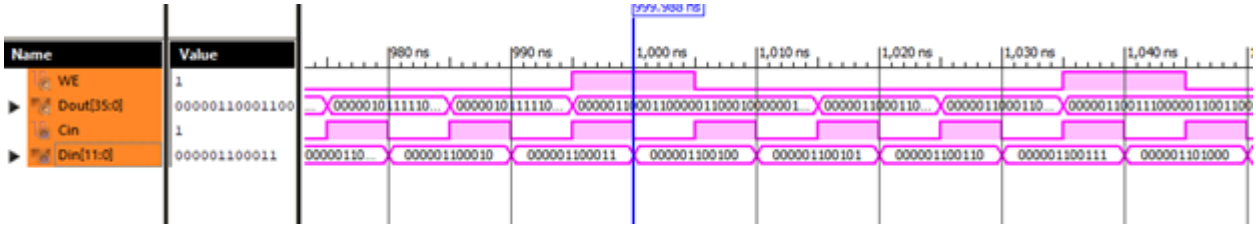


Рисунок 6.6.1 Временная диаграмма работы Десериалайзера 12-36 на языкеVerilog

6.7 Десериалайзер 6-42 Verilog

Десериалайзер 6-42 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

Создать десериалайзер (n входов данных - m выходов данных) на языке Verilog. По варианту принять n=6, m=42. Данные Din[n-1:0] сопровождаются тактовым сигналом Cin (передний фронт). После формирования m-разрядного пакета Dout[m-1:0] должен формироваться импульс WE (WriteEnable). Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

В представленном модуле порт Din - информационный, WE - вход разрешения работы, Cin – вход сигнала тактирования, Dout – информационный выход.

В таблице 6.7.1 показан листинг реализации Десериалайзер 6-42 на языке Verilog

Таблица 6.7.1– Листинг реализации Десериалайзер 6-42 на языке Verilog

```
moduledeserializer(Cin, Din, Dout, WE);
inputCin;
input [5:0] Din;
output [41:0] Dout;
outputreg WE;
reg [41:0] Dout;
reg [2:0] cnt;

initialcnt<= 0;
initial WE <= 0;
always @(posedgeCin)
begin
WE = 0;
   case(cnt)
         3'b000: Dout[5:0] <= Din
         3'b001: Dout[11:6] <= Din;
         3'b0010: Dout[17:12] <= Din;
         3'b0011: Dout[23:18] <= Din;
         3'b0100: Dout[29:24] <= Din;
         3'b0101: Dout[35:30] <= Din;
         3'b0110: Dout[41:36] <= Din;
   endcase
   cnt = cnt + 1;
if(cnt == 5'b10000)
begin
   WE<= 1;
   cnt = 0;
end
end
endmodule
```

Втаблице 6.7.2показанлистинг (вариант 2)реализацииДесериалайзер 6-42 наязыкеVerilog

Таблица 6.7.2– Листинг(вариант 2) реализацииДесериалайзер 6-42 наязыкеVerilog

```
moduledeser6v42( clk, dataIn, dataOut, en);
inputwireclk;
inputwire [5:0] dataIn;
outputreg [41:0] dataOut;
   outputregen;

   reg [5:0] cnt;

   initial
   begin
   en= 0;
cnt = 0;
   end

always @(posedgeclk)
begin
         en = 0;
         //cnt<= cnt + 1;
case (cnt)
```

```
                    6'd6: dataOut[5:0] = dataIn;
                    6'd12: dataOut[11:6] = dataIn;
                    6'd18: dataOut[17:12] = dataIn;
                    6'd24: dataOut[23:18] = dataIn;
                      6'd30: dataOut[29:24] = dataIn;
                      6'd36: dataOut[35:30] = dataIn;
                      6'd42: dataOut[41:36] = dataIn;
            default: ;
            endcase
                    cnt<= cnt + 1;
                    if(cnt == 42 &&clk==0)
                    begin
                          en<= 1;
                          cnt<= 0;
            end
            end

            endmodule
```

Для проверки работоспособности модуля были использованы два тестовых воздействия (две сопутствующие реализации самого модуля представлены в разделе

2). В соответствии с первым тестовым воздействием на информационный вход поступает заданная битовая комбинация 111001.

В таблице 6.7.3 показан листинг проверки Десериалайзер 6-42 на языке Verilog

Таблица 6.7.3– Листинг проверки Десериалайзер 6-42 на языке Verilog

```
            module deser6v42_tb;
              // Inputs
              regclk;
              reg [5:0] dataIn;
              // Outputs
              wire [41:0] dataOut;
              wireen;

              // Instantiate the Unit Under Test (UUT)
              deser6v42uut (
                    .clk(clk),
                    .dataIn(dataIn),
                    .dataOut(dataOut),
                    .en(en)
              );

              initial begin
                    // Initialize Inputs
                    clk = 0;
                    dataIn = 0;

                    dataIn = 6'b111001;

                    // Wait 100 ns for global reset to finish
                    #100;
```

```
                        // Add stimulus here

                end

                        always begin #10 clk =~ clk; end

        endmodule
```

Во втором случае последовательность на информационных входах представляет собой увеличивающееся с каждым тактом на 1 6-ти разрядное число от 000000 до 111111.

В таблице 6.7.4 показан листинг проверки Десериалайзер 6-42 на языке Verilog

Таблица 6.7.4– Листинг проверки Десериалайзер 6-42 на языке Verilog

```
        moduledeserializer_tb;

                // Inputs
                regCin;
                reg [5:0] Din;

                // Outputs
                wire [41:0] Dout;
                wireWE;

                // Instantiate the Unit Under Test (UUT)
                deserializeruut (
                        .Cin(Cin),
                        .Din(Din),
                        .Dout(Dout),
                        .WE(WE)
                );

                initial begin
                        // Initialize Inputs
                        Cin = 0;
                        Din = 0;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                end
        always begin #5 Cin = ~Cin; end
                        always begin #10 Din = Din + 1; end
        endmodule
```

В рисунке 6.7.1 показана временная диаграмма работы Десериалайзера 6-42 на языке Verilog

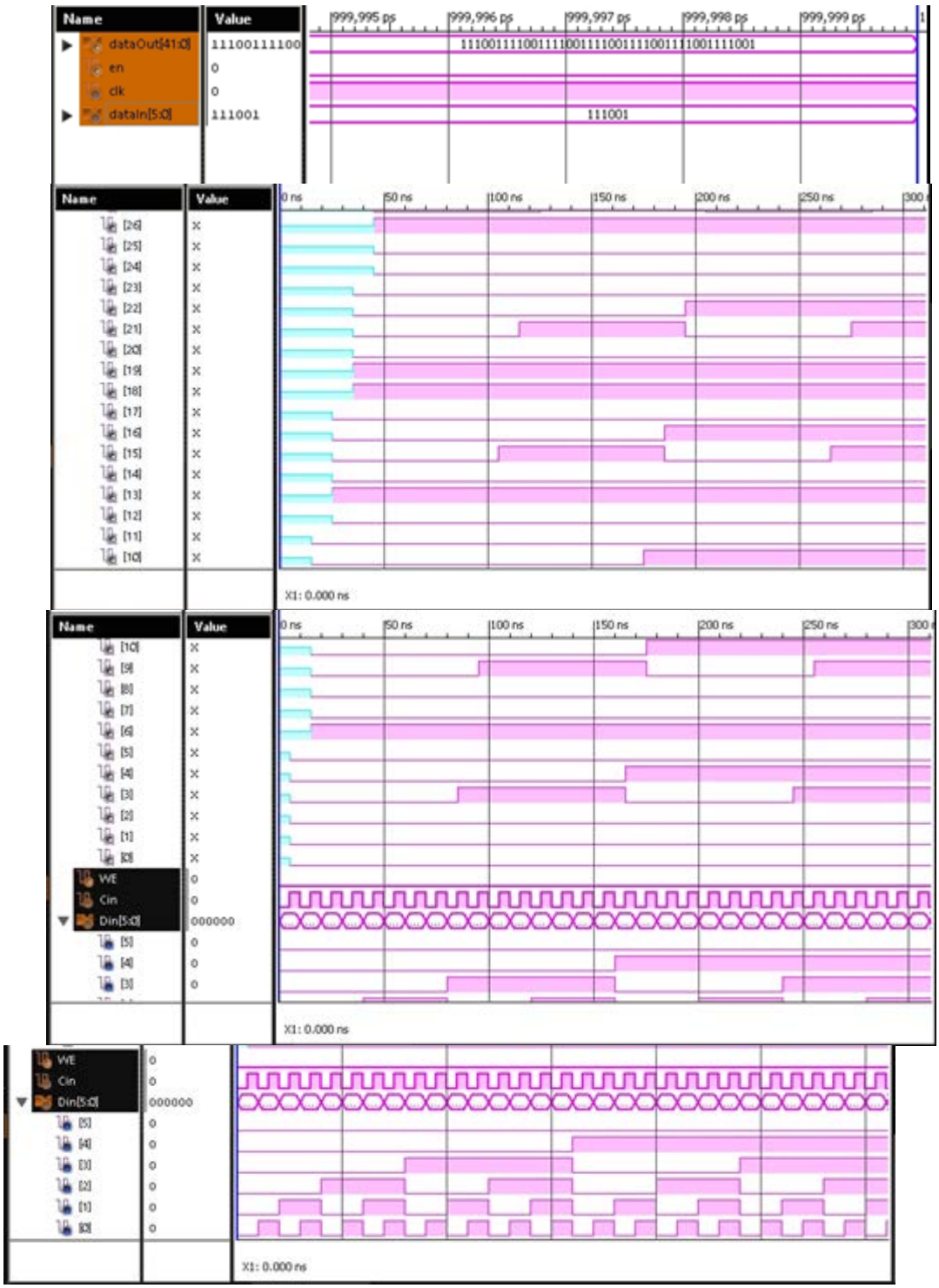


Рисунок 6.7.1 Временные диаграммы работы Десериалайзера 6-42 на языке Verilog

6.8 Десериалайзер 12-48 Verilog

Десериалайзер 12-48 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.8.1 показан листинг реализации Десериалайзер 12-48 на языке Verilog

Таблица 6.8.1– Листинг реализации Десериалайзер 12-48 на языке Verilog

`module twelve_48_t;`
`// Inputs`

```
                                regCin;
                                reg [11:0] Din;

                                // Outputs
                                wire [47:0] Dout;
                                wire WE;
                                wire [3:0] cnt;

                                //  Instantiate  the  Unit  Under  Test
        (UUT)
                                twelve_48 uut (
                                .Cin(Cin),
                                .Din(Din),
                                .Dout(Dout),
                                .WE(WE),
                                .cnt(cnt)
                                );

                                initial begin
                                // Initialize Inputs
                                Cin = 0;
                                Din = 0;

                                //  Wait  100  ns  for  global  reset  to
        finish
                                #100;

                                // Add stimulus here

                                end

                                always begin #5 Cin=1; #5 Cin=0; end
                                always begin #10 Din=Din+12; end

                        endmodule
```

В таблице 6.8.2 показан листинг проверки Десериалайзер 12-48 на языке Verilog

Таблица 6.8.2– Листинг проверки Десериалайзер 12-48 на языке Verilog

```
                        module twelve_48(Cin, Din, Dout, WE, cnt);

                                input Cin;
                                input [11:0] Din;
                                output reg [47:0] Dout;

                                output cnt;
                                reg [3:0] cnt;
                                output reg WE;

                                initial cnt=0;

                                always @(posedgeCin) //переднийфронт
                                begin
                                WE=0;
                                case (cnt)
```

```
                4'b0000: Dout [11:0]=Din;
                4'b0001: Dout [23:12]=Din;
                4'b0010: Dout [35:24]=Din;
                4'b0011: Dout [47:36]=Din;
                endcase

                cnt=cnt+1;
                end

                always @(posedgeCin)
                if (cnt == 4)
                begin
                WE=1;
                cnt=0;

                end

                endmodule
```

В рисунке 6.8.1 показана временная диаграмма работы Десериалайзера 12-48 на языке Verilog

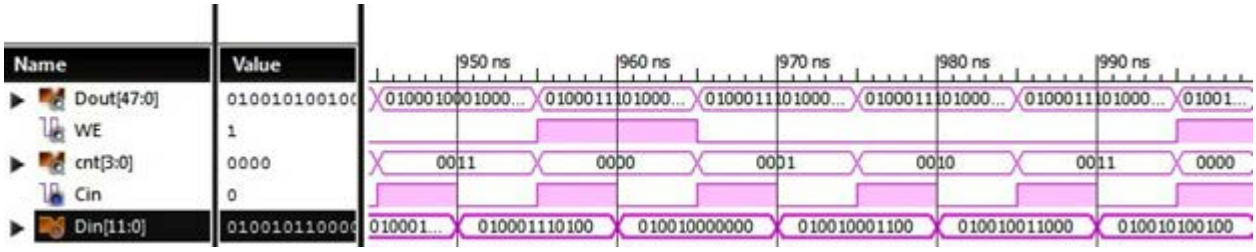


Рисунок 6.8.1 Временные диаграммы работы Десериалайзера 12-48 на языке Verilog

6.9 Десериалайзер 7-28 Verilog

Десериалайзер 7-28 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.9.1 показан листинг реализации Десериалайзер 7-28 на языке Verilog

Таблица 6.9.1– Листинг реализации Десериалайзер 7-28 на языке Verilog

```
                `timescale 1ns / 1ps

               module p7p28(
                    input wire clk,
                    input wire [7:0] dataIn,
                    output reg [28:0] dataOut,
                    output reg [0:0] en
                    );
                    initial en=1'b0;
                    reg [4:0] cnt = 2'b00000;

                    always @(posedge clk)
                      cnt <= cnt + 1;
```

```
                always @* begin
                    en <= 1'd0;
                  case (cnt)
                    5'd7: dataOut[6:0] = dataIn;
                    5'd14: dataOut[13:7] = dataIn;
                    5'd21: dataOut[20:14] = dataIn;
                    5'd28: dataOut[28:21] = dataIn;
                    default: ;
                  endcase
                     if(cnt == 28 && clk==0)
                    begin
                          en <= 1'd1;
                          cnt <= 0;
                end
                end

            endmodule
```

В таблице 6.9.1 показан листинг проверки Десериалайзер 7-28 на языке Verilog

Таблица 6.9.1– Листинг проверки Десериалайзер 7-28 на языке Verilog

```
            `timescale 1ns / 1ps

            ////////////////////////////////////////////////////////////////
////////////
                        ////////
            // Company: BMSTU
            // Engineer: Echeistov
            //
            // Create Date:   17:44:46 03/23/2017
            // Design Name:   p7p28
            //                          Module                          Name:
C:/Users/Vladimir/Desktop/VHDL/dtrig1/p7to28_tb.v
            // Project Name:  dtrig1
            // Target Device:
            // Tool versions:
            // Description:
            //
            // Verilog Test Fixture created by ISE for module: p7p28
            //
            // Dependencies:
            //
            // Revision:
            // Revision 0.01 - File Created
            // Additional Comments:
            //
            ////////////////////////////////////////////////////////////////
////////////


            module p7to28_tb;

               // Inputs
               reg clk;
               reg [7:0] dataIn;
```

```
                    // Outputs
                    wire [28:0] dataOut;

                    // Instantiate the Unit Under Test (UUT)
                    p7p28 uut (
                          .clk(clk),
                          .dataIn(dataIn),
                          .dataOut(dataOut)
                    );

                    initial begin
                          // Initialize Inputs
                          clk = 0;
                          dataIn = 0;

                          // Wait 100 ns for global reset to finish
                          #100;

                          // Add stimulus here

                    end
                       always begin #20 clk=~clk; end
                          always begin #100 dataIn=~dataIn; end
              endmodule
```

В рисунке 6.9.1 показана временная диаграмма работы Десериалайзера 7-28 на языке Verilog

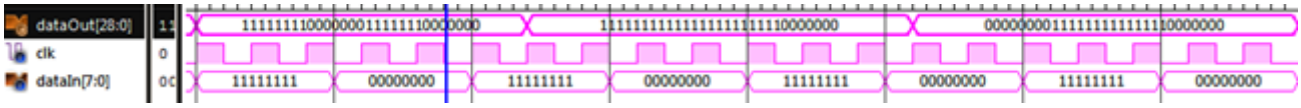


Рисунок 6.9.1 Временные диаграммы работы Десериалайзера 7-28 на языке Verilog

6.10 Десериалайзер 7-28 Verilog. Вариант 2.

Десериалайзер 7-28 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.10.1 показан листинг реализации Десериалайзер 7-28 на языке Verilog

Таблица 6.10.1– Листинг реализации Десериалайзер 7-28 на языке Verilog

```
              module four_six_for(Cin, Din, Dout, WE, cnt);
              input Cin;
              input [6:0] Din;
              output reg [27:0] Dout;

              output cnt;
              output reg WE;
              reg [6:0] cnt;

              initial cnt=0;
```

```
                    always @(posedgeCin) //передний фронт
                    begin
                    WE=0;

                    case (cnt)
                    4'b0000: Dout [6:0]=Din;
                    4'b0001: Dout [13:7]=Din;
                    4'b0010: Dout [20:14]=Din;
                    4'b0011: Dout [27:21]=Din;
                    endcase
                    cnt=cnt+1;
                    if (cnt == 5)
                    begin
                    WE=1;
                    cnt=0;

                    end

                    end
          endmodule
```

В таблице 6.10.2 показан листинг проверки Десериалайзер 7-28 на языке Verilog

Таблица 6.10.2– Листинг проверки Десериалайзер 7-28 на языке Verilog

```
                    // Inputs
                    reg Cin;
                    reg [6:0] Din;

                    // Outputs
                    wire [27:0] Dout;
                    wire WE;

                    // Instantiate the Unit Under Test (UUT)
                    four_six_for uut (
                    .Cin(Cin),
                    .Din(Din),
                    .Dout(Dout),
                    .WE(WE)
                    );
                    *
                    initial begin
                    // Initialize Inputs
                    Cin = 0;
                    Din = 7'b1111111;

                    // Wait 100 ns for global reset to finish
                    #100;

                    // Add stimulus here

                    end

                    always begin #5 Cin=1; #5 Cin=0; end
                    //always begin #5 Din=Din+1; end
```

```
Endmodule
```

В рисунке 6.10.1 показана временная диаграмма работы Десериалайзера 7-28 на языке Verilog

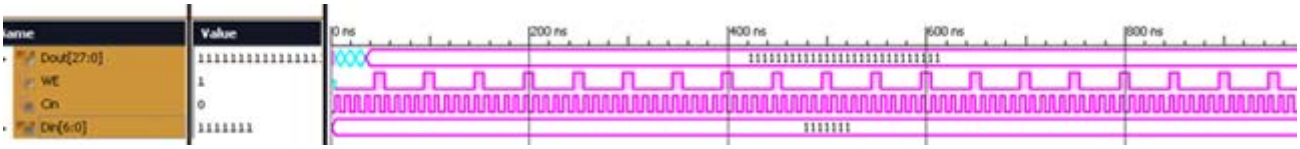


Рисунок 6.10.1 Временные диаграммы работы Десериалайзера 7-28 на языке Verilog

6.11 Десериалайзер 11-44 Verilog

Десериалайзер 11-44 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.11.1 показан листинг реализации Десериалайзер 11-44 на языке Verilog

Таблица 6.11.1– Листинг реализации Десериалайзер 11-44 на языке Verilog

```
module four_six_for(Cin, Din, Dout, WE, cnt);
    inputCin;
    input [10:0] Din;
    outputreg [43:0] Dout;

    outputcnt;
//  -9+
outputreg [63:0] buffer;
    outputreg WE;
    reg [10:0] cnt;

    initialcnt=0;
    always @(posedgeCin) //передний фронт
begin
    WE=0;



    case (cnt)
        4'b0000: Dout [10:0]=Din;
        4'b0001: Dout [21:11]=Din;
        4'b0010: Dout [32:22]=Din;
        4'b0011: Dout [43:33]=Din;
    endcase
    cnt=cnt+1;
    if (cnt == 10)
        begin
            WE=1;
            cnt=0;

  end

end
endmodule
```

В таблице 6.11.2 показан листинг реализации Десериалайзер 11-44 на языке Verilog

Таблица 6.11.2– Листинг реализации Десериалайзер 11-44 на языке Verilog

```
modulesix_four;

 // Inputs
reg Cin;
reg [10:0] Din;

 // Outputs
wire [43:0] Dout;
wireWE;

 // Instantiate the Unit Under Test (UUT)
 four_six_foruut (
     .Cin(Cin),
     .Din(Din),
     .Dout(Dout),
     .WE(WE)
 );
 initial begin
     // InitializeInputs
     Cin = 0;
     Din = 0;

     // Wait 100 ns for global reset to finish
     #100;

     // Addstimulushere
 end
always begin #5 Cin=1; #5 Cin=0; end
always begin #5 Din=Din+1;  end

endmodule
```

В рисунке 6.11.1 показана временная диаграмма работы Десериалайзера 11-44 на языке Verilog

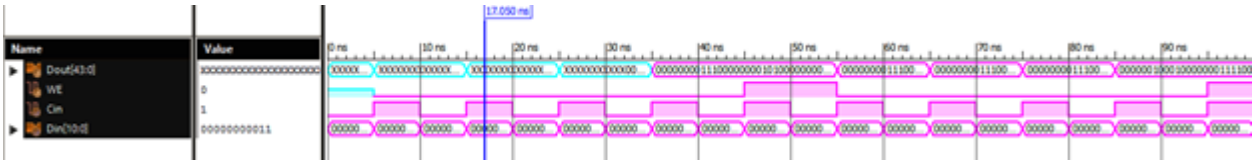


Рисунок 6.11.1 Временные диаграммы работы Десериалайзера 11-44 на языке Verilog

6.12 Десериалайзер 11–44 Verilog. Вариант 2

Десериалайзер 11-44 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

Данные Din[n-1:0] сопровождаются тактовым сигналом Cin (передний фронт). После формирования m-разрядного пакета Dout[m-1:0] должен формироваться импульс WE (WriteEnable).

В таблице 6.12.1 показан листинг реализации Десериалайзер 11-44 на языке Verilog

Таблица 6.12.1– Листинг реализации Десериалайзер 11-44 на языке Verilog

```
module deser11v44( clk, dataIn, dataOut, en);
input wire clk;
    input wire [10:0] dataIn;
    output reg [43:0] dataOut;
    output reg en;

    reg [5:0] cnt;

    initial
    begin
    en= 0;
    cnt = 0;
              end

    always @(posedge clk)
    begin
         en = 0;
         //cnt <= cnt + 1;
      case (cnt)
        6'd11: dataOut[10:0] = dataIn;
        6'd22: dataOut[21:11] = dataIn;
        6'd33: dataOut[32:22] = dataIn;
        6'd44: dataOut[43:33] = dataIn;
        default: ;
      endcase
         cnt <= cnt + 1;
         if(cnt == 44 && clk==0)
         begin
               en <= 1;
               cnt <= 0;
    end
    end

endmodule
```

Втаблице 6.12.2показанлистингпроверкиДесериалайзер 11-44 наязыкеVerilog

Таблица 6.12.2– ЛистингпроверкиДесериалайзер 11-44 наязыкеVerilog

```
moduledeser11v44_tb;

  // Inputs
  regclk;
  reg [10:0] dataIn;

  // Outputs
  wire [43:0] dataOut;
```

```
                wireen;

                // Instantiate the Unit Under Test (UUT)
                deser6v42 uut (
                        .clk(clk),
                        .dataIn(dataIn),
                        .dataOut(dataOut),
                        .en(en)
                );

                initial begin
                        // Initialize Inputs
                        clk = 0;
                        dataIn = 0;

                        dataIn = 6'b111001;

                        // Wait 100 ns for global reset to finish
                        #100;
                        // Add stimulus here

                end
                        always begin #10 clk =~ clk; end
        endmodule
```

В рисунке 6.12.1 показана временная диаграмма работы Десериалайзера 11-44 на языке Verilog

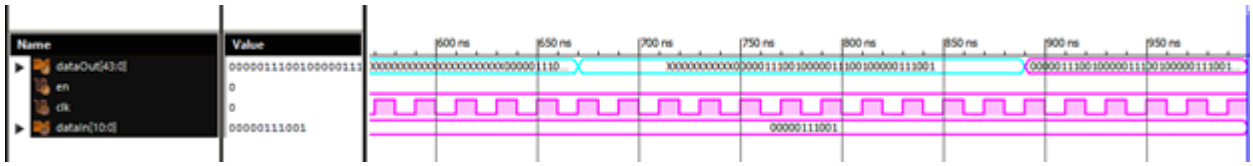


Рисунок 6.12.1 Временные диаграммы работы Десериалайзера 11–44 на языке Verilog

Преобразователь из большего числа разрядов в меньшее с не кратным соотношением разрядностей входов 11 и выходов 2 на языке Verilog.

В таблице 6.12.3 показан листинг реализации Десериалайзер 11-44 на языке Verilog

Таблица 6.12.3– Листинг реализации Десериалайзер 11-44 на языке Verilog

```
        module s11s2(  input wire [in-1:0] dataIn,
                                input wire clkIn,
                                input wire clkToOut,
                                output reg [out-1:0] dataOut,
                                output reg clkOut
                );

                parameter in = 11;
                parameter out = 2;

                reg [in*out-1:0] data = 0;
                integer counterIn = 0;
                integer counterOut = 0;
```

```
                integer start = 0;

                always @(posedge clkIn) begin
                        counterIn = counterIn + 1;
                        if (counterIn > out) begin counterIn = 1; end
                        if (counterIn > 1) begin start = 1; end
                        case (counterIn)
                                1: data[10:0] = dataIn;
                                2: data[21:11] = dataIn;
                                default: ;
                        endcase;
                end

                always @(posedge clkToOut or negedge clkToOut) begin
                        if (start == 1)
                                clkOut = clkToOut;
                        else
                                clkOut = 0;
                end

                always @(posedge clkToOut) begin
                        if (start == 1) begin
                                counterOut = counterOut + 1;
                                if (counterOut > in) begin counterOut = 1;
        end
                                case (counterOut)
                                        1:    dataOut = data[1:0];
                                        2: dataOut = data[3:2];
                                        3: dataOut = data[5:4];
                                        4: dataOut = data[7:6];
                                        5: dataOut = data[9:8];
                                        6: dataOut = data[11:10];
                                        7: dataOut = data[13:12];
                                        8: dataOut = data[15:14];
                                        9: dataOut = data[17:16];
                                        10: dataOut = data[19:18];
                                        11: dataOut = data[21:20];

                                        default: ;
                                endcase;
                        end
                end
        Endmodule
```

Втаблице 6.12.4показанлистингпроверкиДесериалайзер 11-44 наязыкеVerilog

Таблица 6.12.4– ЛистингпроверкиДесериалайзер 11-44 наязыкеVerilog

```
                module s11s2_tb;

                // Inputs
                reg [10:0] dataIn;
                reg clkIn;
                reg clkToOut;

                // Outputs
```

```
                wire [1:0] dataOut;
                wire clkOut;

                // Instantiate the Unit Under Test (UUT)
                s11s2 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .clkToOut(clkToOut),
                        .dataOut(dataOut),
                        .clkOut(clkOut)
                );

                initial begin
                        // Initialize Inputs
                        dataIn = 0;
                        clkIn = 0;
                        clkToOut = 0;

                        dataIn = 11'b10011100100;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                end

                always begin #11 clkIn = ~clkIn; end
                always begin #2 clkToOut = ~clkToOut; end

        endmodule
```

В рисунке 6.12.2 показана временная диаграмма работы Десериалайзера 11-44 на языке Verilog

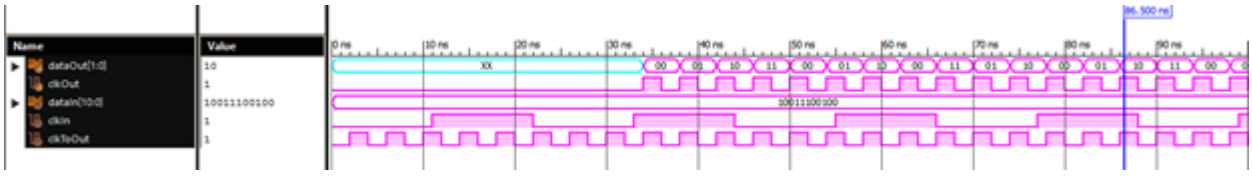


Рисунок 6.12.2 Временные диаграммы работы Десериалайзера 11–44 на языке Verilog

6.13 Сериалайзер 21-3 VHDL

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 21-3 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.13.1 показан листинг реализации Сериалайзер 21-3 на языке VHDL

Таблица 6.13.1– Листинг реализации Сериалайзер 21-3 на языке VHDL

```
        library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity seria_21in3 is
   port( dataIn: in std_logic_vector(20 downto 0);
         clkIn: in std_logic;
         dataOut: out std_logic_vector(2 downto 0);
         clkToOut: in std_logic;
         clkOut: out std_logic;
         RE: out std_logic
         );
end seria_21in3;

architecture Behavioral of seria_21in3 is
   signal data: std_logic_vector(20 downto 0);
   --shared variable counterIn: integer := 0;
   shared variable counterOut: integer := 0;
   shared variable start: integer := 0;
begin
   process(clkIn, dataIn, clkToOut) is
   begin
      if rising_edge(clkIn) then

         data (20 downto 0) <= dataIn;
         start := 1;
      end if;

      if rising_edge(clkToOut) or falling_edge(clkToOut) then
         if start = 1 then
            clkOut <= clkToOut;
         else
            clkOut <= '0';
         end if;
      end if;

      if rising_edge(clkToOut) then

         if start = 1 then
            counterOut := counterOut + 1;
         end if;

         if counterOut > 7 then
            counterOut := 1;
         end if;

         if counterOut > 7 then
            counterOut := 1;
         end if;

         case counterOut is
            when 1 => dataOut <= data(20 downto 18);
            when 2 => dataOut <= data(17 downto 15);
            when 3 => dataOut <= data(14 downto 12);
```

```
                                    when 4 => dataOut <= data(11 downto
            9);
                                    when 5 => dataOut <= data(8 downto 6);
                                    when 6 => dataOut <= data(5 downto 3);
                                    when 7 => dataOut <= data(2 downto 0);
                                    when others => counterOut :=
            counterOut;
                                end case;


                            end if;

                            if start = 0 then
                                  RE <= '0';
                            end if;
                            if start = 1 then
                                  RE <= '1';
                            end if;

                        end process;

                    end Behavioral;
```

Втаблице 6.13.2показанлистингпроверки Сериалайзер 21-3 наязыкеVHDL

Таблица 6.13.2– Листингпроверки Сериалайзер 21-3 наязыкеVHDL

```
                    LIBRARY ieee;
                    USE ieee.std_logic_1164.ALL;

                    -- Uncomment the following library declaration if using
                    -- arithmetic functions with Signed or Unsigned values
                    --USE ieee.numeric_std.ALL;

                    ENTITY tb_seria IS
                    END tb_seria;

                    ARCHITECTURE behavior OF tb_seria IS

                        -- Component Declaration for the Unit Under Test (UUT)

                        COMPONENT seria_21in3
                        PORT(
                             dataIn : IN  std_logic_vector(20 downto 0);
                             clkIn : IN  std_logic;
                             dataOut : OUT  std_logic_vector(2 downto 0);
                             clkToOut : IN  std_logic;
                             clkOut : OUT  std_logic;
                             RE : OUT  std_logic
                            );
                        END COMPONENT;


                        --Inputs
                        signal dataIn : std_logic_vector(20 downto 0) :=
                        b"11100011100011000101";
```

```
                    signal clkIn : std_logic := '0';
                    signal clkToOut : std_logic := '0';

                    --Outputs
                    signal dataOut : std_logic_vector(2 downto 0);
                    signal clkOut : std_logic;
                    signal RE : std_logic;

                    -- Clock period definitions
                    constant clkIn_period : time := 10 ns;
                    constant clkToOut_period : time := 10 ns;
                    constant clkOut_period : time := 10 ns;

                BEGIN

                    -- Instantiate the Unit Under Test (UUT)
                    uut: seria_21in3 PORT MAP (
                           dataIn => dataIn,
                           clkIn => clkIn,
                           dataOut => dataOut,
                           clkToOut => clkToOut,
                           clkOut => clkOut,
                           RE => RE
                         );

                    -- Clock process definitions
                    clkIn_process :process
                    begin
                          clkIn <= '0';
                          wait for clkIn_period/2;
                          clkToOut <= '1';
                          wait for clkToOut_period/14;
                    end process;


                    -- Stimulus process
                    stim_proc: process
                    begin
                       -- hold reset state for 100 ns.
                       wait for 100 ns;

                       wait for clkIn_period*10;

                       -- insert stimulus here

                       wait;
                    end process;

                END;
```

В рисунке 6.13.1 показана временная диаграмма работы Сериалайзера 21-3 на языке VHDL

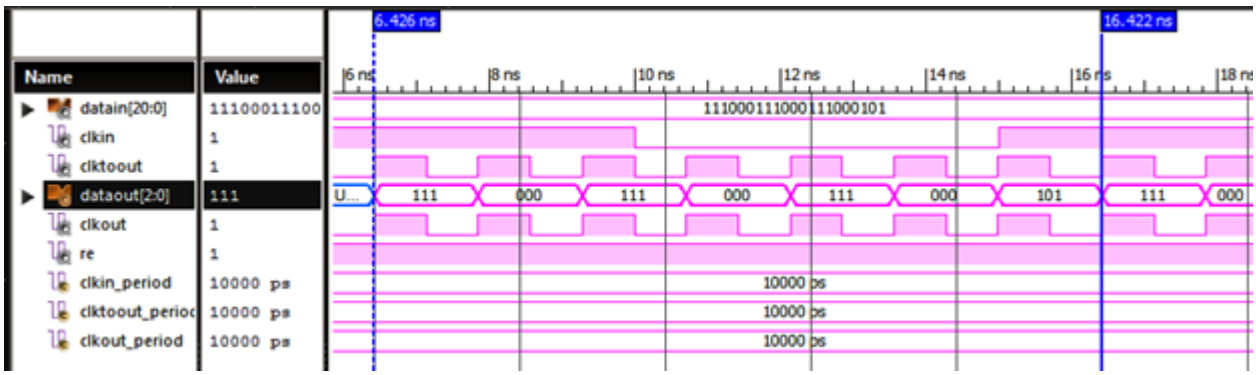


Рисунок 6.13.1 Временные диаграммы работы Сериалайзера 21-3 на языке VHDL

6.14 Сериазайзер 12-3 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 12-3 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 6.14.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 6.14.1– Листинг реализации Сериазайзер 12-3на языке Verilog

```
module s12x3(clk, Din, Dout, WE);
                input clk;
                input [11:0] Din;
                output [2:0] Dout;
                output reg WE;
                reg [2:0] Dout;
                integer cnt;
                initial cnt = 0;
                initial WE <= 0;
        always @(posedge clk)
        begin
                WE <= 0;
                case(cnt)
                        0: Dout <= Din[2:0];
                        1: Dout <= Din[5:3];
                       2: Dout <= Din[8:6];
                        3: Dout <= Din[11:9];
                endcase
                cnt = cnt + 1;
        if(cnt >=4 && WE==0)
        begin
                WE <= 1;
                cnt = 0;
        end
        end
                endmodule
```

В таблице 6.14.1 показан листинг проверкиСериазайзер 12-3 на языке Verilog

Таблица 6.14.1– Листинг проверки Сериазайзер 12-3на языке Verilog

```
                    module s12x3_tb;
            // Inputs
                        reg clk;
                        reg [11:0] Din;
                        // Outputs
                        wire [2:0] Dout;
                        wire WE;
                        // Instantiate the Unit Under Test (UUT)
                        s12x3 uut (
                              .clk(clk),
                              .Din(Din),
                              .Dout(Dout),
                              .WE(WE)
                        );
                        initial begin
                              // Initialize Inputs
                              clk = 1;
                          Din = 724;
                              // Wait 100 ns for global reset to finish
                              #100;
                              // Add stimulus here
                        End
                        always begin #5 clk = ~clk; end
      always begin #40 Din = Din + 11; end
                    endmodule
```

В рисунке 6.14.1 показана временная диаграмма работы Сериалайзера 12-3 на языке VHDL

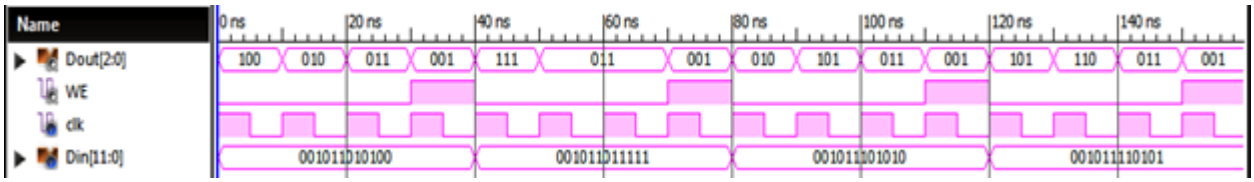


Рисунок 6.14.1 Временные диаграммы работы Сериазайзера 12-3 на языке Verilog

7 НЕКРАТНЫЕ СЕРИАЛАЙЗЕРЫ

7.1 Сериалайзер 5-2 Verilog

Запись в буфер осуществляется синхронно от тактового сигнала CinN, который задаётся в тестовой программе. Вывод данных Dout осуществляется синхронно CoutM, который является задержанным на нужное количество тактов CinM (определяется моментами записи 1-го значения в буфер и чтением из него на Dout).

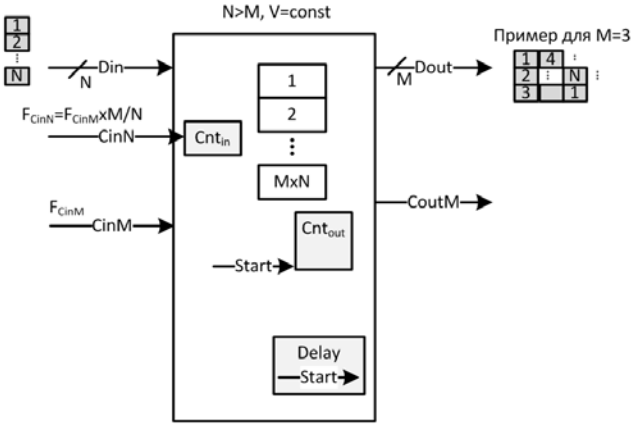


Рисунок 7.1.1 Принцип работы Некратного сериалайзера

То есть полностью синхронный автомат передачи данных без изменения скорости потока V=const. В тестовой программе периоды рекомендуется задавать в целочисленном виде, например для 11-3.hdl : # 11 CinN=~CinN; # 3 CinM=~CinM

В таблице 7.1.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 7.1.1– Листинг реализации Сериазайзер 12-3 на языке Verilog

```
`timescale 1ns / 1ps

module unserial5x2(  input wire [in-1:0] dataIn,
                     input wire clkIn,
                     input wire clkToOut,
                     output reg [out-1:0] dataOut,
                     output reg clkOut
    );

    parameter in = 5;
    parameter out = 2;

    reg [in*out-1:0] data = 0;
    integer counterIn = 0;
    integer counterOut = 0;
    integer start = 0;

    always @(posedge clkIn) begin
        counterIn = counterIn + 1;
        if (counterIn > out) begin counterIn = 1; end
        if (counterIn > 1) begin start = 1; end
        case (counterIn)
            1: data[4:0] = dataIn;
            2: data[9:5] = dataIn;
```

```
                        default: ;
                    endcase;
                end

                always @(posedge clkToOut or negedge clkToOut) begin
                    if (start == 1)
                        clkOut = clkToOut;
                    else
                        clkOut = 0;
                end

                always @(posedge clkToOut) begin
                    if (start == 1) begin
                        counterOut = counterOut + 1;
                        if (counterOut > in) begin counterOut = 1;
        end
                        case (counterOut)
                            1:    dataOut = data[1:0];
                            2: dataOut = data[3:2];
                            3: dataOut = data[5:4];
                            4: dataOut = data[7:6];
                            5: dataOut = data[9:8];
                            default: ;
                        endcase;
                    end
                end

            endmodule
```

В таблице 7.1.2 показан листинг проверкиСериазайзер 12-3 на языке Verilog

Таблица 7.1.2– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
            `timescale 1ns / 1ps

            module unserial5x2tb;

                // Inputs

                reg [4:0] dataIn;
                reg clkIn;
                reg clkToOut;

                // Outputs
                wire [1:0] dataOut;
                wire clkOut;

                // Instantiate the Unit Under Test (UUT)
                unserial5x2 uut (
                    .dataIn(dataIn),
                    .clkIn(clkIn),
                    .clkToOut(clkToOut),
                    .dataOut(dataOut),
                    .clkOut(clkOut)
                );

```

```
                    initial begin
                        // Initialize Inputs
                        dataIn = 5'b00100;
                        clkIn = 1;
                        clkToOut = 0;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                    end
                    always begin #5 clkIn = ~clkIn; end
                    always begin #2 clkToOut = ~clkToOut; end
                    always begin #9 dataIn = 5'b00100; end

                endmodule
```

В рисунке 7.1.2 показана временная диаграмма работы Сериалайзера 5-2 на языке Verilog

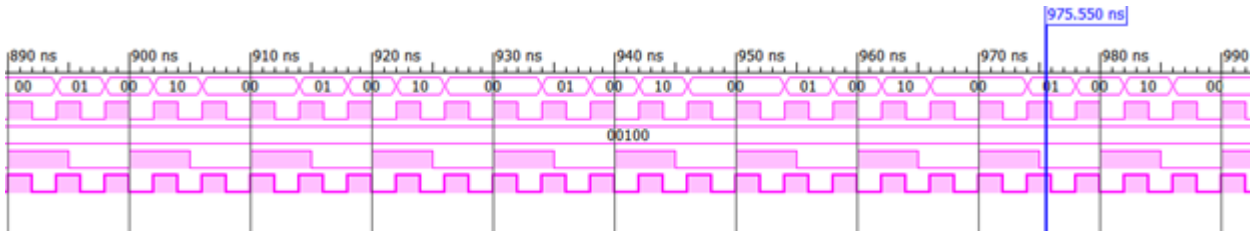


Рисунок 7.1.2 Временные диаграммы работы Сериалайзера 5-2 на языке Verilog

7.2 Сериалайзер 10-3 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 10-3 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.2.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 7.2.1– Листинг реализации Сериазайзер 12-3 на языке Verilog

```
                module s10x3(  input wire [in-1:0] dataIn,
                                    input wire clkIn,
                                    input wire clkToOut,
                                    output reg [out-1:0] dataOut,
                                    output reg clkOut
                    );

                parameter in = 10;
                parameter out = 3;

                reg [in*out-1:0] data = 0;
                integer counterIn = 0;
                integer counterOut = 0;
                integer start = 0;
```

```
                    always @(posedge clkIn) begin
                        counterIn = counterIn + 1;
                        if (counterIn > out) begin counterIn = 1; end
                        if (counterIn > 1) begin start = 1; end
                        case (counterIn)
                            1: data[9:0] = dataIn;
                            2: data[19:10] = dataIn;
                            3:    data[29:20] = dataIn;
                            default: ;
                        endcase;
                    end

                    always @(posedge clkToOut or negedge clkToOut) begin
                        if (start == 1)
                            clkOut = clkToOut;
                        else
                            clkOut = 0;
                    end
                    always @(posedge clkToOut) begin
                        if (start == 1) begin
                            clkOut = clkToOut;
                        else
                            clkOut = 0;
                    end

                    always @(posedge clkToOut) begin
                        if (start == 1) begin
                            counterOut = counterOut + 1;
                            if (counterOut > in) begin counterOut = 1;
            end
                            case (counterOut)
                                1:    dataOut = data[2:0];
                                2: dataOut = data[5:3];
                                3: dataOut = data[8:6];
                                4: dataOut = data[11:9];
                                5: dataOut = data[14:12];
                                6: dataOut = data[17:15];
                                7: dataOut = data[20:18];
                                8: dataOut = data[23:21];
                                9: dataOut = data[26:24];
                                10: dataOut = data[29:27];
                                default: ;
                            endcase;
                        end
                    end
                endmodule
```

В таблице 7.2.2 показан листинг проверкиСериазайзер 12-3 на языке Verilog

Таблица 7.2.2– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
                module s10x3_tb;

                // Inputs
                   reg [9:0] dataIn;
                   reg clkIn;
```

```
                reg clkToOut;

                // Outputs
                wire [2:0] dataOut;
                wire clkOut;
                // Instantiate the Unit Under Test (UUT)
                s10x3 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .clkToOut(clkToOut),
                        .dataOut(dataOut),
                        .clkOut(clkOut)
                );

                initial begin
                        // Initialize Inputs
                        clkIn = 1;
                        clkToOut = 0;

                        dataIn = 10'b0111000111;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                end

                always begin #10 clkIn = ~clkIn; end
                always begin #3 clkToOut = ~clkToOut; end

        endmodule
```

В рисунке 7.2.2 показана временная диаграмма работы Сериалайзера 10-3 на языке Verilog

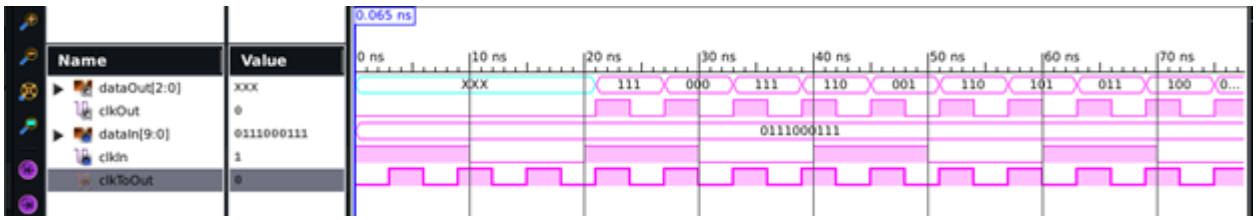


Рисунок 7.2.2 Временные диаграммы работы Сериалайзера 10-3 на языке Verilog

7.3 Сериалайзер 11-3 VHDL

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 11-3 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.3.1 показан листинг реализации Сериазайзер 12-3 на языке VHDL

Таблица 7.3.1– Листинг реализации Сериазайзер 12-3 на языке VHDL

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

entity s16s3 is
   port( dataIn: in bit_vector(10 downto 0);
               clkIn: in std_logic;
               dataOut: out bit_vector(2 downto 0);
               clkToOut: in std_logic;
               clkOut: out std_logic;
               RE: out bit
               );
end s16s3;
architecture Behavioral of s16s3 is
   signal data: bit_vector(32 downto 0);
   shared variable counterIn: integer := 0;
   shared variable counterOut: integer := 0;
   shared variable start: integer := 0;
begin
   process(clkIn, dataIn, clkToOut) is
   begin
         if rising_edge(clkIn) then
               counterIn := counterIn + 1;
               if counterIn> 3 then
                     counterIn := 1;
               end if;
               if counterIn> 1 then
                     start := 1;
               end if;
               case counterIn is
                     when 1 =>data(10 downto 0) <= dataIn;
                     when 2 =>data(21 downto 11) <= dataIn;
                     when 3 =>data(32 downto 22) <= dataIn;
                     when others =>counterIn := counterIn;
               end case;
         end if;

         if rising_edge(clkToOut) or falling_edge(clkToOut)
then
               if start = 1 then
                     clkOut<= clkToOut;
               else
                     clkOut<= '0';
               end if;
         end if;

         if rising_edge(clkToOut) then
               if start = 1 then
                     counterOut := counterOut + 1;
                     if counterOut> 11 then
                           counterOut := 1;
                     end if;
                     case counterOut is
                     when 1 =>dataOut<= data(2 downto 0);
```

```
                                    when 2 =>dataOut<= data(5 downto 3);
                                    when 3 =>dataOut<= data(8 downto 6);
                                    when 4 =>dataOut<= data(11 downto 9);
                                    when 5 =>dataOut<= data(14 downto 12);
                                    when 6 =>dataOut<= data(17 downto 15);
                                    when 7 =>dataOut<= data(20 downto 18);
                                    when 8 =>dataOut<= data(23 downto 21);
                                    when 9 =>dataOut<= data(26 downto 24);
                                    when  10  =>dataOut<=  data(29  downto
            27);
                                    when  11  =>dataOut<=  data(32  downto
            30);
                                    when     others     =>counterOut     :=
            counterOut;
                                    end case;
                              end if;
                        end if;

                        if start = 0 then
                              RE <= '0';
                        end if;
                        if start = 1 then
                              RE <= '1';
                        end if;

                  end process;
            endBehavioral;
```

В таблице 7.3.2 показан листинг проверкиСериазайзер 12-3 на языке VHDL

Таблица 7.3.2– Листинг проверкиСериазайзер 12-3 на языке VHDL

```
            module tb;

                  // Inputs
                  reg [10:0] dataIn;
                  regclkIn;
                  regclkToOut;

                  // Outputs
                  wire [2:0] dataOut;
                  wire clkOut;
                  wire RE;

                  // Instantiate the Unit Under Test (UUT)
                  s16s3 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .dataOut(dataOut),
                        .clkToOut(clkToOut),
                        .clkOut(clkOut),
                        .RE(RE)
                  );

                  initial begin
                        // Initialize Inputs
```

```
                                        dataIn = 11'b01010101010;
                                clkIn = 0;
                                clkToOut = 0;

                                // Wait 100 ns for global reset to finish
                                #100;

                                // Add stimulus here


                                // Wait 100 ns for global reset to finish
                                #100;

                        end

                        always begin #11 clkIn = ~clkIn; end
                        alwaysbegin #3 clkToOut = ~clkToOut; end

                endmodule
```

В рисунке 7.3.1 показана временная диаграмма работы Сериалайзера 11-3 на языке VHDL

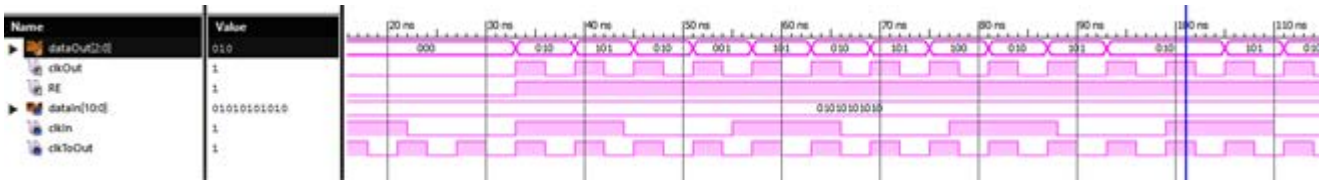


Рисунок 7.3.1 Временные диаграммы работы Сериалайзера 11-3 на языке VHDL

7.4 Сериалайзер 11-4 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 11-4 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.4.1 показан листинг реализации Сериазайзер 11-4 на языке Verilog

Таблица 7.4.1– Листинг реализации Сериазайзер 11-4 на языке Verilog

```
                module seri11_4(     input wire [in-1:0] dataIn,
                                                input wire clkIn,
                                                input wire clkToOut,
                                                output     reg     [out-1:0]
        dataOut,
                                                output reg clkOut
                );

                        parameter in = 11;
                        parameter out = 4;

                        reg [in*out-1:0] data = 0;
                        integer counterIn = 0;
```

```
                integer counterOut = 0;
                integer start = 0;

                always @(posedge clkIn) begin
                        counterIn = counterIn + 1;
                        if (counterIn > out) begin counterIn = 1; end
                        if (counterIn > 1) begin start = 1; end
                        case (counterIn)
                                1: data[10:0] = dataIn;
                                2: data[21:11] = dataIn;
                                3:    data[32:22] = dataIn;
                                4:    data[43:33] = dataIn;

                                default: ;
                        endcase;
                end

                always @(posedge clkToOut or negedge clkToOut) begin
                        if (start == 1)
                                clkOut = clkToOut;
                        else
                                clkOut = 0;
                end

                always @(posedge clkToOut) begin
                        if (start == 1) begin
                                counterOut = counterOut + 1;
                                if (counterOut > in) begin counterOut = 1;
        end
                                case (counterOut)
                                        1:    dataOut = data[3:0];
                                        2: dataOut = data[7:4];
                                        3: dataOut = data[11:8];
                                        4: dataOut = data[15:12];
                                        5: dataOut = data[19:16];
                                        6: dataOut = data[23:20];
                                        7: dataOut = data[27:24];
                                        8: dataOut = data[31:28];
                                        9: dataOut = data[35:32];
                                        10: dataOut = data[39:36];
                                        11: dataOut = data[43:40];

                                        default: ;
                                endcase;
                        end
                end
        endmodule
```

В таблице 7.4.2 показан листинг проверкиСериазайзер 11-4 на языке Verilog

Таблица 7.4.2– Листинг проверкиСериазайзер 11-4 на языке Verilog

```
        `timescale 1ns / 1ps

```

```
module seri11_4tb;

    // Inputs
    reg [10:0] dataIn;
    reg clkIn;
    reg clkToOut;

    // Outputs
    wire [3:0] dataOut;
    wire clkOut;

    // Instantiate the Unit Under Test (UUT)
    seri11_4 uut (
        .dataIn(dataIn),
        .clkIn(clkIn),
        .clkToOut(clkToOut),
        .dataOut(dataOut),
        .clkOut(clkOut)
    );

    initial begin
        // Initialize Inputs
        dataIn = 11'b10010010010;
        clkIn = 0;
        clkToOut = 0;
        #100;
    end   // Wait 100 ns for global reset to finish
    always   begin #11 clkIn = ~clkIn; end
    always   begin #4 clkToOut = ~clkToOut; end

        // Add stimulus here
Endmodule
```

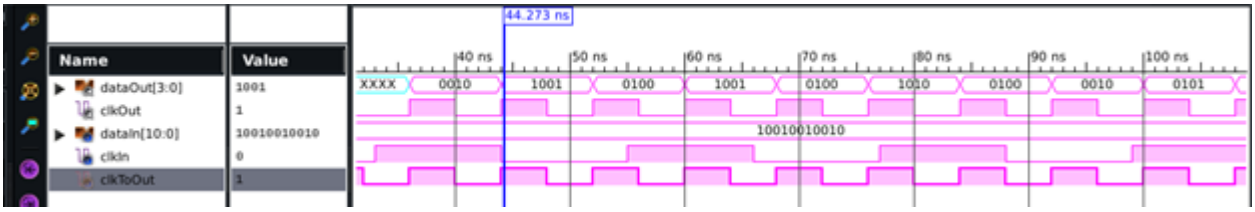


Рисунок 7.4.1 Временные диаграммы работы Сериалайзера 11-4 на языке Verilog

7.5 Сериазайзер 11-5 Verilog

<Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 11-5 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.>

В таблице 6.14.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 6.14.1– Листинг реализации Сериазайзер 12-3на языке Verilog

```
module s11x5(input wire [in-1:0] dataIn,
                                 input wire clkIn,
                                 input wire clkToOut,
                                 output reg [out-1:0] dataOut,
```

```
                                        output reg clkOut
    );
  parameter in = 11;
  parameter out = 5;
  reg [in*out-1:0] data = 0;
            integer counterIn = 0;
  integer start = 0;
  always @(posedge clkIn) begin
      counterIn = counterIn + 1;
    if (counterIn > out) begin counterIn = 1; end
      if (counterIn > 1) begin start = 1; end
      case (counterIn)
          1: data[10:0] = dataIn;
          2: data[21:11] = dataIn;
          3:    data[32:22] = dataIn;
         4:     data[43:33] = dataIn;
          5:    data[54:44] = dataIn;
      default: ;
      endcase;
  end
  always @(posedge clkToOut or negedge clkToOut) begin
      if (start == 1)
          clkOut = clkToOut;
      else
          clkOut = 0;
  end
  always @(posedge clkToOut) begin
      if (start == 1) begin
          counterOut = counterOut + 1;
          if (counterOut > in) begin counterOut = 1;
end
          case (counterOut)
              1:    dataOut = data[4:0];
              2: dataOut = data[9:5];
  3: dataOut = data[14:10];
              4: dataOut = data[19:15];
              5: dataOut = data[24:20];
              6: dataOut = data[29:25];
              7: dataOut = data[34:30];
              8: dataOut = data[39:35];
              9: dataOut = data[44:40];
              10: dataOut = data[49:45];
              11: dataOut = data[54:50];
              default: ;
          endcase;
      end
  end
endmodule
```

В таблице 7.14.1 показан листинг проверкиСериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
module s11x5_tb;
            // Inputs
  reg [10:0] dataIn;
```

```
                reg clkIn;
                reg clkToOut;
                // Outputs
                wire [4:0] dataOut;
                wire clkOut;
                // Instantiate the Unit Under Test (UUT)
                s11x5 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .clkToOut(clkToOut),
                        .dataOut(dataOut),
                        .clkOut(clkOut)
                );
                initial begin
                        // Initialize Inputs
                        clkIn = 1;
                        clkToOut = 0;
                        dataIn = 11'b10011100100;
                        // Wait 100 ns for global reset to finish
                        #100;
                        // Add stimulus here
                end
                always begin #11 clkIn = ~clkIn; end
                always begin #5 clkToOut = ~clkToOut; end
                always begin #22 dataIn = dataIn*2; end
        endmodule
```

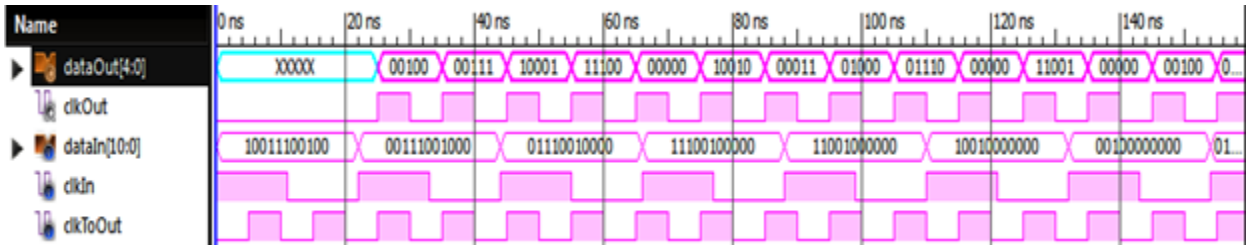


Рисунок 7.4 Временные диаграммы работы Сериазайзера 11-5 на языке Verilog

7.6 Сериалайзер 13-2 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 13-2 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.14.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 12-3 на языке Verilog

```
                `timescale 1ns / 1ps

                module s16s3(  input wire [in-1:0] dataIn,
                                        input wire clkIn,
                                        input wire clkToOut,
                                        output reg [out-1:0] dataOut,
                                        output reg clkOut
                 );
```

```
                        parameter in = 13;
                        parameter out = 2;

                        reg [in*out-1:0] data = 0;
                        integer counterIn = 0;
                        integer counterOut = 0;
                        integer start = 0;

                        always @(posedge clkIn) begin
                              counterIn = counterIn + 1;
                              if (counterIn > out) begin counterIn = 1; end
                              if (counterIn > 1) begin start = 1; end
                              case (counterIn)
                                    1: data[12:0] = dataIn;
                                    2: data[25:13] = dataIn;
                                    default: ;
                              endcase;
                        end

                        always @(posedge clkToOut or negedge clkToOut) begin
                              if (start == 1)
                                    clkOut = clkToOut;
                              else
                                    clkOut = 0;
                        end

                        always @(posedge clkToOut) begin
                              if (start == 1) begin
                                    counterOut = counterOut + 1;
                                    if (counterOut > in) begin counterOut = 1;
                  end
                                    case (counterOut)
                                          1:    dataOut = data[1:0];
                                          2: dataOut = data[3:2];
                                          3: dataOut = data[5:4];
                                          4: dataOut = data[7:6];
                                          5: dataOut = data[9:8];
                                          6: dataOut = data[11:10];
                                          7: dataOut = data[13:12];
                                          8: dataOut = data[15:14];
                                          9: dataOut = data[17:16];
                                          10: dataOut = data[19:18];
                                          11: dataOut = data[21:20];
                                          12: dataOut = data[23:22];
                                          13: dataOut = data[25:24];

                                          default: ;
                                    endcase;
                              end
                        end
                  endmodule
```

В таблице 7.14.1 показан листинг проверкиСериазайзер 12-3 на языке Verilog
Таблица 7.14.1– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
`timescale 1ns / 1ps

module s16s3_tb;

// Inputs
reg [12:0] dataIn;
reg clkIn;
reg clkToOut;

// Outputs
wire [1:0] dataOut;
wire clkOut;

// Instantiate the Unit Under Test (UUT)
s16s3 uut (
	.dataIn(dataIn),
	.clkIn(clkIn),
	.clkToOut(clkToOut),
	.dataOut(dataOut),
	.clkOut(clkOut)
);

initial begin
	// Initialize Inputs
	dataIn = 0;
	clkIn = 0;
	clkToOut = 0;

	dataIn = 13'b0010011100100;

	// Wait 100 ns for global reset to finish
	#100;

	// Add stimulus here

end

always begin #13 clkIn = ~clkIn; end
always begin #2 clkToOut = ~clkToOut; end

endmodule
```

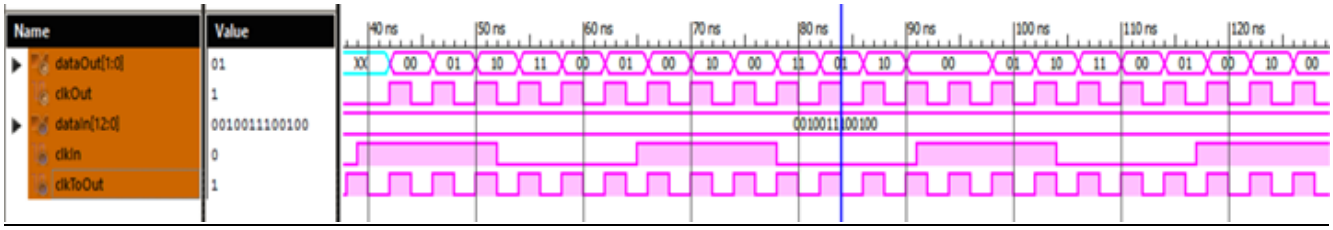


Рисунок 7.5 Временные диаграммы работы Сериалайзера 13-2 на языке Verilog

7.7 Сериалайзер 14-3 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 14-3 используется для создания высокоскоростных

вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.14.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 12-3 на языке Verilog

```
`timescale 1ns / 1ps
module p14p3(
    input wire clk,
    input wire [13:0] dataIn,
    output reg [2:0] dataOut,
    output reg [0:0] re,
    output reg [0:0] clkout
    );
    initial re=1'b0;
    reg [3:0] cnt1 = 4'b0000;
    reg [1:0] cnt2 = 2'b00;
    reg [13:0] dataInBuff = 0;
    reg [0:0] start=0;

    always @(posedge clk) begin
        cnt1 <= cnt1 + 1;
        if (start==1) clkout=clk;
        end
    always @(posedge clk)
        cnt2 <= cnt2 + 1;

    always @* begin
        if(cnt1 == 14)cnt1 <= 1'b0;
        if(cnt2 == 3)cnt2 <= 1'b0;

        re <= 1'd0;
      case (cnt1)
            0: if(cnt2 == 2)
                begin
                    dataOut[2] = dataIn[0];

                    dataOut[1:0]                    =
dataInBuff[13:12];
                end
            1: if(cnt2 == 2)
                begin
                    dataOut[2:1] = dataIn[1:0];

                    dataOut[0] = dataInBuff[13];
                end
          default:
                if(cnt2 == 2)
                    begin
                        //x = cnt1;
                        //y = cnt1-2;
                        dataOut[2] = dataIn[cnt1];
                        dataOut[1]  =  dataIn[cnt1-
1];
                        dataOut[0]  =  dataIn[cnt1-
2];
```

```
                                    end
                endcase
                    if(cnt1 == 13 && clk==0)
                    begin
                          dataInBuff<=dataIn;
                          re <= 1'd1;
                    end
                    if (start==1) clkout=clk;
                    if (start==0 && cnt1==1) start = 1;
             end

        endmodule
```

В таблице 7.14.1 показан листинг проверкиСериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
        `timescale 1ns / 1ps
        module p14p3_tb;

            // Inputs
            reg clk;
            reg [13:0] dataIn;

            // Outputs
            wire [2:0] dataOut;
            wire [0:0] re;
            wire [0:0] clkout;

            // Instantiate the Unit Under Test (UUT)
            p14p3 uut (
                  .clk(clk),
                  .dataIn(dataIn),
                  .dataOut(dataOut),
                  .re(re),
                  .clkout(clkout)
            );

                  initial begin
                  // Initialize Inputs
                  clk = 0;
                  dataIn = 14'b11010100001101;

                  // Wait 100 ns for global reset to finish
                  #100;

                  // Add stimulus here

            end
               always begin #20 clk=~clk; end
        endmodule
```

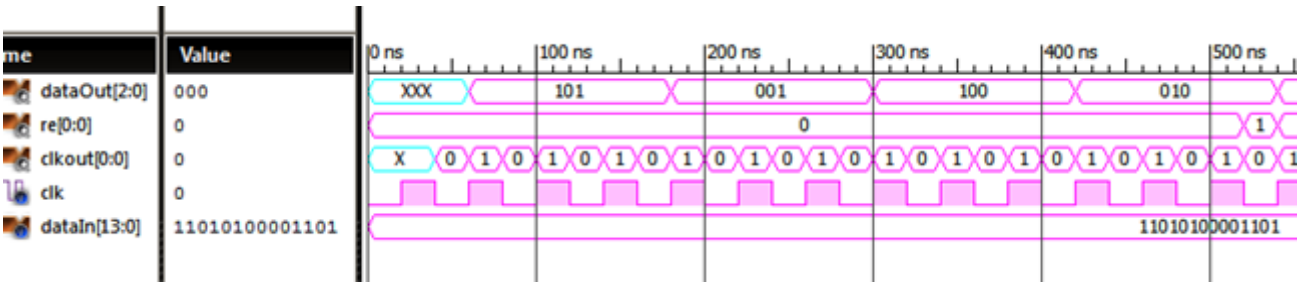


Рисунок 7.6 Временные диаграммы работы Сериалайзера 14-3 на языке Verilog

В таблице 7.14.1 показан листинг реализации Сериазайзер 12-3 на языке VHDL

Таблица 7.14.1– Листинг реализации Сериазайзер 12-3 на языке VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity s14s3 is
port( dataIn: in bit_vector(13 downto 0);
clkIn: in std_logic;
dataOut: out bit_vector(2 downto 0);
clkToOut: in std_logic;
clkOut: out std_logic;RE: out bit);
end s14s3;
architecture Behavioral of s14s3 is signal data: bit_vector(41
downto 0)
      shared variable counterIn: integer := 0;
      shared variable counterOut: integer := 0;
      shared variable start: integer := 0;
begin
            process(clkIn, dataIn, clkToOut) is begin if
            rising_edge(clkIn)
            then counterIn := counterIn + 1;
            if counterIn > 3 then counterIn := 1;
end if;
            if counterIn > 1 then start := 1;end if;
case counterIn is
      when 1 => data(13 downto 0) <= dataIn;when 2 => data(27 downto
      14) <= dataIn;
      when 3 => data(41 downto 28) <= dataIn;when others =>
      counterIn := counterIn;
end case;
end if;
if rising_edge(clkToOut) or falling_edge(clkToOut) then if start =
1 then clkOut <= clkToOut;else clkOut <= '0';end if;
end if;
if rising_edge(clkToOut) then if start = 1 then counterOut :=
1 then clkOut <= clkToOut;else clkOut <= '0';end if;
end if;
if rising_edge(clkToOut) then if start = 1 then counterOut :=
counterOut + 1;
if counterOut > 14 then counterOut := 1;
end if;
      case counterOut is
            when 1 => dataOut <= data(2 downto 0);
            when 2 => dataOut <= data(5 downto 3);
            when 3 => dataOut <= data(8 downto 6);
            when 4 => dataOut <= data(11 downto 9);
```

```
                when 5 => dataOut <= data(14 downto 12);
                when 6 => dataOut <= data(17 downto 15);
                when 7 => dataOut <= data(20 downto 18);
                when 8 => dataOut <= data(23 downto 21);
                when 9 => dataOut <= data(26 downto 24);
                when 10 => dataOut <= data(29 downto 27);
                when 11 => dataOut <= data(32 downto 30);
                when 12 => dataOut <= data(35 downto 33);
                when 13 => dataOut <= data(38 downto 36);
                when 14 => dataOut <= data(41 downto 39);
                when others => counterOut := counterOut;
                end case;end if;
          end if;
    if start = 0 then RE <= '0';
    end if;
    if start = 1 then RE <= '1';
    end if;
    end process;
    end Behavioral;
```

В таблице 7.14.1 показан листинг проверкиСериазайзер 12-3 на языке VHDL

Таблица 7.14.1– Листинг проверкиСериазайзер 12-3 на языке VHDL

```
    module tb;
    // Inputsreg [13:0] dataIn;
    reg clkIn;reg clkToOut;
        // Outputswire [2:0] dataOut;
        wire clkOut;
        wire RE;// Instantiate the Unit Under Test (UUT)s14s3 uut
        (.dataIn(dataIn),
        .clkIn(clkIn),.dataOut(dataOut),.clkToOut(clkToOut),
        .clkOut(clkOut),.RE(RE));
        initial begin// Initialize InputsdataIn = 14'b11111110101010;
        clkIn = 1;clkToOut = 0;
        // Wait 100 ns for global reset to finis#100;
        // Add stimulus here
        // Wait 100 ns for global reset to finish#100;
        Endalways begin #14 clkIn = ~clkIn; endalways begin #3 clkToOut
        = ~clkToOut; end
        endmodule
```

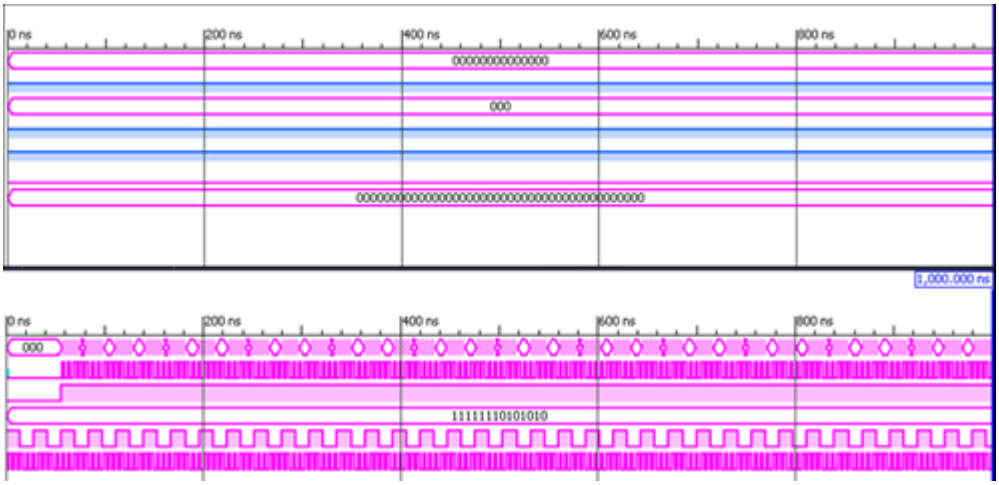


Рисунок 7.7 Временные диаграммы работы Сериазайзера 14-3 на языке VHDL

7.8 Сериазайзер 14-5 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 14-5 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.14.1 показан листинг реализации Сериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 12-3 на языке Verilog

```
`timescale 1ns / 1ps
module ser14_5(
                                    input wire [in-1:0] dataIn,
                                    input wire clkIn,
                                    input wire clkToOut,
                                    output reg [out-1:0] dataOut,
                                    output reg clkOut
 );
parameter in = 14;
parameter out = 5;
reg [in*out-1:0] data = 0;
          integer counterIn = 0;
integer counterOut = 0;
integer start = 0;
always @(posedge clkIn) begin
      counterIn = counterIn + 1;
      if (counterIn > out) begin counterIn = 1; end
      if (counterIn > 1) begin start = 1; end
      case (counterIn)
            1: data[13:0] = dataIn;
            2: data[27:14] = dataIn;
            3:    data[41:28] = dataIn;
            4:    data[55:42] = dataIn;
            5:    data[69:56] = dataIn;
      default: ;
      endcase;
end
always @(posedge clkToOut or negedge clkToOut) begin
      if (start == 1)
            clkOut = clkToOut;
      else
            clkOut = 0;
end
always @(posedge clkToOut) begin
      if (start == 1) begin
            counterOut = counterOut + 1;
            if (counterOut > in) begin counterOut = 1;
end
            case (counterOut)
                  1:    dataOut = data[4:0];
                  2: dataOut = data[9:5];
                  3: dataOut = data[14:10];
                  4: dataOut = data[19:15];
                  5: dataOut = data[24:20];
```

```
                                    6: dataOut = data[29:25];
                                    7: dataOut = data[34:30];
                                    8: dataOut = data[39:35];
                                    9: dataOut = data[44:40];
                                    10: dataOut = data[49:45];
                                    11: dataOut = data[54:50];
                                    12: dataOut = data[59:55];
                                    13: dataOut = data[64:60];
                                    14: dataOut = data[69:65];
                                    default: ;
                              endcase;
                        end
                  end
            endmodule
```

В таблице 7.14.1 показан листинг проверки Сериазайзер 12-3 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 12-3 на языке Verilog

```
            `timescale 1ns / 1ps
            module ser14_5_test;

                  // Inputs
                  reg [13:0] dataIn;
                  reg clkIn;
                  reg clkToOut;

                  // Outputs
                  wire [4:0] dataOut;
                  wire clkOut;

                  // Instantiate the Unit Under Test (UUT)
                  ser14_5 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .clkToOut(clkToOut),
                        .dataOut(dataOut),
                        .clkOut(clkOut)
                  );

                  initial begin
                        // Initialize Inputs
                        dataIn = 14'b01000001101001;
                        clkIn = 1;
                        clkToOut = 0;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                  end
                  always begin #11 clkIn = ~clkIn; end
                  always begin #5 clkToOut = ~clkToOut; end
                  always begin #22 dataIn = dataIn + 1; end
```

```
endmodule
```

Рисунок 7.8 Временные диаграммы работы Сериазайзера 14-5 на языке Verilog

7.9 Сериалайзер 16-3 Verilog

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Сериалайзер 16-3 используется для создания высокоскоростных вычислительных сетей и компьютерных шин. Они позволяют передавать информацию на высоких частотах при помощи дешевых соединений на основе медной витой пары.

В таблице 7.14.1 показан листинг реализации Сериазайзер 16-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 16-3 на языке Verilog

```
`timescale 1ns / 1ps

module s16s3(  input wire [in-1:0] dataIn,
                                   input wire clkIn,
                                   input wire clkToOut,
                                   outputreg [out-1:0] dataOut,
                                   outputregclkOut,
                                   outputreg RE
    );

    parameter in = 16;
    parameter out = 3;

    reg [in*out-1:0] data = 0;
    integercounterIn = 0;
    integercounterOut = 0;
    integer start = 0;

    always @(posedgeclkIn) begin
          counterIn = counterIn + 1;
          if (counterIn> out) begin counterIn = 1; end
          if (counterIn> 1) begin start = 1; end
          case (counterIn)
                1: data[15:0] = dataIn;
                2: data[31:16] = dataIn;
                3: data[47:32] = dataIn;
                default: ;
```

```
                endcase;
            end

            always @(posedgeclkToOut or negedgeclkToOut) begin
                if (start == 1)
                    clkOut = clkToOut;
                else
                    clkOut = 0;
            end

            always @(posedgeclkToOut) begin
                if (start == 1) begin
                    counterOut = counterOut + 1;
                    if (counterOut> in) begin counterOut = 1;
end
                    case (counterOut)
                        1:    dataOut = data[2:0];
                        2: dataOut = data[5:3];
                        3: dataOut = data[8:6];
                        4: dataOut = data[11:9];
                        5: dataOut = data[14:12];
                        6: dataOut = data[17:15];
                        7: dataOut = data[20:18];
                        8: dataOut = data[23:21];
                        9: dataOut = data[26:24];
                        10: dataOut = data[29:27];
                        11: dataOut = data[32:30];
                        12: dataOut = data[35:33];
                        13: dataOut = data[38:36];
                        14: dataOut = data[41:39];
                        15: dataOut = data[44:42];
                        16: dataOut = data[47:45];
                        default: ;
                    endcase;
                end
            end

            always @*
            begin
                if(start == 1) begin RE <= 1; end
                else begin RE <= 0; end
            end
        endmodule
```

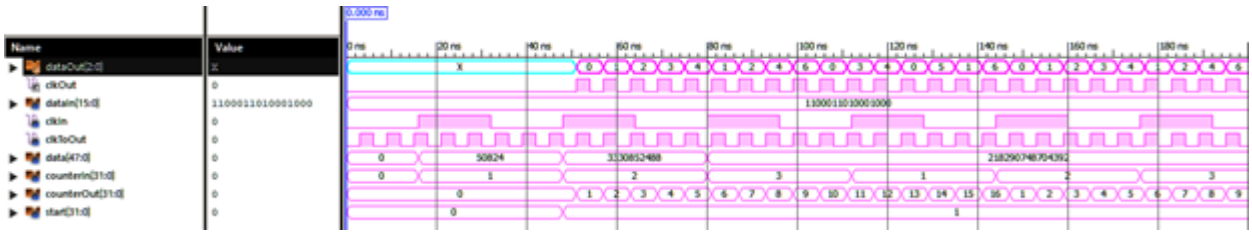


Рисунок 7.9 Временные диаграммы работы Сериалайзера 16-3 на языке Verilog

В таблице 7.14.1 показан листинг реализации Сериазайзер 16-3 на языке VHDL

Таблица 7.14.1– Листинг реализации Сериазайзер 16-3 на языке VHDL

```
        library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity s16s3 is
   port( dataIn: in bit_vector(15 downto 0);
         clkIn: in std_logic;
         dataOut: out bit_vector(2 downto 0);
         clkToOut: in std_logic;
         clkOut: out std_logic;
         RE: out bit
         );
end s16s3;

architecture Behavioral of s16s3 is
   signal data: bit_vector(47 downto 0);
   shared variable counterIn: integer := 0;
   shared variable counterOut: integer := 0;
   shared variable start: integer := 0;
begin
   process(clkIn, dataIn, clkToOut) is
   begin
      if rising_edge(clkIn) then
         counterIn := counterIn + 1;
         if counterIn > 3 then
            counterIn := 1;
         end if;
         if counterIn > 1 then
            start := 1;
         end if;
         case counterIn is
            when 1 =>data(15 downto 0) <= dataIn;
            when 2 =>data(31 downto 16) <= dataIn;
            when 3 =>data(47 downto 32) <= dataIn;
            when others =>counterIn := counterIn;
         end case;
      end if;

      if rising_edge(clkToOut) or falling_edge(clkToOut)
then
         if start = 1 then
            clkOut <= clkToOut;
         else
            clkOut <= '0';
         end if;
      end if;

      if rising_edge(clkToOut) then
         if start = 1 then
            counterOut := counterOut + 1;
            if counterOut > 16 then
               counterOut := 1;
            end if;
            case counterOut is
               when 1 => dataOut <= data(2
downto 0);
               when 2 => dataOut <= data(5
```

```
                downto 3);
                                        when  3  =>  dataOut  <=  data(8
                downto 6);
                                        when  4  =>  dataOut  <=  data(11
                downto 9);
                                        when  5  =>  dataOut  <=  data(14
                downto 12);
                                        when  6  =>  dataOut  <=  data(17
                downto 15);
                                        when  7  =>  dataOut  <=  data(20
                downto 18);
                                        when  8  =>  dataOut  <=  data(23
                downto 21);
                                        when  9  =>  dataOut  <=  data(26
                downto 24);
                                        when  10  =>  dataOut  <=  data(29
                downto 27);
                                        when  11  =>  dataOut  <=  data(32
                downto 30);
                                        when  12  =>  dataOut  <=  data(35
                downto 33);
                                        when  13  =>  dataOut  <=  data(38
                downto 36);
                                        when  14  =>  dataOut  <=  data(41
                downto 39);
                                        when  15  =>  dataOut  <=  data(44
                downto 42);
                                        when  16  =>  dataOut  <=  data(47
                downto 45);
                                        when   others   =>counterOut   :=
                counterOut;
                                    end case;
                                end if;
                            end if;

                            if start = 0 then
                                RE <= '0';
                            end if;
                            if start = 1 then
                                RE <= '1';
                            end if;

                        end process;
                    end Behavioral;
```

Для проверки работоспособности модуля на входы преобразователя поступают три сигнала: тактирование clkIn и clkToOut и данные dataIn. Тактирование происходит с частотой 1/16 и ⅓, что соответствует числу входов и выходов устройства. На вход данных поступает заранее заданная комбинация нулей и единиц, а именно 1100011010001000, которая соответствует последовательности 0,1,2,3 и т.д. Также для проверки тактирование может подаваться комбинация из чередования нулей и единиц.

В таблице 7.14.1 показан листинг проверкиСериазайзер 16-3 на языке VHDL

Таблица 7.14.1– Листинг проверкиСериазайзер 16-3 на языке VHDL

```
                    initial begin
                        // Initialize Inputs
```

```
                dataIn = 0;
                clkIn = 0;
                clkToOut = 0;

                dataIn = 16'b1100011010001000;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

            end

            always begin #16 clkIn = ~clkIn; end
            always begin #3 clkToOut = ~clkToOut; end
```

При разработке модуля появилась необходимость проконтролировать состояние внутренних счётчиков и буфера данных, для этого код модуля изначально был написан на языке Verilog, после чего был перенесён на язык VHDL.

Рисунок 7.10.1 отражает работу модуля на языке Verilog. Согласно данной временной диаграмме можно отследить изменение внутренних счетчиков и заполнение буфера данных. На рисунке 7.10.2 представлена временная диаграмма работы модуля на языке VHDL, которая полностью соответствует результату с рисунка 7.10.1.

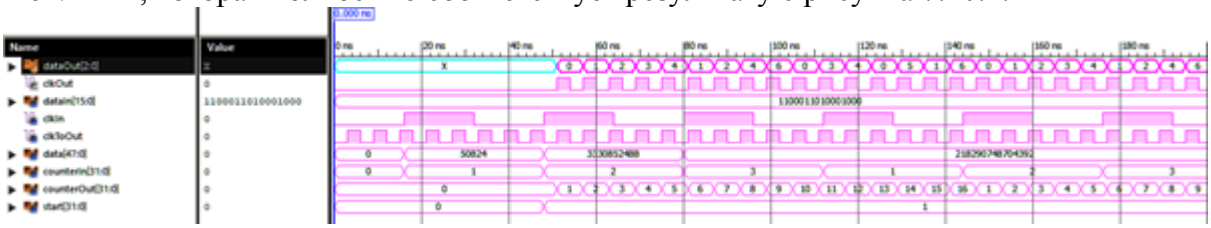


Рисунок 7.10.1 Временная диаграмма работы преобразователя на языкеVerilog

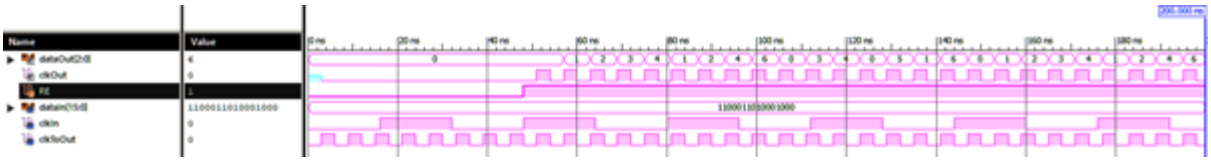


Рисунок 7.10.2 Временная диаграмма работы преобразователя на языкеVHDL

Задача

На основе существующей программы реализовать интерфейс CAN. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

На рисунке 7.10.3 отражена проверка тактирования модуля, а именно поступление данных на выход при их поступлении на вход устройства. На рисунке 7.10.4 представлена временная диаграмма работы модуля в соответствии с кодом отладки, которая подтверждает, что выходная комбинация полностью соответствует входной.

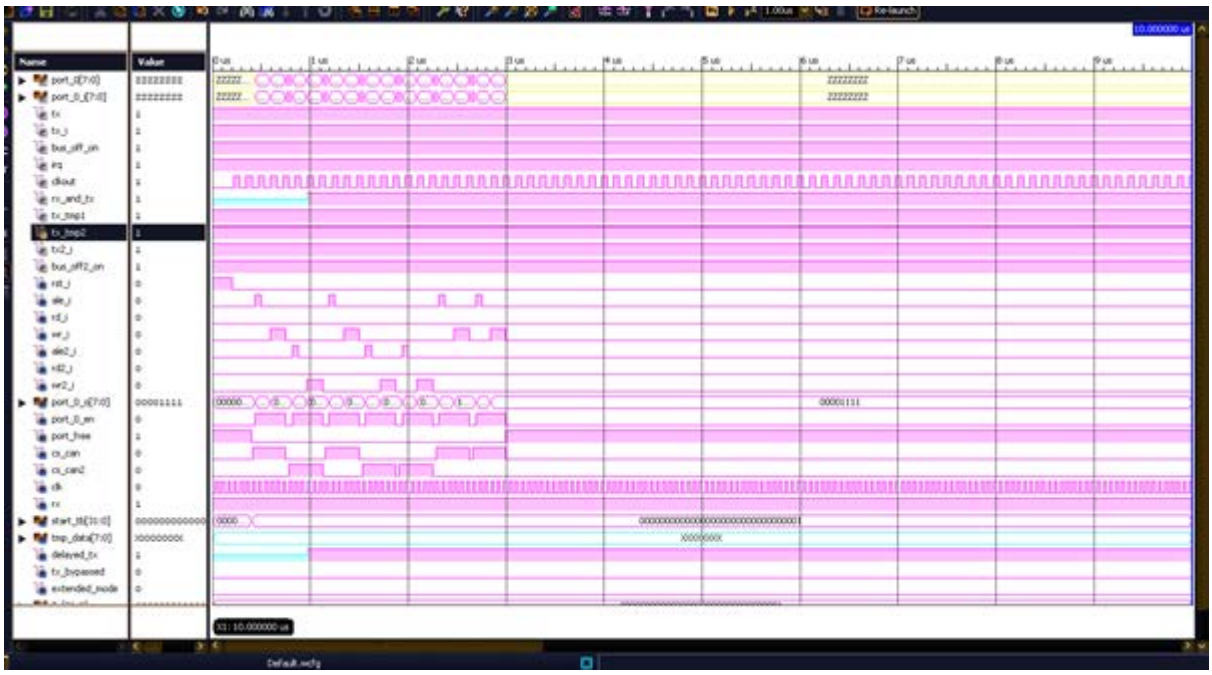


Рисунок 7.10.3 Временная диаграмма работы интерфейса

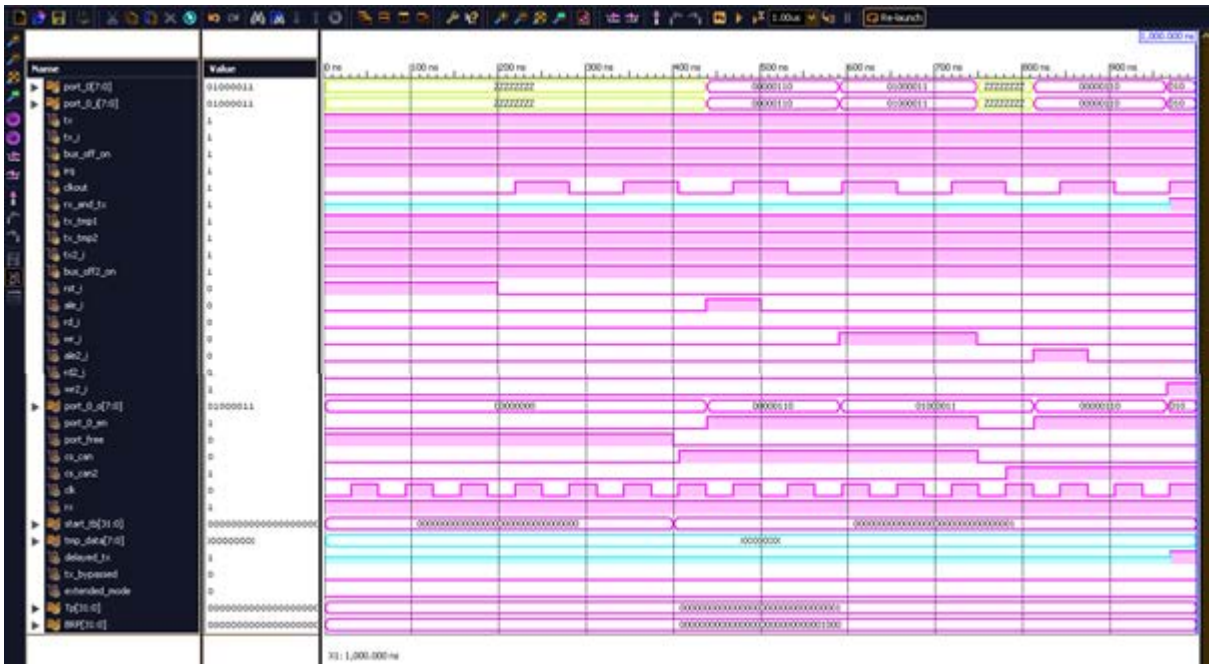


Рисунок 7.10.3 Временная диаграмма работы интерфейса

Сериалайзер 17-2 VHDL

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=17, n=2. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 17 информационных входов и 2 входа тактирования: clkIn - тактирование входных данных и clkToOut - тактирование выходных данных, и 2 информационных выхода, 1 выход clkOut для тактирования выходных данных и выход RE, разрешающий чтение.

В таблице 7.14.1 показан листинг реализации Сериазайзер 17-2 на языке VHDL

Таблица 7.14.1– Листинг реализации Сериазайзер 17-2 на языке VHDL

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

entity ser17v2 is
   port( dataIn: in bit_vector(16 downto 0);
         clkIn: in std_logic;
         dataOut: out bit_vector(1 downto 0);
         clkToOut: in std_logic;
         clkOut: out std_logic;
         RE: out bit
         );
end ser17v2;

architecture Behavioral of ser17v2 is
   signal data: bit_vector(33 downto 0);
   shared variable counterIn: integer := 0;
   shared variable counterOut: integer := 0;
   shared variable start: integer := 0;
begin
   process(clkIn, dataIn, clkToOut) is
   begin
      ifrising_edge(clkIn) then
         counterIn := counterIn + 1;
         ifcounterIn> 2 then
            counterIn := 1;
         end if;
         ifcounterIn> 1 then
            start := 1;
         end if;
         casecounterIn is
            when 1 => data(16 downto 0) <= dataIn;
            when 2 => data(33 downto 17) <= dataIn;
            --when 3 =>data(47 downto 32) <= dataIn;
            when others =>counterIn := counterIn;
         end case;
      end if;

      ifrising_edge(clkToOut) or falling_edge(clkToOut) then
         if start = 1 then
            clkOut<= clkToOut;
         else
            clkOut<= '0';
         end if;
      end if;
```

```
                    ifrising_edge(clkToOut) then
                        if start = 1 then
                            counterOut := counterOut + 1;
                            ifcounterOut> 17 then
                                counterOut := 1;
                            end if;
                            casecounterOut is
                                when 1 =>dataOut<= data(1 downto 0);
                                when 2 =>dataOut<= data(3 downto 2);
                                when 3 =>dataOut<= data(5 downto 4);
                                when 4 =>dataOut<= data(7 downto 6);
                                when 5 =>dataOut<= data(9 downto 8);
                                when  6  =>dataOut<=  data(11 downto 10);
                                when  7  =>dataOut<=  data(13 downto 12);
                                when  8  =>dataOut<=  data(15 downto 14);
                                when  9  =>dataOut<=  data(17 downto 16);
                                when  10  =>dataOut<=  data(19 downto 18);
                                when  11  =>dataOut<=  data(21 downto 20);
                                when  12  =>dataOut<=  data(23 downto 22);
                                when  13  =>dataOut<=  data(25 downto 24);
                                when  14  =>dataOut<=  data(27 downto 26);
                                when  15  =>dataOut<=  data(29 downto 28);
                                when  16  =>dataOut<=  data(31 downto 30);
                                when  17  =>dataOut<=  data(33 downto 32);
                                when  others  =>counterOut  :=  counterOut;
                            end case;
                        end if;
                    end if;

                    if start = 0 then
                        RE <= '0';
                    end if;
                    if start = 1 then
                        RE <= '1';
                    end if;

                end process;
            endBehavioral;
```

В соответствии с тестовым воздействием на входы преобразователя поступают три сигнала: тактирование clkIn и clkToOut и данные dataIn. Соотношение тактовых частот

составляет 2/17. На вход данных поступает заранее заданная комбинация нулей и единиц, 00100111001001111, фактически представляющая собой последовательность 0, 1, 2, 3 и т.д. в двоичной системе счисления.

В таблице 7.14.1 показан листинг проверкиСериазайзер 17-2 на языке VHDL

Таблица 7.14.1– Листинг проверкиСериазайзер 17-2 на языке VHDL

```
                    initial begin
                            // Initialize Inputs
                            dataIn = 0;
                            clkIn = 0;
                            clkToOut = 0;


                            dataIn = 17'b11110010011100100;

                            // Wait 100 ns for global reset to finish
                            #100;

                    end

                    always begin #17 clkIn = ~clkIn; end
                    always begin #2 clkToOut = ~clkToOut; end
```

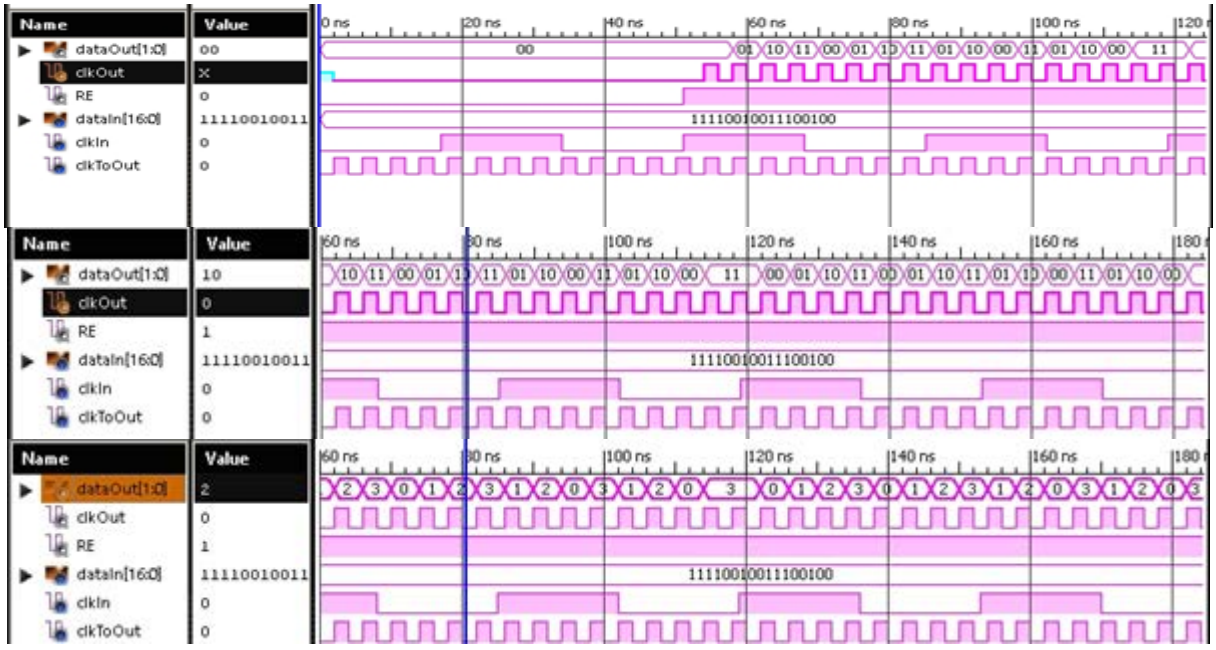


Рисунок 7.11 Временные диаграммы работы Сериалайзера 17-2 на языке VHDL

7.10 Сериалайзер 17-3 Verilog

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для

синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=17, n=3. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 17 информационных входов и 3 входа тактирования: clkIn - тактирование входных данных и clkToOut - тактирование выходных данных, и 2 информационных выхода, 1 выход clkOut для тактирования выходных данных и выход RE, разрешающий чтение.

В таблице 7.14.1 показан листинг реализации Сериазайзер 17-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 17-3 на языке Verilog

```
module seventeen_three

                    (Cin, Din, Dout, buff, start, cnt);
                    input [16:0] Din ;
                    output reg [2:0] Dout=0;
                    input Cin;
                    output reg [50:0] buff=0;
                    output reg [7:0] cnt;

                    output reg start;

                    initial cnt=0;
                    initial start=0;


                    always @(posedgeCin)
                    begin
                    case (cnt)
                    8'b000: buff [16:0]=Din;
                    8'b1011: buff [33:17]=Din;
                    8'b10110: buff [50:34]=Din;

                    endcase

                if (start==1)
                case (cnt)
                8'b1000: Dout [2:0]=buff[2:0];//8
                8'b1100: Dout [2:0]=buff[5:3];//12
                8'b10100: Dout [2:0]=buff[8:6];//16
                8'b10000: Dout [2:0]=buff[11:9];//20
                8'b11000: Dout [2:0]=buff[14:12];//24
                8'b11100: Dout [2:0]=buff[17:15];//28
                8'b100000: Dout [2:0]=buff[20:18];//32

                8'b1000: Dout [2:0]=buff[29:27];//0
                8'b0100: Dout [2:0]=buff[32:30];//4
                8'b0100: Dout [2:0]=buff[35:33];//4
                8'b1000: Dout [2:0]=buff[38:36];//0
                8'b0100: Dout [2:0]=buff[41:39];//4
                8'b0100: Dout [2:0]=buff[44:42];//4
```

```
                    8'b1000: Dout [2:0]=buff[47:45];//0
                    8'b0100: Dout [2:0]=buff[50:48];//4

                    endcase

                    if (cnt==58) cnt=0;
                    else cnt=cnt+1;

                    if (cnt==24) start=1;

                    end

                    endmodule

module seventeen_three_tf;

                    // Inputs
                    regCin;
                    reg [16:0] Din;

                    // Outputs
                    wire [2:0] Dout;
                    wire [50:0] buff;
                    wire start;
                    wire [7:0] cnt;

                    // Instantiate the Unit Under Test (UUT)
                    seventeen_threeuut (
                    .Cin(Cin),
                    .Din(Din),
                    .Dout(Dout),
                    .buff(buff),
                    .start(start),
                    .cnt(cnt)
                    );

                    initial begin
                    // Initialize Inputs
                    Cin = 0;
                    Din = 0;

                    // Wait 100 ns for global reset to finish
                    #100;

                    // Add stimulus here

                    end
                    always begin #5 Cin=~Cin; end
                    always begin #55 Din=Din+11; end
                    endmodule
```

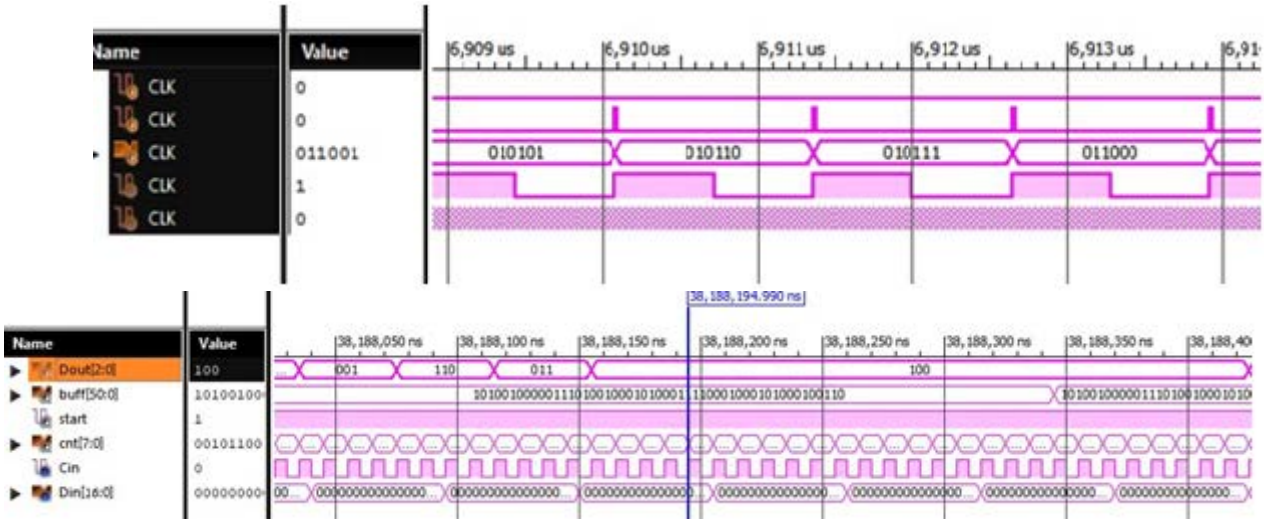


Рисунок 7.12.1 Временные диаграммы работы модуля Сериалайзера 17-3 на языке Verilog

Представленный ниже модуль имеет три входа: A, B, C и один выход S.

В таблице 7.14.1 показан листинг реализации Сериазайзер 17-3 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 17-3 на языке Verilog

```
module delta(A, B, C, S, S_TMP);
   input A, B, C;
   output S, S_TMP;
   reg S, S_TMP;
   always @(A or B or C or S_TMP)
         begin
               S_TMP<=A ^ B;
               S<=S_TMP ^ C;
         end
endmodule
```

В коде тестового воздействия сигналы на входах Аи С по истечении 120ns принимают единичное значение. Сигнал S_TMP, значение которого равно AxorB, также принимает единичное значение, но уже в момент времени t=120ns + tdelta(tdelta– бесконечно малая, не фиксируется, но с целью демонстрации мы можем ввести дополнительную задержку), значение сигнала S станет единичным в момент времени t=120ns + tdelta, но уже в момент t=120ns +2*tdelta примет установившееся значение, равное 0.

В таблице 7.14.1 показан листинг проверкиСериазайзер 17-3 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 17-3 на языке Verilog

```
moduledelta_tb;

   // Inputs
   reg A;
   reg B;
```

```
                reg C;

                // Outputs
                wire S;
                wire S_TMP;

                // Instantiate the Unit Under Test (UUT)
                deltauut (
                        .A(A),
                        .B(B),
                        .C(C),
                        .S(S),
                        .S_TMP(S_TMP)
                );

                initial begin
                        // Initialize Inputs
                        A = 0;
                        B = 0;
                        C = 0;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                end
        always begin #120 C = 1; end
        always begin #120 A = 1; end

        endmodule
```

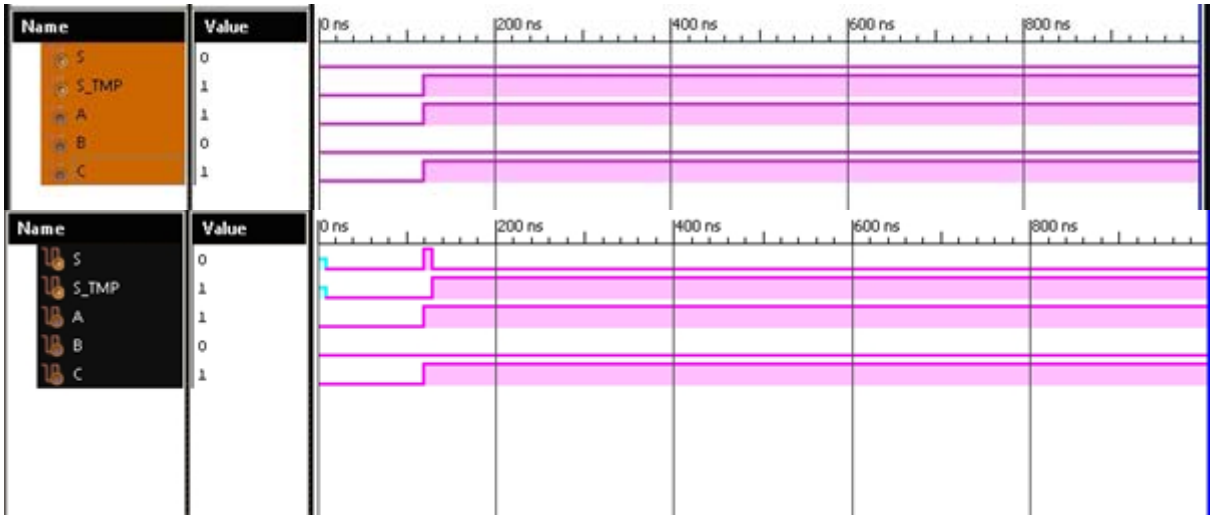


Рисунок 7.12.2 Временные диаграммы работы модуля Сериалайзера 17-3 на языке Verilog в соответствии с тестовым воздействием

7.11 Сериалайзер 9-4 Verilog

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном

преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=9, n=4. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 9 информационных входов и 4 входа тактирования: clkIn - тактирование входных данных и clkToOut - тактирование выходных данных, и 2 информационных выхода, 1 выход clkOut для тактирования выходных данных и выход RE, разрешающий чтение.

В таблице 7.14.1 показан листинг реализации Сериазайзер 9-4 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 9-4 на языке Verilog

```
`timescale 1ns / 1ps

module unserial9x4(  input wire [in-1:0] dataIn,
                     input wire clkIn,
                     input wire clkToOut,
                     output reg [out-1:0] dataOut,
                     output reg clkOut
    );

    parameter in = 9;
    parameter out = 4;

    reg [in*out-1:0] data = 0;
    integer counterIn = 0;
    integer counterOut = 0;
    integer start = 0;

    always @(posedge clkIn) begin
        counterIn = counterIn + 1;
        if (counterIn > out) begin counterIn = 1; end
        if (counterIn > 1) begin start = 1; end
        case (counterIn)
            1: data[8:0] = dataIn;
            2: data[16:9] = dataIn;
            3: data[25:17] = dataIn;
            4: data[35:26] = dataIn;
            default: ;
        endcase;
    end

    always @(posedge clkToOut or negedge clkToOut) begin
        if (start == 1)
            clkOut = clkToOut;
        else
            clkOut = 0;
```

```
                    end

                    always @(posedge clkToOut) begin
                          if (start == 1) begin
                                counterOut = counterOut + 1;
                                if (counterOut > in) begin counterOut = 1;
            end
                                case (counterOut)
                                      1:    dataOut = data[3:0];
                                      2: dataOut = data[7:4];
                                      3: dataOut = data[11:8];
                                      4: dataOut = data[15:12];
                                      5: dataOut = data[19:16];
                                      6: dataOut = data[23:20];
                                      7: dataOut = data[27:24];
                                      8: dataOut = data[31:28];
                                      9: dataOut = data[35:32];
                                      default: ;
                                endcase;
                          end
                    end


                endmodule
```

В таблице 7.14.1 показан листинг проверки Сериазайзер 9-4 на языке Verilog

Таблица 7.14.1– Листинг проверки Сериазайзер 9-4 на языке Verilog

```
                `timescale 1ns / 1ps

               module unserial9x4tb;

                  // Inputs
                  reg [8:0] dataIn;
                  reg clkIn;
                  reg clkToOut;

                  // Outputs
                  wire [3:0] dataOut;
                  wire clkOut;

                  // Instantiate the Unit Under Test (UUT)
                  unserial9x4 uut (
                        .dataIn(dataIn),
                        .clkIn(clkIn),
                        .clkToOut(clkToOut),
                        .dataOut(dataOut),
                        .clkOut(clkOut)
                  );

                  initial begin
                        // Initialize Inputs

                        clkIn = 1;
                        clkToOut = 0;
```

```
                dataIn = 9'b001001110;
                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

            end
            always begin #17 clkIn = ~clkIn; end
            always begin #4 clkToOut = ~clkToOut; end
            always begin #18 dataIn = 17'b001001110;end

        endmodule
```

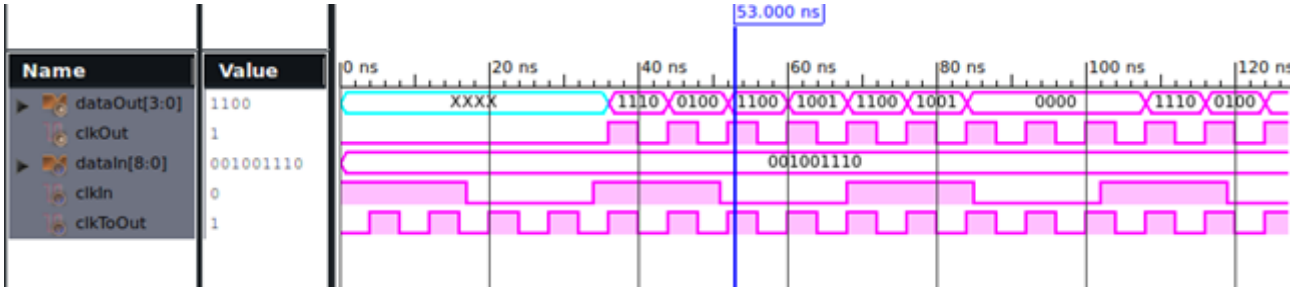


Рисунок 7.13 Временные диаграммы работы модуля Сериалайзера 17-Зна языкеVerilog

7.12 Сериалайзер 8-5 Verilog

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=8, n=5. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 8 информационных входов и 5 входа тактирования.

В таблице 7.14.1 показан листинг реализации Сериазайзер 8-5 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 8-5 на языке Verilog

```
        module s8x5(   input wire [in-1:0] dataIn,
                                    input wire clkIn,
                                    input wire clkToOut,
                                    output reg [out-1:0] dataOut,
                                    output reg clkOut
            );

            parameter in = 8;
            parameter out = 5;
```

```
                    reg [in*out-1:0] data = 0;
                    integer counterIn = 0;
                    integer counterOut = 0;
                    integer start = 0;

                    always @(posedge clkIn) begin
                          counterIn = counterIn + 1;
                          if (counterIn > out) begin counterIn = 1; end
                          if (counterIn > 1) begin start = 1; end
                          case (counterIn)
                                1: data[7:0] = dataIn;
                                2: data[15:8] = dataIn;
                                3:    data[23:16] = dataIn;
                                4:    data[31:24] = dataIn;
                                5:    data[39:32] = dataIn;
                                default: ;
                          endcase;
                    end

                    always @(posedge clkToOut or negedge clkToOut) begin
                          if (start == 1)
                                clkOut = clkToOut;
                          else
                                clkOut = 0;
                    end

                    always @(posedge clkToOut) begin
                          if (start == 1) begin
                                counterOut = counterOut + 1;
                                if (counterOut > in) begin counterOut = 1;
          end
                                case (counterOut)
                                      1:    dataOut = data[4:0];
                                      2: dataOut = data[9:5];
                                      3: dataOut = data[14:10];
                                      4: dataOut = data[19:15];
                                      5: dataOut = data[24:20];
                                      6: dataOut = data[29:25];
                                      7: dataOut = data[34:30];
                                      8: dataOut = data[39:35];

                                      default: ;
                                endcase;
                          end
                    end
              endmodule
```

В таблице 7.14.1 показан листинг проверки Сериазайзер 9-4 на языке Verilog

Таблица 7.14.1– Листинг проверки Сериазайзер 9-4 на языке Verilog

```
              module s8x5_tb;

              // Inputs
```

```
                    reg [7:0] dataIn;
                    reg clkIn;
                    reg clkToOut;

                    // Outputs
                    wire [4:0] dataOut;
                    wire clkOut;

                    // Instantiate the Unit Under Test (UUT)
                    S8x5 uut (
                          .dataIn(dataIn),
                          .clkIn(clkIn),
                          .clkToOut(clkToOut),
                          .dataOut(dataOut),
                          .clkOut(clkOut)
                    );

                    initial begin
                          // Initialize Inputs
                          clkIn = 1;
                          clkToOut = 0;

                          dataIn = 8'b11000111;

                          // Wait 100 ns for global reset to finish
                          #100;

                          // Add stimulus here

                    end

                    always begin #8 clkIn = ~clkIn; end
                    always begin #5 clkToOut = ~clkToOut; end

               endmodule
```

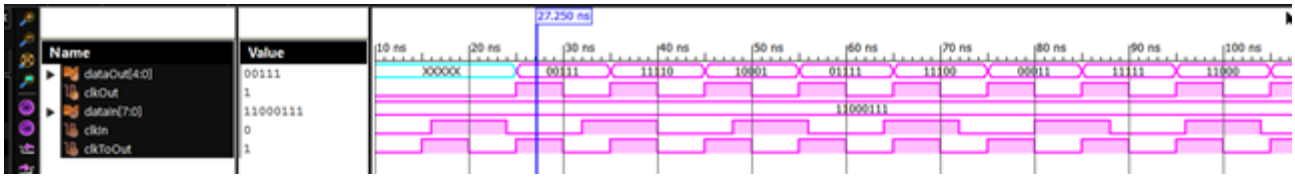


Рисунок 7.14 Временные диаграммы работы модуля Сериалайзера 8-5 на языке Verilog

7.13 Сериалайзер 17-4 Verilog

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для

синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=17, n=4. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 17 информационных входов и 4 входа тактирования.

В таблице 7.14.1 показан листинг реализации Сериазайзер 17-4 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 17-4 на языке Verilog

```
`timescale 1ns / 1ps
module unserial17x4( input wire [in-1:0] dataIn,
                     input wire clkIn,
                     input wire clkToOut,
                     output reg [out-1:0] dataOut,
                     output reg clkOut
    );
    parameter in = 17;
    parameter out = 4;
    reg [in*out-1:0] data = 0;
    integer counterIn = 0;
    integer counterOut = 0;
    integer start = 0;

    always @(posedge clkIn) begin
        counterIn = counterIn + 1;
        if (counterIn > out) begin counterIn = 1; end
        if (counterIn > 1) begin start = 1; end
        case (counterIn)
            1: data[16:0] = dataIn;
            2: data[33:17] = dataIn;
            3: data[50:34] = dataIn;
            4: data[67:51] = dataIn;
            default: ;
        endcase;
    end

    always @(posedge clkToOut or negedge clkToOut) begin
        if (start == 1)
            clkOut = clkToOut;
        else
            clkOut = 0;
    end

    always @(posedge clkToOut) begin
        if (start == 1) begin
            counterOut = counterOut + 1;
            if (counterOut > in) begin counterOut = 1;
end
            case (counterOut)
                1:    dataOut = data[3:0];
                2: dataOut = data[7:4];
                3: dataOut = data[11:8];
                4: dataOut = data[15:12];
```

```
                                    5: dataOut = data[19:16];
                                    6: dataOut = data[23:20];
                                    7: dataOut = data[27:24];
                                    8: dataOut = data[31:28];
                                    9: dataOut = data[35:32];
                                    10: dataOut = data[39:36];
                                    11: dataOut = data[43:40];
                                    12: dataOut = data[47:44];
                                    13: dataOut = data[51:48];
                                    14: dataOut = data[55:52];
                                    15: dataOut = data[59:56];
                                    16: dataOut = data[63:60];
                                    17: dataOut = data[67:64];
                                    default: ;
                                endcase;
                        end
                end


endmodule
```

В таблице 7.14.1 показан листинг проверки Сериазайзер 17-4 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 17-4 на языке Verilog

```
`timescale 1ns / 1ps

module unserial17x4tb;

    // Inputs
    reg [16:0] dataIn;
    reg clkIn;
    reg clkToOut;

    // Outputs
    wire [3:0] dataOut;
    wire clkOut;

    // Instantiate the Unit Under Test (UUT)
    unserial17x4 uut (
        .dataIn(dataIn),
        .clkIn(clkIn),
        .clkToOut(clkToOut),
        .dataOut(dataOut),
        .clkOut(clkOut)
    );

    initial begin
        // Initialize Inputs

        clkIn = 1;
        clkToOut = 0;
dataIn = 17'b00100111001001010;
        // Wait 100 ns for global reset to finish
        #100;
```

```
                                    // Add stimulus here

                                end
                                always begin #17 clkIn = ~clkIn; end
                                always begin #4 clkToOut = ~clkToOut; end
                                always begin #34 dataIn = 17'b00100111011001010;end
                            endmodule
```

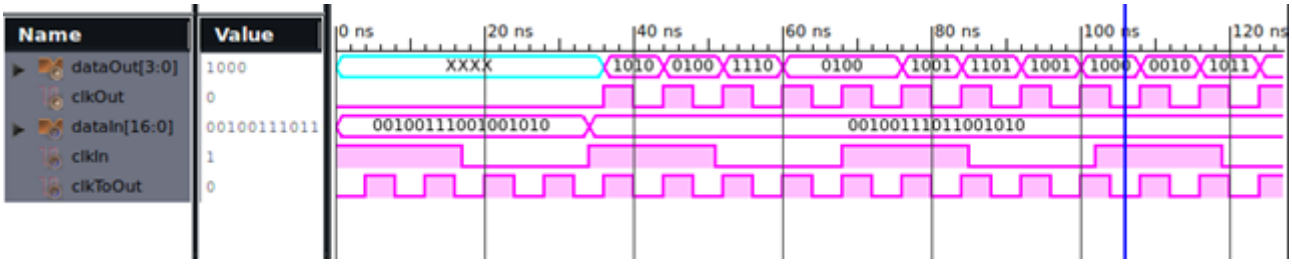


Рисунок 7.15 Временные диаграммы работы модуля Сериалайзера 17-4 на языке Verilog

7.14 Сериалайзер 21-4 Verilog

Требуется реализовать преобразователь из большего числа разрядов в меньшее с некратным соотношением разрядностей входов m и выходов n. (m-n.hdl). В данном преобразователе на вход будут подаваться данные Din большей разрядности m и высокочастотный тактовый сигнал Clk (как-бы от опорного генератора). На выходе формируется непрерывный поток данных Dout меньшей разрядности n и Cout (частота FCout=FClk), старт которого задержан на время первого преобразования. Также для синхронизации процесса захвата входных данных Din должен формироваться импульс RE (ReadEnable). Согласно варианту m=21, n=4. Разработать программу отладки на языке Verilog. Снять временные характеристики и проверить работоспособность модуля.

Разрабатываемый преобразователь имеет 21 информационных входов и 4 входа тактирования.

В таблице 7.14.1 показан листинг реализации Сериазайзер 21-4 на языке Verilog

Таблица 7.14.1– Листинг реализации Сериазайзер 21-4 на языке Verilog

```
                            entity seria_21in4 is
                               port( dataIn: in std_logic_vector(20 downto 0);
                                           clkIn: in std_logic;
                                           dataOut: out std_logic_vector(3 downto 0);
                                           clkToOut: in std_logic;
                                           clkOut: out std_logic;
                                           RE: out std_logic
                                           );
                            end seria_21in4;

                            architecture Behavioral of seria_21in4 is
                               signal data: std_logic_vector(83 downto 0);
                               shared variable counterIn: integer := 0;
                               shared variable counterOut: integer := 0;
```

```
                shared variable start: integer := 0;
            begin

                process(clkIn, dataIn, clkToOut) is
            begin
                if rising_edge(clkIn) then
                    counterIn := counterIn + 1;
                    if counterIn > 4 then
                        counterIn := 1;
                    end if;
                    if counterIn> 0 then
                        start := 1;
                    end if;
                    case counterIn is
                        when 1 =>data(20 downto 0) <= dataIn;
                        when 2 =>data(41 downto 21) <= dataIn;
                        when 3 =>data(62 downto 42) <= dataIn;
                        when 4 =>data(83 downto 63) <= dataIn;
                        when others =>counterIn := counterIn;
                    end case;
                end if;

                if rising_edge(clkToOut) or falling_edge(clkToOut)
        then
                    if start = 1 then
                        clkOut<= clkToOut;
                    else
                        clkOut<= '0';
                    end if;
                end if;

                if rising_edge(clkToOut) then
                    if start = 1 then
                        counterOut := counterOut + 1;
                        if counterOut > 21 then
                            counterOut := 1;
                        end if;
                        case counterOut is
                            when   1   =>dataOut<=   data(83
        downto 80);
                            when   2   =>dataOut<=   data(79
        downto 76);
                            when   3   =>dataOut<=   data(75
        downto 72);
                            when   4   =>dataOut<=   data(71
        downto 68);
                            when   5   =>dataOut<=   data(67
        downto 64);
                            when   6   =>dataOut<=   data(63
        downto 60);
                            when   7   =>dataOut<=   data(59
        downto 56);
                            when   8   =>dataOut<=   data(55
        downto 52);
                            when   9   =>dataOut<=   data(51
        downto 48);
                            when   10   =>dataOut<=   data(47
        downto 44);
```

```
                                    when   12   =>dataOut<=   data(39
        downto 36);
                                    when   13   =>dataOut<=   data(35
        downto 32);
                                    when   14   =>dataOut<=   data(31
        downto 28);
                                    when   15   =>dataOut<=   data(27
        downto 24);
                                    when   16   =>dataOut<=   data(23
        downto 20);
                                    when   17   =>dataOut<=   data(19
        downto 16);
                                    when   18   =>dataOut<=   data(15
        downto 12);
                                    when   19   =>dataOut<=   data(11
        downto 8);
                                    when    20   =>dataOut<=    data(7
        downto 4);
                                    when    21   =>dataOut<=    data(3
        downto 0);
                                    when   others   =>counterOut   :=
        counterOut;
                                end case;
                            end if;
                        end if;

                        if start = 0 then
                              RE <= '0';
                        end if;
                        if start = 1 then
                              RE <= '1';
                        end if;

                end process;
```

В таблице 7.14.1 показан листинг проверки Сериазайзер 21-4 на языке Verilog

Таблица 7.14.1– Листинг проверкиСериазайзер 21-4 на языке Verilog

```
            module seria_21v4_TB;

                // Inputs
                reg [20:0] dataIn;
                reg clkIn;
                reg clkToOut;

                // Outputs
                wire [3:0] dataOut;
                wire clkOut;
                wire RE;

                // Instantiate the Unit Under Test (UUT)
                seria_21in4 uut (
                      .dataIn(dataIn),
                      .clkIn(clkIn),
                      .dataOut(dataOut),
                      .clkToOut(clkToOut),
                      .clkOut(clkOut),
                      .RE(RE)
```

```
                );

                initial begin
                        // Initialize Inputs
                        dataIn = 'b11100011100011100101;
                        clkIn = 0;
                        clkToOut = 0;

                        // Wait 100 ns for global reset to finish
                        #100;

                        // Add stimulus here

                end

                always
                        begin
                                #21 clkIn = ~clkIn;
                        end

                always
                        begin
                                #4 clkToOut = ~clkToOut;
                        end

        endmodule
```

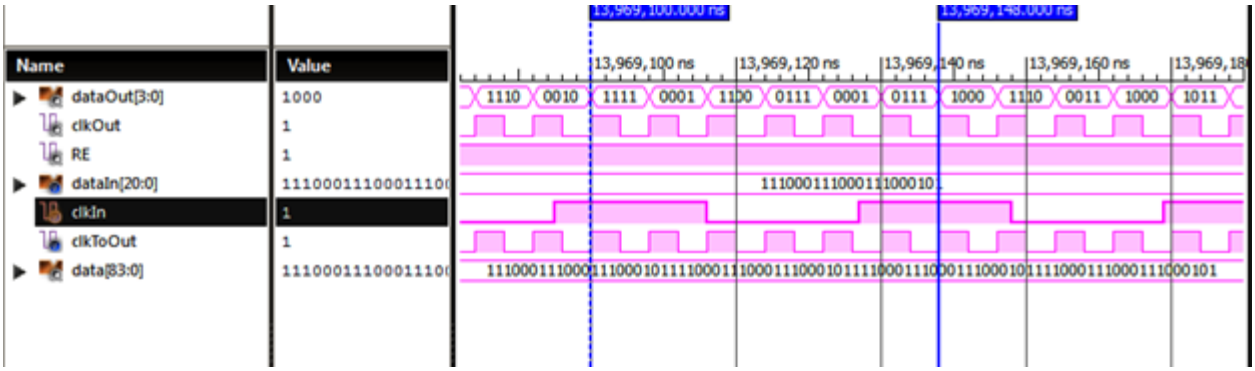


Рисунок 7.16 Временные диаграммы работы модуля Сериалайзера 21-4на языкеVerilog

8 ПАМЯТЬ И ИНТЕРФЕЙСЫ

8.1 Однопортовая синхронная память

Создать однопортовая синхронную память с одним адресом для операции чтения/записи. Создать файл проверки для тестирования созданной памяти.

В таблице 8.1.1 показан листинг реализации однопортовой синхронной памяти на языке Verilog

Таблица 8.1.1– Листинг реализации однопортовой синхронной памяти на языке Verilog

```
`timescale 1ns / 1ps
module ram_sp_sr_sw (
 clk         , // Clock Input
 address     , // Address Input
 data        , // Data bi-directional
 cs          , // Chip Select
 we          , // Write Enable/Read Enable
 oe            // Output Enable
);

 parameter DATA_WIDTH = 8 ;
 parameter ADDR_WIDTH = 8 ;
 parameter RAM_DEPTH = 1 << ADDR_WIDTH;

 //--------------Input Ports-----------------------
 input                  clk         ;
 input [ADDR_WIDTH-1:0] address     ;
 input                  cs          ;
 input                  we          ;
 input                  oe          ;

 //--------------Inout Ports-----------------------
 inout [DATA_WIDTH-1:0]  data       ;

 //--------------Internal variables----------------
 reg [DATA_WIDTH-1:0] data_out ;
 reg [DATA_WIDTH-1:0] mem [0:RAM_DEPTH-1];
 reg                  oe_r;

 //--------------Code Starts Here------------------

 // Tri-State Buffer control
 // output : When we = 0, oe = 1, cs = 1
assign data = (cs && oe &&  ! we) ? data_out : 8'bz;

 // Memory Write Block
 // Write Operation : When we = 1, cs = 1
 always @ (posedge clk)
 begin : MEM_WRITE
    if ( cs && we ) begin
        mem[address] = data;
    end
```

```
end

// Memory Read Block
// Read Operation : When we = 0, oe = 1, cs = 1
always @ (posedge clk)
begin : MEM_READ
  if (cs &&  ! we && oe) begin
    data_out = mem[address];
    oe_r = 1;
  end else begin
    oe_r = 0;
  end
end

endmodule // End of Module ram_sp_sr_sw
```

В таблице 8.1.2 показан листинг проверки однопортовой синхронной памяти на языке Verilog

Таблица 8.1.2– Листинг проверки однопортовой синхронной памяти на языке Verilog

```
`timescale 1ns / 1ps


module ram_sp_sr_sw_tb;

  // Inputs
  reg clk;
  reg [7:0] address;
  reg cs;
  reg we;
  reg oe;
  reg [7:0] data_reg;
  // Bidirs
  wire [7:0] data = data_reg;

  // Instantiate the Unit Under Test (UUT)
  ram_sp_sr_sw uut (
        .clk(clk),
        .address(address),
        .data(data),
        .cs(cs),
        .we(we),
        .oe(oe)
  );

  initial begin
        // Initialize Inputs
        clk = 0;
        address = 1;
        cs = 0;
        we = 1;
        oe = 0;
```

```
                data_reg = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimulus here

        end
         always begin
                #5
                clk = ~clk;
           end

        always begin
           we =~we;
           #100;
        end

           always begin
                #10
                oe = 0;
                cs = 1;
                data_reg = data_reg + 1;
                address = address + 1;
           end

           always begin
                #10
                oe = 1;
                cs = 1;
                address = address + 1;
           end

        endmodule
```

8.2 Двухпортовая синхронная память

Особенности семейства SynchronousDual-PortRAM : синхронный интерфейс с раздельными сигналами синхронизации CLK_R и CLK_L и внутренние счетчики (Internalcounters) для организации пакетного режима передачи данных . Поскольку обязательным условием доступа активных устройств к пространству такой памяти является их взаимная синхронизация от одного системного таймера, никакой дополнительной логики (арбитраж, семафоры или прерывания) для разрешения конфликтных ситуаций не требуется. Операции обращения к ячейкам асинхронной памяти могут выполняться в произвольные моменты времени при условии соблюдения необходимых временных соотношений между сигналами установки адреса, управления, чтения/записи данных. Операции обращения к ячейкам синхронного двух-портового ОЗУ осуществляются строго под управлением внешнего сигнала синхронизации (CLK_R для порта R и CLK_L для порта L).

Архитектура синхронного двухпортового ОЗУ оптимизирована для применения в вычислительных сетях (ATM и Ethernet коммутаторы/маршрутизаторы) и системах беспроводной телефонии (базовые станции), обеспечивая следующие синхронные режимы работы памяти: Pipelined (конвейерный), Flow-through (сквозной) и Burst (пакетный).

Режимы Pipelined и Flow-through отличаются структурой выходного устройства. В Pipelined дополнительный буферный регистр-защелка по выходу (Outputregister) позволяет организовать конвейерный доступ к данным (одновременно с чтением по предыдущему адресу осуществляется запрос по следующему). Это позволяет сократить общее время обращения к памяти. Недостатком этого режима является задержка на один период сигнала синхронизации при считывании первого слова. Во втором случае (режим Flow-through) считываемые данные непосредственно поступают на выходную шину микросхемы памяти (Output). Это позволяет обеспечить минимальную задержку при считывании первого слова. Однако все последующие обращения к памяти в этом режиме будут проходить за более длительное время, чем в режиме Pipelined. Режимы Flow-through и Pipelined задаются пользователем аппаратно.

Создать двухпортовую синхронную память с одним адресом для операции чтения/записи. Создать файл проверки для тестирования созданной памяти.

В таблице 8.2.1 показан листинг реализации двухпортовой синхронной памяти на языке Verilog

Таблица 8.2.1– Листинг реализации двухпортовой синхронной памяти на языке Verilog

```
`timescale 1ns / 1ps
module ram_dp_sr_sw (
clk       , // Clock Input
address_0 , // address_0 Input
data_0    , // data_0 bi-directional
cs_0      , // Chip Select
we_0      , // Write Enable/Read Enable
oe_0      , // Output Enable
address_1 , // address_1 Input
data_1    , // data_1 bi-directional
cs_1      , // Chip Select
we_1      , // Write Enable/Read Enable
oe_1        // Output Enable
);

parameter data_0_WIDTH = 8 ;
parameter ADDR_WIDTH = 8 ;
parameter RAM_DEPTH = 1 << ADDR_WIDTH;

  //--------------Input Ports-----------------------
  input clk ;
  input [ADDR_WIDTH-1:0] address_0 ;
```

```
                    input cs_0 ;
                    input we_0 ;
                    input oe_0 ;
                    input [ADDR_WIDTH-1:0] address_1 ;
                    input cs_1 ;
                    input we_1 ;
                    input oe_1 ;

                    //--------------Inout Ports-----------------------
                    inout [data_0_WIDTH-1:0] data_0 ;
                    inout [data_0_WIDTH-1:0] data_1 ;

                    //--------------Internal variables----------------
                    reg [data_0_WIDTH-1:0] data_0_out ;
                    reg [data_0_WIDTH-1:0] data_1_out ;
                    reg [data_0_WIDTH-1:0] mem [0:RAM_DEPTH-1];

                    //--------------Code Starts Here------------------
                    // Memory Write Block
                    // Write Operation : When we_0 = 1, cs_0 = 1
                    always @ (posedge clk)
                    begin : MEM_WRITE
                      if ( cs_0 && we_0 ) begin
                         mem[address_0] <= data_0;
                      end else if (cs_1 && we_1) begin
                         mem[address_1] <= data_1;
                      end
                   end



                    // Tri-State Buffer control
                    // output : When we_0 = 0, oe_0 = 1, cs_0 = 1
                  assign data_0 = (cs_0 && oe_0 &&  ! we_0) ? data_0_out :
            8'bz;

                    // Memory Read Block
                    // Read Operation : When we_0 = 0, oe_0 = 1, cs_0 = 1
                    always @ (posedge clk)
                    begin : MEM_READ_0
                      if (cs_0 &&  ! we_0 && oe_0) begin
                        data_0_out <= mem[address_0];
                      end else begin
                        data_0_out <= 0;
                      end
                    end

                    //Second Port of RAM
                    // Tri-State Buffer control
                    // output : When we_0 = 0, oe_0 = 1, cs_0 = 1
                  assign data_1 = (cs_1 && oe_1 &&  ! we_1) ? data_1_out :
            8'bz;
                    // Memory Read Block 1
                    // Read Operation : When we_1 = 0, oe_1 = 1, cs_1 = 1
                    always @ (posedge clk)
```

```
begin : MEM_READ_1
  if (cs_1 &&  ! we_1 && oe_1) begin
    data_1_out <= mem[address_1];
  end else begin
    data_1_out <= 0;
  end
end

endmodule // End of Module ram_dp_sr_sw
```

В таблице 8.2.2 показан листинг проверки двухпортовой синхронной памяти на языке Verilog

Таблица 8.2.2– Листинг проверкидвухпортовой синхронной памяти на языке Verilog

```
`timescale 1ns / 1ps
module ram_dp_sr_sw_tb;

  // Inputs
  reg clk;
  reg [7:0] address_0;
  reg cs_0;
  reg we_0;
  reg oe_0;
  reg [7:0] address_1;
  reg cs_1;
  reg we_1;
  reg oe_1;
  reg [7:0] data_0_reg;

  // Bidirs
  wire [7:0] data_0 = data_0_reg;
  wire [7:0] data_1;


  // Instantiate the Unit Under Test (UUT)
  ram_dp_sr_sw uut (
        .clk(clk),
        .address_0(address_0),
        .data_0(data_0),
        .cs_0(cs_0),
        .we_0(we_0),
        .oe_0(oe_0),
        .address_1(address_1),
        .data_1(data_1),
        .cs_1(cs_1),
        .we_1(we_1),
        .oe_1(oe_1)
  );

  initial begin
        // Initialize Inputs
        clk = 0;
        address_0 = 1;
```

```
                cs_0 = 0;
                we_0 = 0;
                oe_0 = 0;
                address_1 = 0;
                cs_1 = 0;
                we_1 = 0;
                oe_1 = 0;
                data_0_reg = 0;

                // Wait 100 ns for global reset to finish
                #100;

                // Add stimuus here

            end
        always begin
                #5
                clk = ~clk;
            end

            always begin
                #10
                we_0 = 1;
                oe_0 = 0;
                cs_0 = 1;
                data_0_reg = data_0_reg + 1;
                address_0 = address_0 + 1;
            end

            always begin
                #10
                we_1 = 0;
                oe_1 = 1;
                cs_1 = 1;
                address_1 = address_1 + 1;
            end

        endmodule
```

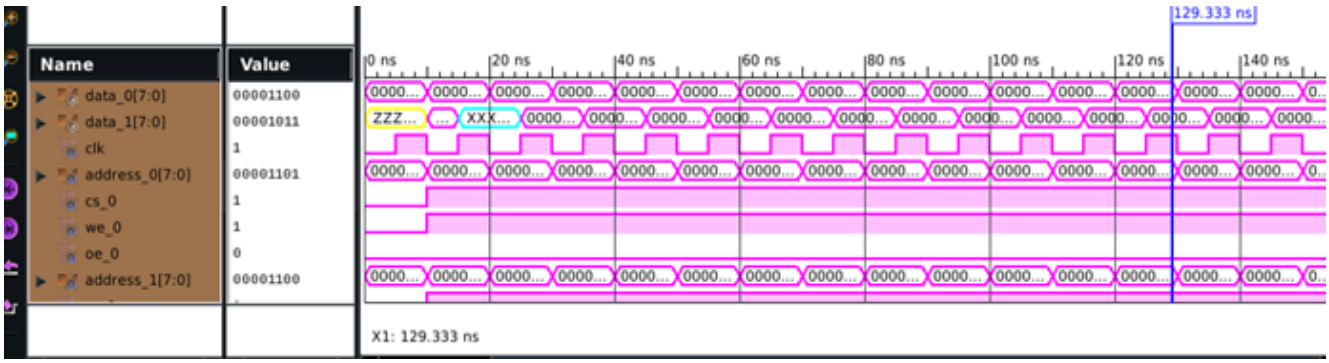


Рисунок 8.2.1 Временная диаграмма работы Двухпортовой синхронной памяти

8.3 RS-232 Transmitter VHDL

В состав устройства входит тестовый файл (TB) и модуль передатчика (taker). Ниже представлена схема.

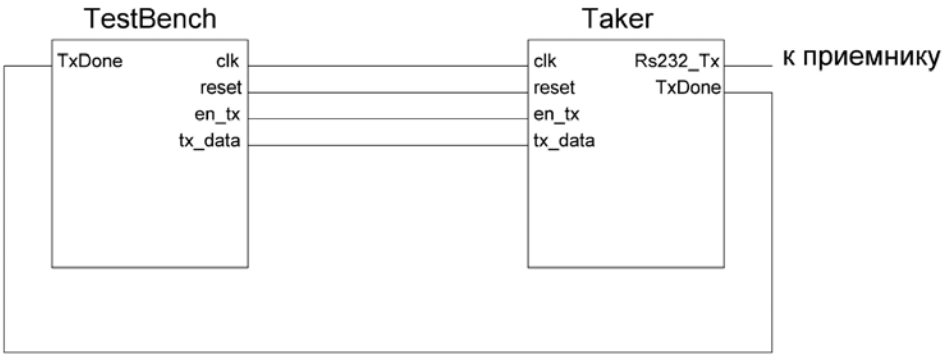


Рисунок 8.3.1 Схема USART - RS 232 передатчика

Программа написана для платы Virtex 5, где частота тактовой шины равна 200 МГц, а скорость передатчика USART 9600 боде.

Когда модуль передатчика отправил одно из слов, он выставляет линию txDone = ‘1’ (которая связывает два модуля и является ОС), и наш ТВ отправляет следующее слово (букву сообщения из 8 бит, изначально записанного в ТВ).

Наше сообщение хранится в модуле ТВ и задано в кодировке ASCII. Оно передается циклически. Т.е. на стороне приемника (компьютера), будет постоянно приниматься данное сообщение (‘HELLOWORLD!’).

В таблице 8.3.1 показан листинг реализации RS-232 Transmitter на языке VHDL

Таблица 8.3.1– Листинг реализации RS-232 Transmitter на языке VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity taker is
   port(
            SysClk: in std_logic;
            SysReset: in std_logic; -- activ = '0'
            TxEnable: in std_logic; -- activ = '1'
            TxData: in std_logic_vector (7 downto 0);

            Rs232Tx: out std_logic; -- bit;
            TxDone: out std_logic
            );
end taker;

architecture Behavioral_taker of taker is

   type state_type is (waiting, writing, ending);
    signal state, next_state : state_type;
   signal counter: std_logic_vector (15 downto 0) := (others
                => '0'); -- for bitTime
   signal Nbit:  std_logic_vector (3 downto 0) := b"0000"; --
```

```
                                9: beginer count
                    constant Rs232BitTime: integer := 20840; -- 9600 boud
                begin

                    --Insert the following in the architecture after the
            begin
                                keyword
                    process(SysClk)
                    begin

                            if (rising_edge(SysClk)) then
                                if (SysReset = '0') then
                                        counter <= (others => '0');
                                elsif (TxEnable = '1') then
                                        if (counter = Rs232BitTime) then
                                                counter <= (others => '0');
                                        else
                                                counter <= counter + 1;
                                        end if;
                                else
                                        counter <= (others => '0');
                                end if;
                            end if;

                            if (rising_edge(SysClk)) then
                                if (SysReset = '0') then
                                        Nbit <= b"0000";
                                elsif (TxEnable = '1') then
                                        if (counter = Rs232BitTime) then
                                                if (Nbit = b"1001") then -- 9:
            beginner
                                count, 8: ending
                                                        Nbit <= b"0000";
                                                else
                                                        Nbit <= Nbit + 1;
                                                end if;
                                        end if;
                                else
                                        Nbit <= b"0000";
                                end if;
                            end if;

                    end process;

                    process (state, counter, TxEnable, Nbit)
                --      variable Nbit: integer range 0 to 9 := 0;
                    begin
                            next_state <= state;   --default is to stay in
            current
                                state

                                case (state) is
                                        when waiting =>
                                                if (TxEnable = '1') then --
                                rising_edge(TxEnable)
```

```
                                                next_state <= writing;
                                            end if;

                                        when writing =>

                                            if (counter = Rs232BitTime) then
                                                if (Nbit = b"1001") then -
- 8: ending
                                count
                                                    next_state        <=
ending;
                                                else
                                                    next_state        <=
writing;
                                                end if;
                                            end if;

                                        when ending =>

                                            if (TxEnable = '0') then
                                                next_state <= waiting;
                                            else
                                                next_state <= writing;
                                            end if;
                                        when others =>
                                            next_state <= waiting;
                                end case;
                        end process;
                        --MEALY State-Machine - Outputs based on state and
inputs
                        OUTPUT_DECODE: process (state, Nbit)
                        begin
                           --insert statements to decode internal output signals
                           --below is simple example
                           case state is
                                when waiting =>
                                    TxDone <= '0';
                                    Rs232Tx <= '1';

                                when writing =>
                                    if (Nbit = b"0000") then -- 9:
beginner count
                                        Rs232Tx <= '0';
                                    elsif (Nbit = b"1001") then
                                        Rs232Tx <= '1';
                                    else
                                        Rs232Tx                    <=
TxData(conv_integer(Nbit)-1);
                                    end if;
                                    TxDone <= '0';

                                when ending =>
                                    TxDone <= '1';
                                    Rs232Tx <= '1';

                                when others =>
```

```
                                    Rs232Tx <= '1';
                        end case;
                    end process;

                end Behavioral_taker;
```

В таблице 8.3.2 показан листингпроверкиRS-232 Transmitter на языке VHDL

Таблица 8.3.2– Листинг проверкиRS-232 Transmitter на языке VHDL

```
                library IEEE;
                use IEEE.STD_LOGIC_1164.ALL;
                use IEEE.STD_LOGIC_arith.ALL;
                use IEEE.STD_LOGIC_UNSIGNED.ALL;--std.textio
                use std.textio.ALL;

                entity testbench is
                end testbench;

                architecture nature of testbench is

                    type state_type is (writing, waiting);

                    component taker
                        port(
                            SysClk: in std_logic;
                            SysReset: in std_logic; -- activ = '0'
                            TxEnable: in std_logic; -- activ = '1'
                            TxData: in std_logic_vector (7 downto 0);

                            Rs232Tx: out std_logic; -- bit;
                            TxDone: out std_logic
                            );
                        end component;

                    signal Ndata: std_logic_vector (3 downto 0) := b"0000";

                    signal state, next_state : state_type;
                    signal cnt: std_logic_vector (27 downto 0) := (others =>
                            '0');
                    constant max_count: std_logic_vector (27 downto 0) :=
                            X"02FC6C0";--1Hz = BEBC200  0032E10


                    signal SysClk: std_logic := '0';
                    signal SysReset: std_logic := '1'; -- activ = '0'
                    signal TxEnable: std_logic := '1'; -- activ = '1'
                    signal TxData: std_logic_vector (7 downto 0);

                    signal Rs232Tx: std_logic; -- bit;
                    signal TxDone: std_logic;

                begin
                    add: taker
                        port map(
```

```
                SysClk => SysClk,
                SysReset => SysReset,
                TxEnable => TxEnable,
                TxData => TxData,

                Rs232Tx => Rs232Tx,
                TxDone => TxDone
                );

        SysSy: process(SysClk, txdone)

        begin
            SysClk <= not SysClk after 2.5 ns;

            if (rising_edge(sysclk)) then
                if (cnt = max_count) then
                    cnt <= (others => '0');
                else
                    cnt <= cnt + 1;
                end if;
            end if;

            if (rising_edge(sysclk)) then
                if (TxDone = '1') then
                    if (Ndata = b"1101") then
                        Ndata <= (others => '0');
                    else
                        Ndata <= Ndata + 1;
                    end if;
                end if;
            end if;

            if (rising_edge(sysclk)) then
                if (TxDone = '1' or cnt = 0) then
                    case (conv_integer(Ndata)) is
                        when 0 =>
                            txdata <= x"48";
                        when 1 =>
                            txdata <= x"65";
                        when 2 =>
                            txdata <= x"6c";
                        when 3 =>
                            txdata <= x"6c";
                        when 4 =>
                            txdata <= x"6f";
                        when 5 =>
                            txdata <= x"09";
                        when 6 =>
                            txdata <= x"57";
                        when 7 =>
                            txdata <= x"6f";
                        when 8 =>
                            txdata <= x"72";
                        when 9 =>
                            txdata <= x"6c";
```

```
                                        when 10 =>
                                                txdata <= x"64";
                                        when 12 =>
                                                txdata <= x"0d";
                                        when 13 =>
                                                txdata <= x"09";
                                        when others =>
                                                txdata <= x"0d";

                                    end case;
                                end if;
                            end if;

                            if (rising_edge(sysclk)) then
                                    state <= next_state;
                            end if;

                    end process;

                    sysload_as: process(state)
                    begin

                            if (state = waiting) then
                                    txenable <= '0';
                            else
                                    txenable <= '1';
                            end if;

                    end process;

                    sysstate: process(state, txdone, cnt)
                    begin

                            next_state <= state;
                                    state <= next_state;
                            end if;

                    end process;

                    sysload_as: process(state)
                    begin

                            if (state = waiting) then
                                    txenable <= '0';
                            else
                                    txenable <= '1';
                            end if;

                    end process;

                    sysstate: process(state, txdone, cnt)
                    begin

                            next_state <= state;

```

```
                        if (state = writing) then
                            if (TxDone = '1' and Ndata = b"1101") then
                                next_state <= waiting;
                            end if;
                        else
                            if (cnt = 0) then
                                next_state <= writing;
                            end if;
                        end if;

                    end process;

                end nature;
```

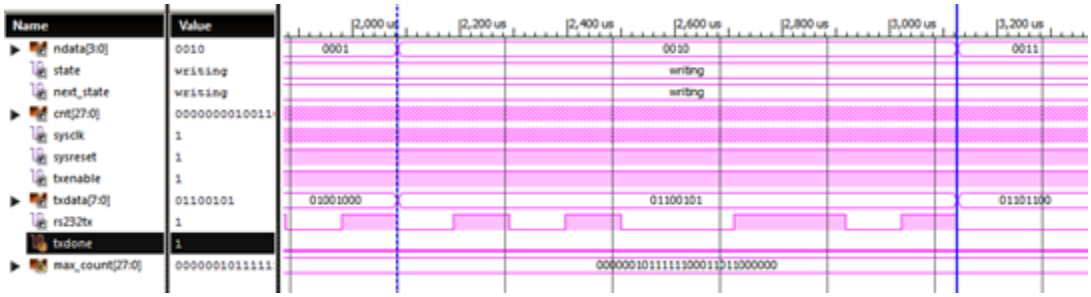


Рисунок 8.3.2 Временная диаграмма передачи 1 слова в обратном порядке бит (выделенная область)

Сигналы модуля TB:

max_count – содержит значение, которое определяет частоту шины virtex 5, равной 200 МГц.

Cnt – счетчик до максимального значения max_count.

State – состояние в котором находится модуль передачи, их 3: ожидание (waiting), запись (передача - writing), конец передачи (ending).

next_state – следующее записанное состояние.

SysReset – общий сброс.

TxEnable – разрешение передачи.

TxData – данные которые надо передать (8 битная шина), содержит одну букву (слово).

Ndata – номер слова, которой нужно передать (от 0 до 13).

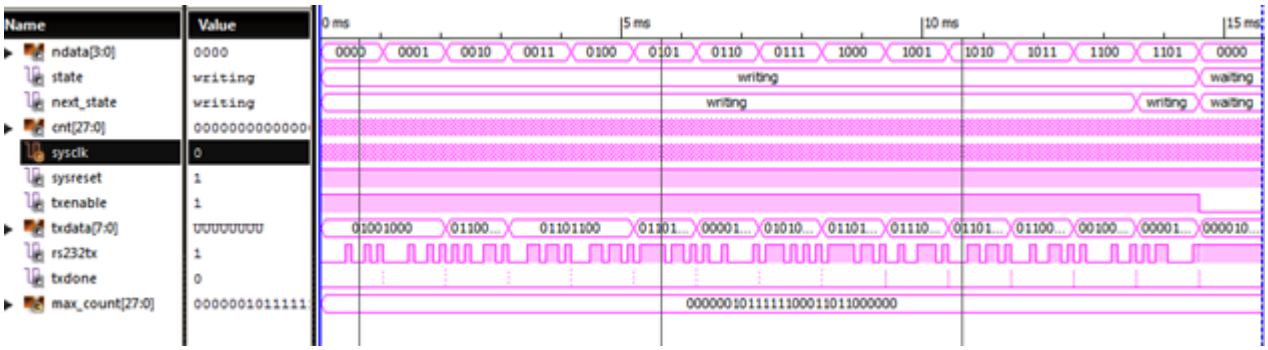


Рисунок 8.3.2 Временные диаграммы передачи сообщения из 13 байт (в конце код символа, соответствующий переносу каретки)

Сигналы модуля передатчика:

Входные:

Clk – тактовый сигнал.

SysReset – общий сброс.

TxEnable – разрешение передачи.

TxData – данные которые надо передать (8 битная шина), содержит одну букву (слово).

Ndata – номер слова, которой нужно передать (от 0 до 13).

max_count – содержит значение, которое определяет частоту шины virtex 5, равной 200 МГц.

Выходные:

TxDone – подтверждение передачи.

Rs232Tx – бит данных.

Внутрение:

counter– счетчик до максимального значения Rs232BitTime, который определяет длительность (кол-во тактов) одного бита.

Nbit – номер текущего бита для передачи – 10 бит (от 0 до 9).

State – состояние в котором находится модуль передачи, их 3: ожидание (waiting), запись (передача - writing), конец передачи (ending).

next_state – следующее записанное состояние.

Rs232BitTime – константа, содержит значение, которое определяет скорость передачи. В примере скорость соответствует 9600 боде (бит/сек, с учетом значащих бит и бит старт и стоп).

8.4 RS-232 на основе USARTVerilog

В состав устройства входит тестовый файл (TB) и модуль передатчика (taker). Ниже представлена схема.

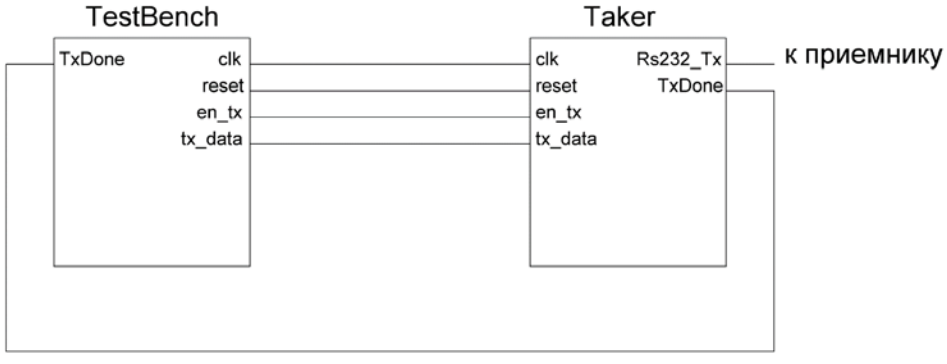


Рисунок 8.3.1 Схема USART - RS 232 передатчика

Программа написана для платы Virtex 5, где частота тактовой шины равна 200 МГц, а скорость передатчика USART 9600 боде.

Когда модуль передатчика отправил одно из слов, он выставляет линию txDone = ‘1’ (которая связывает два модуля и является ОС), и наш ТВ отправляет следующее слово (букву сообщения из 8 бит, изначально записанного в ТВ).

Наше сообщение хранится в модуле ТВ и задано в кодировке ASCII. Оно передается циклически. Т.е. на стороне приемника (компьютера), будет постоянно приниматься данное сообщение (‘HELLOWORLD!’).

В таблице 8.4.1 показан листингuart реализации RS-232 на основе USART на языке Verilog

Таблица 8.4.1– Листинг uatr реализации RS-232 на основе USART на языке Verilog

```
module uart(input wire [7:0] din,
        input wire wr_en,
        input wire clk_50m,
        output wire tx,
        output wire tx_busy,
        input wire rx,
        output wire rdy,
        input wire rdy_clr,
        output wire [7:0] dout);

wire rxclk_en, txclk_en;

baud_rate_genuart_baud(.clk_50m(clk_50m),
                .rxclk_en(rxclk_en),
                .txclk_en(txclk_en));
transmitter uart_tx(.din(din),
          .wr_en(wr_en),
          .clk_50m(clk_50m),
          .clken(txclk_en),
          .tx(tx),
          .tx_busy(tx_busy));
receiver uart_rx(.rx(rx),
          .rdy(rdy),
          .rdy_clr(rdy_clr),
          .clk_50m(clk_50m),
          .clken(rxclk_en),
          .data(dout));

Endmodule
```

В таблице 8.4.2 показан листинг receiver реализации RS-232 на основе USART на языке Verilog

Таблица 8.4.2– Листинг receiver реализации RS-232 на основе USART на языке Verilog

```
module receiver(input wire rx,
        output regrdy,
        input wire rdy_clr,
        input wire clk_50m,
        input wire clken,
        output reg [7:0] data);

initial begin
   rdy = 0;
   data = 8'b0;
end

parameter RX_STATE_START   = 2'b00;
parameter RX_STATE_DATA    = 2'b01;
parameter RX_STATE_STOP    = 2'b10;

reg [1:0] state = RX_STATE_START;
reg [3:0] sample = 0;
reg [3:0] bitpos = 0;
reg [7:0] scratch = 8'b0;

always @(posedge clk_50m) begin
   if (rdy_clr)
        rdy<= 0;

   if (clken) begin
        case (state)
        RX_STATE_START: begin
              /*
              * Start counting from the first low sample, once
            we've
              * sampled a full bit, start collecting data bits.
              */
              if (!rx || sample != 0)
                    sample <= sample + 4'b1;

              if (sample == 15) begin
                    state <= RX_STATE_DATA;
                    bitpos<= 0;
                    sample <= 0;
                    scratch <= 0;
              end
        end
        RX_STATE_DATA: begin
              sample <= sample + 4'b1;
              if (sample == 4'h8) begin
                    scratch[bitpos[2:0]] <= rx;
                    bitpos<= bitpos + 4'b1;
              end
              if (bitpos == 8 && sample == 15)
                    state <= RX_STATE_STOP;
        end
        RX_STATE_STOP: begin
```

```
                                    /*
                                     * Our baud clock may not be running at
                exactly the
                                     * same rate as the transmitter.  If we
                thing that
                                     * we're at least half way into the stop
                bit, allow
                                     * transition into handling the next start
                bit.
                                     */
                                    if (sample == 15 || (sample >= 8 && !rx))
                begin
                                        state <= RX_STATE_START;
                                        data <= scratch;
                                        rdy<= 1'b1;
                                        sample <= 0;
                                    end else begin
                                        sample <= sample + 4'b1;
                                    end
                                end
                                default: begin
                                    state <= RX_STATE_START;
                                end
                                endcase
                            end
                        end

                        endmodule
```

В таблице 8.4.3 показан листинг transmitter реализации RS-232 на основе USART на языке Verilog

Таблица 8.4.3– Листинг transmitter реализации RS-232 на основе USART на языке Verilog

```
                        module transmitter(input wire [7:0] din,
                                    input wire wr_en,
                                    input wire clk_50m,
                                    input wire clken,
                                    output regtx,
                                    output wire tx_busy);

                        initial begin
                           tx = 1'b1;
                        end

                        parameter STATE_IDLE = 2'b00;
                        parameter STATE_START      = 2'b01;
                        parameter STATE_DATA = 2'b10;
                        parameter STATE_STOP = 2'b11;

                        reg [7:0] data = 8'h00;
                        reg [2:0] bitpos = 3'h0;
                        reg [1:0] state = STATE_IDLE;

                        always @(posedge clk_50m) begin
```

```
                case (state)
                STATE_IDLE: begin
                      if (wr_en) begin
                            state <= STATE_START;
                            data <= din;
                            bitpos<= 3'h0;
                      end
                end
                STATE_START: begin
                      if (clken) begin
                            tx<= 1'b0;
                            state <= STATE_DATA;
                      end
                end
                STATE_DATA: begin
                      if (clken) begin
                            if (bitpos == 3'h7)
                                  state <= STATE_STOP;
                            else
                                  bitpos<= bitpos + 3'h1;
                            tx<= data[bitpos];
                      end
                end
                STATE_STOP: begin
                      if (clken) begin
                            tx<= 1'b1;
                            state <= STATE_IDLE;
                      end
                end
                default: begin
                      tx<= 1'b1;
                      state <= STATE_IDLE;
                end
                endcase
          end

          assign tx_busy = (state != STATE_IDLE);

          endmodule
```

В таблице 8.4.4 показан листинг baudrategen реализации RS-232 на основе USART на языке Verilog

Таблица 8.4.4– Листинг baud rate gen реализации RS-232 на основе USART на языке Verilog

```
          module baud_rate_gen(input wire clk_50m,
                        output wire rxclk_en,
                        output wire txclk_en);
          parameter RX_ACC_MAX = 50000000 / (115200 * 16);
          parameter TX_ACC_MAX = 50000000 / 115200;
          parameter RX_ACC_WIDTH = $clog2(RX_ACC_MAX);
          parameter TX_ACC_WIDTH = $clog2(TX_ACC_MAX);
          reg [RX_ACC_WIDTH - 1:0] rx_acc = 0;
          reg [TX_ACC_WIDTH - 1:0] tx_acc = 0;
```

```
assign rxclk_en = (rx_acc == 5'd0);
assign txclk_en = (tx_acc == 9'd0);

always @(posedge clk_50m) begin
   if (rx_acc == RX_ACC_MAX[RX_ACC_WIDTH - 1:0])
         rx_acc<= 0;
   else
         rx_acc<= rx_acc + 5'b1;
end

always @(posedge clk_50m) begin
   if (tx_acc == TX_ACC_MAX[TX_ACC_WIDTH - 1:0])
         tx_acc<= 0;
   else
         tx_acc<= tx_acc + 9'b1;
end

endmodule
```

В таблице 8.4.5 показан листинг проверки RS-232 на основе USART на языке Verilog

Таблица 8.4.5– Листинг проверки RS-232 на основе USART на языке Verilog

```
`include "uart.v"

module uart_tx_test();

reg [7:0] data = 0;
regclk = 0;
reg enable = 0;

wire tx_busy;
wire rdy;
wire [7:0] rxdata;

wire loopback;
regrdy_clr = 0;

uarttest_uart(.din(data),
   .wr_en(enable),
   .clk_50m(clk),
   .tx(loopback),
   .tx_busy(tx_busy),
   .rx(loopback),
   .rdy(rdy),
   .rdy_clr(rdy_clr),
   .dout(rxdata));

initial begin
   $dumpfile("uart.vcd");
   $dumpvars(0, uart_tx_test);
   enable <= 1'b1;
   #2 enable <= 1'b0;
end
```

```
                    always begin
                        #1 clk = ~clk;
                    end

                    always @(posedgerdy) begin
                        #2 rdy_clr<= 1;
                        #2 rdy_clr<= 0;
                        if (rxdata != data) begin
                                $display("FAIL: rx data %x does not match tx %x",
            rxdata,
                                    data);
                                $finish;
                        end else begin
                                if (rxdata == 8'hff) begin
                                    $display("SUCCESS: all bytes verified");
                                    $finish;
                                end
                                data <= data + 1'b1;
                                enable <= 1'b1;
                                #2 enable <= 1'b0;
                        end
                    end

                    endmodule
```

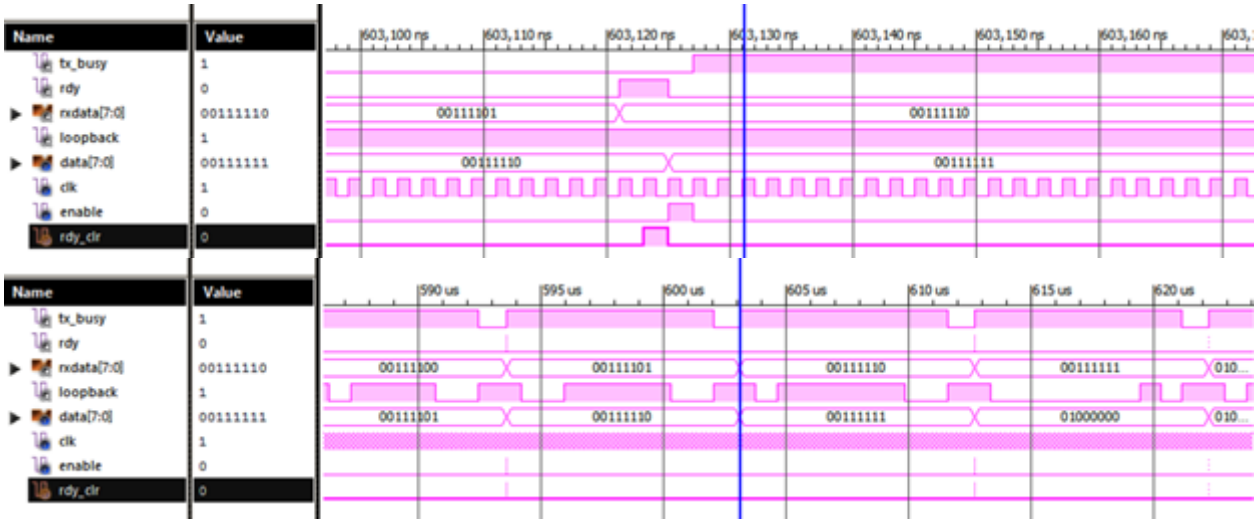


Рисунок 8.4 Временные диаграммы работы RS-232 на основе USARTна языке Verilog

8.5 I2C Verilog

I²C (IIC, англ.Inter-IntegratedCircuit) — последовательная асимметричная шина для связи между интегральными схемами внутри электронных приборов. Использует две двунаправленные линии связи (SDA и SCL), применяется для соединения низкоскоростных периферийных компонентов с процессорами и микроконтроллерами (например, на материнских платах, во встраиваемых системах, в мобильных телефонах). Реализовать данную шину на языке Verilog.

В таблице 8.5.1 показан листинг реализации I2C Verilogна языке Verilog

Таблица 8.5.1– Листинг реализации I2C на языке Verilog

```
moduleI2C_master
#(
parameterfreq = 66
)
(
inout wire          SDA,          // I2C Serial data line, pulled high at
                    board level
inout wire          SCL,          // I2C Serial clock line, pulled high at
                    board level
input wire           sys_clock,   // System clock, wr_ctrl should be
                    synchronous to this
input wire         reset,       // power-on reset - puts I2C bus into idle
                    state
input wire [31:0] ctrl_data,  // Data bus for writing the control
                    register
input wire         wr_ctrl,     // Write enable for control register, also
                    starts I2C cycles
outputreg [31:0] status        // Status of I2C including most recently
                    read data
);
regfloat_sda;   // This is essentially SDA when we are sourcing it (open
                    drain at pin)
regfloat_scl;   // This is essentially SCL when we are sourcing it (open
                    drain at pin)
wiresda_in;     // Feedback from the IOB pin for SDA
wirescl_in;     // Feedback from the IOB pin for SCL

// I/O buffers for the I2C open-drain signals
// Note that even if no slaves drive SCL, you need to use feedback
// to sense the rising edge of SCL to be sure to meet hold time.
assignscl_in = SCL;
assign SCL = float_scl ? 1'bZ : 1'b0;

assignsda_in = SDA;
assign SDA = float_sda ? 1'bZ : 1'b0;

localparamt_hd_sta = 4 * freq,        // Hold time on START condition,
                    4.0us from spec
t_low = 5 * freq,           // SCL low time 4.7us from spec
t_high = 5 * freq,            // SCL high time 4.0us from spec, but cycle
                    time must be 10us
t_su_sta = 5 * freq,            // SDA, SCL high before asserting start,
                    4.7us from spec
```

```
        t_su_dat = (freq>> 2) + 1, // Data valid to SCL rising,
250ns from spec
        t_hold = (freq>> 1) + 1,   // SCL falling to SDA changing 0
from spec,
                        0.5us for AD9888
        t_su_sto = 4 * freq;          // SCL high to SDA high for
STOP condition,
                        4us from spec

        localparamtime_width = clogb2(t_low + 1); // Declare enough
bits to hold
                        maximum delay (5 us)

        reg [time_width-1:0] timer;

        //  Ceiling  of  log  base  2  from  the  Verilog  Language
Reference Manual:
        function integer clogb2;
        input [31:0] value;
        begin
        value = value - 1;
        for (clogb2 = 0; value > 0; clogb2 = clogb2 + 1)
        value = value >> 1;
        end
        endfunction

        reg [31:0] ctrl_reg;    // I2C control register
        // Bit definitions:
        //  31    Read / not write.
        //  30     Repeated Start.  On read cycles setting this bit
uses repeated
        start instad of stop start sequence.
        //  29:24 Reserved.
        //  23:17 7-bit I2C address of slave.
        //  16     Don't care.  Allows use of read or write address
when writing
        this byte.
        //  15:8  Register Subaddress
        //  7:0   Data to write.  Don't care on read cycles.

        //reg [31:0] status;      // I2C status register
        // Bit definitions
        //   31     Busy.  Not ready to accept new control register
writes.
        //  30     Address NACK.  Last address cycle resulted in no
acknowledge
        from slave.
        //  29     Data NACK.  Last data write cycle resulted in no
acknowledge
        from slave.
        //  28     Read.  Most recently completed cycle was a read.
Data in bits
                        7:0 is valid.
        //   27      Overrun.   An attempt was made to write the
ctrl_reg while
        busy.  Cleared
        //        on successful write of ctrl_reg.
        //  26     Initializing - waiting for SDA to go high after
```

```
module reset
      //  25:8  Reserved.  Tied to zero.
      //  7:0   Read data.  Valid after a read cycle when bit 28
is set.

      reg [26:0] shift_reg; // Data to shift out.  Includes ack
positions: 1 =
                  NACK or slave ack.

      reg [4:0] bit_count;  // Counts bits during shift states.

      reg [3:0] state;      // Main state machine state variable
      reg [3:0] rtn_state;  // Return state for "subroutines"

      regsda, scl;          // debounced, deglitched SDA and SCL
inputs

      regwr_cyc;             // Every access starts with a write
cycle for
      subaddress.

      reg [7:0] read_data;  // Input shift register for reads

      reg [3:0] scl_startup_count;   // Clock SCL at least 12
times after SDA
                  is detected high

      // Debounce and deglitch input signals
      reg [3:0] sda_sr, scl_sr;
      always @ (posedgesys_clock or posedge reset)
      if (reset)
      begin
      sda_sr<= 4'b1111;  // Start up assuming quiescent state of
inputs
      sda<= 1;
      end
      else
      begin
      sda_sr<= {sda_sr[2:0], sda_in};
      if (sda_sr == 4'b0000) sda<= 0;
      else if (sda_sr == 4'b1111) sda<= 1;
      scl_sr<= {scl_sr[2:0], scl_in};
      if (scl_sr == 4'b0000) scl<= 0;
      else if (scl_sr == 4'b1111) scl<= 1;
      end

      // Define states:
      localparampre_start_up = 0,
      start_up = 1,
      idle = 2,
      start = 3,
      clock_low = 4,
      shift_data = 5,
      clock_high = 6,
      stop = 7,
      spin = 15;
```

```
always @ (posedgesys_clock or posedge reset)
if (reset)
begin
timer<= t_low;
state<= pre_start_up;
rtn_state<= pre_start_up;
ctrl_reg<= 0;
status<= 32'h84000000; // Busy, initializing
shift_reg<= {27{1'b1}};
bit_count<= 0;
float_sda<= 1;
float_scl<= 1;
wr_cyc<= 1;
read_data<= 0;
scl_startup_count<= 0;
end
else
begin
if (wr_ctrl)
begin
if (status[31]) // busy
begin
status[27] <= 1;   // Set overrun flag on unsuccessful attempt to write
                        ctrl_reg
end
else // not busy
begin
ctrl_reg<= ctrl_data;
status[27] <= 0;  // Clear overrun flag on successful write to ctrl_reg
end
end
case (state)
      // In pre-start-up state wait for SDA to go high while clocking
                        SCL as necessary
pre_start_up:
begin
if (timer == 0) // every 5 us
begin
if (float_scl)
begin
if (sda&& (scl_startup_count == 12))  // quiescent?
begin
scl_startup_count<= 0;
state<= start_up;
end
else
float_scl<= 0; // Start another SCL clock cycle if SDA is still low
timer<= t_low;
scl_startup_count<= scl_startup_count + 1;
end
end
else  // Currently driving SCL
```

```
                begin
                float_scl<= 1; // Release SCL
                timer<= t_low;
                end
                end
                else if (scl | !float_scl) // Start timing after rising edge of SCL if
                                not driven
                begin
                timer<= timer - 1;
                end
                end
                      // In start-up state, generate a start and stop with no clocks in
                                between
                start_up:
                if (timer == 0) // every 5 us
                begin
                timer<= t_low;
                scl_startup_count<= scl_startup_count + 1;
                if (scl_startup_count == 2) float_sda<= 0; // Create a start condition
                if (scl_startup_count == 12) float_sda<= 1; // Create a stop condition
                if (scl_startup_count == 15) state <= idle;
                end
                else
                begin
                timer<= timer - 1;
                end
                end
                idle:
                begin
                float_sda<= 1;
                float_scl<= 1;
                wr_cyc<= 1;
                status[31] <= 0;  // Not busy
                status[26] <= 0;  // Done initialization
                if (wr_ctrl& !status[31])  // successful write to ctrl_reg
                begin
                state<= start;
                status[31] <= 1;  // go busy
                end
                end
                start:
                begin
                          // Create high to low transition on SDA.
                          // Both SDA and SCL were high at least 4.7us before entering
                                this state
                float_sda<= 0;
                float_scl<= 1;
                if (!sda) // Continue when we see sda driven low
                begin
                           //  7-bit  ADDR,  R/WN,  Slave  Ack,  8-bit Subaddr, Slave Ack,
```

```
                        8-bit Data, Slave Ack
                        // Data byte and final Slave Ack do not apply
for reads
                        // For Stop then start, SDA must be low after
last ack
      cycle.
                        // For repeated start, SDA must be high after
last ack
      cycle.
      if (ctrl_reg[31])   // reading requires subaddr write then
data read
      if (wr_cyc)
      shift_reg<=
      {ctrl_reg[23:17],1'b0,1'b1,ctrl_reg[15:8],1'b1,ctrl_reg[30]
,7'b0,1'b0};
      else
      shift_reg<=
{ctrl_reg[23:17],1'b1,1'b1,8'hff,1'b1,8'b0,1'b0};
      else                // Writing
      shift_reg<=

{ctrl_reg[23:17],1'b0,1'b1,ctrl_reg[15:8],1'b1,ctrl_reg[7:0
                        ],1'b1};
      bit_count<= 0;
      timer<= t_hd_sta;  // 4.0us from the spec
      rtn_state<= clock_low;
      state<= spin;
      end
      end
      clock_low:
      begin
                  // Assert SCL low and when it is low, wait for
t_hold before
                        changing SDA
      float_scl<= 0;
      if (!scl) // Continue when SCL line has gone low
      begin
      timer<= t_hold; // extra 0.5 us for AD9888
      rtn_state<= shift_data;
      state<= spin;
      end
      end
      shift_data:
      begin
                  // Shift data onto the SDA line
      float_sda<= shift_reg[26];
      shift_reg<= {shift_reg[25:0],1'b0};  // shift left
      timer<= t_low; // 4.7us min from spec
      rtn_state<= clock_high;
      state<= spin;
      end
      clock_high:
      begin
                  // Release low drive on SCL and when it goes high
                  // sample SDA and move on
      float_scl<= 1;
      if (scl)
```

```
                begin
                bit_count<= bit_count + 1;
                if (bit_count == 8) // Address ACK cycle
                begin
                status[30] <= sda;  // SDA should be driven low for slave
ACK
                end
                else if ((bit_count == 17) &wr_cyc || (bit_count == 26))
// Data ACK
                                cycles
                begin
                status[29] <= sda;  // SDA should be driven low for slave
ACK
                end
                if ((bit_count == 18) &ctrl_reg[31]    // Reading and past
first data
                                ack
                                     || (bit_count == 27))                 //
Past second
                                data ack
                begin
                timer<= t_su_sto;  // 4.0us from spec
                rtn_state<= stop;
                state<= spin;
                end
                else
                begin
                if (bit_count != 17) read_data<= {read_data[6:0],sda};  //
shift data
                                in, MSB first
                timer<= t_high;  // 4.0us from spec, but use 5.0 instead to
meet cycle
                                 time
                rtn_state<= clock_low;
                state<= spin;
                end
                end
                end
                stop:
                begin
                                // We get here twice for read cycles, once for
writes.  On
                                reads if
                                // using repeated start we don't need to wait as
long before
                                asserting SDA
                                // for start since t_su_sto has already elapsed
(4.0us)
                float_sda<= 1;   // SDA will already be high in the case of
repeated
                                start
                if (sda)
                begin
                if (ctrl_reg[31]) // reading
                begin
                if (wr_cyc)   // just finished sending subaddress
                begin
                if (ctrl_reg[30]) // repeated start
```

```
                timer<= t_su_sta - t_su_sto;
                else
                timer<= t_su_sta;
                rtn_state<= start;
                end
                else
                begin
                status[7:0] <= read_data;
                status[28] <= 1;
                timer<= t_su_sta;  // Setup to start is same as bus-free,
        4.7us from
                                spec
                rtn_state<= idle;  // For writes we're all done
                end
                wr_cyc<= 0;
                state<= spin;
                end
                else              // writing
                begin
                status[28] <= 0;
                timer<= t_su_sta;  // Setup to start is same as bus-free,
        4.7us from
                                spec
                rtn_state<= idle;  // For writes we're all done
                state<= spin;
                end
                end
                end
                spin:
                begin
                                // stay in this state for requested time period
        then "return"
                if (timer > 0)
                begin
                timer<= timer - 1;
                end
                else
                begin
                state<= rtn_state;
                end
                end
                endcase
                end

                endmodule

                `default_nettype wire
```

В таблице 8.5.2 показан листинг проверки I2C Verilogна языке Verilog

Таблица 8.5.2– Листинг проверки I2C на языке Verilog

```
                module i2c_master_tb;
                // Inputs
                regsys_clock;
                reg reset;
```

```
reg [31:0] ctrl_data;
regwr_ctrl;
// Outputs
wire [31:0] status;
// Bidirs
wire SDA;
wire SCL;

pullup (SDA);
pullup (SCL);

// Instantiate the Unit Under Test (UUT)
I2C_master
#(
  .freq         (100)
)
uut
(
  .SDA          (SDA),
  .SCL          (SCL),
  .sys_clock    (sys_clock),
  .reset        (reset),
  .ctrl_data    (ctrl_data),
  .wr_ctrl      (wr_ctrl),
  .status       (status)
);
initial begin
  // Initialize Inputs
sys_clock = 0;
reset = 1;
  // Write 'h44 to register 'h55 in I2C slave 'h66
ctrl_data = 32'h00665544;
wr_ctrl = 0;
  // Wait 100 ns for global reset to finish
  #101;
reset = 0;
  // Add stimulus here
  #220000
  @ (posedgesys_clock) wr_ctrl<= #1 1;
  @ (posedgesys_clock) wr_ctrl<= #1 0;
end
alwayssys_clock = #5 !sys_clock;
endmodule
```

Рисунок 8.5 Временные диаграммы работы I2Cна языке Verilog

Теперь проведем тоже самое на языке VHDL.

Задание: реализация I2C интерфейса на языке VHDL

Первоисточник:https://www.eewiki.net/display/LOGIC/I2C%20Master%20(VHDL)#I2CMaster(VHDL)-TheoryofOperation

Представленная реализация интерфейса представляет собой мастер, построенный на базе спецификации «NXPUM10204 I2C-busspecification». Включает в себя следующий функционал:

- определяемые пользователем параметры, определяющие частоту внутреннего тактового генератора и скорость шины (SCL);

- система в генерирует следующий комплекс условий: Start,Stop, RepeatedStart и Acknowledge;

- используется 7-битная адресация подчиненных устройств.

На рисунке 8.6.1 представлена структура моделируемой системы.

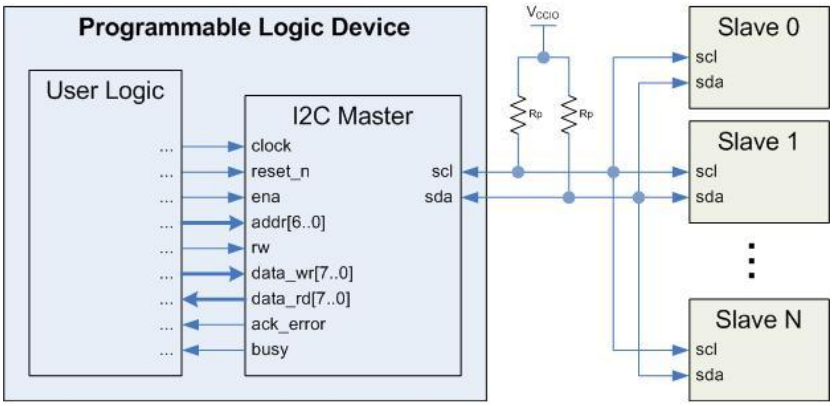


Рисунок 8.6.1– Структура моделируемой системы I2CVHDL

За программную реализацию мастера (I2CMaster) отвечает главная программа, а за код логики пользователя (UserLogic) – тестовая программа.

В таблице 8.5.3 показан листинг реализации I2C Verilogна языке VHDL

Таблица 8.5.3– Листинг реализации I2C на языке VHDL

-- FileName: i2c_master.vhd
-- Dependencies: none
-- Design Software: Quartus II 64-bit Version 13.1 Build 162 SJ Full Version
--
-- HDL CODE IS PROVIDED "AS IS." DIGI-KEY EXPRESSLY DISCLAIMS ANY
-- WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
-- LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
-- PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
-- BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
-- DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
-- PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS

```
--     BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE
                THEREOF),
--     ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--------------------------------------------------------------------
                --------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY i2c_master IS
  GENERIC(
    input_clk : INTEGER := 50_000_000; --input clock speed from user
                logic in Hz
    bus_clk   : INTEGER := 400_000);   --speed the i2c bus (scl) will
                run at in Hz
  PORT(
    clk       : IN     STD_LOGIC;                    --system clock
    reset_n   : IN     STD_LOGIC;                    --active low reset
    ena       : IN     STD_LOGIC;                    --latch in command
    addr      : IN     STD_LOGIC_VECTOR(6 DOWNTO 0); --address of target
                slave
    rw        : IN     STD_LOGIC;                    --'0' is write, '1
                is read
    data_wr   : IN     STD_LOGIC_VECTOR(7 DOWNTO 0); --data to write to
                slave
    busy      : OUT    STD_LOGIC;                    --indicates
                transaction in progress
    data_rd   : OUT    STD_LOGIC_VECTOR(7 DOWNTO 0); --data read from
                slave
    ack_error : BUFFER STD_LOGIC;                    --flag if improper
                acknowledge from slave
    sda       : INOUT  STD_LOGIC;                    --serial data
                output of i2c bus
    scl       : INOUT  STD_LOGIC);                   --serial clock
                output of i2c bus
END i2c_master;
ARCHITECTURE logic OF i2c_master IS
  CONSTANT divider  :  INTEGER := (input_clk/bus_clk)/4; --number of
                clocks in 1/4 cycle of scl
  TYPE machine IS(ready, start, command, slv_ack1, wr, rd, slv_ack2
                mstr_ack, stop); --needed states
```

```
                SIGNAL state         : machine;                          --
        state machine
                SIGNAL data_clk      : STD_LOGIC;                        --
        data clock
                        for sda
                SIGNAL data_clk_prev : STD_LOGIC;                        --
        data clock
                        during previous system clock
                SIGNAL scl_clk       : STD_LOGIC;                        --
        constantly
                        running internal scl
                SIGNAL scl_ena       : STD_LOGIC := '0';                 --
        enables
                        internal scl to output
                SIGNAL sda_int       : STD_LOGIC := '1';                 --
        internal sda
    SIGNAL sda_ena_n     : STD_LOGIC;                      --enables
                        internal sda to output
                SIGNAL addr_rw       : STD_LOGIC_VECTOR(7 DOWNTO 0);     --
        latched in
                        address and read/write
                SIGNAL data_tx       : STD_LOGIC_VECTOR(7 DOWNTO 0);     --
        latched in
                        data to write to slave
                SIGNAL data_rx       : STD_LOGIC_VECTOR(7 DOWNTO 0);     --
        data received
                        from slave
                SIGNAL bit_cnt       : INTEGER RANGE 0 TO 7 := 7;        --
        tracks bit
                       number in transaction
                SIGNAL stretch       : STD_LOGIC := '0';                 --
        identifies if
                        slave is stretching scl
             BEGIN

    --generate the timing for the bus clock (scl_clk) and the data clock
                        (data_clk)
                PROCESS(clk, reset_n)
                  VARIABLE count   :   INTEGER RANGE 0 TO divider*4;    --
        timing for clock
                        generation
                BEGIN
                  IF(reset_n = '0') THEN               --reset asserted
                    stretch <= '0';
                    count := 0;
                  ELSIF(clk'EVENT AND clk = '1') THEN
                    data_clk_prev <= data_clk;            --store previous
        value of data
                        clock
                    IF(count = divider*4-1) THEN          --end of timing
        cycle
                      count := 0;                       --reset timer
          ELSIF(stretch = '0') THEN          --clock stretching from slave
                        not detected
                      count := count + 1;                 --continue clock
        generation
                        timing
                    END IF;
                    CASE count IS
```

```
                    WHEN 0 TO divider-1 =>              --first 1/4 cycle
of clocking
                      scl_clk <= '0';
                      data_clk <= '0';
                    WHEN divider TO divider*2-1 =>     --second 1/4 cycle
of clocking
                      scl_clk <= '0';
                      data_clk <= '1';
                    WHEN divider*2 TO divider*3-1 =>   --third 1/4 cycle
of clocking
                      scl_clk <= '1';                 --release scl
                      IF(scl = '0') THEN                --detect if slave
is
                       stretching clock
                        stretch <= '1';
                      ELSE
                        stretch <= '0';
                      END IF;
                      data_clk <= '1';
                    WHEN OTHERS =>                     --last 1/4 cycle of
clocking
                      scl_clk <= '1';
                      data_clk <= '0';
                  END CASE;
                END IF;
              END PROCESS;

              --state machine and writing to sda during scl low (data_clk
rising
                     edge)
              PROCESS(clk, reset_n)
              BEGIN
                IF(reset_n = '0') THEN                --reset asserted
                  state <= ready;                          --return to
initial state
                  busy <= '1';                          --indicate not
available
                  scl_ena <= '0';                       --sets scl high
impedance
                  sda_int <= '1';                       --sets sda high
impedance
                  ack_error <= '0';                              --clear
acknowledge error
                       flag
                  bit_cnt <= 7;                         --restarts data
bit counter
                  data_rd <= "00000000";                --clear data read
port
                ELSIF(clk'EVENT AND clk = '1') THEN
                  IF(data_clk = '1' AND data_clk_prev = '0') THEN  --data
clock
                     rising edge
                    CASE state IS
                      WHEN ready =>                     --idle state
                        IF(ena = '1') THEN                --transaction
requested
                          busy <= '1';                  --flag busy
             addr_rw <= addr & rw;          --collect requested slave
                        address and command
```

```
                data_tx <= data_wr;             --collect requested data to
                        write
                          state <= start;                   --go to start
        bit
                        ELSE                                --remain idle
                          busy <= '0';                      --unflag busy
                          state <= ready;                   --remain idle
                        END IF;
                      WHEN start =>                          --start bit of
        transaction
                        busy <= '1';                        --resume busy if
        continuous
                      mode
                        sda_int  <=  addr_rw(bit_cnt);        --set  first
        address bit to
                      bus
                        state <= command;                   --go to command
                    WHEN command =>                         --address and
        command byte
                      of transaction
                        IF(bit_cnt  =  0)  THEN                 --command
        transmit finished
                sda_int <= '1';                 --release sda for slave
                      acknowledge
                bit_cnt <= 7;                   --reset bit counter for
                        "byte" states
                          state <= slv_ack1;                --go to slave
        acknowledge
                        (command)
                        ELSE                                 --next clock
        cycle of
                      command state
                          bit_cnt <= bit_cnt - 1;         --keep track of
        transaction
                      bits
                          sda_int    <=    addr_rw(bit_cnt-1);    --write
        address/command bit
                      to bus
                          state <= command;                 --continue with
        command
                        END IF;
           WHEN slv_ack1 =>                     --slave acknowledge bit
                      (command)
                        IF(addr_rw(0) = '0') THEN         --write command
                          sda_int  <=  data_tx(bit_cnt);     --write  first
        bit of data
                          state <= wr;                      --go to write
        byte
                        ELSE                                --read command
                          sda_int <= '1';                   --release sda
        from incoming
                      data
                          state <= rd;                       --go to read
        byte
                        END IF;
                      WHEN wr =>                             --write byte of
        transaction
                        busy <= '1';                        --resume busy if
        continuous
```

```
                        mode
                         IF(bit_cnt = 0) THEN                --write byte
            transmit
                        finished
                           sda_int <= '1';                   --release sda
            for slave
                        acknowledge
                           bit_cnt <= 7;                      --reset bit
            counter for
                        "byte" states
                  state <= slv_ack2;             --go to slave acknowledge
                        (write)
                         ELSE                                --next clock
            cycle of write
                        state
                           bit_cnt <= bit_cnt - 1;          --keep track of
            transaction
                        bits
                           sda_int <= data_tx(bit_cnt-1); --write next bit
            to bus
                           state <= wr;                        --continue
            writing
                         END IF;
                         END IF;

               busy <= '1';                     --resume busy if continuous
                        mode
               IF(bit_cnt = 0) THEN             --read byte receive
                        finished
                           IF(ena = '1' AND addr_rw = addr & rw) THEN
                        --continuing with another read at same
                                                             --address
                             sda_int <= '0';                --acknowledge
            the byte has
                        been received
                           ELSE                              --stopping or
            continuing
                        with a write
                             sda_int <= '1';                 --send a no-
            acknowledge
                        (before stop or repeated start)
                           END IF;
                           bit_cnt <= 7;                      --reset bit
            counter for
                        "byte" states
                           data_rd  <=  data_rx;                --output
            received data
                           state <= mstr_ack;               --go to master
            acknowledge
                         ELSE                                --next clock
            cycle of read
                        state
                           bit_cnt <= bit_cnt - 1;          --keep track of
            transaction
                        bits
                           state <= rd;                        --continue
            reading
                         END IF;
            WHEN slv_ack2 =>                    --slave acknowledge bit
```

```
                        (write)
                         IF(ena = '1') THEN                    --continue
            transaction
                           busy <= '0';                        --continue is
            accepted
                  addr_rw <= addr & rw;          --collect requested slave
                        address and command
                           data_tx <= data_wr;                  --collect
            requested data to
                        write
                           IF(addr_rw = addr & rw) THEN      --continue
            transaction with
                        another write
                             sda_int <= data_wr(bit_cnt);  --write first
            bit of data
                             state <= wr;                    --go to write
            byte
                           ELSE                                --continue
            transaction with
                        a read or new slave
                             state <= start;              --go to repeated
            start
                           END IF;
                         ELSE                                  --complete
            transaction
                           state <= stop;                 --go to stop bit
                         END IF;
              WHEN mstr_ack =>                  --master acknowledge bit
                        after a read
                         IF(ena = '1') THEN                    --continue
            transaction
                           busy <= '0';                        --continue is
            accepted and
                        data received is available on bus
                 addr_rw <= addr & rw;          --collect requested slave
                        address and command
                  data_tx <= data_wr;           --collect requested data to
                        write
                  IF(addr_rw = addr & rw) THEN   --continue transaction with
                        another read
                    sda_int <= '1';             --release sda from incoming
                        data
                             state <= rd;                     --go to read
            byte
                           ELSE                                --continue
            transaction with
                        a write or new slave
                             state <= start;              --repeated start
                           END IF;
                         ELSE                                  --complete
            transaction
                           state <= stop;                 --go to stop bit
                         END IF;
                       WHEN stop =>                          --stop bit of
            transaction
                         busy <= '0';                    --unflag busy
                         state <= ready;                    --go to idle
            state
                    END CASE;
```

```
                    ELSIF(data_clk = '0' AND data_clk_prev = '1') THEN  --data
          clock
                        falling edge
                        CASE state IS
                          WHEN start =>
                            IF(scl_ena = '0') THEN                          --
          starting new
                        transaction
                              scl_ena <= '1';                           --enable
          scl output
                              ack_error <= '0';                          --reset
          acknowledge
                        error output
                            END IF;
             WHEN slv_ack1 =>                           --receiving slave
                        acknowledge (command)
               IF(sda /= '0' OR ack_error = '1') THEN  --no-acknowledge or
                        previous no-acknowledge
                 ack_error <= '1';                      --set error output
                        if no-acknowledge
                            END IF;
             WHEN rd =>                                 --receiving slave
                        data
               data_rx(bit_cnt) <= sda;                 --receive current
                        slave data bit
             WHEN slv_ack2 =>                           --receiving slave
                        acknowledge (write)
               IF(sda /= '0' OR ack_error = '1') THEN  --no-acknowledge or
                        previous no-acknowledge
                              ack_error <= '1';                         --set
          error output
                        if no-acknowledge
                            END IF;
                        WHEN stop =>
                            scl_ena <= '0';                           --disable
          scl
                        WHEN OTHERS =>
                            NULL;
                      END CASE;
                    END IF;
                  END IF;
                END PROCESS;

                --set sda output
                WITH state SELECT
        sda_ena_n <= data_clk_prev WHEN start,     --generate start
                        condition
                             NOT  data_clk_prev  WHEN  stop,    --generate
          stop condition
                  sda_int WHEN OTHERS;          --set to internal sda
                        signal

                --set scl and sda outputs
                scl <= '0' WHEN (scl_ena = '1' AND scl_clk = '0') ELSE 'Z';
                sda <= '0' WHEN sda_ena_n = '0' ELSE 'Z';

              END logic;
```

В таблице 8.5.4 показан листинг проверки I2C Verilogна языке VHDL

Таблица 8.5.4– Листинг проверки I2C на языке VHDL

```
`timescale 1ns / 1ps
module i2c_master_tb;

    // Inputs
    reg clk;
    reg reset_n;
    reg ena;
    reg [6:0] addr;
    reg rw;
    reg [7:0] data_wr;

    // Outputs
    wire busy;
    wire [7:0] data_rd;
    wire ack_error;

    // Bidirs
    wire sda;
    wire scl;

    // Instantiate the Unit Under Test (UUT)
    i2c_master uut (
        .clk(clk),
        .reset_n(reset_n),
        .ena(ena),
        .addr(addr),
        .rw(rw),
        .data_wr(data_wr),
        .busy(busy),
        .data_rd(data_rd),
        .ack_error(ack_error),
        .sda(sda),
        .scl(scl)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        reset_n = 1;//отсутствие сброса (отрицательная логика)
        ena = 1;    //сигнал разрешения
        addr = 7'b1110001;//адрес подчиненного
        rw = 0;//прием - 1, передача - 0
        data_wr = 8'b10101010;//регистр передаваемых данных


        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
```

end
always begin #5 clk=~clk; end //сигнал внутреннего
тактового //генератора
endmodule

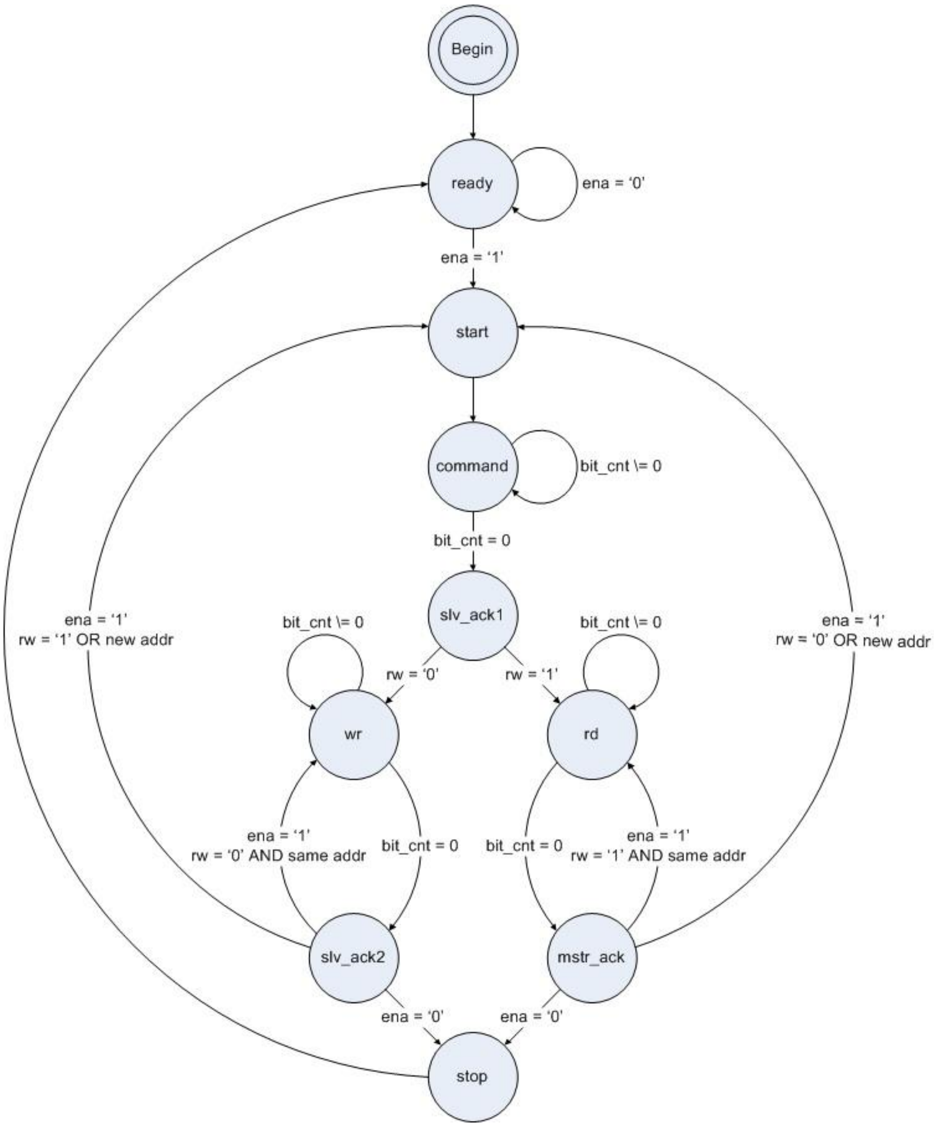


Рисунок 8.6.2 – Граф состояний мастера

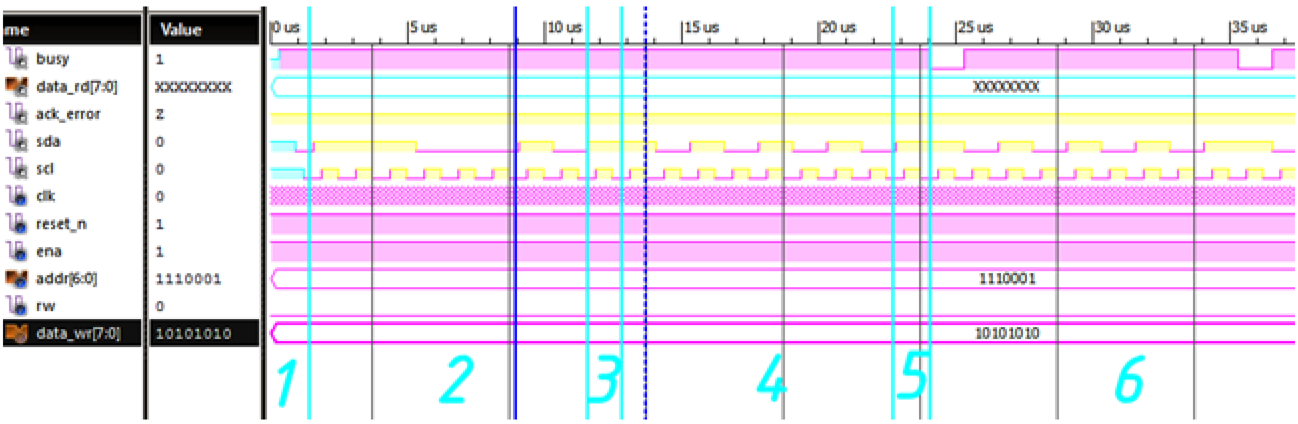


Рисунок 8.6.3 – Временная диаграмма работы I2Сна языке VHDL

Описание участков временной диаграммы:

1. выдача сигнала «Start»;

2. выдача передача 7-битного адреса на шину SDA и 1 бита «чтение/запись»;

3. опрос подтверждения (acknowledge), которое отсутствует в виду того, что подчиненных на шине нет;

4. отправка октета данных;

5. опрос подтверждения;

6. передача последующих данных.

8.6 SPIVerilog

SPI (англ. Serial Peripheral Interface, SPI bus — последовательный периферийный интерфейс, шина SPI) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным (англ. four-wire) интерфейсом. Реализовать данный тип шины на языке Verilog.

В таблице 8.6.1 показан листинг реализации SPIна языке Verilog

Таблица 8.6.1– Листинг реализации SPI на языке Verilog

```
module spi_master(rstb,clk,mlb,start,tdat,cdiv,din,
                  ss,sck,dout,done,rdata);
inputrstb,clk,mlb,start;
input [7:0] tdat;  //transmit data
input [1:0] cdiv;  //clock divider
   input din;
   outputregss;
   outputregsck;
   outputregdout;
outputreg done;
   outputreg [7:0] rdata; //received data

parameter idle=2'b00;
parameter send=2'b10;
parameter finish=2'b11;
reg [1:0] cur,nxt;

   reg [7:0] treg,rreg;
   reg [3:0] nbit;
   reg [4:0] mid,cnt;
   regshift,clr;

//FSM i/o
always @(start or cur or nbit or cdiv or rreg) begin
         nxt=cur;
         clr=0;
```

```
                shift=0;//ss=0;
                case(cur)
                      idle:begin
                            if(start==1)
                begin
                                                case (cdiv)
                                                      2'b00: mid=2;
                                                      2'b01: mid=4;
                                                      2'b10: mid=8;
                                                      2'b11: mid=16;
                                                endcase
                                                shift=1;
                                                done=1'b0;
                                                nxt=send;
                                                end
                end //idle
                      send:begin
                            ss=0;
                            if(nbit!=8)
                                  begin shift=1; end
                            else begin
                                                rdata=rreg;done=1'b1;
                                                nxt=finish;
                                  end
                            end//send
                      finish:begin
                                  shift=0;
                                  ss=1;
                                  clr=1;
                                  nxt=idle;
                            end
                      default: nxt=finish;
          endcase
          end//always

          //state transistion
          always@(negedgeclk or negedgerstb) begin
          if(rstb==0)
          cur<=finish;
          else
          cur<=nxt;
          end

          //setup falling edge (shift dout) sample rising edge (read din)
          always@(negedgeclk or posedgeclr) begin
          if(clr==1)
                begincnt=0; sck=1; end
          else begin
             if(shift==1) begin
                cnt=cnt+1;
             if(cnt==mid) begin
                sck=~sck;
                cnt=0;
                end //mid
```

```
                    end //shift
                end //rst
                end //always

                //sample @ rising edge (read din)
                always@(posedgesck or posedgeclr ) begin // or negedgerstb
                                nbit=0;  rreg=8'hFF;  end
                else begin
                        if(mlb==0) //LSB first, din@msb -> right shift
                                begin  rreg={din,rreg[7:1]};  end
                        else  //MSB first, din@lsb -> left shift
                                begin  rreg={rreg[6:0],din};  end
                        nbit=nbit+1;
                end //rst
                end //always

                always@(negedgesck or posedgeclr) begin
                if(clr==1) begin
                   treg=8'hFF;  dout=1;
                end
                else begin
                        if(nbit==0) begin //load data into TREG
                                treg=tdat; dout=mlb?treg[7]:treg[0];
                        end //nbit_if
                        else begin
                                if(mlb==0) //LSB first, shift right
                                        begintreg={1'b1,treg[7:1]};
        dout=treg[0]; end
                                else//MSB first shift LEFT
                                        begintreg={treg[6:0],1'b1};
        dout=treg[7]; end
                        end
                end //rst
                end //always
                endmodule


                modulespi_slave            (rstb,ten,tdata,mlb,ss,sck,sdin,
        sdout,done,rdata);
                inputrstb,ss,sck,sdin,ten,mlb;
                input [7:0] tdata;
                outputsdout;           //slave out   master in
                outputreg done;
                outputreg [7:0] rdata;

                reg [7:0] treg,rreg;
                reg [3:0] nb;
                wiresout;

                assignsout=mlb?treg[7]:treg[0];
                  assign sdout=( (!ss)&&ten )?sout:1'bz; //if 1=> send data
        else
                        TRI-STATE sdout

                //read from  sdout
```

```
                always @(posedgesck or negedgerstb)
                begin
                if (rstb==0)
                        beginrreg = 8'h00;  rdata = 8'h00; done = 0; nb =
        0; end
                          //
                   else if (!ss) begin
                              if(mlb==0)   //LSB  first,  in@msb  ->  right
        shift
                                    beginrreg ={sdin,rreg[7:1]}; end
                              else     //MSB first, in@lsb -> left shift
                                    beginrreg ={rreg[6:0],sdin}; end
                        //increment bit count
                              nb=nb+1;
                              if(nb!=8) done=0;
                              else  beginrdata=rreg; done=1; nb=0; end
                        end    //if(!ss)_END  if(nb==8)
                end

                //send to  sdout
                always @(negedgesck or negedgerstb)
                begin
                   if (rstb==0)
                        begintreg = 8'hFF; end
                   else begin
                        if(!ss) begin
                              if(nb==0) treg=tdata;
                              else begin
                              if(mlb==0)   //LSB  first,  out=lsb  ->  right
        shift
                                    begintreg   =   {1'b1,treg[7:1]};
        end
                              else     //MSB first, out=msb -> left shift
                                    begintreg   =   {treg[6:0],1'b1};
        end

                              end
                        end //!ss
                   end //rstb

                end //always

                endmodule
```

В таблице 8.6.2 показан листинг проверкиSPIна языке Verilog

Таблица 8.6.2– Листинг проверкиSPI на языке Verilog

```
                moduleTB_SPI_MasSlv;
                regrstb;
                regclk = 1'b0;
                regmlb = 1'b0;
                reg start = 1'b0;
                reg [7:0] m_tdat = 8'b00000000;
                reg [1:0] cdiv = 0;
                wire din;
                wiress;
```

```
wiresck;
wiredout;
wireMdone;
wire [7:0] Mrdata;
   reg ten = 1'b0;
reg [7:0] s_tdata = 8'b00000000;
wireSLVdone;
wire [7:0] SLVrdata;

parameter PERIOD = 50;
parameter real DUTY_CYCLE = 0.5;
parameter OFFSET = 100;
initial begin  // Clock process for clk
        #OFFSET;
forever
clk = 1'b0;
#(PERIOD-(PERIOD*DUTY_CYCLE)) clk = 1'b1;
#(PERIOD*DUTY_CYCLE);
end
end
// to end simulation
   initial #10000 $stop;

//uut MASTER instantiation
spi_master MAS (
        .rstb(rstb),
        .clk(clk),
        .mlb(mlb),
        .start(start),
        .tdat(m_tdat),
        .cdiv(cdiv),
        .din(din),
        .sck(sck),
        .dout(dout),
        .done(Mdone),
        .rdata(Mrdata));
//uut SLAVE instantiation
   spi_slave SLV (
        .rstb(rstb),
        .ten(ten),
        .tdata(s_tdata),
        .mlb(mlb),
        .ss(ss),
        .sck(sck),
        .sdin(dout),
        .sdout(din),
        .done(SLVdone),
        .rdata(SLVrdata));

// timed contrl signals
initial begin
        #10 rstb = 1'b0;
        #100;
                rstb = 1'b1;start = 1'b0;
                m_tdat = 8'b01111100;
```

```
                                cdiv = 2'b00;
                        #100  start = 1'b1;ten=1; //s_tdata=8'hAC;
                        #100  start = 1'b0;

            #1800 mlb = 1'b1; cdiv=2'b01;
                              m_tdat=8'b00011100;//s_tdata=8'h64;
                        #100  start = 1'b1;
                         #100  start = 1'b0;
                         #2202;

                        #100  start = 1'b1;
                         #100  start = 1'b0;
                         #2000;

                m_tdat=~m_tdat;
                        #100  start = 1'b1;
                         #100  start = 1'b0;
                         #2000;
                end

                always @ (rstb or Mrdata) begin
                if(rstb==0)
                s_tdata = 8'hAA;
                else
                begin
                           # 10 s_tdata = Mrdata;
                end
                end
                endmodule
```

Рисунок 8.7 – Временная диаграмма работы SPIна языке Verilog

Теперь седалем тоже самое на языке VHDL.

В таблице 8.6.3 показан листинг реализации SPIна языке VHDL

Таблица 8.6.3– Листинг реализации SPI на языке VHDL

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

```
use ieee.numeric_std.all;

entity spi_controller is
generic(
  N                     : integer := 8;      -- number of bit
                        to serialize
  CLK_DIV                  : integer := 100 );  -- input clock
                        divider to generate output serial clock; o_sclk frequency =
                        i_clk/(2*CLK_DIV)
 port (
  i_clk  : in  std_logic;
  i_rstb  : in  std_logic;
  i_tx_start  : in  std_logic;  -- start TX on serial line
  o_tx_end  : out std_logic;  -- TX data completed;
                        o_data_parallel available
  i_data_parallel   : in   std_logic_vector(N-1 downto 0); --
                        data to sent
  o_data_parallel   : out  std_logic_vector(N-1 downto 0); --
                        received data
  o_sclk  : out std_logic;
  o_ss  : out std_logic;
  o_mosi  : out std_logic;
  i_miso  : in  std_logic);
end spi_controller;

architecture rtl of spi_controller is
type t_spi_controller_fsm is (
                                ST_RESET   ,
                                ST_TX_RX   ,
                                ST_END     );

signal r_counter_clock  : integer range 0 to CLK_DIV*2;
signal r_sclk_rise  :std_logic;
signal r_sclk_fall  :std_logic;
signal r_counter_clock_ena  :std_logic;

signal r_counter_data  : integer range 0 to N;
signal w_tc_counter_data  :std_logic;

signal r_st_present  :t_spi_controller_fsm;
signal w_st_next  :t_spi_controller_fsm;
signal r_tx_start  :std_logic;  -- start TX on serial line
signal r_tx_data  :std_logic_vector(N-1 downto 0);  -- data to sent
signal  r_rx_data   :std_logic_vector(N-1  downto  0);   -- received data

begin

w_tc_counter_data<= '0' when(r_counter_data>0) else '1';
-------------------------------------------------------------
```

```
                    -- FSM
                    p_state : process(i_clk,i_rstb)
                    begin
                       if(i_rstb='0') then
                             r_st_present<= ST_RESET;
                       elsif(rising_edge(i_clk)) then
                             r_st_present<= w_st_next;
                       end if;
                    end process p_state;

                    p_comb : process(
                    r_st_present  ,
                    w_tc_counter_data  ,
                    r_tx_start  ,
                    r_sclk_rise  ,
                    r_sclk_fall  )
                    begin
                       case r_st_present is
                             when  ST_TX_RX      =>
                                   if       (w_tc_counter_data='1') and
                                  (r_sclk_rise='1')  then    w_st_next<=  ST_END
            ;
                                   else
                                  w_st_next<= ST_TX_RX     ;
                             r_st_present<= ST_RESET;
                       elsif(rising_edge(i_clk)) then
                             r_st_present<= w_st_next;
                       end if;
                    end process p_state;

                    p_comb : process(
                    r_st_present  ,
                    w_tc_counter_data  ,
                    r_tx_start  ,
                    r_sclk_rise  ,
                    r_sclk_fall  )
                    begin
                       case r_st_present is
                             when  ST_TX_RX      =>
                                   if       (w_tc_counter_data='1') and
                                  (r_sclk_rise='1')  then    w_st_next<=  ST_END
            ;
                                   else
                                  w_st_next<= ST_TX_RX     ;
                                   end if;

                             when  ST_END      =>
                                   if(r_sclk_fall='1') then
                                         w_st_next<= ST_RESET    ;
                                   else
                                         w_st_next<= ST_END    ;
                                   end if;

                             when  others           =>  -- ST_RESET
                                   if(r_tx_start='1')   then         w_st_next<=
```

```
                ST_TX_RX ;
                                else                                    w_st_next<=
                ST_RESET ;
                                end if;
                        end case;
                    end process p_comb;

                    p_state_out : process(i_clk,i_rstb)
                    begin
                        if(i_rstb='0') then
                                r_tx_start<= '0';
                                r_tx_data<= (others=>'0');
                                r_rx_data<= (others=>'0');
                                o_tx_end<= '0';
                                o_data_parallel<= (others=>'0');

                                r_counter_data<= N-1;
                                r_counter_clock_ena<= '0';

                                o_sclk<= '1';
                                o_ss<= '1';
                                o_mosi<= '1';
                        elsif(rising_edge(i_clk)) then
                                r_tx_start<= i_tx_start;

                                case r_st_present is
                                        when ST_TX_RX         =>
                                                o_tx_end<= '0';
                                                r_counter_clock_ena<= '1';
                                                if(r_sclk_rise='1') then
                                                        o_sclk<= '1';
                                                        r_rx_data<= r_rx_data(N-2 downto
                                                0)&i_miso;
                                                        if(r_counter_data>0) then
                                                                r_counter_data<=
                ST_TX_RX ;
                                                        end if;
                                                elsif(r_sclk_fall='1') then
                                                        o_sclk<= '0';
                                                        o_mosi<= r_tx_data(N-1);
                                                        r_tx_data<=  r_tx_data(N-2  downto
                0)&'1';
                                                end if;
                                                o_ss<= '0';

                                        when ST_END          =>
                                                o_tx_end<= r_sclk_fall;
                                                o_data_parallel<= r_rx_data;
                                                r_counter_data<= N-1;
                                                r_counter_clock_ena<= '1';
                                                o_ss<= '0';

                                        when others                =>  -- ST_RESET
                                                r_tx_data<= i_data_parallel;
                                                o_tx_end<= '0';
                                                r_counter_data<= N-1;
```

```
                                    r_counter_clock_ena<= '0';
                                    o_sclk<= '1';
                                    o_ss<= '1';
                           end case;
                     end if;
               end process p_state_out;


               p_counter_clock : process(i_clk,i_rstb)
               begin
                  if(i_rstb='0') then
                        r_counter_clock<= 0;
                        r_sclk_rise<= '0';
                        r_sclk_fall<= '0';
                  elsif(rising_edge(i_clk)) then
                        if(r_counter_clock_ena='1') then   --sclk = '1' by
          default
                              if(r_counter_clock=CLK_DIV-1)  then    --firse
          edge =
                           fall
                                    r_counter_clock<= r_counter_clock + 1;
                                    r_sclk_rise<= '0';
                                    r_sclk_fall<= '1';
                              elsif(r_counter_clock=(CLK_DIV*2)-1) then
                                    r_counter_clock<= 0;
                                    r_sclk_rise<= '1';
                                    r_sclk_fall<= '0';
                              else
                                    r_counter_clock<= r_counter_clock + 1;
                                    r_sclk_rise<= '0';
                                    r_sclk_fall<= '0';
                              end if;
                               else
                              r_counter_clock<= 0;
                              r_sclk_rise<= '0';
                              r_sclk_fall<= '0';
                        end if;
                  end if;
               end process p_counter_clock;

               end rtl;
```

В таблице 8.6.4 показан листинг проверкиSPIна языке VHDL

Таблица 8.6.4– Листинг проверкиSPI на языке VHDL

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

               entity tb_spi_controller is
               end tb_spi_controller;

               architecture rtl of tb_spi_controller is

               component spi_controller
```

```
        generic(
          N                    : integer := 8;      -- number of bit
                               to serialize
          CLK_DIV              : integer := 100 );  -- input clock
                               divider to generate output serial clock; o_sclk frequency =
                               i_clk/(2*CLK_DIV)
        port (
          i_clk  : in  std_logic;
          i_rstb  : in  std_logic;
          i_tx_start  : in  std_logic;  -- start TX on serial line
          o_tx_end  : out std_logic;  -- TX data completed;
                               o_data_parallel available
          i_data_parallel  : in  std_logic_vector(N-1 downto 0); --
                               data to sent
          o_data_parallel  : out std_logic_vector(N-1 downto 0); --
                               received data
          o_sclk  : out std_logic;
          o_ss  : out std_logic;
          o_mosi  : out std_logic;
          i_miso  : in  std_logic);
        end component;

        constant N                     : integer := 16;      -- number of bit to
                              serialize
        constant CLK_DIV               : integer := 100;  -- input clock divider
                              to generate output serial clock; o_sclk frequency =
                              i_clk/(2*CLK_DIV)

        signal i_clk  :std_logic:='0';
        signal i_rstb  :std_logic;
        signal i_tx_start  :std_logic;  -- start TX on serial line
        signal o_tx_end   :std_logic;   -- TX data completed; o_data_parallel
                              available
        signal i_data_parallel  :std_logic_vector(N-1 downto 0);  -- data to
                              sent
        signal o_data_parallel  :std_logic_vector(N-1 downto 0);  -- received
                              data
        signal o_sclk  :std_logic;
        signal o_ss  :std_logic;
        signal o_mosi  :std_logic;

        signal mosi_test  :std_logic_vector(N-1 downto 0);  -- tx data
        signal miso_test   :std_logic_vector(N-1 downto 0);   -- received data
        signal count_rise  : integer;
        signal count_fall  : integer;
```

```
                    begin

                    i_clk<= not i_clk after 5 ns;
                    i_rstb<= '0', '1' after 163 ns;


                    u_spi_controller :spi_controller
                    generic map(
                       N                    => N                    ,
                       CLK_DIV              => CLK_DIV              )
                    port map(
                       i_clk                      =>i_clk  ,
                       i_rstb                     =>i_rstb  ,
                       i_tx_start                 =>i_tx_start  ,
                       o_tx_end                   =>o_tx_end  ,
                       i_data_parallel            =>i_data_parallel  ,
                       o_data_parallel            =>o_data_parallel  ,
                       o_sclk                     =>o_sclk  ,
                       o_ss                       =>o_ss  ,
                       o_mosi                     =>o_mosi  ,
                       i_miso                     =>i_miso  );


                    ------------------------------------------------------------
              ---------
                    -- FSM
                    p_control : process(i_clk,i_rstb)
                    variable v_control  : unsigned(12 downto 0);
                    begin
                       if(i_rstb='0') then
                             v_control  := (others=>'0');
                             i_tx_start<= '0';
                             i_data_parallel<=
                                 std_logic_vector(to_unsigned(16#92#,N));
                       elsif(rising_edge(i_clk)) then
                             v_control  := v_control + 1;
                             if(v_control=10) then
                                   i_tx_start<= '1';
                             else
                                   i_tx_start<= '0';
                             end if;

                             if(o_tx_end='1') then
                                   i_data_parallel<=
                                 std_logic_vector(unsigned(i_data_parallel)+1);
                             end if;
                       end if;
                    end process p_control;

                    p_control_sclk : process(o_sclk)
                    begin
                       if(i_rstb='0') then
                                        mosi_test<=    mosi_test(N-2    downto
              0)&o_mosi;
```

```
                                        count_rise<= count_rise+1;
                                    else
                                        count_rise<= 0;
                                    end if;
                                end if;

                                if(falling_edge(o_sclk)) then
                                    if(o_ss='0') then
                                        miso_test<=
                    std_logic_vector(rotate_right(unsigned(miso_test),1));
                                        i_miso<= miso_test(N-1) after 63 ns;
                                        count_fall<= count_fall+1;
                                    else
                                        count_fall<= 0;
                                    end if;
                                end if;
                            end if;
                        end process p_control_sclk;

                        end rtl;
```

Рисунок 8.8 – Временная диаграмма работы SPIна языке VHDL

9 СИНТЕЗАТОРЫ ЧАСТОТЫ

9.1 Sin PolinomVerilog

Синтезатор частот — устройство для генерации электрических гармонических колебаний с помощью линейных повторений (умножением, суммированием, разностью) на основе одного или нескольких опорных генераторов. Синтезаторы частот служат источниками стабильных (по частоте) колебаний в радиоприёмниках, радиопередатчиках, частотомерах, испытательных генераторах сигналов и других устройствах, в которых требуется настройка на разные частоты в широком диапазоне и высокая стабильность выбранной частоты. Стабильность обычно достигается применением фазовой автоподстройки частоты или прямого цифрового синтеза (DDS) с использованием опорного генератора с кварцевой стабилизацией. Синтез частот обеспечивает намного более

высокую точность и стабильность, чем традиционные электронные генераторы с перестройкой изменением индуктивности или ёмкости, очень широкий диапазон перестройки без каких-либо коммутаций и практически мгновенное переключение на любую заданную частоту. Реализовать данное устройство на языке Verilog.

В таблице 9.1.1 показан листинг реализации SinPolinomна языке Verilog

Таблица 9.1.1– Листинг реализации Sin Polinom на языке Verilog

```
`timescale 1us / 1ns

moduletestbench();

//assume basic clock is 10Mhz
regclk;
initialclk=0;
always
        #0.05 clk = ~clk;

//make reset signal at begin of simulation
reg reset;
initial
begin
        reset = 1;
        #0.1;
        reset = 0;
end

//function calculating sinus
function real sin;
input x;
real x;
real x1,y,y2,y3,y5,y7,sum,sign;
  begin
        sign = 1.0;
        x1 = x;
        if (x1<0)
        begin
              x1 = -x1;
              sign = -1.0;
        end
        while (x1 > 3.14159265/2.0)
        begin
              x1 = x1 - 3.14159265;
              sign = -1.0*sign;
        end
        y = x1*2/3.14159265;
        y2 = y*y;
        y3 = y*y2;
        y5 = y3*y2;
        y7 = y5*y2;
        sum = 1.570794*y - 0.645962*y3 +
                                        0.079692*y5                -
0.004681712*y7;
        sin = sign*sum;
```

```
   end
endfunction

//generate requested "freq" digital
integerfreq;
reg [31:0]cnt;
regcnt_edge;
always @(posedgeclk or posedge reset)
begin
        if(reset)
              begin
              cnt<=0;
              cnt_edge<= 1'b0;
        end
endfunction

//generate requested "freq" digital
integerfreq;
regcnt_edge;
always @(posedgeclk or posedge reset)
begin
        if(reset)
        begin
              cnt<=0;
              cnt_edge<= 1'b0;
        end
        else
        if(cnt>=(10000000/(freq*64)-1) )
        begin
              cnt<=0;
              cnt_edge<= 1'b0;
        end
        else
        if(cnt>=(10000000/(freq*64)-1) )
        begin
              cnt<=0;
              cnt_edge<= 1'b1;
        end
        else
        begin
              cnt<=cnt+1;
              cnt_edge<= 1'b0;
        sin = sign*sum;

        end
end

realmy_time;
realsin_real;
reg signed [15:0]sin_val;

//generate requested "freq" sinus
always @(posedgecnt_edge)
begin
   sin_real<= sin(my_time);
```

```
                sin_val<= sin_real*32000;
                        my_time<= my_time+3.14159265*2/64;
                end

                initial
                begin
                        $dumpfile("out.vcd");
                        $dumpvars(0,testbench);
                        my_time=0;

                        freq=500;
                              #10000;
                        freq=1000;
                              #10000;
                        freq=1500;
                              #10000;

                        $finish;
                end
```

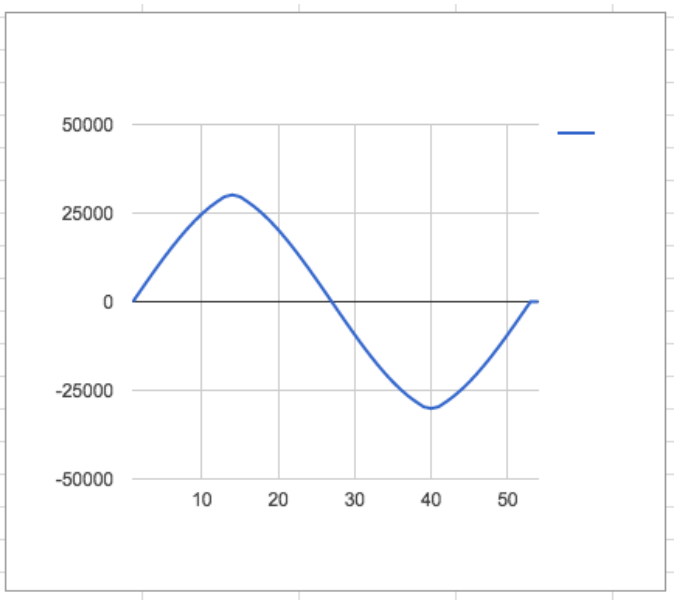


Рисунок 9.1 – Временная диаграмма работы SinPolinomна языке Verilog

9.2 Sin LUT 4 bit

Синтезатор частот — устройство для генерации электрических гармонических колебаний с помощью линейных повторений (умножением, суммированием, разностью) на основе одного или нескольких опорных генераторов. Синтезаторы частот служат источниками стабильных (по частоте) колебаний в радиоприёмниках, радиопередатчиках, частотомерах, испытательных генераторах сигналов и других устройствах, в которых требуется настройка на разные частоты в широком диапазоне и высокая стабильность выбранной частоты. Стабильность обычно достигается применением фазовой автоподстройки частоты или прямого цифрового синтеза (DDS) с использованием опорного генератора с кварцевой стабилизацией. Синтез частот обеспечивает намного более

высокую точность и стабильность, чем традиционные электронные генераторы с перестройкой изменением индуктивности или ёмкости, очень широкий диапазон перестройки без каких-либо коммутаций и практически мгновенное переключение на любую заданную частоту. Реализовать данное устройство на языке Verilog.

В таблице 9.2.1 показан листинг реализации Sin LUT 4 bitна языке Verilog

Таблица 9.2.1– Листинг реализации Sin LUT 4 bit на языке Verilog

```
`timescale 1ns / 1ps
module SinLUT (clk,sin,cnt);
input clk;
output reg [3:0] sin; // по амплитуде sin
outputreg [3:0] cnt; // счетчик от 0 до 15, период sin по времени
initial begin sin=0;cnt=0; end
always @(posedge clk)
begin
cnt=cnt+1; // икремент по времени
case (cnt) // период функции sin
4'd0 :sin = 7; // середина амплитуды sin
4'd1 : sin = 10;
4'd2 : sin = 13;
4'd3 : sin = 15; // max(sin)
4'd4 : sin = 13;
4'd5 : sin = 10;
4'd6 : sin = 7;
4'd7 :sin = 7; // середина амплитуды sin
4'd8 : sin = 7;
4'd9 : sin = 5;
4'd10 : sin = 3;
4'd11 : sin = 0; // min(sin)
4'd12 : sin = 0;
4'd13 : sin = 3;
4'd14 : sin = 5;
4'd15 :sin = 7; // середина амплитуды sin
endcase
end
endmodule
```

В таблице 9.2.2 показан листинг проверкиSin LUT 4 bitна языке Verilog

Таблица 9.2.2– Листинг проверкиSin LUT 4 bit на языке Verilog

```
`timescale 1ns / 1ps
//
module sinLUT_tb;

// Inputs
reg clk;

// Outputs
wire [3:0] sin;
wire [3:0] cnt;

```

```
                    // Instantiate the Unit Under Test (UUT)
                    SinLUT uut (
                    .clk(clk),
                    .sin(sin),
                    .cnt(cnt)
                    );

                    initial begin
                    // Initialize Inputs
                    clk = 0;

                    // Wait 100 ns for global reset to finish
                    #100;

                    // Add stimulus here

                    end
                    always begin #5 clk = ~clk; end
          endmodule
```

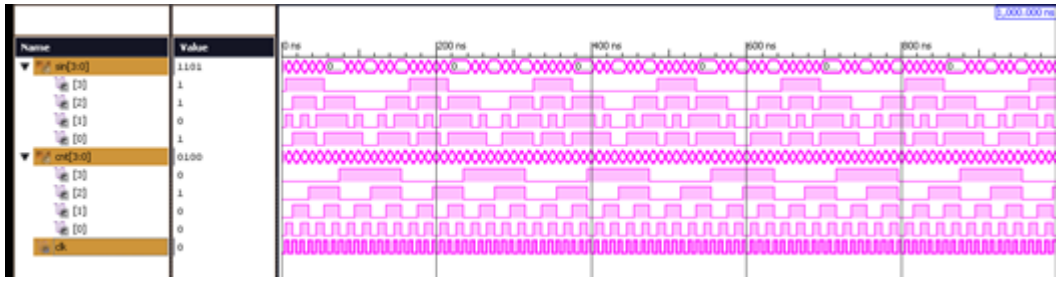


Рисунок 9.2 – Временная диаграмма работы SinLUT 4 bit

9.3 Sin LUT 16 bit

Синтезатор частот — устройство для генерации электрических гармонических колебаний с помощью линейных повторений (умножением, суммированием, разностью) на основе одного или нескольких опорных генераторов. Синтезаторы частот служат источниками стабильных (по частоте) колебаний в радиоприёмниках, радиопередатчиках, частотомерах, испытательных генераторах сигналов и других устройствах, в которых требуется настройка на разные частоты в широком диапазоне и высокая стабильность выбранной частоты. Стабильность обычно достигается применением фазовой автоподстройки частоты или прямого цифрового синтеза (DDS) с использованием опорного генератора с кварцевой стабилизацией. Синтез частот обеспечивает намного более высокую точность и стабильность, чем традиционные электронные генераторы с перестройкой изменением индуктивности или ёмкости, очень широкий диапазон перестройки без каких-либо коммутаций и практически мгновенное переключение на любую заданную частоту. Реализовать данное устройство на языке Verilog.

В таблице 9.3.3 показан листинг реализации Sin LUT 16 bitна языке Verilog

Таблица 9.3.3– Листинг реализации Sin LUT 16 bit на языке Verilog

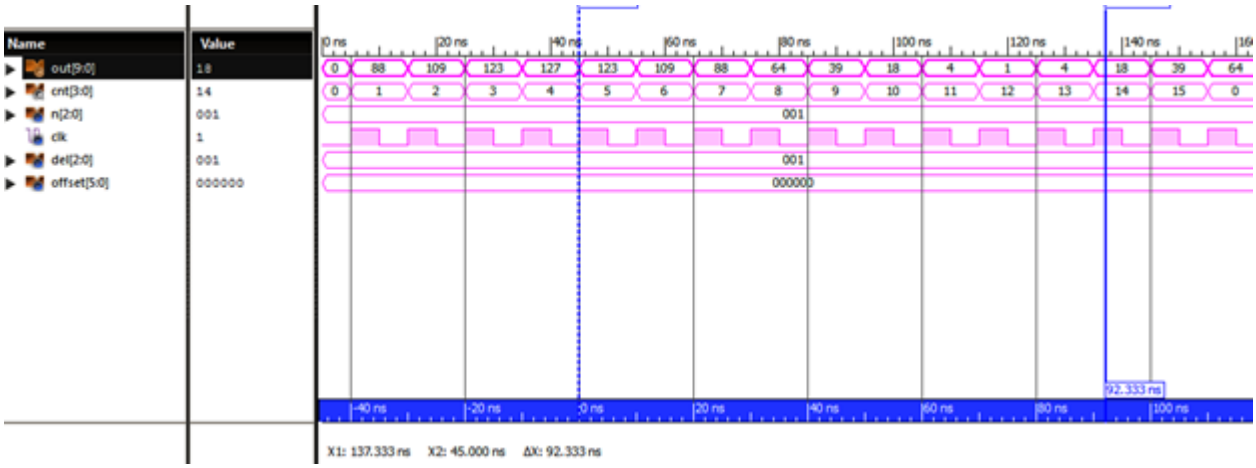
```
`timescale 1ns / 1ps
module sin(n,clk,del,offset,out,cnt);
input [2:0] n; //множетель амплитуды
input clk;
input [2:0] del;//делитель частоты
input [5:0] offset; //задание смещения пи/2=8 пи=16 1.5пи=24
outputreg [9:0] out; // выходной регистр
outputreg [3:0] cnt; // счетчик от 0 до 32, период сигнала
reg [1:0] flag;
reg [2:0] bufer;
initial begin out=0; cnt=0; bufer=0; flag=0; end
always @(posedge clk)
begin
if (flag==0)
   begin
   flag=1;
   cnt=offset;
   end
bufer=bufer+1;
if (bufer==del)
   begin
   cnt=cnt+1; // икремент по времени
   bufer=0;
   end
//max частота 3,125 МГц
case (cnt) // период функции sin
   4'd0 :out = n*64; // середина амплитуды sin
   4'd1 : out = n*88;
   4'd2 : out = n*109;
   4'd3 : out = n*123;
   4'd4 : out = n*127;
   4'd5 : out = n*123;
   4'd6 : out = n*109;
   4'd7 : out = n*88;
   4'd8 : out = n*64;
   4'd9 : out = n*39;
   4'd10 : out = n*18;
   4'd11 : out = n*4;
   5'd12 : out = n*1;
   5'd13 : out = n*4;
   5'd14 : out = n*18;
   5'd15 : out = n*39;

endcase
end
endmodule
```

В таблице 9.3.4 показан листинг проверкиSin LUT 16 bitна языке Verilog

Таблица 9.3.4– Листинг проверкиSin LUT 16 bit на языке Verilog

```
`timescale 1ns / 1ps
module test;

```

```
                    // Inputs
                    reg [2:0] n;
                    reg clk;
                    reg [2:0] del;
                    reg [5:0] offset;

                    // Outputs
                    wire [9:0] out;
                    wire [3:0] cnt;

                    // Instantiate the Unit Under Test (UUT)
                    sin uut (
                          .n(n),
                          .clk(clk),
                          .del(del),
                          .offset(offset),
                          .out(out),
                          .cnt(cnt)
                    );

                    initial begin
                          // Initialize Inputs
                          n = 1;
                          clk = 0;
                          del = 1;
                          offset = 0;

                          // Wait 100 ns for global reset to finish
                          #100;

                          // Add stimulus here

                    end
                    always begin #5 clk = ~clk; end
              endmodule
```



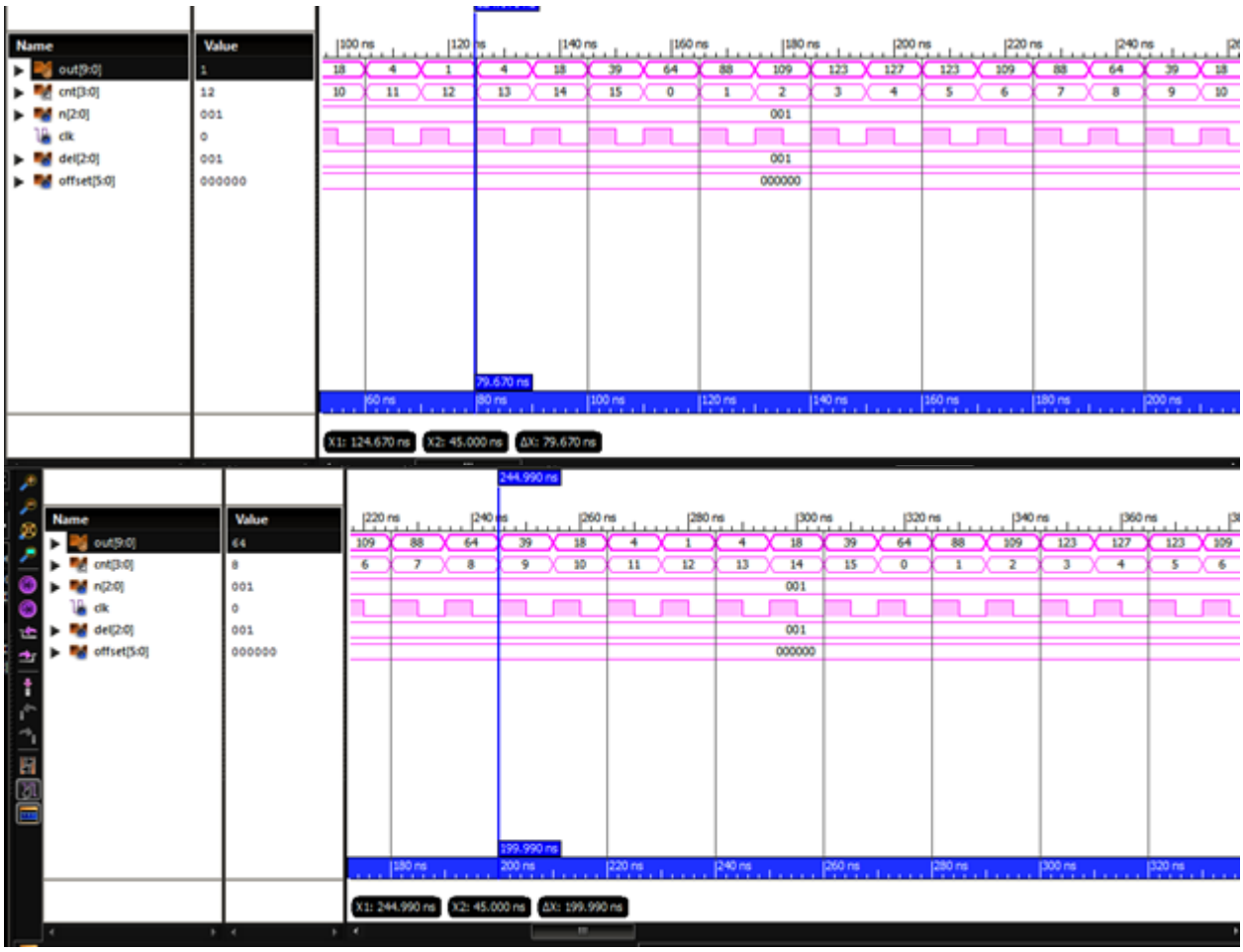


Рисунок 9.3 – Временные диаграммы работы SinLUT 16 bit

9.4 Sin LUT 24 bit

Синтезатор частот — устройство для генерации электрических гармонических колебаний с помощью линейных повторений (умножением, суммированием, разностью) на основе одного или нескольких опорных генераторов. Синтезаторы частот служат источниками стабильных (по частоте) колебаний в радиоприёмниках, радиопередатчиках, частотомерах, испытательных генераторах сигналов и других устройствах, в которых требуется настройка на разные частоты в широком диапазоне и высокая стабильность выбранной частоты. Стабильность обычно достигается применением фазовой автоподстройки частоты или прямого цифрового синтеза (DDS) с использованием опорного генератора с кварцевой стабилизацией. Синтез частот обеспечивает намного более высокую точность и стабильность, чем традиционные электронные генераторы с перестройкой изменением индуктивности или ёмкости, очень широкий диапазон перестройки без каких-либо коммутаций и практически мгновенное переключение на любую заданную частоту. Реализовать данное устройство на языке Verilog.

В таблице 9.4.1 показан листинг реализации SinLUT24 bit на языке Verilog

Таблица 9.4.1– Листинг реализации SinLUT24 bit на языке Verilog

```
`timescale 1ns / 1ps
module sin(n,clk,del,offset,out,cnt);
```

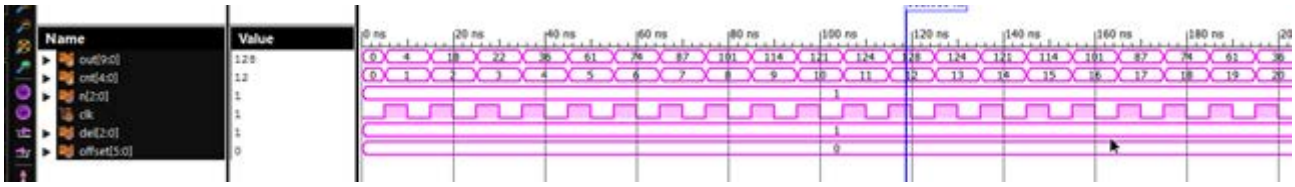
```
input [2:0] n; //множетель амплитуды
input clk;
input [2:0] del;//делитель частоты
input [5:0] offset; //задание смещения пи/2=8 пи=16 1.5пи=24
outputreg [9:0] out; // выходной регистр
outputreg [4:0] cnt; // счетчик от 0 до 32, период сигнала
reg [1:0] flag;
reg [2:0] bufer;
initial begin out=0; cnt=0; bufer=0; flag=0; end
always @(posedge clk)
begin
if (flag==0)
   begin
   flag=1;
   cnt=offset;
   end
bufer=bufer+1;
if (bufer==del)
   begin
   cnt=cnt+1; // икремент по времени
   if(cnt > 23)
   begin
        cnt = 0;
   end;

   bufer=0;
   end
   //max частота 3,125 МГц
     case (cnt) // период функции sin
        4'd0 :out = n*0; // середина амплитуды sin
        4'd1 : out = n*4;
        4'd2 : out = n*18;
        4'd3 : out = n*22;
        4'd4 : out = n*36;
        4'd5 : out = n*61;
        4'd6 : out = n*74;
        4'd7 : out = n*87;
        4'd8 : out = n*101;
        4'd9 : out = n*114;
        4'd10 : out = n*121;
        4'd11 : out = n*124;
        5'd12 : out = n*128;
        5'd13 : out = n*124;
        5'd14 : out = n*121;
        5'd15 : out = n*114;
        5'd16 : out = n*101;
        5'd17 : out = n*87;
        5'd18 : out = n*74;
        5'd19 : out = n*61;
        5'd20 : out = n*36;
        5'd21 : out = n*22;
        5'd22 : out = n*18;
      5'd23 : out = n*4;
```

```
                endcase
            end
        endmodule
```

В таблице 9.4.2 показан листинг проверкиSinLUT24 bit на языке Verilog

Таблица 9.4.2– Листинг проверкиSinLUT24 bit на языке Verilog

```
`timescale 1ns / 1ps
module test;

    // Inputs
    reg [2:0] n;
    reg clk;
    reg [2:0] del;
    reg [5:0] offset;

    // Outputs
    wire [9:0] out;
    wire [4:0] cnt;

    // Instantiate the Unit Under Test (UUT)
    sin uut (
        .n(n),
        .clk(clk),
        .del(del),
        .offset(offset),
        .out(out),
        .cnt(cnt)
    );

    initial begin
        // Initialize Inputs
        n = 1;
        clk = 0;
        del = 1;
        offset = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end
    always begin #5 clk = ~clk; end

endmodule
```



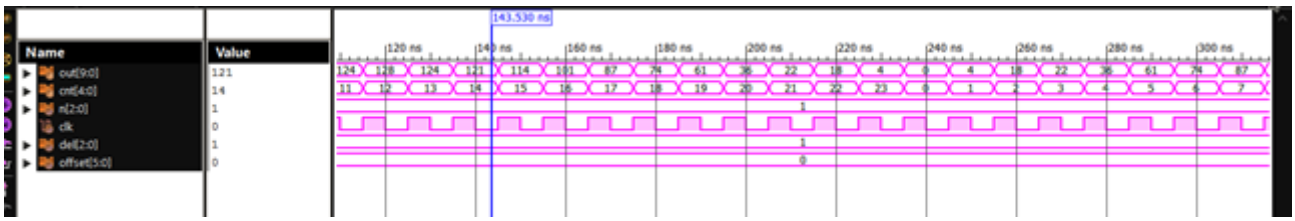


Рисунок 9.4.1 – Временные диаграммы работы SinLUT 24 bit

9.5 Sin LUT 32 bit

Синтезатор частот — устройство для генерации электрических гармонических колебаний с помощью линейных повторений (умножением, суммированием, разностью) на основе одного или нескольких опорных генераторов. Синтезаторы частот служат источниками стабильных (по частоте) колебаний в радиоприёмниках, радиопередатчиках, частотомерах, испытательных генераторах сигналов и других устройствах, в которых требуется настройка на разные частоты в широком диапазоне и высокая стабильность выбранной частоты. Стабильность обычно достигается применением фазовой автоподстройки частоты или прямого цифрового синтеза (DDS) с использованием опорного генератора с кварцевой стабилизацией. Синтез частот обеспечивает намного более высокую точность и стабильность, чем традиционные электронные генераторы с перестройкой изменением индуктивности или ёмкости, очень широкий диапазон перестройки без каких-либо коммутаций и практически мгновенное переключение на любую заданную частоту. Реализовать данное устройство на языке Verilog.

В таблице 9.5.1 показан листинг реализации SinLUT32bit на языке Verilog

Таблица 9.5.1– Листинг реализации SinLUT32bit на языке Verilog

```
module Sin (n,clk,del,offset,out,cnt);
        input [2:0] n; //множетель амплитуды
        input clk;
        input [2:0] del;//делитель частоты
input [5:0] offset; //задание смещения пи/2=8 пи=16 1.5пи=24
outputreg [9:0] out; // выходной регистр
outputreg [4:0] cnt; // счетчик от 0 до 32, период сигнала
reg [1:0] flag;
        reg [2:0] bufer;
        initial begin out=0; cnt=0; bufer=0; flag=0; end
        always @(posedge clk)
        begin
        if (flag==0)
    begin
flag=1;
cnt=offset;
end
        bufer=bufer+1;
        if (bufer==del)
begin
cnt=cnt+1; // икремент по времени
bufer=0;
```

```
                end
                        //max частота 3,125 МГц
                case (cnt) // период функции sin
                        5'd0 :out = n*64; // середина амплитуды sin
                5'd1 : out = n*76;
                        5'd2 : out = n*88;
                        5'd3 : out = n*99;
                        5'd4 : out = n*109;
                        5'd5 : out = n*117;
                        5'd6 : out = n*123;
                        5'd7 : out = n*126;
                        5'd8 : out = n*127;
                        5'd9 : out = n*126;
                        5'd10 : out = n*123;
                        5'd11 : out = n*117;
                        5'd12 : out = n*109;
                        5'd13 : out = n*99;
                        5'd14 : out = n*88;
                        5'd15 : out = n*76;
                        5'd16 : out = n*64;
                        5'd17 : out = n*51;
                        5'd18 : out = n*39;
                        5'd19 : out = n*28;
                        5'd20 : out = n*18;
                        5'd21 : out = n*10;
                        5'd22 : out = n*4;
                        5'd23 : out = n*2;
                        5'd24 : out = n*1;
                        5'd25 : out = n*2;
                        5'd26 : out = n*4;
                        5'd27 : out = n*10;
                        5'd28 : out = n*18;
                        5'd29 : out = n*28;
                        5'd30 : out = n*39;
                        5'd31 : out = n*51;
                        endcase
                        end
                        endmodule
```

В таблице 9.5.2 показан листинг проверкиSinLUT32bit на языке Verilog

Таблица 9.5.2– Листинг проверкиSinLUT32bit на языке Verilog

```
            module Sin_tb;
                // Inputs
                reg [2:0] n
                reg clk;
                reg [2:0] del;
                reg [5:0] offset;
                // Outputs
                wire [9:0] out;
                wire [4:0] cnt;
                // Instantiate the Unit Under Test (UUT)
                Sin uut (
                      .n(n),
```

```
                .clk(clk),
                .del(del),
                .offset(offset),
                .out(out),
                .cnt(cnt)
            );
            initial begin
                // Initialize Inputs
                n = 1;
                clk = 0;
                del = 1;
                offset = 0;

                // Wait 100 ns for global reset to finish
                #100;
                // Add stimulus here
            end
             always begin #5 clk = ~clk; end
        endmodule
```

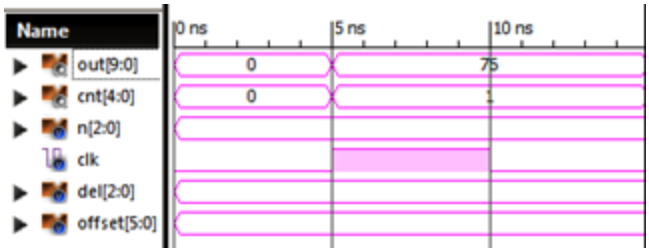


Рисунок 9.5.1 – Временные диаграммы работы SinLUT 32 bit

Первый период синусоиды начинается с нуля, так что будет рассматривать второй период.

Рисунок 9.5.2 – Временные диаграммы работы SinLUT 32 bitc вторым периодом

Выходной сигнал с параметрами n=1, del=1,offset=0;

Такт											0	1	2	3	4	5
Значение	4	6	8	9	09	17	23	26	27	26	23	17	09	9	8	6
Такт	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Значение	4	1	9	8	8	0						0	8	8	9	1

Таблица 9.1 Аппроксимация выходного сигнала модуля SinLUT 32 bit

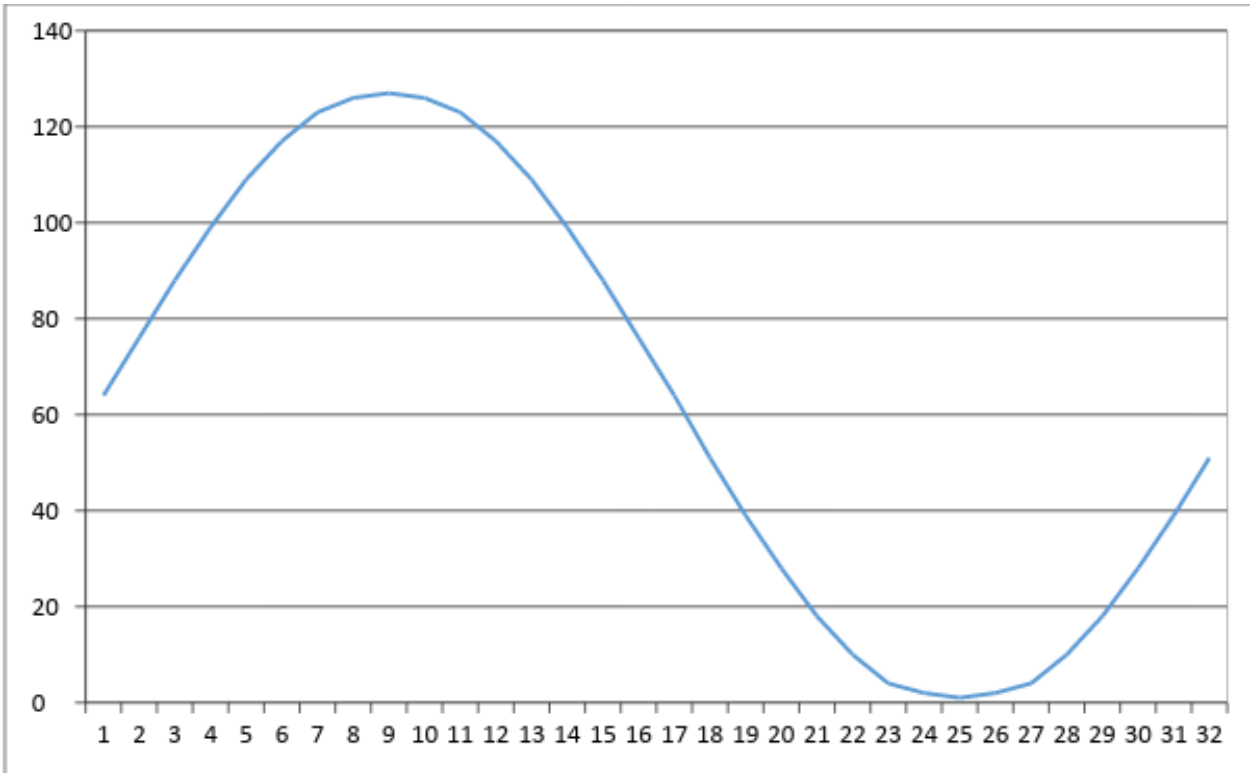


Рисунок 9.5.3– График изменения амплитуды SinLUT 32 bit

Рисунок 9.5.3 – Временные диаграммы работы SinLUT 32 bit с параметрами выходного сигнала n=2, del=1,offset=0

Изменим частоты изменив параметры входного сигнала:

Рисунок 9.5.4 – Временные диаграммы работы SinLUT 32 bit с параметрами выходного сигнала n=1, del=2,offset=0

Изменим смещения изменив параметры входного сигнала:

Рисунок 9.5.5 – Временные диаграммы работы SinLUT 32 bit с параметрами выходного сигнала n=1, del=1,offset=8

10 КАДРОВЫЙ СЕЛЕКТОР

10.1 CS1

Система кадровой синхронизации состоит из схемы временного стробирования и формирователя импульсо запуска задающего генератора кадровой развёртки (ФИ).

Схема временного стробирования предназначена для повышения помехоустойчивости устройства кадровой синхронизации. Она пропускает на свой вход только синхроимпульсы полей, и препятствует прохождению через неё импульсных помех, которые могут появиться на её входе. Схема может работать в двух режимах: в режиме поиска синхроимпульса полей и в режиме слежения за временным положением этого импульса.

Реализовать данное устройство.

В таблице 10.1.1 показан листинг реализации CS1на языке Verilog

Таблица 10.1.1– Листинг реализации CS1 на языке Verilog

```
module sc1 (Din, CLK, cLS, Start);

input wire CLK;
input wire Din;

reg [127:0] S;

initial
   begin
         cLS = 0;
   end

always @(posedge CLK) begin S<={S[30:0],Din}; end


output reg Start;

output reg [5:0] cLS;

initial
begin
   Start = 0;

end

always @(negedge CLK)
   begin
   if (S == 32'b11111111000000001111111110000000) begin
            Start<=1; end
   if (S == 32'b11111111111111110000000000000000 & Start<=1)
begin
            cLS<=cLS+1; end

```

```
end

endmodule
```

В таблице 10.1.2 показан листинг проверкиCS1на языке Verilog

Таблица 10.1.2– Листинг проверкиCS1 на языке Verilog

```
module SC1_TB;

    // Inputs
    reg Din;
    reg CLK;

    // Outputs
    wire [5:0] cLS;
    wire Start;

    // Instantiate the Unit Under Test (UUT)
    sc1 uut (
        .Din(Din),
        .CLK(CLK),
        .cLS(cLS),
        .Start(Start)
    );

    initial begin
        // Initialize Inputs
        Din = 0;
        CLK = 0;

        // Wait 100 ns for global reset to finish
        #100;
        // Add stimulus here

    end

always begin #5 CLK = ~CLK; end
always begin #80 Din = ~Din; end

endmodule
```

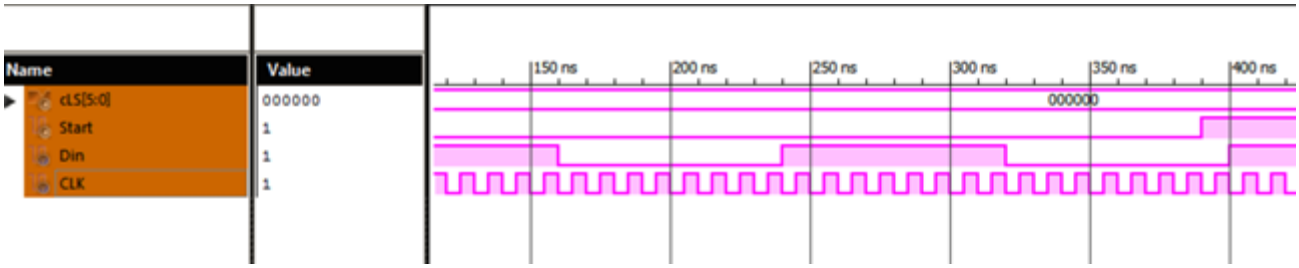


Рисунок 10.1.1 – Временные диаграммы работы CS1

10.2 CS2

Система кадровой синхронизации состоит из схемы временного стробирования и формирователя импульсо запуска задающего генератора кадровой развёртки (ФИ).

Схема временного стробирования предназначена для повышения помехоустойчивости устройства кадровой синхронизации. Она пропускает на свой вход только синхроимпульсы полей, и препятствует прохождению через неё импульсных помех, которые могут появиться на её входе. Схема может работать в двух режимах: в режиме поиска синхроимпульса полей и в режиме слежения за временным положением этого импульса.

Реализовать данное устройство.

В таблице 10.2.1 показан листинг реализации CS2 на языке Verilog

Таблица 10.2.1– Листинг реализации CS2 на языке Verilog

```
`timescale 1ns / 1ps
module sc1 (Din, CLK, cLS, Start);

input wire CLK;
input wire Din;
//input wire CLKd;

//output wire CS;
//output wire LS;

reg [127:0] S;

initial
   begin
         cLS = 0;
   end

always @(posedge CLK) begin S<={S[30:0],Din}; end
//always  @(posedge LS) begin cLS<=cLS + 1; end

output reg Start;

output reg [5:0] cLS;

initial
begin
   Start = 0;
   //LS = 0;
end

always @(negedge CLK)
   begin
   if (S == 32'b11111111000000001111111110000000) begin
            Start<=1; end
if (S == 32'b11111111111111110000000000000000 & Start<=1) begin
            cLS<=cLS+1; end
//
//wire CS;

end
```

```
endmodule
```

В таблице 10.2.2 показан листинг проверкиCS2 на языке Verilog

Таблица 10.2.2– Листинг проверкиCS2 на языке Verilog

```
module SC1_TB;

// Inputs
reg Din;
reg CLK;

// Outputs
wire [5:0] cLS;

// Instantiate the Unit Under Test (UUT)
sc1 uut (
	.Din(Din),
	.CLK(CLK),
	.cLS(cLS),
	.Start(Start)
);

initial begin
	// Initialize Inputs
	Din = 0;
	CLK = 0;

	// Wait 100 ns for global reset to finish
	#100;

	// Add stimulus here

end

always begin #5 CLK = ~CLK; end
always begin #80 Din = ~Din; end

endmodule
```

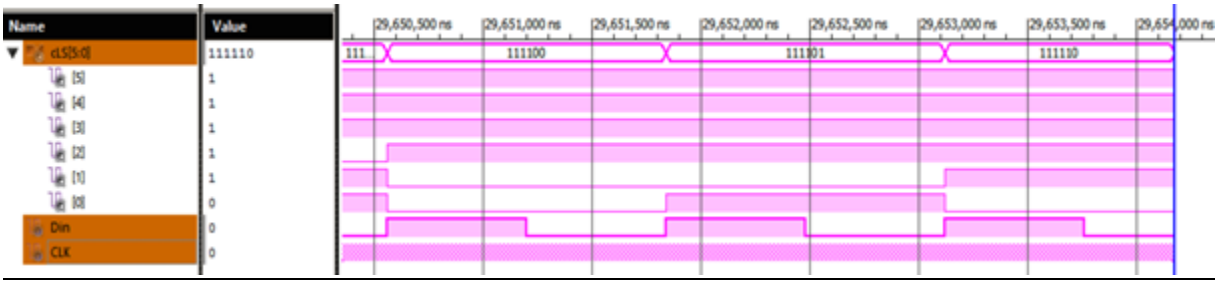


Рисунок 10.2.1 – Временные диаграммы работы CS2

10.3 CS3

Система кадровой синхронизации состоит из схемы временного стробирования и формирователя импульсо запуска задающего генератора кадровой развёртки (ФИ).

Схема временного стробирования предназначена для повышения помехоустойчивости устройства кадровой синхронизации. Она пропускает на свой вход только синхроимпульсы полей, и препятствует прохождению через неё импульсных помех, которые могут появиться на её входе. Схема может работать в двух режимах: в режиме поиска синхроимпульса полей и в режиме слежения за временным положением этого импульса.

Реализовать данное устройство.

В таблице 10.3.1 показан листинг реализации CS3 на языке Verilog

Таблица 10.3.1– Листинг реализации CS3 на языке Verilog

```
module sc3 (Din, CLK, CS, LS, cLS);

                inputwire CLK;
                inputwireDin;
                //inputwireCLKd;

                outputwire CS;
                outputwire LS;

                reg [127:0] S;

                initial
                 begin
                cLS = 0;
                end

                always @(posedge CLK) begin S<={S[126:0],Din};
        end
                always @(posedge LS) begincLS<=cLS + 1; end

                //regStart;

                outputreg [5:0] cLS;

                //initial
                //begin
                //Start = 0;
                //LS = 0;
                //end

                //always @(negedge CLK)
                // begin
                // if (S ==
        128'b11111111111111111111111111111100000000000000000000000
        00000000011111111111111111111111111111000000000000000000
```

```
                                000000000000000) beginStart<=1; end
                                // if (S ==
        128'b11111111111111111111111111111111111111111111111111111111
        11111111110000000000000000000000000000000000000000000000000000
                                000000000000000   &Start<=1)   begincLS<=cLS+1;
        end
                                //
                                //wire CS;
                                //assign CS
        =(~S[0]|~S[1])&~S[2]&~S[3]&~S[4]&~S[5]&~S[6]&~S[7]&~S[8]&~S
        [9]&~S[10]&~S[11]&~S[12]&~S[13]&~S[14]&~S[15]&~S[16]&~S[17]
              &~S[18]&~S[19]&~S[20]&~S[21]&~S[22]&~S[23]&~S[24]&~S[25]&~S
        [26]&~S[27]&~S[28]&~S[29]&(~S[30]|~S[31])&(S[32]|S[33])&S[3
        4]&S[35]&S[36]&S[37]&S[38]&S[39]&S[40]&S[41]&S[42]&S[43]&S[
        44]&S[45]&S[46]&S[47]&S[48]&S[49]&S[50]&S[51]&S[52]&S[53]&S
        [54]&S[55]&S[56]&S[57]&S[58]&S[59]&S[60]&S[61]&(S[62]|S[63]
        )&(~S[64]|~S[65])&~S[66]&~S[67]&~S[68]&~S[69]&~S[70]&~S[71]
              &~S[72]&~S[73]&~S[74]&~S[75]&~S[76]&~S[77]&~S[78]&~S[79]&~S
        [80]&~S[81]&~S[82]&~S[83]&~S[84]&~S[85]&~S[86]&~S[87]&~S[88
        ]&~S[89]&~S[90]&~S[91]&~S[92]&~S[93]&(~S[94]|~S[95])&(S[96]
        |S[97])&S[98]&S[99]&S[100]&S[101]&S[102]&S[103]&S[104]&S[10
        5]&S[106]&S[107]&S[108]&S[109]&S[110]&S[111]&S[112]&S[113]&
        S[114]&S[115]&S[116]&S[117]&S[118]&S[119]&S[120]&S[121]&S[1
                                22]&S[123]&S[124]&S[125]&S[126]&S[127];
                                //wire LS;
                                assign CS
        =(~S[0]|~S[1]|~S[2]|~S[3])&(~S[4]|~S[5]|~S[6]|~S[7])&(~S[8]
        |~S[17]|~S[18]|~S[19])&(~S[20]|~S[21]|~S[22]|~S[23])&(~S[24
        ]|~S[25]|~S[26]|~S[27])&(~S[28]|~S[29]|~S[30]|~S[31])&(S[32
        ]|S[33]|S[34]|S[35])&(S[36]|S[37]|S[38]|S[39])&(S[40]|S[41]
        |S[42]|S[43])&(S[44]|S[45]|S[46]|S[47])&(S[48]|S[49]|S[50]|
        S[51])&(S[52]|S[53]|S[54]|S[55])&(S[56]|S[57]|S[58]|S[59])&
        (S[60]|S[61]|S[62]|S[63])&(~S[64]|~S[65]|~S[66]|~S[67])&(~S
        [68]|~S[69]|~S[70]|~S[71])&(~S[72]|~S[73]|~S[74]|~S[75])&(~
        S[76]|~S[77]|~S[78]|~S[79])&(~S[80]|~S[81]|~S[82]|~S[83])&(
        ~S[84]|~S[85]|~S[86]|~S[87])&(~S[88]|~S[89]|~S[90]|~S[91])&
        (~S[92]|~S[93]|~S[94]|~S[95])&(S[96]|S[97]|S[98]|S[99])&(S[
```

```
100]|S[101]|S[102]|S[103])&(S[104]|S[105]|S[106]|S[107])&(S
[108]|S[109]|S[110]|S[111])&(S[112]|S[113]|S[114]|S[115])&(
S[116]|S[117]|S[118]|S[119])&(S[120]|S[121]|S[122]|S[123])&
                    (S[124]|S[125]|S[126]|S[127]);
                    assign LS
=(~S[0]|~S[1])&~S[2]&~S[3]&~S[4]&~S[5]&~S[6]&~S[7]&~S[8]&~S
[9]&~S[10]&~S[11]&~S[12]&~S[13]&~S[14]&~S[15]&~S[16]&~S[17]
      &~S[18]&~S[19]&~S[20]&~S[21]&~S[22]&~S[23]&~S[24]&~S[25]&~S
[26]&~S[27]&~S[28]&~S[29]&~S[30]&~S[31]&~S[32]&~S[33]&~S[34
]&~S[35]&~S[36]&~S[37]&~S[38]&~S[39]&~S[40]&~S[41]&~S[42]&~
S[43]&~S[44]&~S[45]&~S[46]&~S[47]&~S[48]&~S[49]&~S[50]&~S[5
1]&~S[52]&~S[53]&~S[54]&~S[55]&~S[56]&~S[57]&~S[58]&~S[59]&
~S[60]&~S[61]&(~S[62]|~S[63])&(S[64]|S[65])&S[66]&S[67]&S[6
8]&S[69]&S[70]&S[71]&S[72]&S[73]&S[74]&S[75]&S[76]&S[77]&S[
78]&S[79]&S[80]&S[81]&S[82]&S[83]&S[84]&S[85]&S[86]&S[87]&S
[88]&S[89]&S[90]&S[91]&S[92]&S[93]&S[94]&S[95]&S[96]&S[97]&
S[98]&S[99]&S[100]&S[101]&S[102]&S[103]&S[104]&S[105]&S[106
]&S[107]&S[108]&S[109]&S[110]&S[111]&S[112]&S[113]&S[114]&S
[115]&S[116]&S[117]&S[118]&S[119]&S[120]&S[121]&S[122]&S[12
                    3]&S[124]&S[125]&S[126]&S[127];

                    endmodule

          module sc3_tb;

                    // Inputs
                    reg Din;
                    reg CLK;

                    // Outputs
                    wire CS;
                    wire LS;
                    wire [5:0] cLS;

                    // Instantiate the Unit Under Test (UUT)
                    sc3 uut (
                    .Din(Din),
                    .CLK(CLK),
                    .CS(CS),
                    .LS(LS)
                    .cLS(cLS)
                    );
```

```
initial begin
// Initialize Inputs
Din = 0;
CLK = 0;

// Wait 100 ns for global reset to finish
#100;

// Add stimulus here

end

always begin #5 CLK = ~CLK; end
always begin #640 Din = ~Din; end

endmodule
```

10.4 CS4

Система кадровой синхронизации состоит из схемы временного стробирования и формирователя импульсо запуска задающего генератора кадровой развёртки (ФИ).

Схема временного стробирования предназначена для повышения помехоустойчивости устройства кадровой синхронизации. Она пропускает на свой вход только синхроимпульсы полей, и препятствует прохождению через неё импульсных помех, которые могут появиться на её входе. Схема может работать в двух режимах: в режиме поиска синхроимпульса полей и в режиме слежения за временным положением этого импульса.

Реализовать данное устройство.

В таблице 10.4.1 показан листинг реализации CS4 на языке Verilog

Таблица 10.4.1– Листинг реализации CS4 на языке Verilog

```
module sc4 (Din, CLK, CS, LS, cLS);

input wire CLK;
input wire Din;

output wire CS;
output wire LS;

reg [127:0] S;

initial
   begin
         cLS = 0;
   end

always @(posedge CLK) begin S<={S[126:0],Din}; end
always  @(posedge LS) begin cLS<=cLS + 1; end
```

```
      outputreg [5:0] cLS;

      assign CS
=(~S[0]|~S[1])&~S[2]&~S[3]&~S[4]&~S[5]&~S[6]&~S[7]&~S[8]&~S
[9]&~S[10]&~S[11]&~S[12]&~S[13]&~S[14]&~S[15]&~S[16]&~S[17]
      &~S[18]&~S[19]&~S[20]&~S[21]&~S[22]&~S[23]&~S[24]&~S[25]&~S
[26]&~S[27]&~S[28]&~S[29]&(~S[30]|~S[31])&(S[32]|S[33])&S[3
4]&S[35]&S[36]&S[37]&S[38]&S[39]&S[40]&S[41]&S[42]&S[43]&S[
44]&S[45]&S[46]&S[47]&S[48]&S[49]&S[50]&S[51]&S[52]&S[53]&S
[54]&S[55]&S[56]&S[57]&S[58]&S[59]&S[60]&S[61]&(S[62]|S[63]
)&(~S[64]|~S[65])&~S[66]&~S[67]&~S[68]&~S[69]&~S[70]&~S[71]
      &~S[72]&~S[73]&~S[74]&~S[75]&~S[76]&~S[77]&~S[78]&~S[79]&~S
[80]&~S[81]&~S[82]&~S[83]&~S[84]&~S[85]&~S[86]&~S[87]&~S[88
]&~S[89]&~S[90]&~S[91]&~S[92]&~S[93]&(~S[94]|~S[95])&(S[96]
|S[97])&S[98]&S[99]&S[100]&S[101]&S[102]&S[103]&S[104]&S[10
5]&S[106]&S[107]&S[108]&S[109]&S[110]&S[111]&S[112]&S[113]&
S[114]&S[115]&S[116]&S[117]&S[118]&S[119]&S[120]&S[121]&S[1
                  22]&S[123]&S[124]&S[125]&S[126]&S[127];

      assign  LS
=(~S[0]|~S[1])&~S[2]&~S[3]&~S[4]&~S[5]&~S[6]&~S[7]&~S[8]&~S
[9]&~S[10]&~S[11]&~S[12]&~S[13]&~S[14]&~S[15]&~S[16]&~S[17]
      &~S[18]&~S[19]&~S[20]&~S[21]&~S[22]&~S[23]&~S[24]&~S[25]&~S
[26]&~S[27]&~S[28]&~S[29]&~S[30]&~S[31]&~S[32]&~S[33]&~S[34
]&~S[35]&~S[36]&~S[37]&~S[38]&~S[39]&~S[40]&~S[41]&~S[42]&~
S[43]&~S[44]&~S[45]&~S[46]&~S[47]&~S[48]&~S[49]&~S[50]&~S[5
1]&~S[52]&~S[53]&~S[54]&~S[55]&~S[56]&~S[57]&~S[58]&~S[59]&
~S[60]&~S[61]&(~S[62]|~S[63])&(S[64]|S[65])&S[66]&S[67]&S[6
8]&S[69]&S[70]&S[71]&S[72]&S[73]&S[74]&S[75]&S[76]&S[77]&S[
78]&S[79]&S[80]&S[81]&S[82]&S[83]&S[84]&S[85]&S[86]&S[87]&S
[88]&S[89]&S[90]&S[91]&S[92]&S[93]&S[94]&S[95]&S[96]&S[97]&
S[98]&S[99]&S[100]&S[101]&S[102]&S[103]&S[104]&S[105]&S[106
]&S[107]&S[108]&S[109]&S[110]&S[111]&S[112]&S[113]&S[114]&S
[115]&S[116]&S[117]&S[118]&S[119]&S[120]&S[121]&S[122]&S[12
                  123]&S[124]&S[125]&S[126]&S[127];
```

```
endmodule
```

В таблице 10.4.2 показан листинг проверкиCS4 на языке Verilog

Таблица 10.4.2– Листинг проверкиCS4 на языке Verilog

```
module sc4_tb;

    // Inputs
    reg Din;
    reg CLK;

    // Outputs
    wire CS;
    wire LS;
    wire [5:0] cLS;

    // Instantiate the Unit Under Test (UUT)
    sc4uut (
          .Din(Din),
          .CLK(CLK),
          .CS(CS),
          .LS(LS),
          .cLS(cLS)
              );

    initial begin
          // Initialize Inputs
          Din = 0;
          CLK = 0;

          // Wait 100 ns for global reset to finish
          #100;

          // Add stimulus here

    end

always begin #5 CLK = ~CLK; end
always begin #640 Din = ~Din; end

endmodule
```

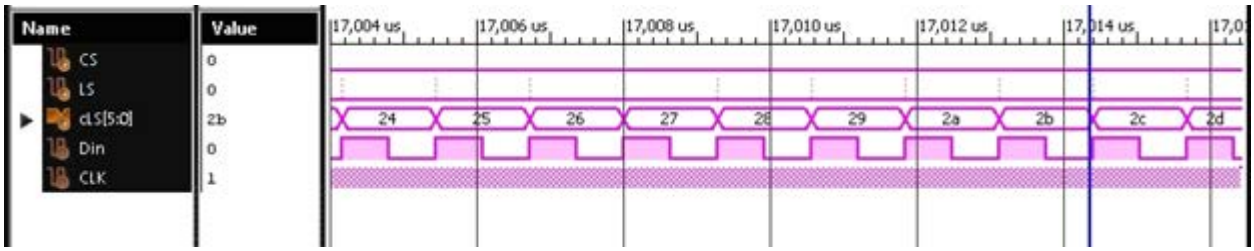


Рисунок 10.4.1 – Временные диаграммы работы CS4

11 ПРАКТИКА JTAG

11.1 Задание

Задание: исследовать электрическую коммутацию программатораJTAG с ПЛИС-отладочной платой фирмы Xilinx семейства Spartan-6 F. Внешний вид программатора представлен на рисунке 11.1 , а отладочной платы – на рисунке 11.2.

Рисунок 11.1.1 – Программатор XUPUSB-JTAG

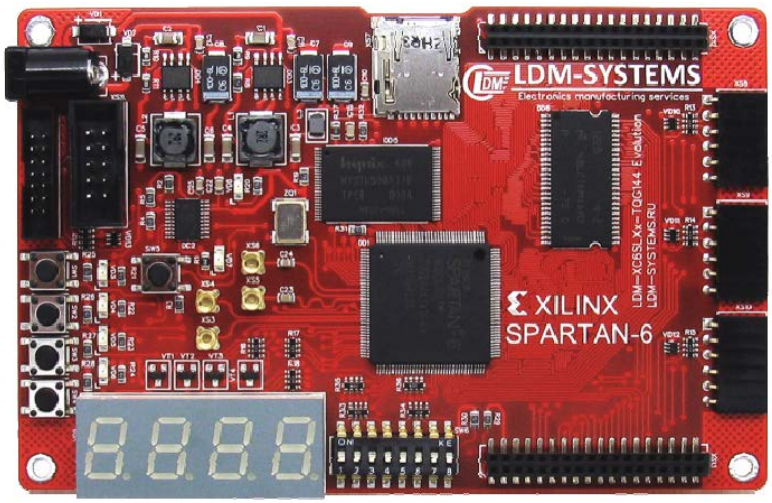


Рисунок 11.1.2 – Отладочная плата XB-XC6SLXx-TQ144 Evolution

На рисунке 11.1.3 представлена диаграмма тестирования электрической коммутации программатора с отладочной платой.

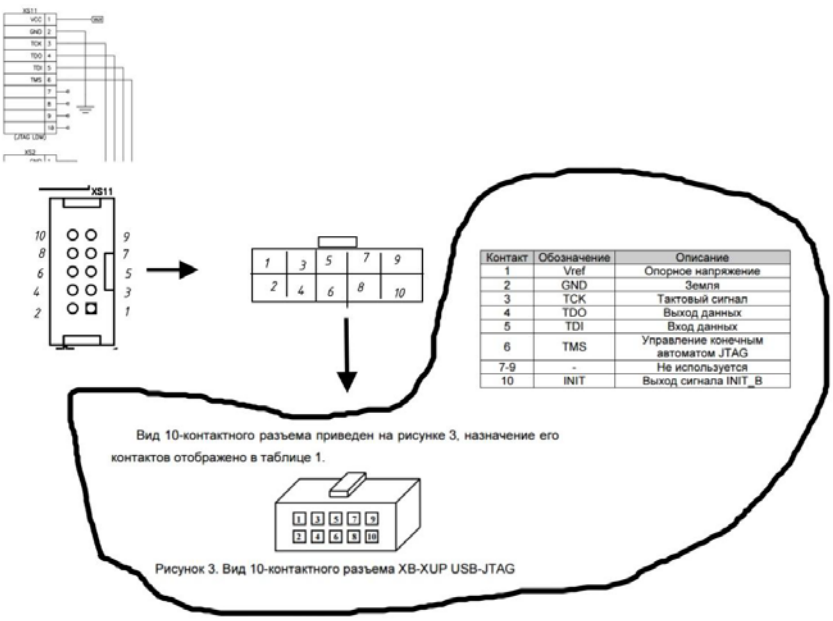


Рисунок 11.1.3 – Диаграмма тестировки

Сначала из документации на отладочную плату (https://ldm-systems.ru/f/doc/catalog/XB-XC6SLXX-TQ144/XB-XC6SLXX-TQ144.pdf) была найдено описание контактов отладочного разъема. Следом – после коммутации с вышеназванным разъемом – был исследован соединительный кабель. А затем – был исследован ответный разъем программатора.