Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Дисциплина: Функциональная схемотехника

Лабораторная работа 2

Вариант 1

Выполнил:

Гурьянов Кирилл Алексеевич

Группа: P33302

Преподаватель:

Табунщик Сергей Михайлович

Санкт-Петербург

2024

Цель работы

Получить навыки описания арифметических блоков на RTL-уровне с использованием языка описания аппаратуры Verilog HDL. Разработать цифровой блок со сложной логикой работы для выполнения заданной вариантом арифметической операции.

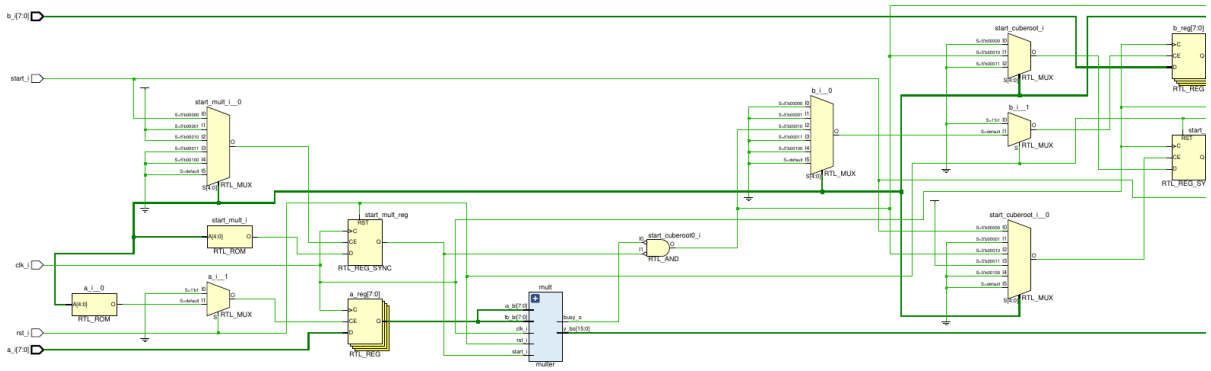
Задание

Функция: $y = a^2 + \sqrt[3]{b}$, ограничения: 1 сумматор и 2 умножителя

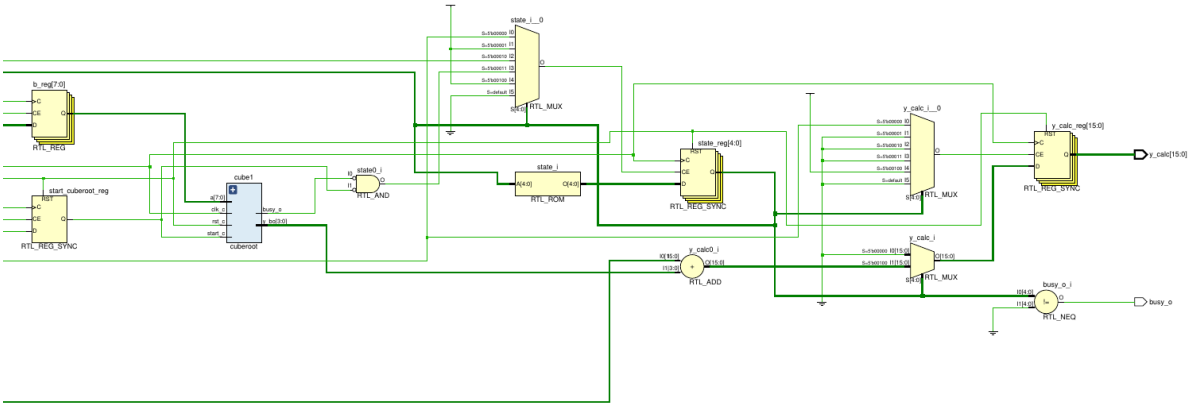
1. Разработайте и опишите на Verilog HDL схему, вычисляющую значение функции в соответствии с заданными ограничениями согласно варианту задания.
2. Определите область допустимых значений функции.
3. Разработайте тестовое окружение для разработанной схемы. Тестовое окружение должно проверять работу схемы 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

Схема разработанного блока вычисления функции



"Рисунок 1. Схема блока вычисления часть 1"



"Рисунок 2. Схема блока вычисления часть 2"

Код разработанного модуля

```
`timescale 1ns / 1ps

module function_calculator(
    input clk_i,
    input rst_i,
    input [7:0] a_i,
    input [7:0] b_i,
    input start_i,
    output busy_o,
    output reg [15:0] y_calc
 );

    localparam IDLE = 0,
    STEP = 1,
    WORK_MULT = 2,
    WORK_CUBEROOT = 3,
    WORK_SUM = 4;

    reg [4:0] state;
    reg [7:0] a;
    reg [7:0] b;
    wire [15:0] square_of_a;
    wire [3:0] cuberoot_result;
    reg [15:0] result;
    reg start_mult;
    reg start_cuberoot;
    wire mult_busy;
    wire cube_busy;
    reg [15:0] mult_result;
```

```
reg [3:0] cube_result;

multer mult(
    .clk_i(clk_i),
    .rst_i(rst_i),
    .a_bi(a),
    .b_bi(a),
    .start_i(start_mult),
    .busy_o(mult_busy),
    .y_bo(square_of_a)
);

cuberoot cube1(
    .clk_c(clk_i),
    .rst_c(rst_i),
    .a(b),
    .start_c(start_cuberoot),
    .busy_o(cube_busy),
    .y_bo(cuberoot_result)
);

assign busy_o = (state != IDLE);

always @(posedge clk_i) begin
    if (rst_i) begin
        y_calc <= 0;
        start_cuberoot <= 0;
        start_mult <= 0;
        state <= IDLE;
    end else begin
        case (state)
            IDLE:
                begin
                    if (start_i) begin
                        y_calc <= 0;
                        start_cuberoot <= 0;
                        state <= STEP;
                        start_mult <= 0;


                    end
                end
            STEP:
                begin
                    start_mult <= 1;
                    a <= a_i;
```

```
                    state <= WORK_MULT;
                end
            WORK_MULT:
                begin
                    start_mult <= 0;
                    if (~mult_busy && ~start_mult) begin
                        state <= WORK_CUBEROOT;
                        b <= b_i;
                        start_cuberoot <= 1;
                    end
                end
            WORK_CUBEROOT:
                begin
                    start_cuberoot <= 0;
                    if (~cube_busy && ~start_cuberoot) begin
                        state <= WORK_SUM;
                    end
                end
            WORK_SUM:
                begin
                    y_calc <= square_of_a + cuberoot_result;
                    state <= IDLE;
                end
        endcase
    end
end

endmodule
```

Описание работы разработанного блока

На вход модуль получается тактовый сигнал clk, по которому происходит синхронизация, сигнал rst, который является сигналом сброса. При активном значении сигнала сброса модуль сбрасывает все значения, а также переходит в состояние ожидание — IDLE. Сигнал start нужен для определения момента, когда блок должен начать вычисления.

Также на вход модулю подаются 2 восьмибитных числа, a и b, по значениям которых вычисляется функция, в соответствии с вариантом.

Выход схемы состоит из 3 сигналов: ready, сигнализирующий о том, что блок готов начать вычисления, busy, сигнализирующий, что блок занят

вычислением и y_calc, по которому блок возвращает результат вычисления.

Блок использует для вычисления значения функции 3 других блока: 1 сумматор, 1 умножитель и 1 блок вычисления кубического корня. Описания работы блока вычисления кубического корня будет приведено позже.

Блок может пребывать в 4 состояниях: ожидать работы, выполнять вычисление квадрата a, выполнять вычисление кубического корня из b, выполнять сложение полученных результатов.

Ожидание работы:

```
IDLE:
      begin
            if (start_i) begin
                  y_calc <= 0;
                  state <= WORK_MULT;
                  start_mult <= 1;
                  a <= a_i;
            end
      end
```

Если блок готов к работе и поступил сигнал старт, то подается сигнал старта на умножитель и ему передается значение a_i.

Вычисление квадрата a:

```
WORK_MULT
      begin
            start_mult <= 0;
            if (~mult_busy && ~start_mult) begin
                  state = WORK_CUBEROOT;
                  b <= b_i;
                  start_cuberoot <= 1;
            end
       end
```

Если блок умножителя закончил вычисление, то переводим наш блок в состояние вычисления кубического корня из числа, подав на вход b число b_i и подав сигнал старта работы блока.

Вычисление кубического корня b:

```
WORK_CUBEROOT:
      begin
            start_cuberoot <= 0;
            if (~cube_busy && ~start_cuberoot) begin
                  state <= WORK_SUM;
            end
```

```
        end
```

Если блок вычислителя кубического корня закончил вычисление, то мы переходим в состояние вычисления суммы функции, в котором мы записываем сумму результата вычисления блока умножителя и вычислителя корня в вывод блока функции.

Код модуля кубического корня

```
`timescale 1ns / 1ps
module cuberoot(
    input clk_c,
    input rst_c,
    input [7:0] a,
    input start_c,
    output busy_o,
    output reg [3:0] y_bo
    );
    wire end_step;
    reg [7:0] x;
    reg [1:0] i;
    reg [4:0] state;
    reg [3:0] ans;
    reg [7:0] local_mask;
    reg [7:0] mask;
    reg [7:0] tmp;
    reg [15:0] tmp_mult_result;
    reg mult_start;
    wire mult_busy;

    reg [7:0] mult_a;
    reg [7:0] mult_b;
    wire [15:0] mult_result;
    multer multiplier(
        .clk_i(clk_c),
        .rst_i(rst_c),
        .a_bi(mult_a),
        .b_bi(mult_b),
        .start_i(mult_start),
        .busy_o(mult_busy),
        .y_bo(mult_result)
    );

    localparam IDLE = 0,
           CALC_SHIFT = 1,
```

```
        MASK = 2,
        NEW_ANS_BIT = 3,
        MUL1 = 4,
        WAIT_MUL1 = 5,
        MUL2 = 6,
        WAIT_MUL2 = 7,
        CHECK_X = 8,
        INVERT_ANS_BIT = 9,
        SUB_I = 10;
assign end_step = (i == 2'b11);
assign busy_o = (state != IDLE);

always @(posedge clk_c) begin
   if (rst_c) begin
        y_bo <= 0;
        i <= 0;
        mult_start <= 0;
        state <= IDLE;
        i <= 2'b10;
        local_mask <= 3'b111;
        mask <= 0;
   end else begin
        case(state)
            IDLE:
                begin
                    if (start_c) begin
                        y_bo <= 0;
                        state <= CALC_SHIFT;
                        mult_start <= 0;
                        local_mask <= 3'b111;
                        mask <= 0;
                        x <= a;
                        i <= 2'b10;
                        ans <= 0;
                     end
                 end
             CALC_SHIFT:
                begin
                    mult_a <= 3;
                    mult_b <= i;
                    mult_start <= 1;
                    state <= MASK;
                end
             MASK:
                begin
                    if (end_step) begin
```

```
                y_bo <= ans;
                state <= IDLE;
            end else begin
                mult_start <= 0;
                if (~mult_busy && ~mult_start) begin
                 mask <= mask | (local_mask << mult_result);
                    state <= NEW_ANS_BIT;
                end
            end
        end
    NEW_ANS_BIT:
        begin
            tmp <= x & mask;
            ans <= ans ^ (1 << i);
            state <= MUL1;
        end
    MUL1:
        begin
            mult_a <= ans;
            mult_b <= ans;
            mult_start <= 1;
            state <= WAIT_MUL1;
        end
    WAIT_MUL1:
        begin
            mult_start <= 0;
            if (~mult_busy && ~mult_start) begin
                tmp_mult_result <= mult_result;
                state <= MUL2;
            end
        end
    MUL2:
        begin
            mult_a <= tmp_mult_result[7:0];
            mult_b <= ans;
            mult_start <= 1;
            state <= CHECK_X;
        end
    CHECK_X:
        begin
           mult_start <= 0;
           if (~mult_busy && ~mult_start) begin
                tmp_mult_result <= mult_result;
                state <= INVERT_ANS_BIT;
           end
        end
```

```
                INVERT_ANS_BIT:
                    begin
                        if (tmp < tmp_mult_result) begin
                            ans <= ans ^ (1 << i);
                        end
                        state <= SUB_I;
                    end
                SUB_I:
                    begin
                        if (i == 2'b00) begin
                            i <= 2'b11;
                        end else begin
                            i <= i - 1;
                        end
                        state <= CALC_SHIFT;
                    end
            endcase
        end
    end

endmodule
```

Описание работы модуля кубического корня

Модуль реализует алгоритм извлечение кубического корня из числа в двоичной системе счисления. Суть алгоритма заключается в следующем, мы разбиваем число по тройкам с конца. Ставим первую цифру ответа в единицу, после чего сдвигаем ее на количество полученных троек в исходном числе влево. Находим куб полученного числа и сравниваем его с первой слева тройкой исходного числа смещенной влево на (N-1) * 3 бит, где N – количество троек в исходном числе. Если полученный куб меньше смещенной тройки, то переходим на 2 круг алгоритма, если больше или равен, то меняем единицу в ответе на 0 и продолжаем алгоритм.

Область допустимых значений и множество значений разработанного модуля

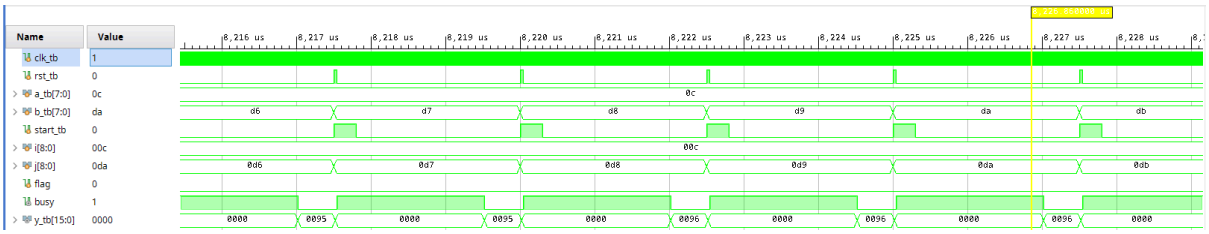
$0 \leq a \leq 255$

$0 \leq b \leq 255$

$y = a^2 + \sqrt[3]{b}$

$$0 \leq a^2 \leq 65025$$
$$0 \leq \sqrt[3]{b} \leq 6$$
$$0 \leq y \leq 65031$$

Результаты тестирования разработанного модуля



"Рисунок 3. Временная диаграмма разработанного модуля"

На данной диаграмме мы можем видеть, что при a = 0c (т.е. 12) и b = d7 (т.е. 215), y = 0095 (т.е. 149).

Давайте проверим:

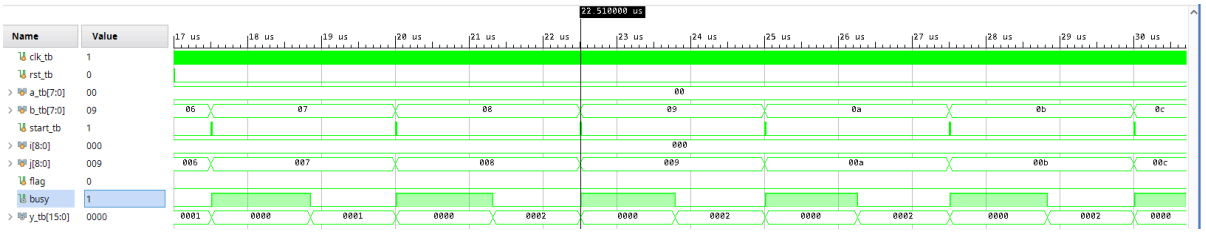
$$y = a^2 + \sqrt[3]{b} = 12^2 + \sqrt[3]{215} = 144 + 5 = 149$$

Также на диаграмме видно, что в при a = 0c и b = 0d8 (т.е. 216), y = 0096 (т.е. 150)

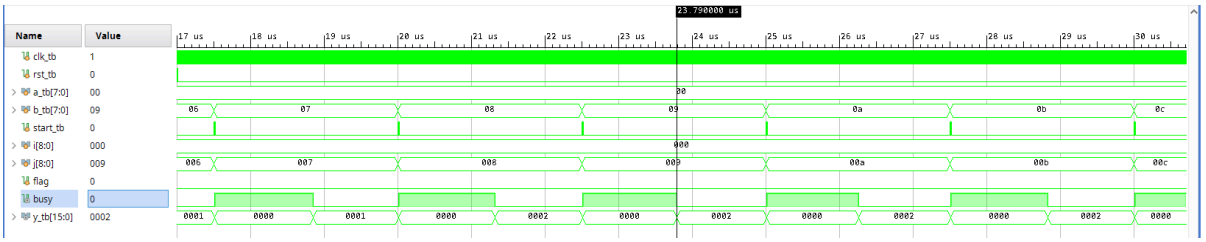
Проверка:

$$y = a^2 + \sqrt[3]{b} = 12^2 + \sqrt[3]{216} = 144 + 6 = 150$$

Время вычисления результата



"Рисунок 4. Время начала вычисления значения"



"Рисунок 5. Время окончания вычисления значения"

При тактовой частоте сигнала в 100 МГц время вычисления результата модулем составляет: $t = 23790 \text{ нс} - 22510 \text{ нс} = 1280 \text{ нс}$

Вывод

В ходе выполнения лабораторной работы, я приобрел навыки описания арифметических блоков на уровне Register Transfer Level (RTL) с использованием языка описания аппаратуры Verilog HDL. Разработав цифровой блок, реализующий функцию, согласно варианту, с ограничением на использование 1 сумматора и 2 умножителей, я углубил свое понимание принципов работы цифровых схем и их оптимизации. Определение области допустимых значений функции, создание тестового окружения для проверки схемы на различных входных векторах, моделирование при тактировании от сигнала с частотой 100 МГц и анализ времени вычислений позволили мне овладеть методами верификации и оптимизации цифровых схем.