Отчет

Подготовил Пашенцев Павел Владимирович

Оглавление

Задание 2
Выполнение 3
Схема цифрового устройства 3
Описание цифровой схемы 3
Обязательные требования входных значений 3
Дополнительные требования входных значений 3
Верификация модуля 4
Проверка корректности верификации 4
Возможные способы защиты 5
Оценка аппаратных ресурсов 5
Оценка максимальной тактовой частоты 6
Приложение 7
Q_calculator.sv 7
tb_Q_calculator.sv 9
main.py 11
Tests.csv 11

Задание

Дано следующее математическое выражение:

$$Q = \frac{(a - b) \cdot (1 + 3c) - 4d}{2}$$

1. Нарисуйте в любом графическом редакторе цифровую схему, реализующую вычисление заданного выражения (Visio, draw.io или аналогичный)
2. Используя любой HDL язык (Verilog, SystemVerilog, VHDL) опишите цифровую схему, отвечающую заданным требованиям:
3. входные параметры *a, b, c, d* являются целыми числами со знаком (signed)
4. набор параметров *a, b, c, d* должен подаваться на вход схемы синхронно
5. разрядность данных должна определяться параметром
6. схема должна обеспечивать возможность получения нового набора входных параметров *a, b, c, d* каждый такт
7. латентность схемы должна быть оптимальной
8. по возможности реализовать подтверждение входных и выходных данных сигналом valid
9. Верифицируйте описанную схему с помощью testbench. В качестве симулятора можно использовать САПР Vivado, ModelSim или аналогичный
10. Для проверки корректности работы схемы и testbench, разработайте программу на Python, решающую заданное математическое выражение
11. Описать возможные способы защиты от ошибок переполнения разрядной сетки

При наличии инструментальной возможности (оценивается отдельно):

12. Оценить аппаратный ресурс, требуемый для реализации схемы по результатам синтеза, выполненного в САПР Vivado или Quartus
13. Оценить максимальную тактовую частоту работы схемы

Выполнение

Схема цифрового устройства

Составим цифровую схему требуемого устройства (рис. 1). Заметим, что некоторые блоки можно вычислять параллельно.

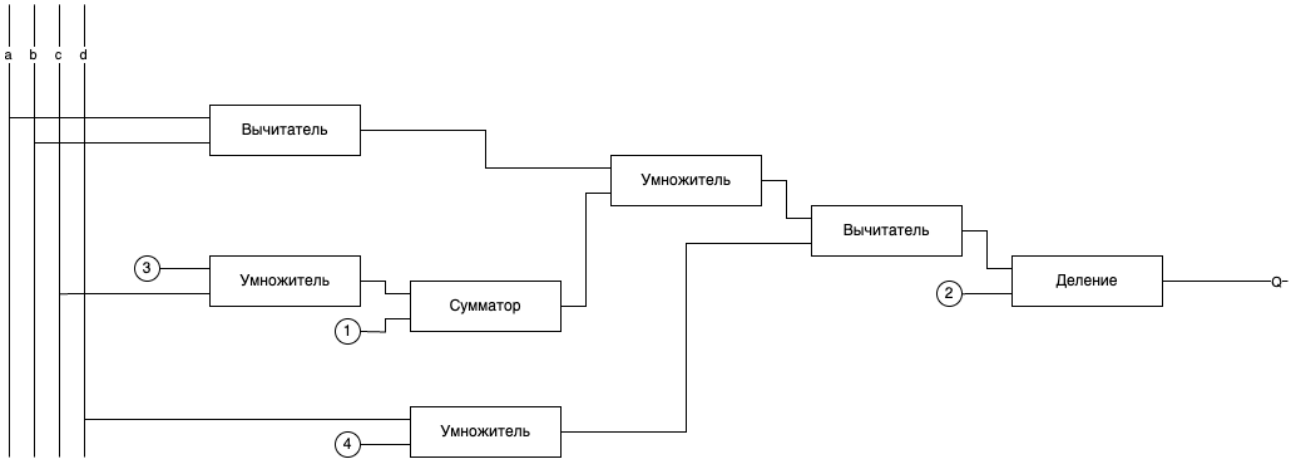

Рисунок 1. Схема устройства

Описание цифровой схемы

Теперь напишем код модуля (приложение Q_calculator.sv) на языке System Verilog для реализации цифровой схемы. Имеются 4 входа (a, b, c, d) и один выход функции в соответствии заданию.

После компиляции получим следующую RTL-модель (рис. 2).

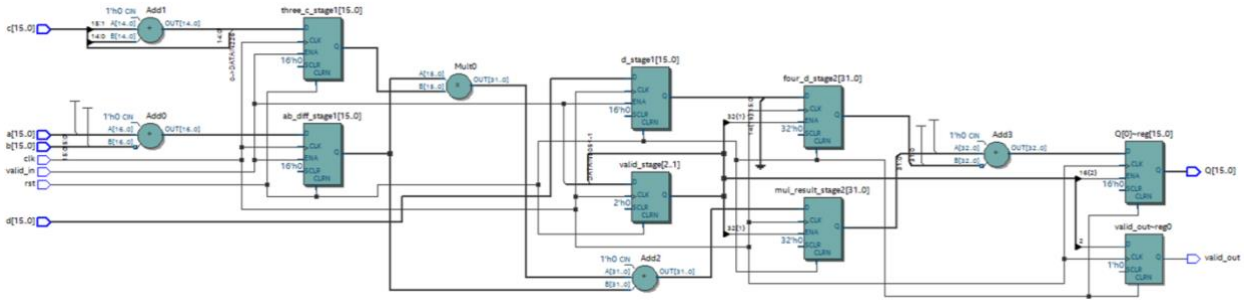

Рисунок 2. RTL-модель

Обязательные требования входных значений

Входные параметры a, b, c, d объявлены как знаковые числа (пункт 3), подаются синхронно по сигналу valid_in (пункт 4), что обеспечивает одновременную обработку всего набора. Разрядность данных задаётся параметром WIDTH, позволяя гибко адаптировать модуль под нужную точность (пункт 5).

Дополнительные требования входных значений

Схема реализована как конвейер с тремя стадиями и поддержкой сигнала valid, что позволяет принимать новый набор параметров a, b, c, d на каждом такте

(пункт 6). Латентность сведена к минимуму за счёт распараллеливания операций по стадиям (пункт 7), а сигналы valid_in и valid_out обеспечивают подтверждение актуальности входных и выходных данных (пункт 8).

Верификация модуля

Был разработан тестбенч для проверки работы разрабатываемого модуля (приложение tb_Q_calculator.sv). Waveform-диаграмма приведена на рисунке 3. На рисунке 4 приведен текстовый отчет пройденных тестов.

Рисунок 3. Waveform диаграмма тестбенча

```
VSIM 40> run -all
# PASS test 0: Q=-1336
# PASS test 1: Q=-2502
# PASS test 2: Q=1334
# PASS test 3: Q=165
# PASS test 4: Q=2099
# PASS test 5: Q=-13108
# PASS test 6: Q=3483
# PASS test 7: Q=5215
# PASS test 8: Q=300
# PASS test 9: Q=6752
# PASS test 10: Q=12284
# PASS test 11: Q=3126
# PASS test 12: Q=-13998
# PASS test 13: Q=3523
# PASS test 14: Q=4203
# PASS test 15: Q=3323
# PASS test 16: Q=3626
# PASS test 17: Q=96
# PASS test 18: Q=-9180
# PASS test 19: Q=932
```

Рисунок 4. Текстовый отчет

Проверка корректности верификации

Для проверки корректности верификационного модуля был написан скрипт на Python (приложение main.py) для проверки входного вектора и выходного значения. Также был сформирован файл тестирующей выборки для

дальнейшей верификации (приложение Tests.csv), выборка из этого файла представлена на рисунке 5.

	a	b	c	d	Q_expected
0	-74	-34	20	58	-1336
1	-72	-6	23	96	-2502
2	78	29	16	-67	1334
3	37	27	6	-35	165
4	6	-87	15	20	2099
5	66	7	15	4	1349

Рисунок 5. Выборка из Tests.csv

Возможные способы защиты

В коде защита от переполнения достигается расширением разрядности: все промежуточные операции выполняются в удвоенной ширине EXT = 2 * WIDTH, что предотвращает потерю значащих битов. Дополнительно можно реализовать проверку старших битов или флаг переполнения для контроля ошибок.

Оценка аппаратных ресурсов

В качестве аппаратной платформы была выбрана плата DE10-Lite с чипом 10M50DAF484C7G. Для оценки работы потребовалось скомпилировать проект (рис. 6).

По результатам синтеза итоговая схема использует всего 98 логических элементов из 49 760 (<1 %), 81 регистр и 2 встроенных 9-битных умножителя из 288 (<1 %). Это означает, что схема занимает крайне малую часть доступных ресурсов, оставляя большой запас для дальнейшего расширения или интеграции с другими модулями. Память, PLL, UFM и другие блоки не используются.

Flow Status	Successful - Tue May 06 16:57:40 2025
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	TestTask
Top-level Entity Name	Q_calculator
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	98 / 49,760 (< 1 %)
Total registers	81
Total pins	84 / 360 (23 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	2 / 288 (< 1 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Рисунок 6. Результат компиляции

Оценка максимальной тактовой частоты

Для оценки максимальной тактовой частоты использовался TimeQuest Timing Analyzer. Максимальная тактовая частота была получена при помощи Fmax Summary (рис. 7). Также можно проанализировать распространение сигнала внутри схемы (рис. 8) при помощи Report Clocks.

Slow 1200mV 85C Model

	Fmax	Restricted Fmax	Clock Name
1	153.59 MHz	153.59 MHz	clk

Рисунок 7. Максимальная тактовая частота

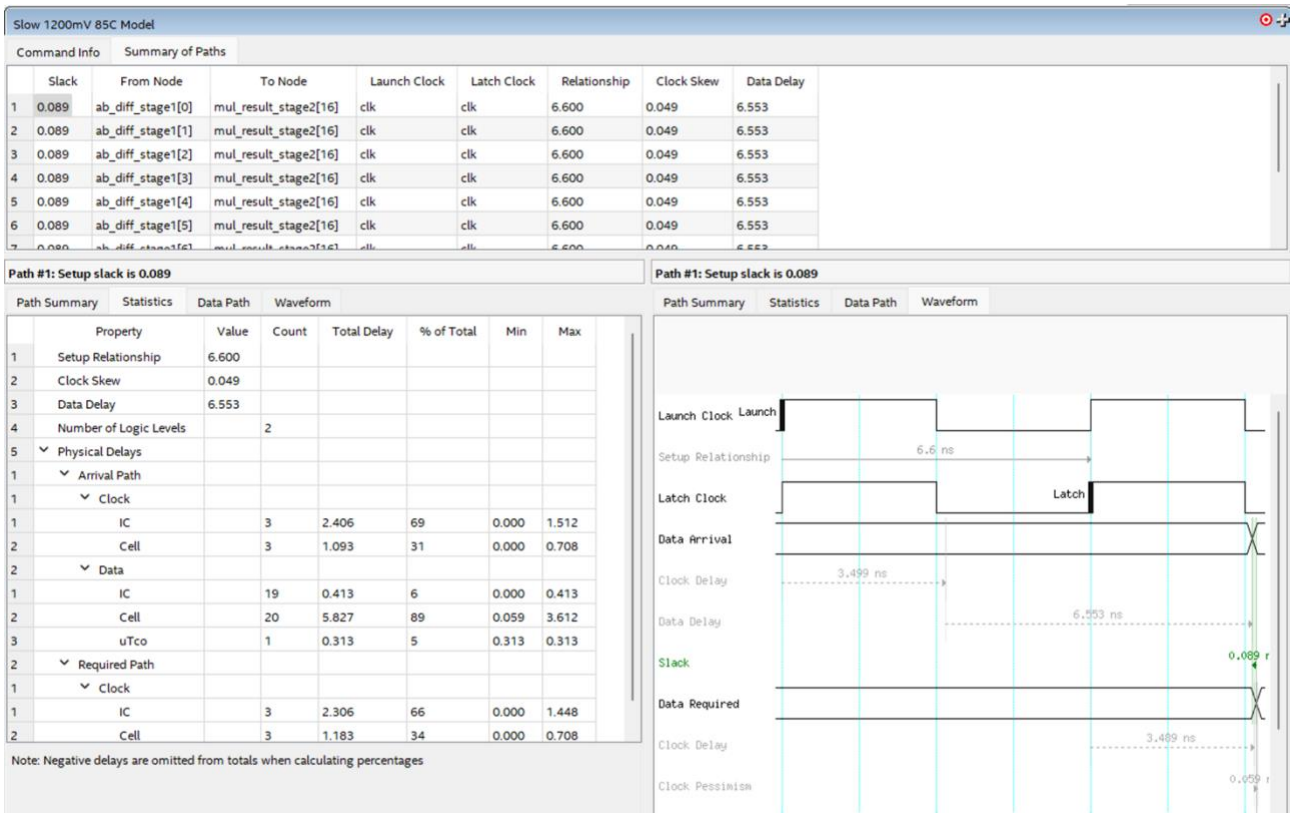

Рисунок 8. Распостранение сигнала в схеме

Выводы

В рамках задания была разработана цифровая схема для вычисления заданного выражения с параметризуемой разрядностью и синхронной подачей входных данных. Схема реализована на SystemVerilog в виде трёхстадийного конвейера с поддержкой сигналов valid и защитой от переполнения за счёт расширения разрядности. Проведена верификация с помощью тестбенча и Python-скрипта, подтверждающая корректность работы. Синтез показал минимальное использование аппаратных ресурсов и высокую максимальную тактовую частоту.

Приложение

Q_calculator.sv

```
module Q_calculator #(
    parameter WIDTH = 16  // разрядность входных данных
)(
    input  wire clk,
    input  wire rst,
    input  wire valid_in,
    input  wire signed [WIDTH-1:0] a,
    input  wire signed [WIDTH-1:0] b,
    input  wire signed [WIDTH-1:0] c,
    input  wire signed [WIDTH-1:0] d,
    output reg  signed [WIDTH-1:0] Q,
    output reg  valid_out
);

    // Удвоенная разрядность для промежуточных значений
    localparam EXT = 2 * WIDTH;

    // Регистры для стадии 1
    reg signed [WIDTH-1:0] ab_diff_stage1;
    reg signed [WIDTH-1:0] three_c_stage1;
    reg signed [WIDTH-1:0] d_stage1;
    reg                    valid_stage1;

    // Регистры для стадии 2 (расширенные)
    reg signed [EXT-1:0] mul_result_stage2;
    reg signed [EXT-1:0] four_d_stage2;
    reg                  valid_stage2;

    // Стадия 1: вычисляем a - b, 3c, сохраняем d
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            ab_diff_stage1   <= 0;
            three_c_stage1   <= 0;
            d_stage1         <= 0;
            valid_stage1     <= 0;
        end else begin
            if (valid_in) begin
                ab_diff_stage1 <= a - b;
                three_c_stage1 <= 3 * c;
                d_stage1       <= d;
                valid_stage1   <= 1;
            end else begin
```

```
                valid_stage1 <= 0;
            end
        end
    end

    // Стадия 2: основная арифметика (всё в расширенной
разрядности)
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            mul_result_stage2 <= 0;
            four_d_stage2     <= 0;
            valid_stage2      <= 0;
        end else begin
            if (valid_stage1) begin
                // расширяем до EXT с сохранением знака
                mul_result_stage2  <=  $signed({{(EXT  -
WIDTH){ab_diff_stage1[WIDTH-1]}}, ab_diff_stage1}) *
                                       $signed({{(EXT     -
WIDTH){three_c_stage1[WIDTH-1]}}, three_c_stage1}) +
                                       $signed({{(EXT     -
WIDTH){ab_diff_stage1[WIDTH-1]}}, ab_diff_stage1});  // +1
                four_d_stage2        <= $signed({{(EXT  -
WIDTH){d_stage1[WIDTH-1]}}, d_stage1}) <<< 2; // 4*d
                valid_stage2      <= 1;
            end else begin
                valid_stage2 <= 0;
            end
        end
    end

    // Стадия 3: вычитание и деление на 2, вывод
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            Q         <= 0;
            valid_out <= 0;
        end else begin
            if (valid_stage2) begin
                Q         <= ($signed(mul_result_stage2 -
four_d_stage2)) >>> 1;
                valid_out <= 1;
            end else begin
                valid_out <= 0;
            end
        end
    end
```

```
endmodule
```

tb_Q_calculator.sv

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

  parameter int WIDTH      = 16;
  parameter int CLK_PERIOD = 10;
  parameter int NUM_TESTS  = 20;

  logic clk = 0;
  logic rst;
  logic valid_in;
  logic signed [WIDTH-1:0] a, b, c, d;
  logic signed [WIDTH-1:0] Q;
  logic valid_out;

  Q_calculator #(.WIDTH(WIDTH)) dut (
    .clk       (clk),
    .rst       (rst),
    .valid_in  (valid_in),
    .a         (a),
    .b         (b),
    .c         (c),
    .d         (d),
    .Q         (Q),
    .valid_out (valid_out)
  );

  always #(CLK_PERIOD/2) clk = ~clk;

  typedef struct packed {
    logic signed [WIDTH-1:0] a, b, c, d;
  } tv_t;

  tv_t tests[NUM_TESTS] = '{
    '{-74, -34,  20,  58},
    '{-72,  -6,  23,  96},
    '{ 78,  29,  16, -67},
    '{ 37,  27,   6, -35},
    '{  6, -87,  15,  20},
    '{  3, -99, -85,  77},
    '{-46,  35, -28, -61},
```

```
  '{ 32,  89, -61, -14},
  '{-56, -48, -17, -50},
  '{-79,  45, -36, -59},
  '{  5, -97,  79, -73},
  '{-22,  74, -23,  69},
  '{-78,  26,  90, -47},
  '{-63,  50, -20, -95},
  '{ 61,  32,  99,  59},
  '{-58, -89,  67, -96},
  '{-88,  64, -17,  87},
  '{-88, -87,  33, -73},
  '{ 49, -34, -75, -58},
  '{-73, -88,  33, -91}
};

function automatic logic signed [WIDTH-1:0] reference
  (input logic signed [WIDTH-1:0] aa,
   input logic signed [WIDTH-1:0] bb,
   input logic signed [WIDTH-1:0] cc,
   input logic signed [WIDTH-1:0] dd);
  reference = ((aa - bb) * (3*cc + 1) - 4*dd) >>> 1;
endfunction

logic signed [WIDTH-1:0] exp_Q;
int i;

initial begin
  rst = 1;
  valid_in = 0;
  a = 0; b = 0; c = 0; d = 0;
  #(2*CLK_PERIOD);
  rst = 0;

  for (i = 0; i < NUM_TESTS; i++) begin
    a = tests[i].a;
    b = tests[i].b;
    c = tests[i].c;
    d = tests[i].d;
    valid_in = 1;
    @(posedge clk);
    valid_in = 0;

    wait (valid_out == 1);
    exp_Q = reference(a, b, c, d);
```

```
      if (Q !== exp_Q)
        $error("FAIL test %0d: exp=%0d got=%0d", i, exp_Q,
Q);
      else
        $display("PASS test %0d: Q=%0d", i, Q);

      @(posedge clk);
    end

    $finish;
  end

endmodule
```

main.py

```
import random
import pandas as pd

# Определение функции на Python, которая рассчитывает
значение Q
def compute_Q(a: int, b: int, c: int, d: int) -> int:
    return ((a - b) * (1 + 3 * c) - 4 * d) // 2

# Генерация 20 случайных тестов с входами в диапазоне от -
100 до 100
test_vectors = []
for _ in range(20):
    a = random.randint(-100, 100)
    b = random.randint(-100, 100)
    c = random.randint(-100, 100)
    d = random.randint(-100, 100)
    q = compute_Q(a, b, c, d)
    test_vectors.append((a, b, c, d, q))

df = pd.DataFrame(test_vectors, columns=["a", "b", "c",
"d", "Q_expected"])
```

Tests.csv

```
a,b,c,d,Q_expected
-74,-34,20,58,-1336
-72,-6,23,96,-2502
78,29,16,-67,1334
37,27,6,-35,165
6,-87,15,20,2099
66,7,15,4,1349
```

```
15,45,-45,-9,2028
28,11,-100,-72,-2398
82,-85,38,-10,9622
-49,-54,42,-99,515
74,-3,-44,86,-5216
68,-95,-92,34,-22481
-53,-73,38,-83,1316
-87,5,-3,83,202
18,-45,-85,70,-8141
18,34,71,94,-1900
0,23,33,-69,-1012
95,-91,-25,93,-7068
-49,-15,-93,56,4614
49,-82,42,58,8202
```