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 Стецюк В.В.

ИВБО-08-22

Принял ассистент кафедры ВТ Дуксин Н.А.

Лабораторная работа выполнена «__»_____2024 г.

«Зачтено» «__»_____2024 г.

Москва 2024

АННОТАЦИЯ

Данная работа включает в себя 6 рисунков, 4 листинга и 1 таблица. Количество страниц в работе — 20.

СОДЕРЖАНИЕ

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

Название модуля – «fifo_one_clock». Далее в скобках записываются параметры «MEM_SIZE», «DATA_SIZE» и локальный параметр «ADDR_SIZE», отвечающие за размер хранимых данных, размер одного хранимого значения в битах и значение размера шин указателей на хранилища «MEM_SIZE» элементов. Затем в скобках записываются входные и выходные значения. «data_in» является входом данных, clk – входом синхроимпульса, «read_mode» – вход инициализирующий запись, «write_mode» – вход инициализирующий чтение, «enable» – вход подтверждения работы, «reset» – вход сброса. Все входы являются однобитными, за исключением «data_in», который имеет шину размером «DATA_SIZE». Далее в скобках идут выходы «data_out» с шиной размера «DATA_SIZE», отвечающий за вывод данных, и однобитные регистры «full», «empty» и «valid», отвечающие за заполненность памяти модуля, пустоту памяти модуля и корректность выходных данных соответственно.

Далее объявляется ряд регистров. «mem» – буфер памяти, хранящий записываемые значения, имеет размер «MEM_SIZE», в котором каждый элемент имеет размер «DATA_SIZE». «next_full» и «next_empty» регистры, которые отвечают за следующее состояния регистров «full» и «empty». Далее идут регистры «write_pointer» и «read_pointer», которые являются указателями на текущий ячейку чтения и записи соответственно, а также регистр «i», который будет использоваться в качестве счётчика. Следующими после них идут их следующие состояния «next_write_pointer» и «next_read_pointer». Указатели имеют размер «ADDR_SIZE».

Затем следует блок инициализации автомата, в котором происходит привод автомата в начальные состояния.

Следующим идёт блок «always», работающий по переднему фронту синхроимпульса «clk». В данном блоке происходит операция записи информации со входа «data_in» в буфер «mem» при выполнении условия, что устройство включено, запись разрешена и буфер не полный.

Далее идёт блок «always», работающий по переднему фронту синхроимпульса «clk». В данном блоке происходит операция чтения из буфера «mem» на выходную шину «data_out» при выполнении условия, что устройство включено, запись чтение разрешено и буфер не пустой. Так же на выход «valid» будет подана единица. Иначе, если чтение невозможно, на «valid» будет подан ноль.

Следующим идёт блок «always», работающий по переднему фронту синхроимпульса «clk». В данном блоке происходит установка значений для указателей. В первую очередь проверяется подан ли сигнал на вход «reset», и, если подан, значение указателей и выходов «full» и «empty» сбрасывается. Иначе, при условии поданного сигнала на вход «enable» происходит присвоения указателям из следующих состояний.

Далее следует функция, которая принимает на вход указатель и возвращает следующее значение или возвращает нулевое значение.

С помощью ключевого слова «localparam» объявляются локальные параметры, упрощающее чтение кода. Были объявлены следующие параметры: «NONE» равный 0, «READ» равный 1, «WRITE» равный 2 и «READ_AND_WRITE» равный 3. Так же был объявлен двухбитный регистр «op», в который будет записываться один из вышеперечисленных параметров.

Следующий блок «always», в связи с одно процессный стилем написания, является чувствительным на каждую переменную входящую в блок. С помощью блока «case», на основании шины из «write_mode» и «read_mode», происходит установка одного из состояний в регистр «op». При наличии единицы только на входе «read_mode» будет выполнен переход к операции чтения и регистру «op»

присвоено значение параметра «READ» если на «empty» не единица, иначе будет «NONE».

При наличии единицы только на входе «write_mode» будет выполнен переход к операции записи и регистру «op» присвоено значение параметра «WRITE» если на «full» не единица, иначе будет «NONE». При наличии единиц на обоих входах будет произведён выбор с использованием ещё одного «case» по «full» и «empty». Если на «full» единица и на «empty» единица будет выполнена установка «READ», если, иначе если на «full» ноль, а на «empty» единица, то «WRITE». Если единица и на «full», и на «empty», то будет выполнена установка «READ_AND_WRITE».

Следующий блок «case» по значению регистра «op» определяет, какие значения будут установлены указателям. При «NONE» значения следующих указателей не изменятся, при «READ» значение «write_pointer» не изменится, а «next_read_pointer» будет установлено значение, следующее после текущего. Значение «full» будет обнулено, так как после чтения модуль не может быть пустым, а «empty» будет присвоен результат логического выражения, проверяющий, что указатель записи достигнет указателя чтения. При значении «op» равным «WRITE» будет выполнена обратная «READ» логика. При значении «op» равным «READ_AND_WRITE» указателям чтения и записи будут установлены их следующие значения, а значения регистров «full» и «empty» не будут изменены.

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];
```

Продолжение листинга 1.1

```
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

// 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;
```

Продолжение листинга 1.1

```
            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;
                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;
```

Продолжение листинга 1.1

```
                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.

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;
```

Продолжение листинга 1.2

```
    read = 0;
    write = 1;
    #20
    read = 1;
    write = 0;
    #20
    read = 1;
    write = 1;
    #100
    read = 1;
    write = 0;
    #40
    read = 0;
    write = 1;
    #40
    read = 1;
    write = 0;
    #260

    $stop;
end

endmodule
```

Результат тестирования представлен на Рисунке 1.1.

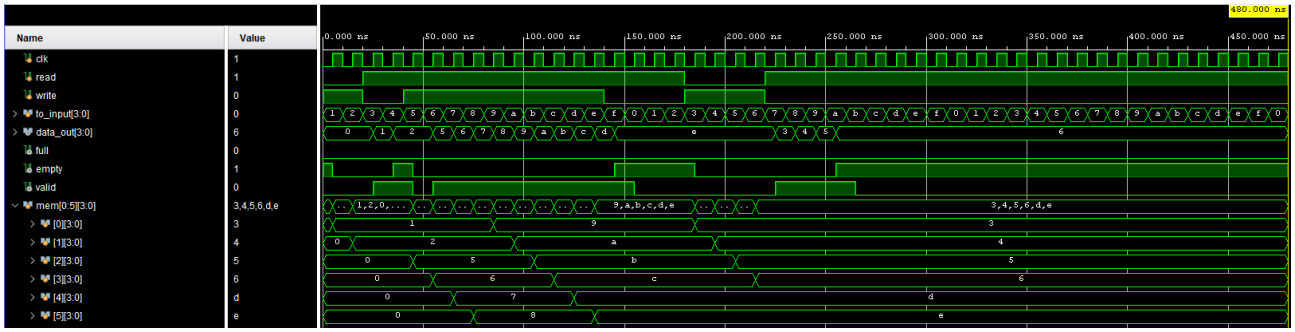

Рисунок 1.1 — Результат тестирования модуля буфера FIFO

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

2.1 Создание модуля

Название модуля – «fifo_two_clock». 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_two». 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
10	30
20	5
30	10

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
```

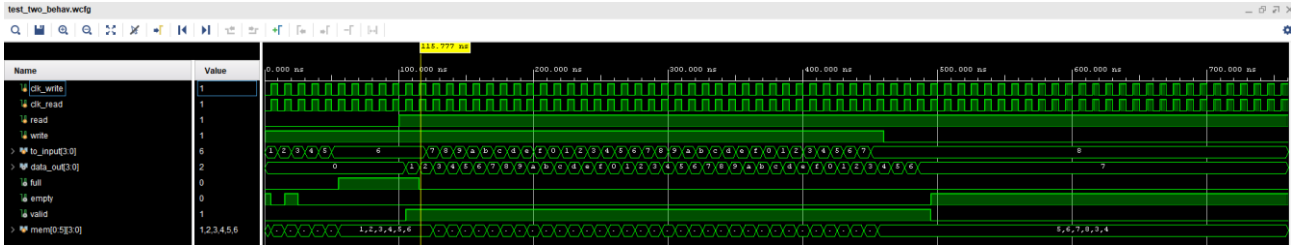

Рисунок 2.1 — Результат тестирования модуля буфера FIFO с одинаковыми синхросигналами

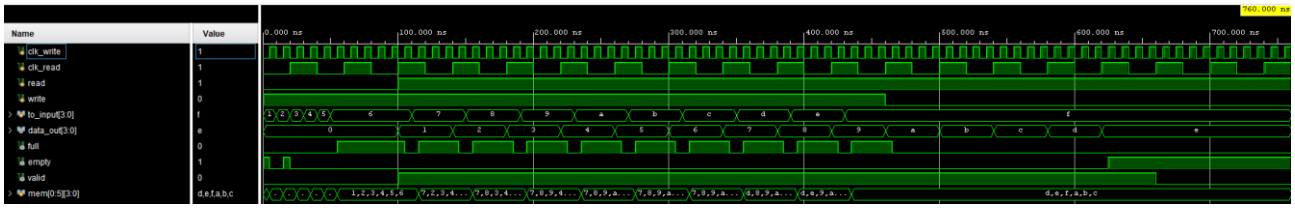

Рисунок 2.2 — Результат тестирования модуля буфера FIFO с синхросигналами 5 и 20

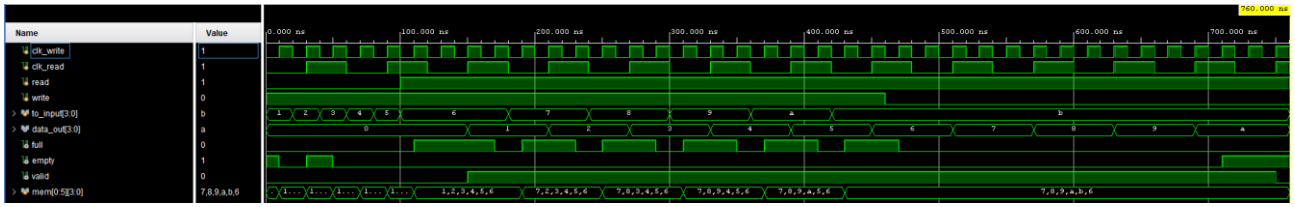

Рисунок 2.3 — Результат тестирования модуля буфера FIFO с синхросигналами 10 и 30

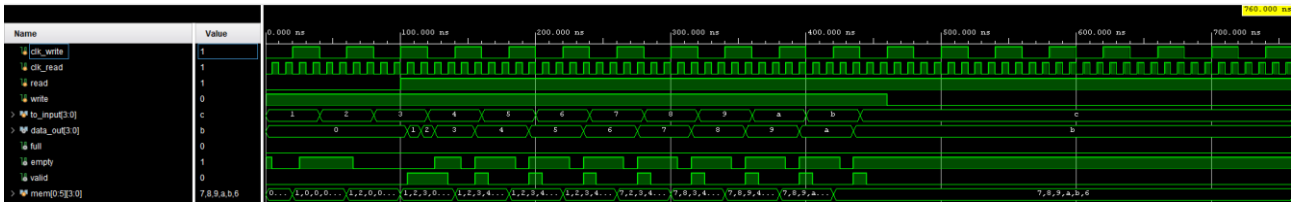

Рисунок 2.4 — Результат тестирования модуля буфера FIFO с синхросигналами 20 и 5

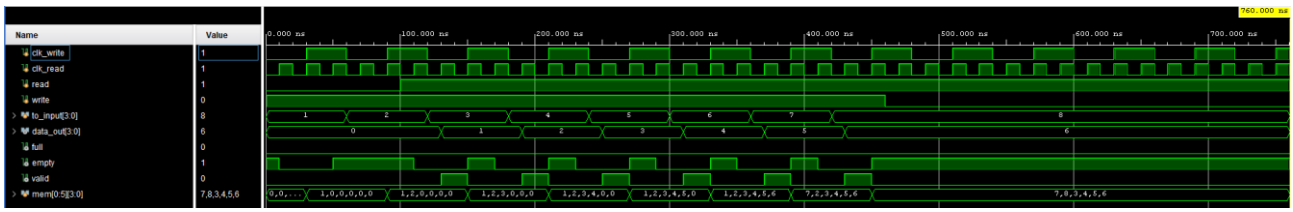

Рисунок 2.5 — Результат тестирования модуля буфера FIFO с синхросигналами 30 и 10

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 № 2 — URL: https://online-edu.mirea.ru/pluginfile.php?file=%2F1225653%2Fmod_assign%2Fintroattachment%2F0%2F%D0%9F%D1%80%D0%B5%D0%B7%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F_%D0%9F%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0_2.pptx&forcedownload=1 (Дата обращения: 12.03.2024).

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

4. 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).