Федеральное государственное автономное образовательное учреждение высшего

образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

**Отчет**

**По лабораторной работе №2**

По дисциплине «Основы схемотехники»

Вариант 6.

Выполнил: Чураков Александр Алексеевич, группа P3331

Преподаватель: Лукашов И. В.

Санкт-Петербург

2025

Оглавление

[Цель работы 3](#_Toc211948879)

[Задание 3](#_Toc211948880)

[Отчет о проделанной работе 4](#_Toc211948881)

[Область допустимых значений 4](#_Toc211948882)

[Умножитель 16х8 4](#_Toc211948883)

[Функция ab+a3 4](#_Toc211948884)

[Описание работы 4](#_Toc211948885)

[Тестовое окружение 6](#_Toc211948886)

[Результат тестирования 8](#_Toc211948887)

[Выводы 9](#_Toc211948888)

# Цель работы

Получить навыки описания арифметических блоков на RTL-уровне с использованием языка описания аппаратуры Verilog HDL.

# Задание

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Содержимое, созданное искусственным интеллектом, может быть неверным.

1. Разработайте и опишите на Verilog HDL схему, вычисляющую значение функции в соответствии с заданными ограничениями согласно варианту задания.

2. Определите область допустимых значений функции.

3. Разработайте тестовое окружение для разработанной схемы. Тестовое окруже ыние должно проверять работу схемы не менее, чем на 10 различных тестовых векторах.

4. Проведите моделирование работы схемы и определите время вычисления результата. Схема должна тактироваться от сигнала с частотой 100 МГц.

5. Составьте отчет по результатам выполнения работы.

# Отчет о проделанной работе

## Область допустимых значений

8-битные входные значения – [0; 255]

24-битное выходное значение [0; 16646400]

## Умножитель 16х8

|  |
| --- |
| module mult16x8 #(  parameter WA = 16, WB = 8  )(  input wire clk,  input wire rst,  input wire start,  input wire [WA-1:0] A,  input wire [WB-1:0] B,  output reg busy,  output reg done,  output reg [WA+WB-1:0] Y  );  localparam WOUT = WA + WB;  reg [WOUT-1:0] a\_ext;  reg [WB-1:0] b\_reg;  reg [WOUT-1:0] acc;  reg [7:0] cnt;  wire [WOUT-1:0] add\_term = b\_reg[0] ? a\_ext : {WOUT{1'b0}};  wire [WOUT-1:0] acc\_next = acc + add\_term;  localparam IDLE=1'b0, WORK=1'b1;  reg state;  always @(posedge clk) begin  if (rst) begin  state<=IDLE; busy<=0; done<=0; Y<=0;  a\_ext<=0; b\_reg<=0; acc<=0; cnt<=0;  end else begin  done <= 1'b0;  case (state)  IDLE: begin  if (start) begin  busy <= 1'b1;  a\_ext <= {{(WOUT-WA){1'b0}}, A};  b\_reg <= B;  acc <= {WOUT{1'b0}};  cnt <= 0;  state <= WORK;  end  end  WORK: begin  acc <= acc\_next;  a\_ext <= a\_ext << 1;  b\_reg <= b\_reg >> 1;  cnt <= cnt + 1;  if (cnt == (WB-1)) begin  Y <= acc\_next;  busy <= 1'b0;  done <= 1'b1;  state<= IDLE;  end  end  endcase  end  end  endmodule |

## Функция ab+a3

### Описание работы

* IDLE: при start=1 захватываем a,b в регистры, busy=1; MUX ставим A16={0,a}, B8=b, даём start\_mul=1 → MUL\_AB.
* MUL\_AB: ждём mul\_done; пишем RegAB←m\_Y и запускаем след.: A16={0,a}, B8=a, start\_mul=1 → MUL\_A2.
* MUL\_A2: по mul\_done сохраняем RegA2←m\_Y[15:0]; запускаем: A16={0,RegA2}, B8=a, start\_mul=1 → MUL\_A3.
* MUL\_A3: по mul\_done фиксируем RegA3←m\_Y → ADD.
* ADD: выполняем Ytmp = RegAB + RegA3, RegY←Ytmp → DONE.
* DONE: выставляем valid=1 (1 такт), busy=0 и возвращаемся в IDLE.

|  |
| --- |
| module func (  input wire clk,  input wire rst,  input wire start,  input wire [7:0] a,  input wire [7:0] b,  output reg busy,  output reg valid,  output reg [23:0] y  );  reg m\_start;  wire m\_busy, m\_done;  reg [15:0] m\_A;  reg [7:0] m\_B;  wire [23:0] m\_Y;  mult16x8 mul (  .clk(clk), .rst(rst),  .start(m\_start),  .A(m\_A), .B(m\_B),  .busy(m\_busy), .done(m\_done),  .Y(m\_Y)  );  reg [7:0] a\_r, b\_r;  reg [23:0] ab24, a3\_24;  reg [15:0] a2\_16;  localparam S\_IDLE=3'd0, S\_AB=3'd1, S\_A2=3'd2, S\_A3=3'd3, S\_ADD=3'd4, S\_DONE=3'd5;  reg [2:0] st;  always @(posedge clk) begin  if (rst) begin  st<=S\_IDLE; busy<=0; valid<=0; y<=0;  m\_start<=0; m\_A<=0; m\_B<=0;  a\_r<=0; b\_r<=0; ab24<=0; a2\_16<=0; a3\_24<=0;  end else begin  m\_start <= 1'b0;  valid <= 1'b0;  case (st)  S\_IDLE: begin  busy <= 1'b0;  if (start) begin  busy <= 1'b1;  a\_r <= a; b\_r <= b;  m\_A <= {8'b0, a};  m\_B <= b;  m\_start<= 1'b1;  st <= S\_AB;  end  end  S\_AB: if (m\_done) begin  ab24 <= m\_Y;  m\_A <= {8'b0, a\_r};  m\_B <= a\_r;  m\_start<= 1'b1;  st <= S\_A2;  end  S\_A2: if (m\_done) begin  a2\_16 <= m\_Y[15:0];  m\_A <= {8'b0, m\_Y[15:0]};  m\_B <= a\_r;  m\_start<= 1'b1;  st <= S\_A3;  end  S\_A3: if (m\_done) begin  a3\_24 <= m\_Y;  st <= S\_ADD;  end  S\_ADD: begin  y <= ab24 + a3\_24;  st <= S\_DONE;  end  S\_DONE: begin  valid <= 1'b1;  busy <= 1'b0;  st <= S\_IDLE;  end  endcase  end  end  endmodule |

## Тестовое окружение

|  |
| --- |
| `timescale 1ns/1ps  module func\_tb;  reg clk = 0;  reg rst = 1;  reg start = 0;  reg [7:0] a = 0, b = 0;  wire busy;  wire valid;  wire [23:0] y;  func dut (  .clk(clk), .rst(rst),  .start(start),  .a(a), .b(b),  .busy(busy), .valid(valid), .y(y)  );  always #5 clk = ~clk;  integer cycle = 0;  always @(posedge clk) cycle <= cycle + 1;  task run\_case(input [7:0] ta, input [7:0] tb, input [23:0] expected, input [255:0] name);  integer c\_start, c\_done, lat;  begin  @(negedge clk);  a <= ta; b <= tb;  start <= 1'b1;  c\_start = cycle;  @(negedge clk);  start <= 1'b0;    @(posedge valid);  c\_done = cycle;  lat = c\_done - c\_start;    if (y !== expected) begin  $display("[FAIL] %0s: a=%0d b=%0d got=%0d exp=%0d", name, ta, tb, y, expected);  $fatal(1);  end  $display("[OK] ? %0s: y=%0d LAT = %0d cycles (~%0d ns)",  name, y, lat, lat\*10);    repeat(2) @(posedge clk);  end  endtask  initial begin  $dumpfile("func\_tb.vcd");  $dumpvars(0, func\_tb);  repeat(5) @(posedge clk);  rst <= 1'b0;  run\_case(8'd3, 8'd4, 24'd39, "SMALL");  run\_case(8'd255, 8'd255, 24'd16646400, "MAX");  run\_case(8'd0, 8'd0, 24'd0, "T0");  run\_case(8'd1, 8'd0, 24'd1, "T1"); // 1\*0 + 1^3 = 1  run\_case(8'd0, 8'd1, 24'd0, "T2"); // 0\*1 + 0^3 = 0  run\_case(8'd7, 8'd5, 24'd378, "T3"); // 7\*5 + 343 = 378  run\_case(8'd10, 8'd10, 24'd1100, "T4"); // 100 + 1000  run\_case(8'd15, 8'd2, 24'd3405, "T5"); // 30 + 3375  run\_case(8'd37, 8'd19, 24'd51356, "T6"); // 703 + 50653  run\_case(8'd128, 8'd1, 24'd2097280, "T7"); // 128 + 2,097,152  run\_case(8'd200, 8'd53, 24'd8010600, "T8"); // 10,600 + 8,000,000  run\_case(8'd255, 8'd0, 24'd16581375, "T9"); // 0 + 255^3  $finish;  end  endmodule |

### Результат тестирования

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как снимок экрана, программное обеспечение, Мультимедийное программное обеспечение, текст

Содержимое, созданное искусственным интеллектом, может быть неверным.

# Выводы

В ходе работы был спроектирован модуль умножителя, один его вход был расширен до 16 разрядов, т. к. для вычисления a3 необходимо сохранить промежуточный результат вычисления квадрата. Этот умножитель был использован для разработки схему вычисления функции a\*b+a3.

По результатам тестирования схема показала корректные вычисления на разных тестовых значениях. Вычисление функции всегда занимает 33 такта (3 умножения по 10 тактов, 1 такт сложение и 2 такта на выход в DONE/valid), так как данный последовательный умножитель делает 8 итераций независимо от данных.