|  |
| --- |
|  |
| 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 4](#_Toc210505924)

[1 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc210505925)

[2 Реализация методов конвейеризации 6](#_Toc210505926)

[2.1 Реализация трех модулей конвейеризации 6](#_Toc210505927)

[ЗАКЛЮЧЕНИЕ 15](#_Toc210505928)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc210505929)

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.

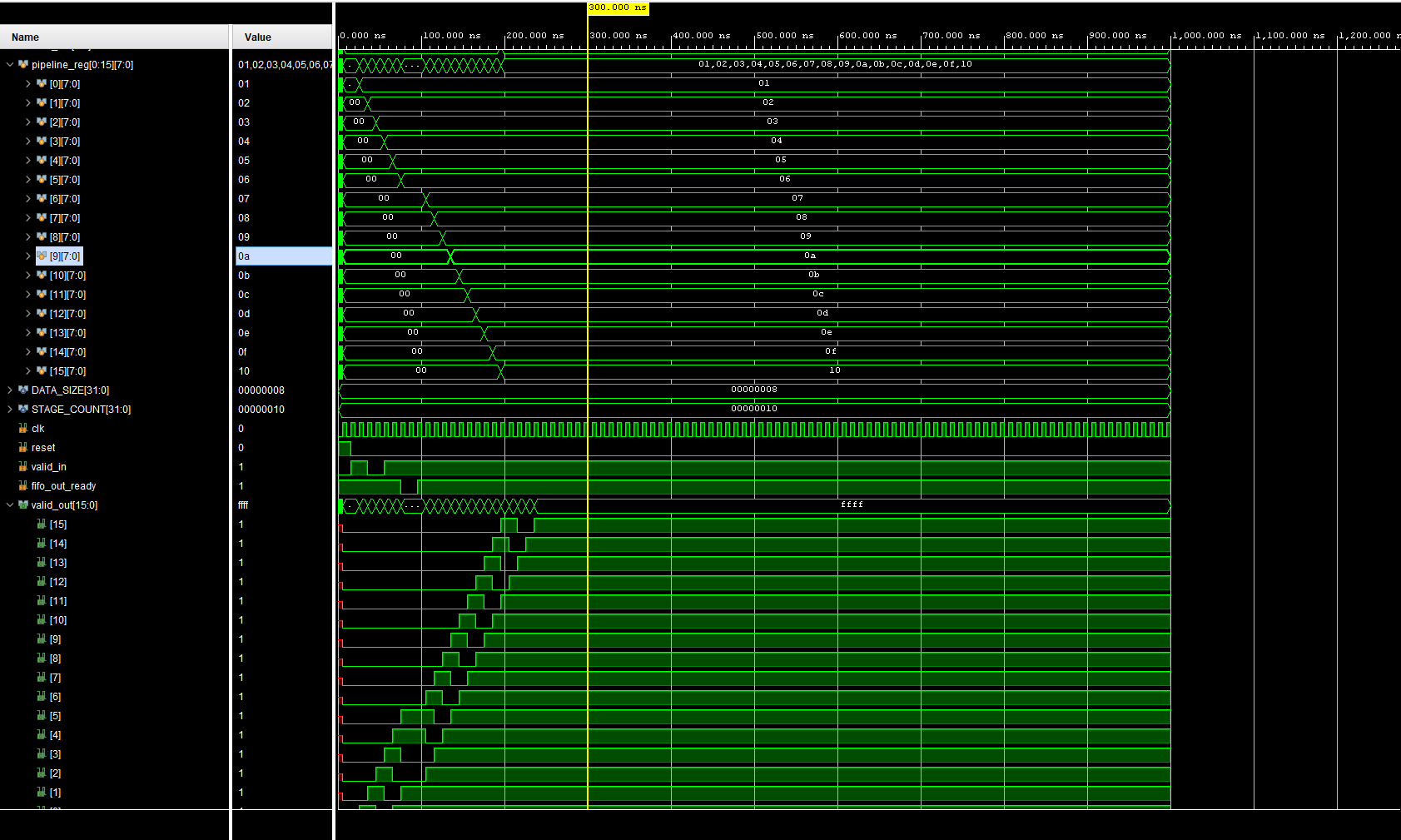


Рисунок 2.1 — Симуляция модуля конвейеризации.

На рисунках 2.2 – 2.3 представлена симуляция последовательного метода конвейеризации.

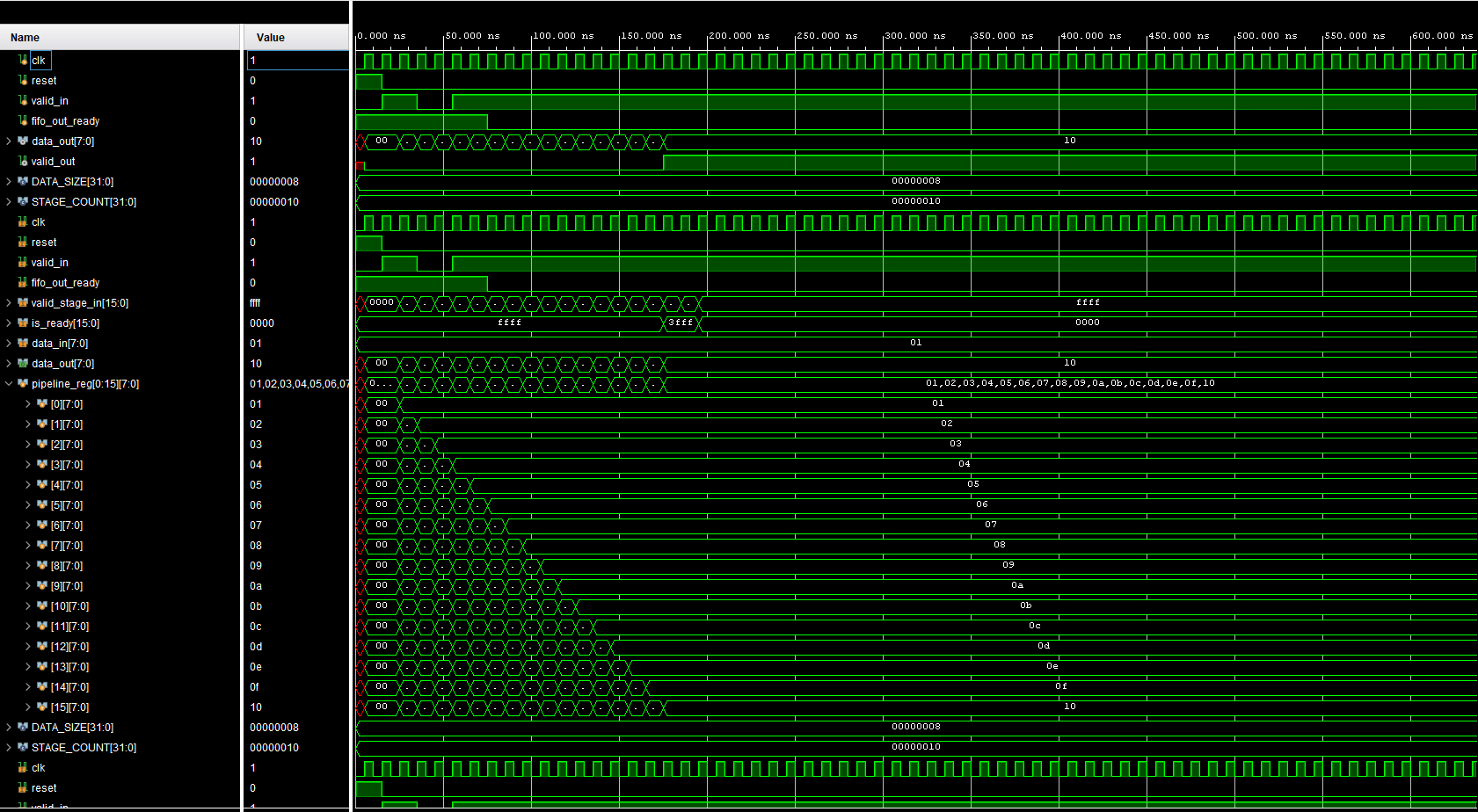


Рисунок 2.2 — Симуляция модуля конвейеризации последовательным методом

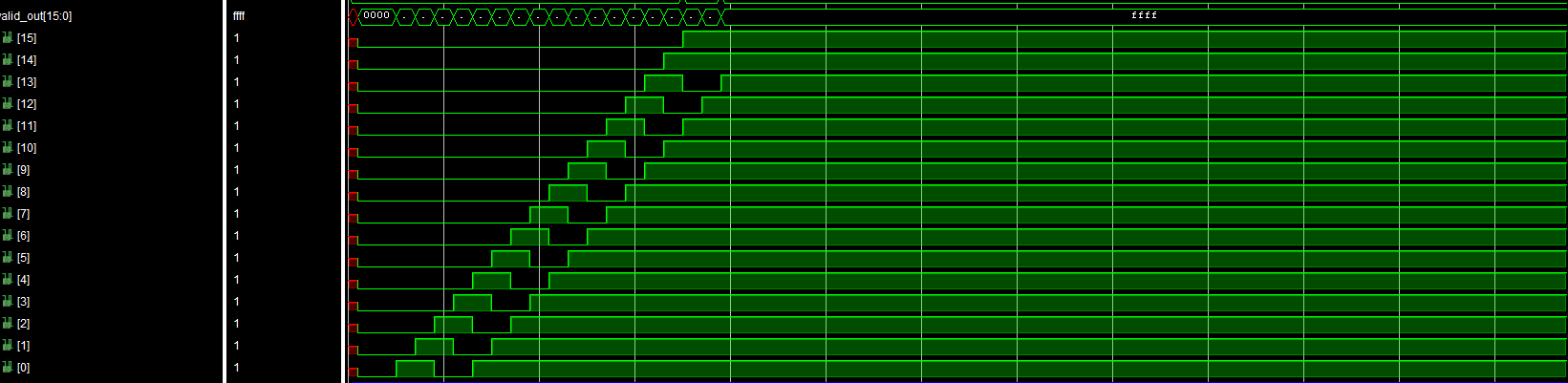


Рисунок 2.3 — Симуляция модуля конвейеризации последовательным методом

На рисунках 2.4 – 2.5 представлена симуляция параллельного метода конвейеризации



Рисунок 2.4 — Симуляция модуля конвейеризации параллельным методом

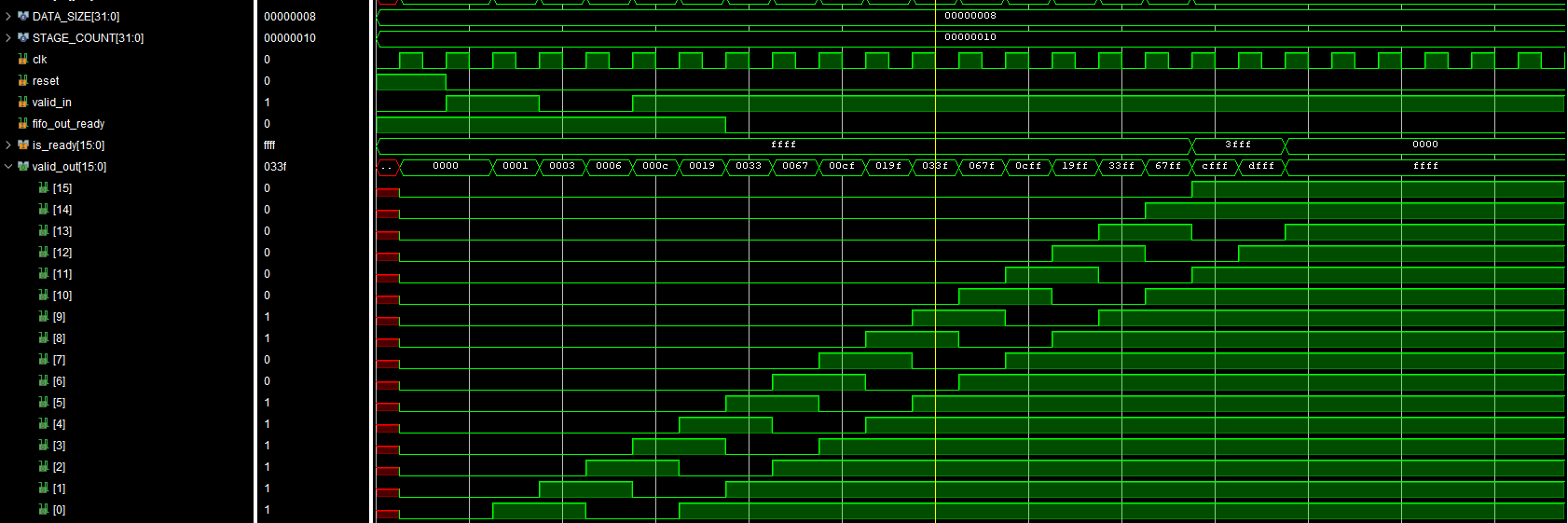


Рисунок 2.5 — Симуляция модуля конвейеризации параллельным методом

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.