|  |
| --- |
|  |
| 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 № 5**

«Организация буферов FIFO»

по дисциплине

«Схемотехника устройств компьютерных систем»

|  |  |
| --- | --- |
| Выполнил студент группы  ИВБО-11-23 | Туктаров Т.А. |
| Принял преподаватель кафедры ВТ | Дуксин Н.А. |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_2025 г. |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2025 г. |

Москва 2025

АННОТАЦИЯ

Данная работа включает в себя 4 рисунков, 4 листинга и 1 таблицу. Количество страниц в работе — 17.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc164890992)

[1 ПРОЕТИРОВАНИЕ И ТЕСТИРОВАНИЕ БУФЕРА FIFO, РАБОТАЮЩЕГО ПО ОДНОМУ ТАКТОВОМУ СИГНАЛУ 5](#_Toc164890993)

[1.1 Создание модуля 5](#_Toc164890994)

[1.2 Создание тестов и верификация модуля 7](#_Toc164890995)

[2 ПРОЕТИРОВАНИЕ И ТЕСТИРОВАНИЕ БУФЕРА FIFO, РАБОТАЮЩЕГО ПО ДВУМ ТАКТОВЫМ СИГНАЛАМ 10](#_Toc164890996)

[2.1 Создание модуля 10](#_Toc164890997)

[2.2 Создание тестов и верификация модуля 13](#_Toc164890998)

[ЗАКЛЮЧЕНИЕ 16](#_Toc164890999)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc164891000)

Введение

В практической работе рассматриваются вопросы построения и применения буферов типа FIFO при организации многокомпонентных систем. Целью работы является построение с использованием языка Verilog буферов FIFO, работающих по одному и по двум синхросигналам, а также тестирование полученной схемы

# 1 ПРОЕТИРОВАНИЕ И ТЕСТИРОВАНИЕ БУФЕРА FIFO, РАБОТАЮЩЕГО ПО ОДНОМУ ТАКТОВОМУ СИГНАЛУ

## 1.1 Создание модуля

Для начала создадим модуль FIFO с одним синхросигналом.

Код модуля представлен в Листинге 1.1.

Листинг 1.1 — Код модуля на языке Verilog для буфера FIFO

|  |
| --- |
| module fifo\_one\_clock#(  MEM\_SIZE = 6,  DATA\_SIZE = 4,  localparam ADDR\_SIZE = $clog2(MEM\_SIZE)  )  (  input [DATA\_SIZE - 1 : 0] data\_in,  input clk, read\_mode, write\_mode, enable, reset,    output reg [DATA\_SIZE - 1 : 0] data\_out,  output reg full, empty, valid  );  reg [DATA\_SIZE - 1 : 0] mem [0 : MEM\_SIZE - 1];  reg next\_full, next\_empty;  reg [ADDR\_SIZE - 1 : 0] write\_pointer, read\_pointer;  reg [ADDR\_SIZE - 1 : 0] next\_write\_pointer, next\_read\_pointer;  reg [ADDR\_SIZE - 1 : 0] i;  initial  begin  write\_pointer = {ADDR\_SIZE{1'b0}};  read\_pointer = {ADDR\_SIZE{1'b0}};    next\_write\_pointer = {ADDR\_SIZE{1'b0}};  next\_read\_pointer = {ADDR\_SIZE{1'b0}};    full = 1'b0;  next\_full = 1'b0;    empty = 1'b1;  next\_empty = 1'b1;    valid = 1'b0;    data\_out = {DATA\_SIZE{1'b0}};    for (i = 0; i < MEM\_SIZE; i = i + 1)  mem[i] = {DATA\_SIZE{1'b0}};  end |

Продолжение листинга 1.1

|  |
| --- |
| // Операция записи  always @(posedge clk)  if ( enable && write\_mode && !full )  mem[write\_pointer] <= data\_in;    // Операция чтения в буфер  always @(posedge clk)  begin  if ( enable && read\_mode && !empty )  begin  data\_out <= mem[read\_pointer];  valid <= 1;  end  else  valid <= 0;  end  // Установка значений для указателей  always@(posedge clk)  if (reset)  begin  write\_pointer <= {ADDR\_SIZE{1'b0}};  read\_pointer <= {ADDR\_SIZE{1'b0}};  next\_write\_pointer = {ADDR\_SIZE{1'b0}};  next\_read\_pointer = {ADDR\_SIZE{1'b0}};    full <= 1'b0;  empty <= 1'b1;  end  else if (enable)  begin  write\_pointer <= next\_write\_pointer;  read\_pointer <= next\_read\_pointer;    full <= next\_full;  empty <= next\_empty;  end  function [ADDR\_SIZE-1:0] next (input [ADDR\_SIZE-1:0] pointer);  if (pointer == MEM\_SIZE - 1)  next = {ADDR\_SIZE{1'b0}};  else  next = pointer + 1;  endfunction  localparam NONE = 0, READ = 1, WRITE = 2, READ\_AND\_WRITE = 3;  reg [1:0] op;  //  always @\*  begin  case({write\_mode, read\_mode})  2'b01:  op <= !empty ? READ : NONE;  2'b10:  op <= !full ? WRITE : NONE;  2'b11:  case({full, empty})  2'b10: op <= READ; |

Продолжение листинга 1.1

|  |
| --- |
| 2'b01: op <= WRITE;  default: op <= READ\_AND\_WRITE;  endcase  default:  op <= NONE;  endcase    case(op)  NONE:  begin  next\_write\_pointer <= write\_pointer;  next\_read\_pointer <= read\_pointer;    next\_full <= full;  next\_empty <= empty;  end    READ:  begin  next\_write\_pointer <= write\_pointer;  next\_full <= 0;  next\_empty <= next\_read\_pointer == write\_pointer;  next\_read\_pointer <= next(read\_pointer);  end    WRITE:  begin  next\_read\_pointer <= read\_pointer;  next\_full <= next\_write\_pointer == read\_pointer;  next\_empty <= 0;  next\_write\_pointer <= next(write\_pointer);  end    READ\_AND\_WRITE:  begin  next\_empty <= empty;  next\_full <= full;  next\_read\_pointer <= next(read\_pointer);  next\_write\_pointer <= next(write\_pointer);  end    endcase    end  endmodule |

## 1.2 Создание тестов и верификация модуля

Верификационное окружение для проведения тестов конечного автомата представлено представлен модулем «test». Тестовый модуль проводит симуляцию подачи значений и считывания данных из модуля. Значения «to\_input» подаются на вход и «data\_out» является выходной шиной.[2]

Код тестового модуля представлен в Листинге 1.2.

Листинг 1.2 — Реализация тестового модуля буфера FIFO

|  |
| --- |
| `timescale 1ns / 1ps  module test;  reg clk;  reg read;  reg write;  reg[3:0] to\_input;  initial clk = 0;  always #5 clk <= ~clk;  always #10 to\_input = to\_input + 1;  wire [3:0] data\_out;  wire full, empty, valid;  fifo\_one\_clock uut  (  .data\_in(to\_input),  .clk(clk),  .read\_mode(read),  .write\_mode(write),  .enable(1),  .reset(0),    .data\_out(data\_out),  .full(full),  .empty(empty),  .valid(valid)    );  initial  begin  to\_input = 1;  read = 0;  write = 1;  #40  read = 1;  write = 0;  #40  read = 1;  write = 1;  #80  read = 1;  write = 0;  #20  read = 0;  write = 1;  #40  read = 1;  write = 0;  #260    $stop;  end |

Результат тестирования представлен на Рисунке 1.1.

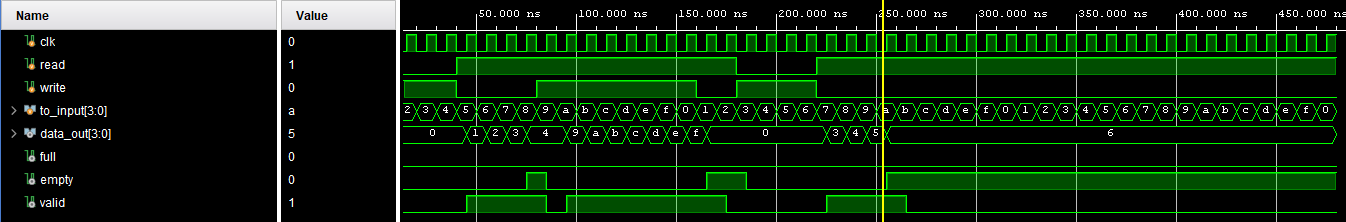


Рисунок 1.1 — Результат тестирования модуля буфера FIFO

# 2 ПРОЕТИРОВАНИЕ И ТЕСТИРОВАНИЕ БУФЕРА FIFO, РАБОТАЮЩЕГО ПО ДВУМ ТАКТОВЫМ СИГНАЛАМ

## 2.1 Создание модуля

Название модуля – «test». Модуль имеет практически аналогичную с «fifo\_one\_clock» реализацию. Далее будут рассмотрены только новые и отличительные моменты данного модуля по сравнению с «fifo\_one\_clock».

Был убран «clk» и добавлены входы «clk\_read» и «clk\_write», для соответствующих синхронных входов.

Операция записи и чтения в данном модуле происходит по переднему фронту синхросигналов «clk\_write» и «clk\_read» соответственно, а также сбрасывание указателей происходит в тех же блоках «always».

Операции сброса full и перехода в следующее состояние «full» и «empty» теперь происходит по переднему фронту синхросигналов или «clk\_read» или «clk\_write».

Внутри блока «case» работающему по регистру «op» были изменены условия установки значения для будущих указателей. Из «NONE», «READ» и «WRITE» были убраны изменения указателей, не относящихся к текущей операции. Так же во всех состояниях «op» были изменены условия установки «next\_full» и «next\_empty» на одинаковые логические выражения проверки, описанные в прошлом модуле.

Код модуля представлен в Листинге 2.1.

Листинг 2.1 — Код модуля на языке Verilog для буфера FIFO

|  |
| --- |
| module fifo\_two\_clock#(  MEM\_SIZE = 6,  DATA\_SIZE = 4,  localparam ADDR\_SIZE = $clog2(MEM\_SIZE)  )  (  input [DATA\_SIZE - 1 : 0] data\_in,  input clk\_write, clk\_read, read\_mode, write\_mode, enable, reset,    output reg [DATA\_SIZE - 1 : 0] data\_out,  output reg full, empty, valid  );  reg [DATA\_SIZE - 1 : 0] mem [0 : MEM\_SIZE - 1];  reg next\_full, next\_empty;  reg [ADDR\_SIZE - 1 : 0] write\_pointer, read\_pointer;  reg [ADDR\_SIZE - 1 : 0] next\_write\_pointer, next\_read\_pointer;  reg [ADDR\_SIZE - 1 : 0] i;  initial  begin  write\_pointer = {ADDR\_SIZE{1'b0}};  read\_pointer = {ADDR\_SIZE{1'b0}};    next\_write\_pointer = {ADDR\_SIZE{1'b0}} + 1;  next\_read\_pointer = {ADDR\_SIZE{1'b0}} + 1;    full = 1'b0;  next\_full = 1'b0;    empty = 1'b1;  next\_empty = 1'b1;    valid = 1'b0;    data\_out = {DATA\_SIZE{1'b0}};    for (i = 0; i < MEM\_SIZE; i = i + 1)  mem[i] = {DATA\_SIZE{1'b0}};  end  // Операция записи  always @(posedge clk\_write)  begin  if ( reset )  write\_pointer <= {ADDR\_SIZE{1'b0}};  else  begin  if ( enable && write\_mode && !full )  begin  mem[write\_pointer] <= data\_in;  write\_pointer <= next\_write\_pointer;  end  end  end    // Операция чтения в буфер  always @(posedge clk\_read)  begin  if ( reset ) |

Продолжение листинга 2.1

|  |
| --- |
| begin  read\_pointer <= {ADDR\_SIZE{1'b0}};  end  else  begin  if ( enable && read\_mode && !empty )  begin  read\_pointer <= next\_read\_pointer;  data\_out <= mem[read\_pointer];  valid <= 1;  end  else  valid <= 0;  end  end    always @(posedge clk\_read, posedge clk\_write)  begin  if ( reset )  begin  full <= 1'b0;  empty <= 1'b1;  end  else if ( enable )  begin  full = next\_full;  empty = next\_empty;  end  end    function [ADDR\_SIZE-1:0] next (input [ADDR\_SIZE-1:0] pointer);  begin  if (pointer == MEM\_SIZE - 1)  next = {ADDR\_SIZE{1'b0}};  else  next = pointer + 1;  end  endfunction  localparam NONE = 0, READ = 1, WRITE = 2, READ\_AND\_WRITE = 3;  reg [1:0] op;  //  always @\*  begin  case({write\_mode, read\_mode})  2'b01:  op <= !empty ? READ : NONE;  2'b10:  op <= !full ? WRITE : NONE;  2'b11:  case({full, empty})  2'b10: op <= READ;  2'b01: op <= WRITE;  default: op <= READ\_AND\_WRITE;  endcase  default:  op <= NONE;  endcase    case(op)  NONE: |

Продолжение листинга 2.1

|  |
| --- |
| begin  next\_full <= full;  next\_empty <= empty;  end    READ:  begin  next\_full <= next\_write\_pointer == read\_pointer;  next\_empty <= next\_read\_pointer == write\_pointer;  next\_read\_pointer <= next(read\_pointer);  end    WRITE:  begin  next\_full <= next\_write\_pointer == read\_pointer;  next\_empty <= next\_read\_pointer == write\_pointer;  next\_write\_pointer <= next(write\_pointer);  end    READ\_AND\_WRITE:  begin  next\_empty <= next\_read\_pointer == write\_pointer;  next\_full <= next\_write\_pointer == read\_pointer;    next\_read\_pointer <= next(read\_pointer);  next\_write\_pointer <= next(write\_pointer);  end    endcase    end  endmodule |

## 2.2 Создание тестов и верификация модуля

Верификационное окружение для проведения тестов конечного автомата представлено представлен модулем «test». Тестовый модуль проводит симуляцию подачи значений и считывания данных из модуля на разных наборах синхросигналов. Значения «to\_input» подаются на вход и «data\_out» является выходной шиной.

Для выявления ошибок были последовательно протестированы следующие случайные наборы тактовых сигналов, представленные в Таблице 2.1. Результаты тестирования представлены в Рисунках 2.1 – 2.5

Таблица 2.1 — Таблица протестированных значений

|  |  |
| --- | --- |
| clk\_write | clk\_read |
| 5 | 5 |
| 5 | 20 |
| 20 | 5 |

Код тестового модуля представлен в Листинге 2.2.

Листинг 2.2 — Реализация тестового модуля буфера FIFO

|  |
| --- |
| `timescale 1ns / 1ps  module test\_two;  reg clk\_write;  reg clk\_read;  reg read;  reg write;  reg[3:0] to\_input;  initial clk\_write = 0;  initial clk\_read = 0;  integer clk\_write\_timming = 5;  always #clk\_write\_timming clk\_write <= ~clk\_write;  integer clk\_read\_timming = 5;  always #clk\_read\_timming clk\_read <= ~clk\_read;  wire [3:0] data\_out;  wire full, empty, valid;  fifo\_two\_clock uut  (  .data\_in(to\_input),  .clk\_write(clk\_write),  .clk\_read(clk\_read),  .read\_mode(read),  .write\_mode(write),  .enable(1),  .reset(0),    .data\_out(data\_out),  .full(full),  .empty(empty),  .valid(valid)    );  always #(clk\_write\_timming\*2)  if(!full && write)  to\_input = to\_input + 1;  initial  begin  to\_input = 1; |

Продолжение листинга 2.2

|  |
| --- |
| read = 0;  write = 1;  #100  read = 1;  write = 1;  #360  read = 1;  write = 0;  #300    $stop;  end  endmodule |

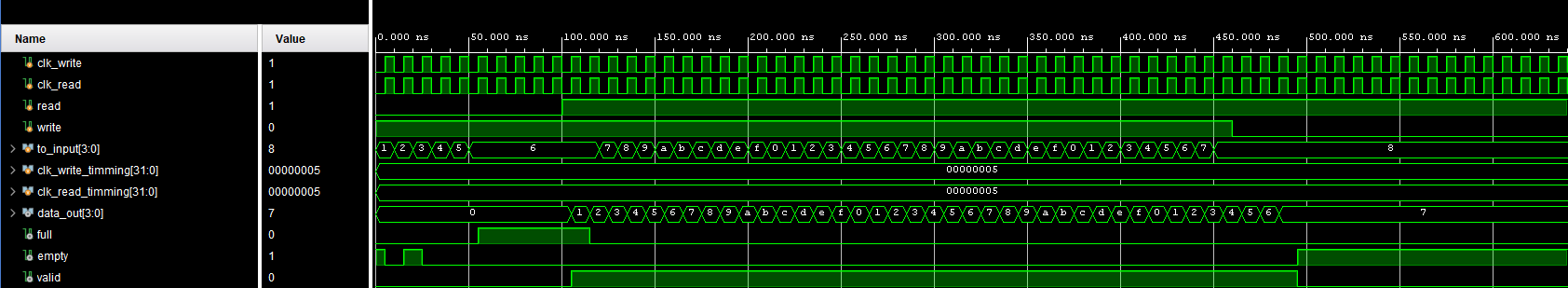


Рисунок 2.1 — Результат тестирования модуля буфера FIFO с одинаковыми синхросигналами

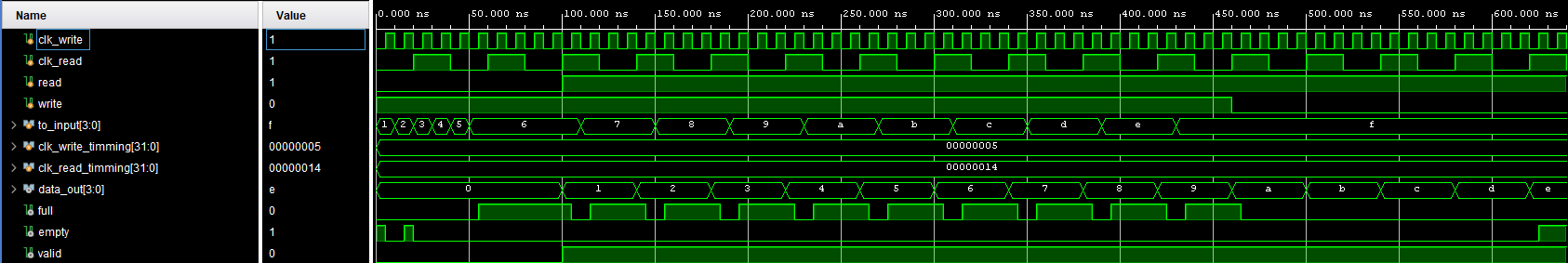


Рисунок 2.2 — Результат тестирования модуля буфера FIFO с синхросигналами 5 и 20

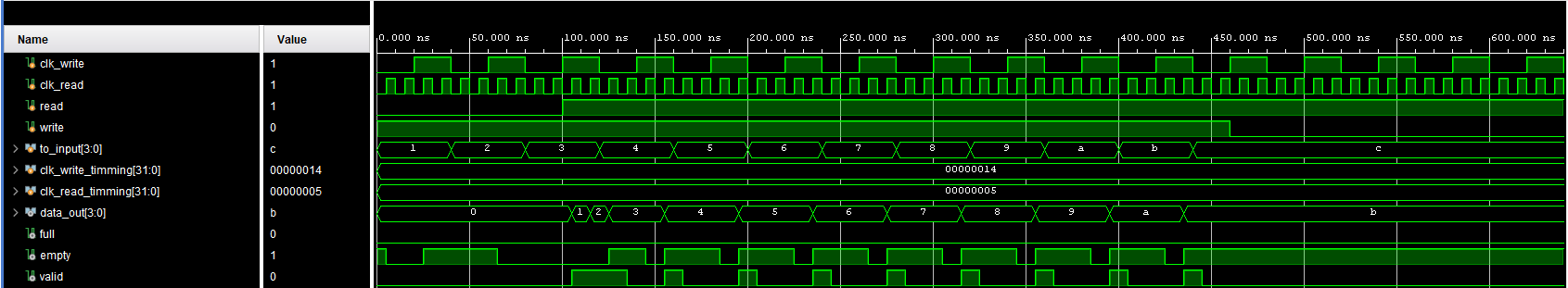


Рисунок 2.4 — Результат тестирования модуля буфера FIFO с синхросигналами 20 и 5

Все последовательно записанные значения были так же последовательно получены. Результаты тестирования подтверждают корректность работы написанного модуля.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены принципы, а также получены навыки построения буферов типа «FIFO». Рассмотрены и реализованы варианты построения буфера FIFO, тактируемых одним синхросигналом и двумя синхросигналами.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методические указания по ПР № 4 — URL: https://online-edu.mirea.ru/mod/resource/view.php?id=405132.

2. Смирнов С.С. Информатика [Электронный ресурс]: Методические указания по выполнению практических и лабораторных работ / С.С. Смирнов — М., МИРЭА — Российский технологический университет, 2018. — 1 электрон. опт. диск (CD-ROM).

3. Тарасов И.Е. ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования. — М.: Горячая линия — Телеком, 2021. — 538 с.: ил.

4. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие / Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).

1. 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.