ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

		А. В. Морозов
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ

РЕАЛИЗОВАТЬ МОДУЛЬ РАБОТЫ С ВНЕШНЕЙ СТАТИЧЕСКОЙ ПАМЯТЬЮ SRAM

по курсу: СХЕМОТЕХНИКА

РАБОТУ ВЫПОЛНИЛИ

СТУДЕНТ ГР. № 4941		Н. С. Горбунов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

СОДЕРЖАНИЕ

1. Вариант задания

Реализовать модуль работы с внешней статической памятью (SRAM). Модуль должен предоставлять интерфейс для чтения и записи со стороны других модулей.

Также необходимо реализовать файл тестбенча для проверки работоспособности модуля и интерфейса.

2. Средство проектирования

Verilog, Verilog HDL (англ. Verilog Hardware Description Language) — это язык описания аппаратуры, используемый для описания и моделирования электронных систем. Verilog HDL, не следует путать с VHDL (конкурирующий язык), наиболее часто используется в проектировании, верификации и реализации (например, в виде СБИС) аналоговых, цифровых и смешанных электронных систем на различных уровнях абстракции.

Разработчики Verilog сделали его синтаксис очень похожим на синтаксис языка C, что упрощает его освоение. Verilog имеет препроцессор, очень похожий на препроцессор языка C, и основные управляющие конструкции «if», «while» также подобны одноимённым конструкциям языка C. Соглашения по форматированию вывода также очень похожи.

Следует отметить, что описание аппаратуры, написанное на языке Verilog (как и на других HDL-языках), принято называть программами, но в отличие от общепринятого понятия программы как последовательности инструкций, здесь программа задает структуру системы. Также для языка Verilog не применим термин «выполнение программы».

Существует подмножество инструкций языка Verilog, называемое синтезируемым. Модули, которые написаны на этом подмножестве, называют RTL (англ. register transfer level — Уровень регистровых передач). Они могут быть физически реализованы с использованием САПР синтеза. Данные САПР по определенным алгоритмам преобразуют абстрактный исходный код на Verilog в netlist — логически эквивалентное описание, состоящее из элементарных логических примитивов (например, AND, OR, NOT, триггеры), которые доступны в выбранной технологии производства СБИС или программирования БМК и ПЛИС. Дальнейшая обработка netlist в конечном итоге порождает фотошаблоны

для литографии или прошивку для FPGA.

3. Схематичный вид SRAM для хранения по 4м адресам

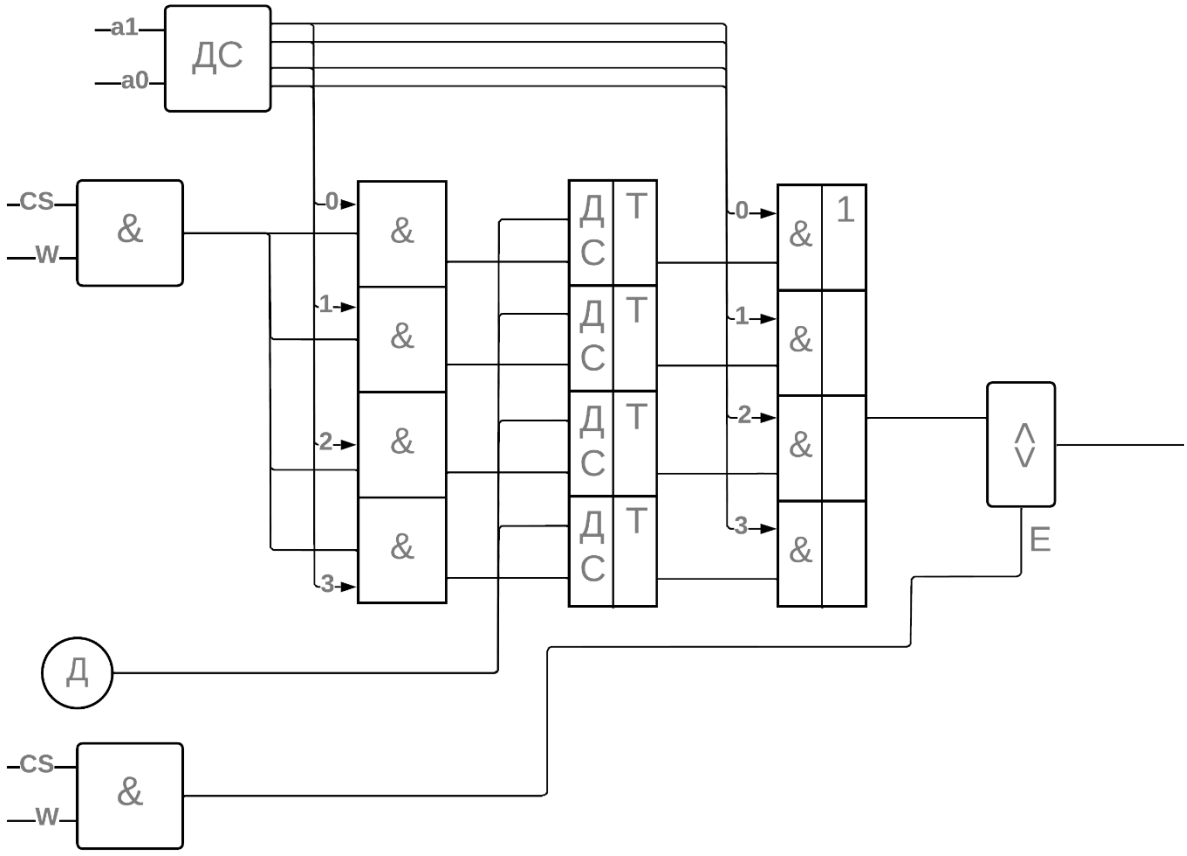

Рисунок 1. Схематичный вид SRAM на 4 адреса

4. Описание созданной микросхемы

Созданный программный модуль представляет из себя модуль статической памяти произвольного доступа с 16 ячейками памяти величиной 8 бит каждая.

Для модуля памяти реализован интерфейс подключения к внешним модулям. Частота работы внешнего модуля может быть не такой как собственная частота работы статической памяти. К примеру, в тесте к написанному модулю было использовано следующее отношение – частота работы внешнего подключаемого модуля в два раза выше собственной частоты работы модуля статической памяти. В связи с этим при записи в память значений происходят потери данных. Именно из-за разности в частотах работы – на два фронта памяти приходится 4 фронта внешнего модуля.

Все взаимодействие внешнего модуля с памятью происходит через написанный интерфейс. При установлении 1 на пин записи интерфейса и 0 на пин чтения, происходит запись по входящему в интерфейс значению адреса входящих в интерфейс данных по 8-рязрядной шине. При установлении 0 на пин записи интерфейса и 1 на пин чтения интерфейса, происходит чтение данных по выходному каналу из интерфейса и по входящему в интерфейс адресу.

5. Код модулей

Модуль памяти

```
`timescale 1ns / 1ps

module sram_sp_srsw
#(parameter DW = 8, AW = 4)
(
	input[AW-1:0]a, //address
	input clk,
	input rd, // enable read
	input we, // enable write
	inout [DW-1:0] d// data
);

parameter DP = 1 << AW; //depth
reg [DW-1:0]mem[0:DP-1];
reg [DW-1:0]reg_d;

always@(posedge clk)
begin
	if (we)// to write
	begin
		mem[a] <= d; //push in cage data
	end
	else if (!we & rd) //read
	begin
		reg_d<= mem[a];
	end
	else
	begin //do not change anything
		mem[a] <= mem[a];
		reg_d <= reg_d;
	end
end

assign d = (!we & rd) ? reg_d : {DW{1'bz}};
//if reading and not writing then push in data wire the chosen
memory cage. Else Z statement
endmodule
```

Интерфейс памяти

```
`timescale 1ns / 1ps

module sram_interface
#(parameter DW = 8, AW = 4)
(
	input clock,
	input read,
	input write,
	input[AW-1:0]address, //address
	input [DW-1:0] datain,
	output reg [DW-1:0] dataout,
	output reg read_ready,

	//for external memory
	inout [DW-1:0] memory_wire,
	output memory_reading,
	output memory_writing,
	output [AW-1:0] memory_address
);

assign memory_reading = read;
assign memory_writing = write;
assign memory_address = address;
assign read_ready = !write;

assign memory_wire = write & !read? datain: 8'bzzzz_zzzz;
assign dataout = read & !write? memory_wire: 8'bzzzz_zzzz;

endmodule
```

Тестовый скрипт

```
`timescale 1ns / 1ps

module test;
reg [3:0]address; //address
reg [3:0]memory_address; //address for sram
reg clk;
reg memory_clock;
reg reading;
reg writing;
reg memory_reading;
reg memory_writing;
wire read_ready;

wire [7:0]inout_data_buf;
reg [7:0]inout_data_buf_reg;
wire [7:0]d;
reg [7:0]din;

assign inout_data_buf =
(memory_writing)?inout_data_buf_reg:8'bzzzz_zzzz;
assign inout_data_buf_reg =
(memory_reading)?inout_data_buf:8'bzzzz_zzzz;

initial
begin
	clk = 1'b0;
	forever #10 clk = ~clk; // period 20
end
initial
begin
	memory_clock = 1'b0;
	forever #20 memory_clock = ~memory_clock; // period 40
end

initial
begin
	// write
	reading = 0;
	writing = 1;
	#20
	address = 4'b0000;
```

```
        din = 8'd1;
        repeat(15) begin
                #20 address = address+1'b1;
                din = din+1'b1;
        end
        writing = 0;
        //read
        reading = 1;
        repeat(15) begin
                #20 address= address-1'b1;
        end

end


        sram_interface uut(
                .clock(clk), //из теста
                .read(reading), //из теста
                .write(writing), //из теста
                .address(address), //из теста
                .datain(din), //из теста
                .dataout(d), //в тест
                .read_ready(read_ready), //в тест
                .memory_wire(inout_data_buf), //в память
                .memory_reading(memory_reading), //в память
                .memory_writing(memory_writing), //в память
                .memory_address(memory_address) //в память
        );

        sram_sp_srsw mem(
                .clk(memory_clock),
                .a(memory_address),
                .oe(memory_reading),
                .we(memory_writing),
                .d(inout_data_buf)
                );
endmodule
```

6. Результаты работы симулятора

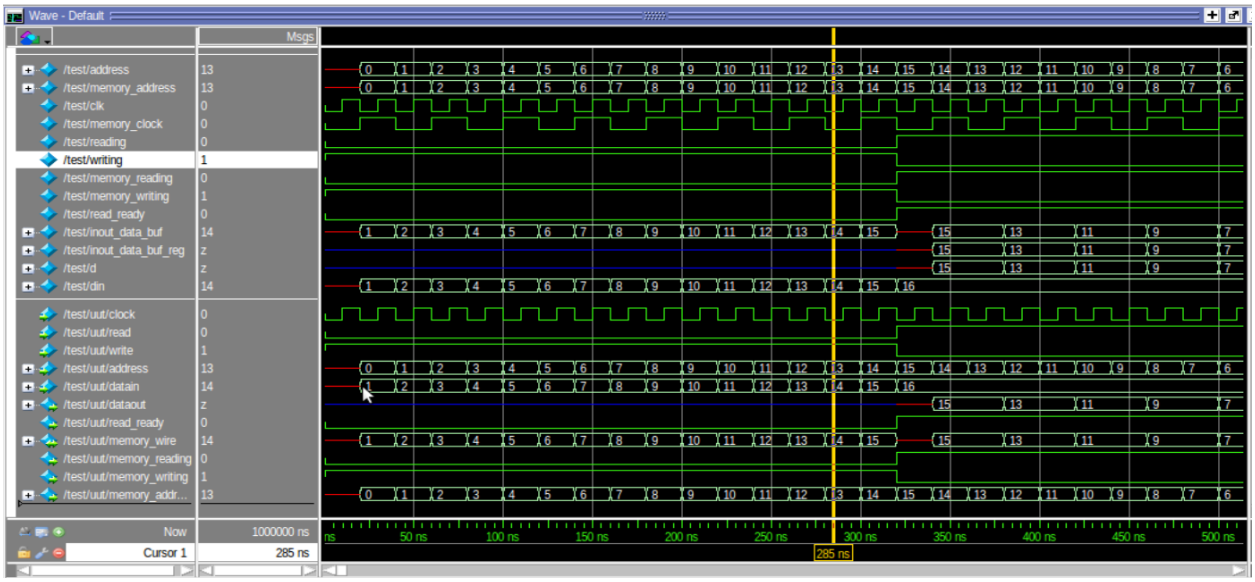

Рисунок 2. Процесс записи

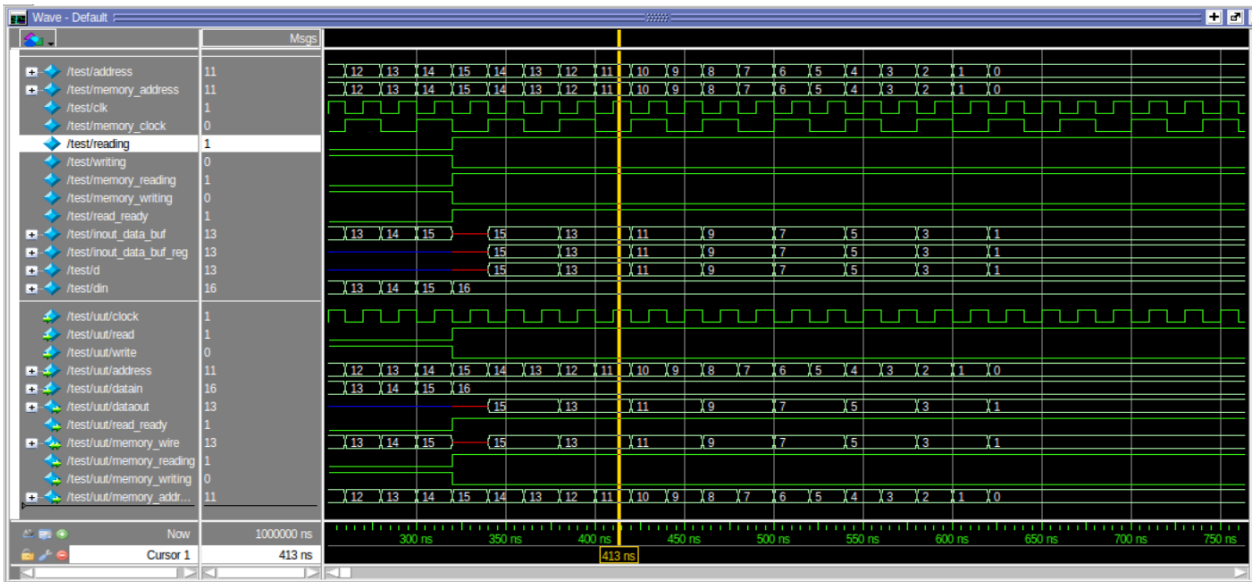

Рисунок 3. Процесс чтения

7. Заключение

В результате выполнения курсовой работы был разработан модуль для организации произвольного доступа к модулю статической памяти по 16 адресам по 8 бит каждый при помощи интерфейса. Модуль обеспечивает возможности произвольного чтения и записи из ячеек модуля статической памяти. Все указанное было реализовано при помощи программного кода на языке Verilog HDL. Также был разработан файл тестирования модуля и интерфейса к нему для проверки корректной работы.