МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ОТЧЁТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2

«Организация продвижения данных по конвейеру»

по дисциплине

«Схемотехника устройств компьютерных систем»

Выполнил студент группы ИВБО-11-23 — Туктаров Т.А.

Принял старший преподаватель кафедры ВТ — Дуксин Н.А.

Практическая работа выполнена «__»_____2025 г.

«Зачтено» «__»_____2025 г.

Москва 2025

АННОТАЦИЯ

Данная работа включает в себя 5 рисунков, 6 листингов. Количество страниц в работе — 17.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Метод конвейеризации подразумевает передачу данных между регистрами, позволяющий использовать параллелизм операций, относящихся к разным итерациям цикла. Цель — построить расписание, при котором последовательные итерации цикла запускались бы с некоторым постоянным интервалом, и свести к минимуму этот интервал за счёт перекрытия различных итераций исходного цикла.

1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: знакомство студентов с возможными аппаратными ресурсами, которые могут быть задействованы при проектировании, а также связь описания устройства при помощи языков описания аппаратуры и полученными результатами синтеза.

Задание: реализовать модули разных методов конвейеризаии на языке Verilog в САПР vivado.

2 РЕАЛИЗАЦИЯ МЕТОДОВ КОНВЕЙЕРИЗАЦИИ

2.1 Реализация трех модулей конвейеризации

При помощи языка описания аппаратуры Verilog средствами САПР Vivado реализуем три метода конвейеризации. Реализация параллельного метода представлена в двух вариантах в Листинге 2.1.1 и 2.1.2

Листинг 2.1.1 — Модуль par_bubble.v

```
`timescale 1ns / 1ps

module par_bubble #(STAGE_COUNT = 16)(
    input clk, reset, fifo_out_ready,
    input [STAGE_COUNT-1:0] valid_stage_in,
    output wire [STAGE_COUNT-1:0] is_ready
    );

assign is_ready[STAGE_COUNT - 1] = ~valid_stage_in[STAGE_COUNT - 1] ||
fifo_out_ready;
genvar i;
generate
    for(i = STAGE_COUNT - 2; i >= 0; i = i - 1)
    begin
        // assign is_ready[i] = ~valid_stage_in[i] || is_ready[i+1];
        assign is_ready[i] = ~valid_stage_in[i] || fifo_out_ready  ||
is_ready[STAGE_COUNT-1:i+1];
    end
endgenerate

endmodule
```

Листинг 2.1.2 — Модуль par_bubble.v

```
`timescale 1ns / 1ps

module par_bubble #(STAGE_COUNT = 16)(
    input clk, reset, fifo_out_ready,
    input [STAGE_COUNT-1:0] valid_stage_in,
    output wire [STAGE_COUNT-1:0] is_ready
    );

assign is_ready[STAGE_COUNT - 1] = ~valid_stage_in[STAGE_COUNT - 1] ||
fifo_out_ready;

genvar i;
genvar j;
generate
    for(i = STAGE_COUNT - 2; i >= 0; i = i - 1)
    begin
        // assign is_ready[i] = ~valid_stage_in[i] || is_ready[i+1];

        for(j = i + 1; j < STAGE_COUNT; j = j + 1)
        begin
            assign is_ready[i] = ~valid_stage_in[i] || fifo_out_ready  ||
is_ready[j];
        end
    end
endgenerate

endmodule
```

Делее представлена реализация последовательного метода, код которого продемонстрирован в Листингах 2.2-2.4.

Листинг 2.2 — Модуль bubble

```
`timescale 1ns / 1ps

module bubble #(STAGE_COUNT = 16)(
    input clk, reset, fifo_out_ready,
    input [STAGE_COUNT-1:0] valid_stage_in,
    output wire [STAGE_COUNT-1:0] is_ready
    );

assign is_ready[STAGE_COUNT - 1] = ~valid_stage_in[STAGE_COUNT - 1] ||
fifo_out_ready;

genvar i;

generate
    for(i = STAGE_COUNT - 2; i >= 0; i = i - 1)
    begin
        assign is_ready[i] = ~valid_stage_in[i] || is_ready[i+1];
    end
endgenerate

endmodule
```

Листинг 2.3 — Модуль valid_chain.v

```
`timescale 1ns / 1ps

module valid_chain2
# (STAGE_COUNT = 16)(
    input clk, reset, valid_in, fifo_out_ready,
    input [STAGE_COUNT-1:0] is_ready,
    output reg [STAGE_COUNT-1:0] valid_out
    );

// reg [STAGE_COUNT-1:0] buffer;

always@(posedge clk)
    if(reset)
        valid_out[0] <= 0;
    else if (is_ready[0])
        valid_out[0] <= valid_in;

genvar i;

wire [STAGE_COUNT-1:0] allow;

generate
    for(i = 1; i < STAGE_COUNT; i = i + 1)
    begin
    always@(posedge clk)
        if(reset)
            valid_out[i] <= 0;
        else if (is_ready[i])
        begin
            valid_out[i] <= valid_out[i-1];
        end
    end
endgenerate

endmodule
```

Листинг 2.4 — Модуль pipeline.v

```
`timescale 1ns / 1ps

module pipeline2 #(DATA_SIZE = 8, STAGE_COUNT = 16)(
    input clk, reset, valid_in, fifo_out_ready,
    input [STAGE_COUNT-1:0] valid_stage_in, // было
    input [STAGE_COUNT-1:0] is_ready,
    // input [STAGE_COUNT-1:0] is_ready, // добавлено
    input [DATA_SIZE-1:0] data_in,
    output [DATA_SIZE-1:0] data_out
    );

reg [DATA_SIZE - 1:0] pipeline_reg [0:STAGE_COUNT - 1];

always@(posedge clk)
    if (reset)
        pipeline_reg[0] <= 0;
    else if (is_ready[0]) // valid_in && fifo_out_ready
         pipeline_reg[0] <= data_in;

genvar i;
generate
    for(i = 1; i < STAGE_COUNT; i = i + 1)
    begin
        always@(posedge clk)
            if (reset)
                pipeline_reg[i] <= 0;
            else if (is_ready[i]) // было valid_stage_in[i-1] &&
fifo_out_ready
            //else if (is_ready[i - 1]) // стало
                 pipeline_reg[i] <= pipeline_reg[i-1] + 1;
    end
endgenerate

assign data_out = pipeline_reg[STAGE_COUNT - 1];

endmodule
```

В Листинге 2.5 реализован модуль классического метода конвейеризации.

Листинг 2.5 – Простая конвертизация в модуле dipeline.

```
`timescale 1ns / 1ps

module dipeline #(DATA_SIZE = 8, STAGE_COUNT = 16)(
    input clk, reset, valid_in, fifo_out_ready,
    input [STAGE_COUNT-1:0] valid_stage_in,
    input [DATA_SIZE-1:0] data_in,
    output [DATA_SIZE-1:0] data_out
    );

reg [DATA_SIZE - 1:0] pipeline_reg [0:STAGE_COUNT - 1];

always@(posedge clk)
    if (reset)
        pipeline_reg[0] <= 0;
    else if (valid_in && fifo_out_ready)
         pipeline_reg[0] <= data_in;

genvar i;
generate
    for(i = 1; i < STAGE_COUNT; i = i + 1)
    begin
        always@(posedge clk)
            if (reset)
                pipeline_reg[i] <= 0;
            else if (valid_stage_in[i-1] && fifo_out_ready)
                 pipeline_reg[i] <= pipeline_reg[i-1] + 1;
    end
endgenerate

assign data_out = pipeline_reg[STAGE_COUNT - 1];

endmodule
```

В Листингах 2.6-2.7 реализованы модули верхнего уровня и тестовый модуль.

Листинг 2.5 – Top.

```
` `timescale 1ns / 1ps

module top2
#(DATA_SIZE = 8, STAGE_COUNT = 16)(
    input clk, reset, valid_in, fifo_out_ready,
    input [DATA_SIZE-1:0] data_in,
    output [DATA_SIZE-1:0] data_out,
    output valid_out
    );

pipeline2 #(
    .DATA_SIZE(DATA_SIZE), .STAGE_COUNT(STAGE_COUNT)
) dataflow (
    .clk(clk),
    .reset(reset),
    .fifo_out_ready(fifo_out_ready),
    .valid_in(valid_in),
    .data_in(data_in),
    .valid_stage_in(valid_stage), // было оригинально
    .is_ready(is_ready),
    // .is_ready(val_chain), // Добавлено
    .data_out(data_out)
);

wire [STAGE_COUNT-1:0] valid_stage;

valid_chain2 #(.STAGE_COUNT(STAGE_COUNT))controlflow(
    .clk(clk),
    .reset(reset),
    .fifo_out_ready(fifo_out_ready),
    .valid_in(valid_in),
    .is_ready(is_ready),
    // .data_out(data_out),
    .valid_out(valid_stage)
);

wire [STAGE_COUNT-1:0] is_ready;

bubble #(.STAGE_COUNT(STAGE_COUNT)) readiness(
    .clk(clk),
    .reset(reset),
    .fifo_out_ready(fifo_out_ready),
    .valid_stage_in(valid_stage),
    .is_ready(is_ready)
);

assign valid_out = valid_stage[STAGE_COUNT - 1];

endmodule
```

Листинг 2.6 – test.

```
`timescale 1ns / 1ps

module test;

reg clk = 0;
always #5 clk <= ~clk;

localparam DATA_SIZE = 8;
localparam STAGE_COUNT = 16;

reg reset, valid_in, fifo_out_ready;
// reg [DATA_SIZE-1:0] data_in;
wire [DATA_SIZE-1:0] data_out;
wire valid_out;

top2 #(.STAGE_COUNT(STAGE_COUNT), .DATA_SIZE(DATA_SIZE)) uut(
    .clk(clk),
    .reset(reset),
    .data_in(1),
    .fifo_out_ready(fifo_out_ready),
    .valid_in(valid_in),
    .data_out(data_out),
    .valid_out(valid_out)
);

initial
begin
    fifo_out_ready <= 1;
    valid_in <= 0;
    reset <= 1;
    @(posedge clk);
    @(posedge clk);
    reset <= 0;
    valid_in <= 1;
    @(posedge clk);
    @(posedge clk);
    valid_in <= 0;
    @(posedge clk);
    @(posedge clk);
    valid_in <= 1;
    @(posedge clk);
    @(posedge clk);
    fifo_out_ready <= 0;
    @(posedge clk);
    @(posedge clk);
    // fifo_out_ready <= 1;
end
endmodule
```

Далее представлены результаты симуляции обычного метода конвейеризации на рисунке 2.1.

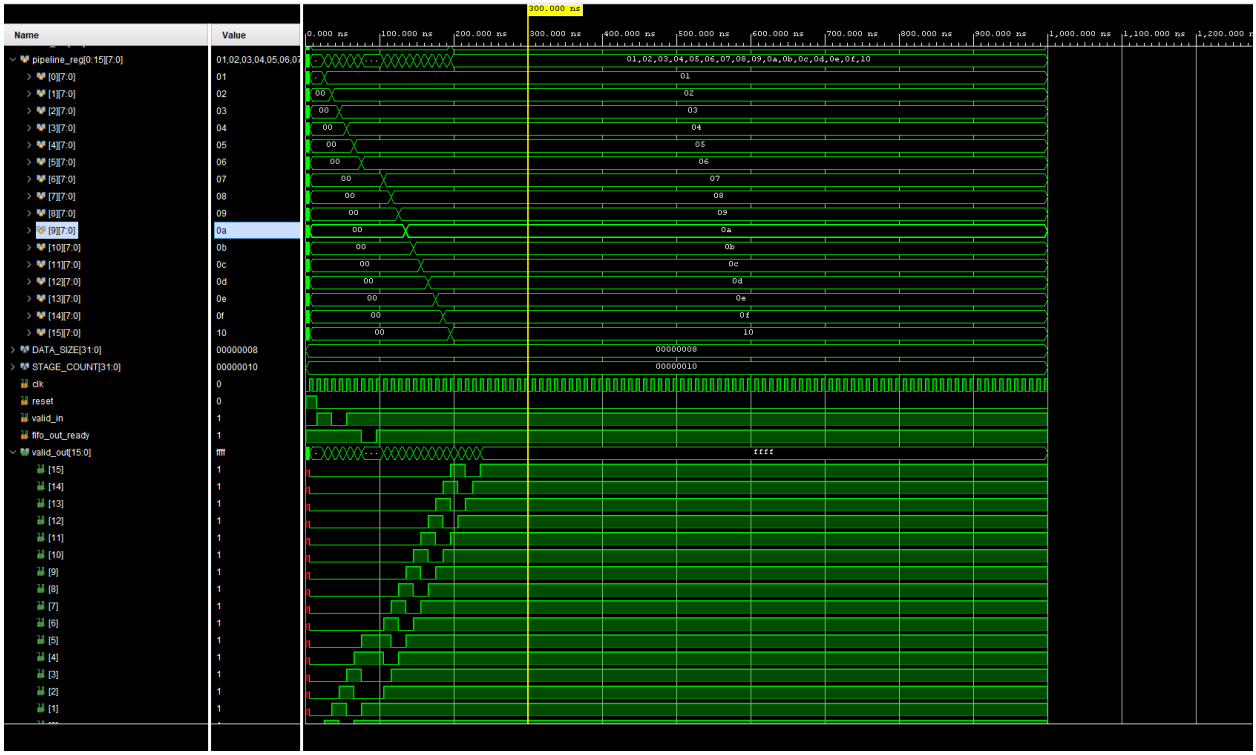

Рисунок 2.1 — Симуляция модуля конвейеризации.

На рисунках 2.2 – 2.3 представлена симуляция последовательного метода конвейеризации.

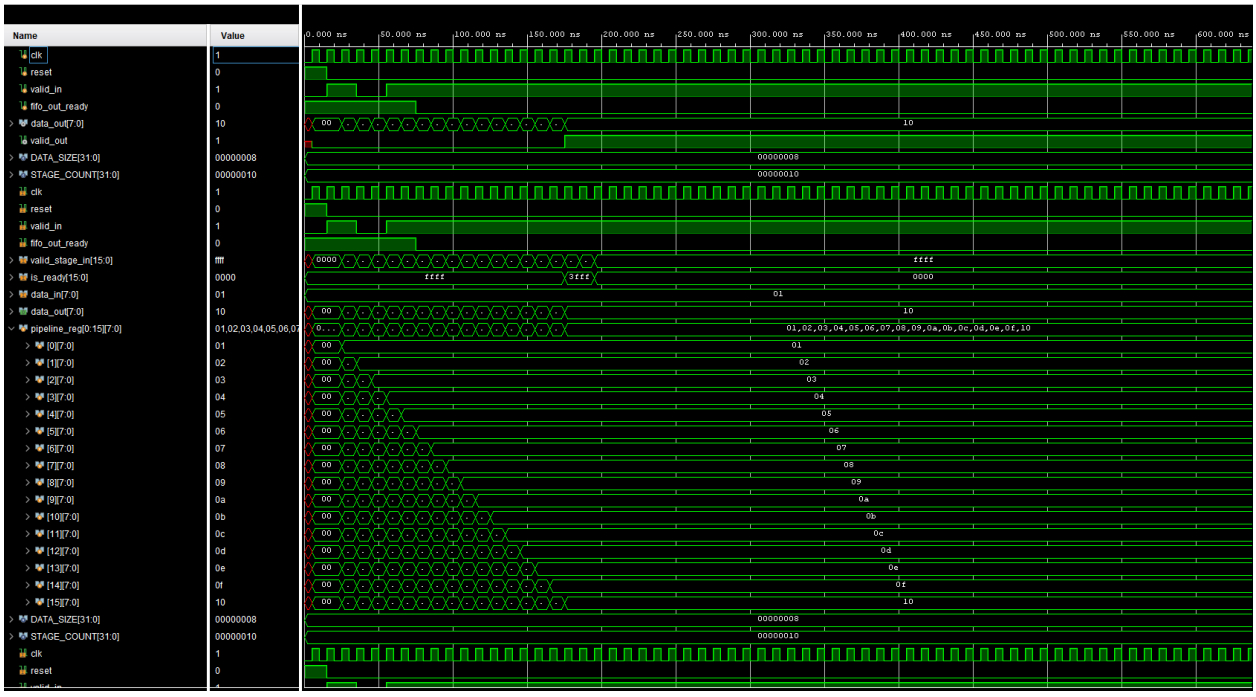

Рисунок 2.2 — Симуляция модуля конвейеризации последовательным методом

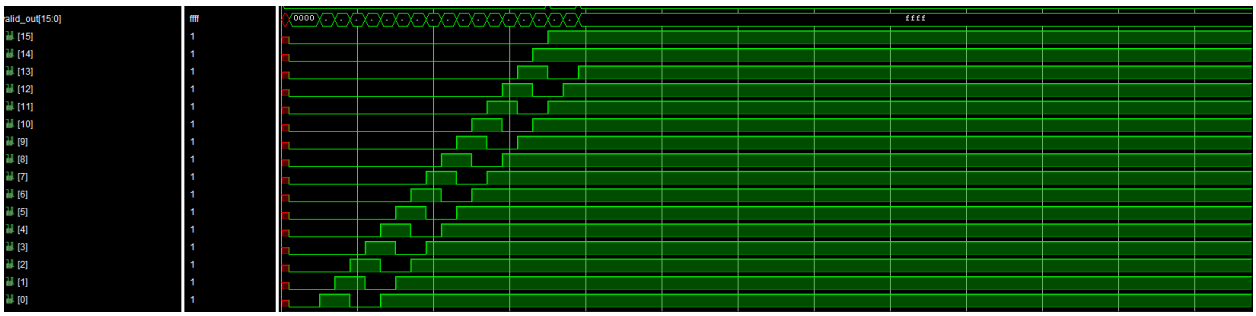

Рисунок 2.3 — Симуляция модуля конвейеризации последовательным методом

На рисунках 2.4 – 2.5 представлена симуляция параллельного метода конвейеризации

Рисунок 2.4 — Симуляция модуля конвейеризации параллельным методом

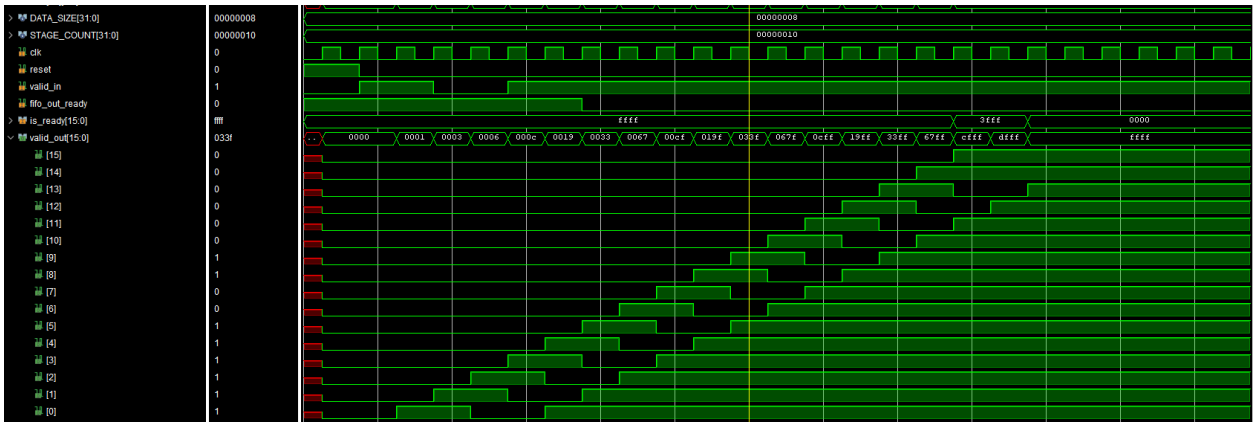

Рисунок 2.5 — Симуляция модуля конвейеризации параллельным методом

Далее представлена таблица 1, показывающая задержки для каждого метода конвейеризации при разных количествах регистров.

Таблица 1 – Таблица временных задержек

Метод конвейеризации/Количество регистров	3	10	16
Классический метод	7.446	7.462	7.607
Последовательный метод	7.488	7.593	7.914
Параллельный метод	7.229	7.379	7.431

По записанным результатам можно сделать выводы о временных задержках в данных методах, заметно отличается длина slack у параллельного метода.

ЗАКЛЮЧЕНИЕ

По завершении практической работы были реализованы модули с разными вариантами реализации контейнеризации. Были написаны модули, описывающие стандартный метод, последовательный и параллельный на Verilog HDL. Произведена симуляция полученных модулей средствами САПР Vivado. Были проанализированы результаты симуляции и рассмотрены их различия. Также получены базовые представления о реализации конвейеризации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Тарасов И. Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. М.: Издательство: Горячая линия - Телеком, 2019 г. ISBN: 978-5-9912-0802-4

2. Орлов С.А. Организация ЭВМ и систем: Учебник для вузов. 3-е изд. Стандарт третьего поколения / С.А. Орлов, Б.Я. Цилькер. – Санкт-Петербург: Питер, 2014. - 688 с. - ISBN 978-5-496-01145-7.

3. Паттерсон Д., Хеннесси Дж. Архитектура компьютера и проектирование компьютерных систем. 4-е изд. СПб.: Питер, 2012. – ISBN 978- 5-459-00291-1.

4. Рабан, Жан.М., Чандракасан, А., Николич, Б. Цифровые интегральные схемы. Методология проектирования. 2-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2016. – 912 с.: ил. – Паралтит. англ. ISBN 978-5-8459- 1116-2 (рус.).

5. Шафер Д., Фатрелл Р., Шафер Л. Управление программными проектами: достижение оптимального качества при минимуме затрат: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 1136 с.: ил. – Парал.тит.англ.