Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия
Дисциплина «Функциональная схемотехника»

# Лабораторная работа №2

Вариант FIFO

Выполнили:
Комягин Д.А.
Чмурова М.В.
Кузенина В.Н.
*P3332*

Преподаватель:
Кустарёв П.В.

Санкт-Петербург
2024 г.

## Цель работы

Разработать цифровой сложно-функциональный блок, который обрабатывает и хранит данные в соответствии с требованиями, указанными в варианте задания. Разработка включает в себя следующие этапы:

1. Определение и согласование интерфейсов;
2. Выбор алгоритма и согласование микроархитектуры;
3. Описание устройства средствами HDL;
4. Функциональная верификация;
5. Прототипирование разработанного блока на FPGA.

## Задание

Очередь, работающая по принципу "first in, first out" (FIFO). Глубина очереди - 8 ячеек. Входные данные – целые 8 битные положительные числа

## Описание алгоритма работы устройства.

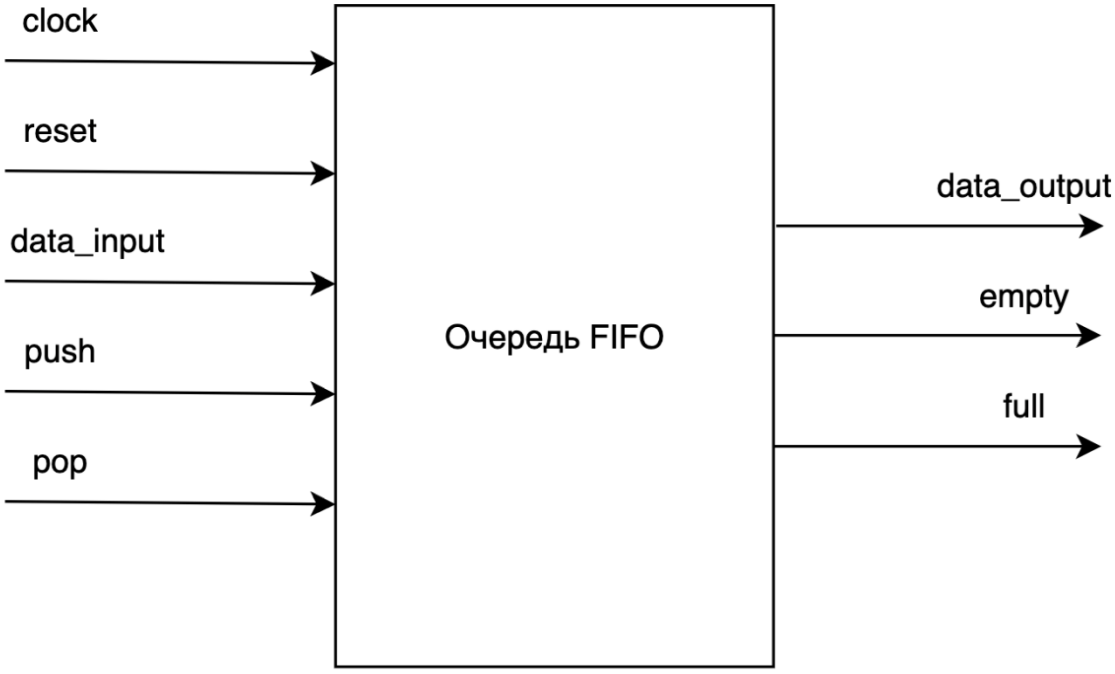


*Рисунок 1. Общий интерфейс FIFO*

- Входные сигналы:

clock – тактовый генератор

reset – сброс сигналов

data_input – входные данные, 8 бит

push – сигнал записи данных в память

pop – сигнал вывода данных из памяти

- Выходные сигналы:

data_output – выходные данные, 8 бит

empty – сигнал, означающий, что память пуста

full – сигнал, означающий, что память заполнена

**Описание работы:**

При запуске системы и подаче сигнала *push* число получается из *data_input* и записывается в память по адресу, на который указывает *write_pointer*, после чего *write_pointer* инкрементируется. При подаче сигнала *pop* число берется из той ячейки памяти, на которую указывает *read_pointer*, после чего *read_pointer* инкрементируется.

Указатели работают по принципу кругового буфера, после перехода значения 0 указателя инвертируется значение флага *write_flag* / *read_flag*, в зависимости от операции *push* / *pop* соответственно.

Значение *empty* устанавливается при равенстве указателей и равенстве флагов (*read_pointer* догнал *write_pointer*, соответственно буфер пустой)

Значение full устанавливается при равенстве указателей и неравенстве флагов (*write_pointer* догнал *read_pointer*, сделал на один круг больше, следовательно буфер полный)

## Микроархитектурная диаграмма

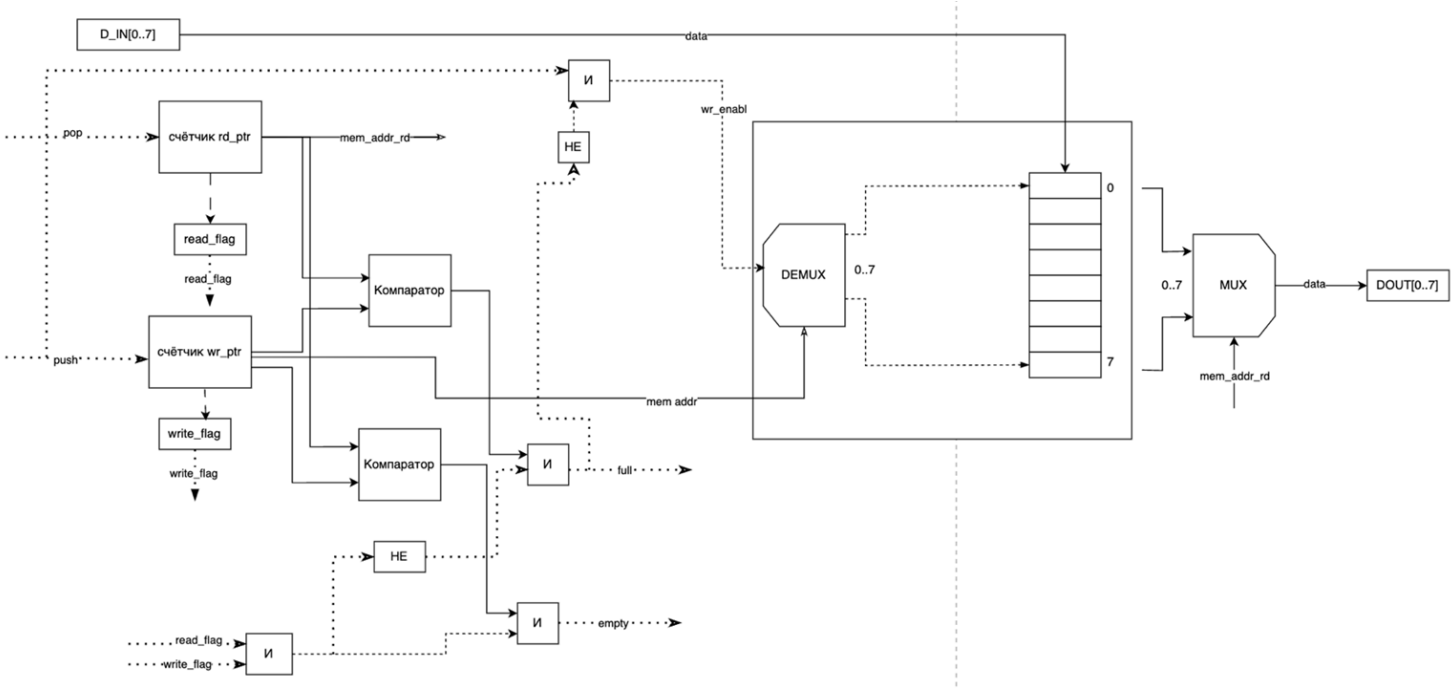


*Рисунок 2. Микоархитектура*

## Описание микроархитектурной диаграммы

На вход поступают два управляющих сигнала: **pop** и **push**. Эти сигналы управляют работой счетчиков **read_pointer** и **write_pointer**, которые, в свою очередь, определяют адреса чтения и записи в памяти FIFO.

**Логика выставления флагов:**

- **Флаг read_flag** устанавливается, когда счетчик **read_pointer** обнуляется. Аналогично, **флаг write_flag** устанавливается при обнулении счетчика **write_pointer**.
- **Сигнал full** (переполнение) выставляется, если значения счетчиков **read_pointer** и **write_pointer** равны, что определяется с помощью логического элемента — компаратора. Для окончательного формирования сигнала **full** используется логический элемент "И", который принимает на вход результат сравнения (сигнал "equal") и

инвертированное значение равенства регистров **read_flag** и **write_flag**.

- **Сигнал empty** (опустошение) формируется аналогично, но вместо инвертированного значения равенства флагов используется прямой результат их логической операции "И". Таким образом, на вход логического элемента "И" поступает сигнал **equal** от компаратора для счетчиков и результат сравнения регистров **read_flag** и **write_flag**.

Сигналы **full** и **empty** направляются на выход системы и служат индикаторами состояния FIFO.

**Логика записи данных:**

Данные, поступающие на вход регистра **D_IN**, записываются в память через демультиплексор. Демультиплексор использует значение счетчика **write_pointer** для выбора адреса записи в памяти. По сигналу **write_enable** данные защелкиваются в выбранной ячейке памяти.

**Логика чтения данных:**

Чтение данных из памяти осуществляется через мультиплексор, входами которого являются все ячейки памяти. Управляющий сигнал от счетчика **read_pointer** выбирает нужное значение, которое передается на выход в регистр **D_OUT**.

Эта архитектура обеспечивает корректное управление чтением и записью, а также контроль состояния FIFO, минимизируя риск переполнения или некорректного чтения.

# Временная диаграмма

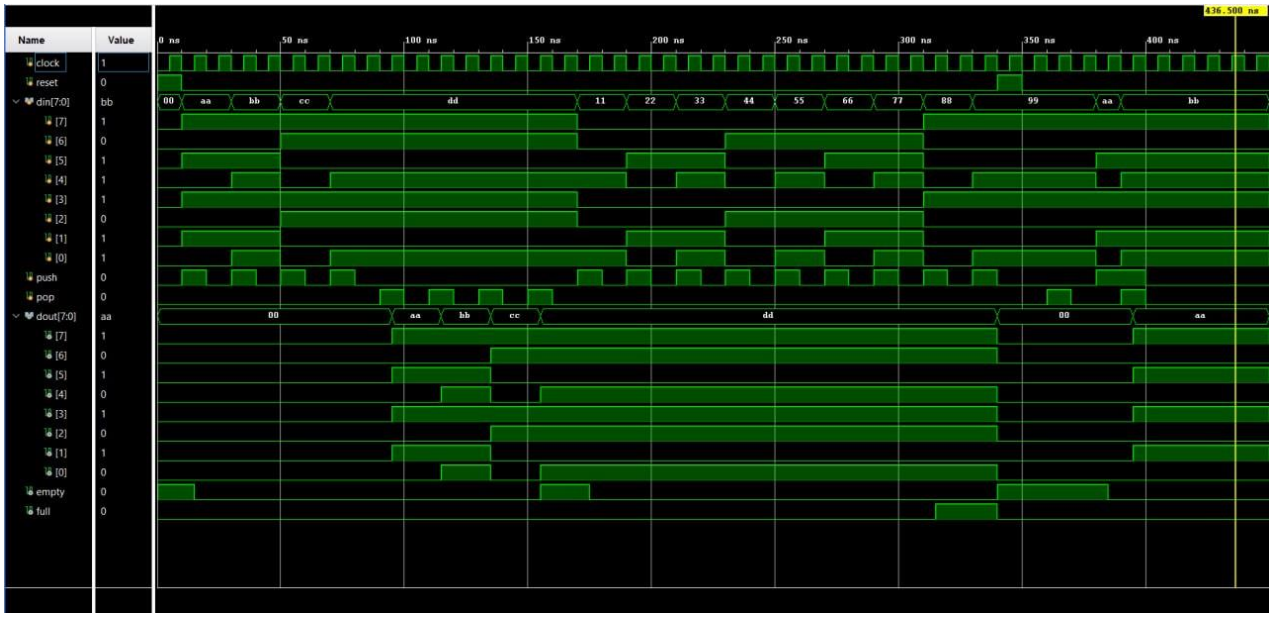


*Рисунок 3. Временная диаграмма*

# Код разработанного модуля

```
`timescale 1ns / 1ps

module fifo
#(
    parameter DATA_WIDTH = 8,
    parameter PTR_WIDTH = 3
)
(
    input clock,
    input reset,
    input [DATA_WIDTH-1:0] din,
    input push,
    input pop,
    output reg [DATA_WIDTH-1:0] dout,
    output wire empty,
    output wire full
);
    reg [PTR_WIDTH-1:0] read_pointer;
    reg [PTR_WIDTH-1:0] write_pointer;
    reg read_flag, write_flag;
    reg [DATA_WIDTH-1:0] mem [0:(2**PTR_WIDTH)-1];

    assign empty = (read_pointer == write_pointer) && (read_flag == write_flag);
    assign full = (read_pointer == write_pointer) && (read_flag != write_flag);


    always @(posedge clock or posedge reset) begin
        if (reset) begin
            read_pointer <= 0;
            write_pointer <= 0;
            read_flag <= 0;
            write_flag <= 0;
            dout <= 0;
        end else begin
            if (push && pop) begin
```

```
                dout <= mem[read_pointer];
                read_pointer <= (read_pointer + 1) % (2**PTR_WIDTH);
                mem[write_pointer] <= din;
                write_pointer <= (write_pointer + 1) % (2**PTR_WIDTH);
            end else if (push && !full) begin
                mem[write_pointer] <= din;
                write_pointer <= (write_pointer + 1) % (2**PTR_WIDTH);
                if (write_pointer == 0) write_flag <= ~write_flag;
            end else if (pop && !empty) begin
                dout <= mem[read_pointer];
                read_pointer <= (read_pointer + 1) % (2**PTR_WIDTH);
                if (read_pointer == 0) read_flag <= ~read_flag;
            end
        end
    end
endmodule
```

## Описание процесса функциональной верификации

В рамках разработки набора тестов, покрывающих функциональные возможности блока, были разработаны тесты для проверки:

- Корректной установки флагов после сброса системы (reset)
- Записи данных в память (push)
- Вывода данных из памяти (pop)
- Корректной установки флагов после опустошения системы (empty)
- Корректной установки флагов после переполнения памяти (full)
- Попытки записи в полную память
- Попытки чтения из пустой памяти
- Корректной обработки одновременных сигналов pop и push

Таким образом, тесты позволяют покрыть все возможные тестовые случаи для анализа корректности функционирования разработанной микроархитектуры.

## Код для проведения тестирования

```
`timescale 1ns / 1ps

module fifo_golden_tb;
    reg clock;
    reg reset;
    reg [7:0] din;
    reg push;
    reg pop;
    wire [7:0] dout;
    wire empty;
```

```
    wire full;

    fifo #(.DATA_WIDTH(8), .PTR_WIDTH(3)) uut (
        .clock(clock),
        .reset(reset),
        .din(din),
        .push(push),
        .pop(pop),
        .dout(dout),
        .empty(empty),
        .full(full)
    );

    always #5 clock = ~clock;
    initial begin
        clock = 0;
        reset = 1;
        push = 0;
        pop = 0;
        din = 0;
        // 1. Сброс FIFO
        #10 reset = 0;
        $display("TEST 1: Reset");
        $display("Checking empty and full flags...");
        if (empty !== 1 || full !== 0)
            $display("FAILED: FIFO is not empty after reset!");
        else
            $display("PASSED");
        // 2. Запись в FIFO
        $display("TEST 2: Push data into FIFO");
        din = 8'hAA; push = 1; #10; push = 0; #10;
        din = 8'hBB; push = 1; #10; push = 0; #10;
        din = 8'hCC; push = 1; #10; push = 0; #10;
        din = 8'hDD; push = 1; #10; push = 0; #10;
        if (full !== 0)
            $display("FAILED: FIFO should not be full after 4 pushes!");
        else
            $display("PASSED");
        // 3. Чтение из FIFO
        $display("TEST 3: Pop data from FIFO");
        pop = 1; #10; pop = 0; #10;
        if (dout !== 8'hAA) $display("FAILED: Expected 0xAA, got 0x%h", dout);
        pop = 1; #10; pop = 0; #10;
        if (dout !== 8'hBB) $display("FAILED: Expected 0xBB, got 0x%h", dout);
        pop = 1; #10; pop = 0; #10;
        if (dout !== 8'hCC) $display("FAILED: Expected 0xCC, got 0x%h", dout);
        pop = 1; #10; pop = 0; #10;
        if (empty !== 1)
            $display("FAILED: FIFO should be empty after 4 pops!");
        else
            $display("PASSED");
        // 4. Проверка состояния "empty"
        $display("TEST 4: Empty signal");
        if (empty !== 1) $display("FAILED: Empty signal should be high!");
        else $display("PASSED");
        // 5. Переполнение FIFO
        $display("TEST 5: Full signal");
        din = 8'h11; push = 1; #10; push = 0; #10;
        din = 8'h22; push = 1; #10; push = 0; #10;
        din = 8'h33; push = 1; #10; push = 0; #10;
        din = 8'h44; push = 1; #10; push = 0; #10;
        din = 8'h55; push = 1; #10; push = 0; #10;
        din = 8'h66; push = 1; #10; push = 0; #10;
        din = 8'h77; push = 1; #10; push = 0; #10;
        din = 8'h88; push = 1; #10; push = 0; #10;
        if (full !== 1 && empty !== 0)
            $display("FAILED: Full signal should be high!");
        else
```

```
            $display("PASSED");
        // 6. Попытка записи в полный FIFO
        $display("TEST 6: Attempt to push into full FIFO");
        din = 8'h99; push = 1; #10; push = 0;
        if (full !== 1 && empty !== 0)
            $display("FAILED: FIFO should remain full!");
        else
            $display("PASSED");
        // 7. Попытка чтения из пустого FIFO
        $display("TEST 7: Read from empty FIFO");
        reset = 1; #10 reset = 0; #10;
        pop = 1; #10; pop = 0; #10;
        if (empty !== 1)
            $display("FAILED: FIFO should be empty!");
        else
            $display("PASSED");
        // 8. Одновременный push и pop
        $display("TEST 8: Simultaneous Push and Pop");
        din = 8'hAA; push = 1; #10;
        din = 8'hBB; pop = 1; #10;
        pop = 0; push = 0;
        if (dout !== 8'hAA)
            $display("FAILED: Simultaneous push/pop failed!");
        else
            $display("PASSED");
        // Завершение
        #50 $finish;
    end
endmodule
```

## Вывод

Достоинства решения заключаются, во-первых, в простоте реализации, архитектура основывается на базовых компонентах(счётчики, мультиплексор, демультиплексор), что делает реализацию относительно простой.
Во-вторых, контроль переполнения и опустошения памяти, наличие флагов full и empty гарантирует корректную работу системы.
В-третьих, заложена корректная обработка одновременного нажатия push и pop, что предотвращает неопределенное поведение системы

Вместо использования логики формирования отдельных сигналов full и empty  через 2 дополнительных регистра можно ввести 3ий счётчик, который отслеживает количество элементов в очереди

Мы решили не вводить третий счётчик, так как он предполагает использование дополнительных регистров для хранения значения и логики его обновления. Также использования третьего счётчика требует синхронизации с существующими. Значения регистров зависят и вычисляются динамически от состояния уже существующих счётчиков.

Проведенное тестирование подтвердило работоспособность устройства на основании следующих факторов:

1. Были протестированы критические и граничные состояния системы, включая сценарии переполнения памяти, полного опустошения и одновременной активации сигналов **push** и **pop**.
2. Также проведены проверки стандартных случаев работы, таких как добавление одного числа с последующим его чтением, а также последовательное добавление нескольких чисел и их последующее чтение из памяти.

Во всех тестовых сценариях устройство продемонстрировало корректное функционирование, что подтверждает его надежность в рамках заявленной функциональности.

В ходе выполнения лабораторной работы были выявлены следующие проблемы:

1. Начальное пустое состояние системы и недостаточный контроль сигналов, что затрудняло запись данных в память.
2. Слишком высокая рабочая частота платы, которая приводила к необходимости значительного замедления работы системы для обеспечения корректного функционирования.

Указанные сложности были устранены в процессе выполнения работы путем соответствующих изменений в архитектуре и настройке системы.