Лабораторная работа №2	Б10	2022
Моделирование схем в Verilog	Барковская Мария Александровна	

Цель работы: построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog.

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 10 и новее (полезные материалы: Verilog.docx). В отчёте нужно указать, какой версией вы пользовались (можно также приложить ссылку на онлайн-платформу). Использовать SystemVerilog допустимо, главное, чтобы код компилился под Icarus 10, 11 или 12. Далее в этом документе Verilog+SystemVerilog обозначается как Verilog.

Описание: в качестве варианта нам были даны некоторые параметры системы и задача, которую надо сначала решить аналитически на основании полученных параметров системы, а затем промоделировать ее на языке Verilog и сравнить полученные результаты (а именно: сколько тактов займет выполнение предоставленной нам программы).

Вычисление недостающих параметров системы.

- **CACHE_SIZE:** весь размер кэша = количество кэш-линий на размер кэш-линий (то есть `CACHE_LINE_SIZE * `CACHE_LINE_COUNT) = 16 * 64 = 1024
- **CACHE_SETS_COUNT:** ассоциативность равна двум, то есть все кэш линии мы храним в наборах по 2, тогда количество наборов кэщ линий равно `CACHE_LINE_COUNT / `CACHE_WAY = 64 / 2 = 32
- **CACHE_SET_SIZE:** логарифм от количества наборов, то есть 5
- **CACHE_OFFSET_SIZE:** offset задает смещение внутри кэш-линии, поэтому его размер – логарифм от размера кэш-линии, то есть 4
- **ADDR1_BUS_SIZE**, **ADDR2_BUS_SIZE:** максимальный размер передаваемой в один момент (в нашем случае – это первый такт из двух, так как во второй передается offset, занимающий всего 4 бита) времени информации равен `SET_SIZE + `CACHE_TAG_SIZE = 15 бит
- **CTR1_BUS_SIZE**, **CTR2_BUS_SIZE:** размер шин для передачи команд равен количеству всех возможных для передачи по этой шине команд (записываем в бинарной записи, поэтому нужно будет взять двоичный логарифм от десятичного числа команд, округленный вверх). Получаем размер шин, равный 3 и 2 соответственно

Аналитическое решение задачи

Поставленная задача была решена аналитически в виде кода на языке программирования kotlin (код представлен ниже). Функция main симулирует задачу, а класс Cache – схематичную работу кэша (те его действия, которые помогают посчитать количество тактов).

В кэше хранятся наборы по две кэш-линии (каждая кэш-линия представляет из себя сами данные и набор информации о них и их состоянии (valid, dirty, tag, lru (lru = true, если в своем наборе кэш-линия не использовалась дольше))).

При получении запроса к памяти (то есть соответстующей команды) кэш проверяет, есть ли у него нужные данные (то есть, находятся ли они в какой-то из кэш-линий (для этого нужно пройтись по двум кэш-линиям из набора с номером set и если у какой-то из них совпал tag с запрошенным, то в кэше есть нужные данные, иначе – нужно обращаться к памяти)). В каждом случае (нашлись данные или нет) дополнительно учитывается время ответа кэша, а в случае, если кэш не нашел у себя этих данных, и при этом кэш-линия, которую мы «вытесняем», “dirty”, то учитываем, что обращения к памяти

необходимо два: для сохранения текущей информации этой кэш-линии и для получения в нее из памяти новой.

tickCounter подсчитывает количество тактов, затраченных на выполнение поставленной нам задачи (учитывая не только работу кэша и памяти, но и сложение, умножение и так далее).

Моделирование заданной системы на Verilog + Воспроизведение задачи на Verilog

Так как мы моделировали систему “процессор-кэш-память”, то в моей реализации есть три основных блока, эту самую систему реализующие: cpu, cache и memory.

Права на чтение/запись данных для модулей в системе такие: при clk == 1 есть право записывать данные на провода, при clk == 0 – считывать с них (эта разница в такт позволяет удобно пользоваться inout шинами, так как мы не можем записать данные, пока с другой стороны не прочитали предыдущие).

Во время реализации возникла определенная проблема ожидания внутри always блоков: неопределенное значение clk после выхода из задержки и возникновение гонок. Для решения этой проблемы мои одногруппники (Артем Пешков и Тимофей Малов) придумали макрос ‘delay’ (реализацию можно найти в листинге кода).

Модуль cache

Реализованный в Verilog-е, кэш работает очень похоже на свой аналитический вариант, с той лишь поправкой, что теперь работа с памятью прописана напрямую. К Кэшу подключены шины, отвечающие за взаимодействие с процессором и памятью, а также в кэше хранятся сами данные кэша (те самые наборы кэш-линий, о которых говорилось в описании аналитического решения), и массив, соответствующий lru, описанному ранее. Всего мы можем получить от процессора три типа команд (не считая C1_NOP):

- Read: позволяет прочитать 8, 16 или 32 бита, в зависимости от команды (указано в ее названии). Происходит чтение адреса по шине a1 за два такта (сначала tag и set, потом offset), проверка на наличие данных в кэше, обработка в случае промаха (если не нашли, то обращаемся к памяти, получая в кэш-линию нужные данные) и возвращение процессору запрошенных данных
- Write: позволяет записать 8, 16 или 32 бита, в зависимости от команды (указано в ее названии). Происходит чтение адреса по шине a1 за два

такта (сначала tag и set, потом offset), проверка на наличие данных в кэше, обработка в случае промаха (если не нашли, то обращаемся к памяти, записывая туда данные, передаваемые нам процессором (не забываем про обработку случая 'dirty')).

- InvalidLine: ставим линии бит валидности 0, при этом если линия была ‘dirty’, записываем ее данные перед этим в память

Общение с модулем памяти происходит следующим образом: ставим на C2 нужную нам команду (чтение/запись), на A2 – необходимый нам адрес, и ставим значение 1 для соответствующего флага, означающего для кэша нашу работу с памятью (reading/writing) и переходим в режим ожидания, пока значение этой переменной не станет обратно равно 0 (что будет значить, что мы закончили работать с памятью). В это время активизируется другой always-блок, в нем мы ждем соответствующей нам конфигурации некоторых из переменных: reading/writing == 1, clk == 0 или 1, в зависимости от того, хотим ли мы считать данные из памяти или записать в нее, а также c2 == `C2_RESPONSE для чтения (что будет нам говорить, что память передает нужные нам данные). Потратив восемь тактов на необходимые нам чтение/запись, возвращаем значение переменной reading/writing в ноль, выходя из этого always-блока.

Модуль memory

К памяти подключены шины, отвечающие за взаимодействие с кэшем. Сама память хранит данные (заполняется изначально в initial-блоке так, как указано в задании). При получении команды, она считывает адрес с шины a2 за два такта, а дальше за 8 тактов в зависимости от полученной команды либо считывает в себя данные с шины d2, либо записывает свои данные в шину d2. При этом учитываем, что, согласно условию, от первого такта команды до первого такта ответа должно пройти 100 тактов.

Модуль cpu

Основная задача просимулирована именно здесь.

К процессору подключены шины, отвечающие за взаимодействие с кэшем. Реализованы таски read_data_8, read_data_16, write_data_32, необходимые для симуляции задачи, а также task_simulation, являющаяся самой симуляцией. Запускается все через initial-блок. Из него же получаем ответ

Сравнение полученных результатов

Из аналитического решения получили:

Total ticks: 5090975
Total accesses: 249600
Cache hits: 228080

Cache misses: 21520
Part of hits: 0.9137820512820513

Из решения на Verilog:

Total ticks: 5042495
Total memory accesses: 249600
Cache hits: 228080
Cache misses: 21520
Part of hits: 0.913782

Листинг кода

Аналитическое решение:

```
var tickCounter = 0
val cache_line_count = 64
val cache_way = 2
val cache_offset_bits = 4 //log_2(cache_line_size)
val cache_set_size = 32 //cache_line_count/cache_way
val cache_set_bits = 5 //log_2(cache_set_size)
val tag_bits = 10

data class Cache_Line(
    var lru: Boolean,
    var valid: Boolean,
    var dirty: Boolean,
    var tag: UInt
)

class Cache {
    var cacheMissCounter = 0
    var cacheHitCounter = 0

    private val cache_lines = List(cache_line_count) { Cache_Line(lru = true, valid = false, dirty
= false, tag = 0u) }

    fun read_data(address: Int, bytes: Int) {
        val address = address.toUInt()
        val offset = address.shl(32 - cache_offset_bits).shr(32 - cache_offset_bits) //last
cache_offset_bits bits
        val set = address.shr(cache_offset_bits).shl(32 - cache_set_bits).shr(32 - cache_set_bits)
//last cache_set_bits bits
        val tag = address.shr(cache_offset_bits + cache_set_bits).shl(32 - tag_bits).shr(32 -
tag_bits) //last tag_bits bits

        val left = set.toInt() * cache_way
        val right = (set.toInt()+1) * cache_way - 1

        for (i in left..right) {
```

```
            if (cache_lines[i].tag == tag) {
                cache_lines[i].lru = false
                cache_lines[i xor 1].lru = true

                if (cache_lines[i].valid) {
                    // нашли в кэше
                    tickCounter += 6 //время, через которое в результате кэш попадания, кэш начинает отвечать
                    tickCounter += 1 //отправка данных по шине d1
                    cacheHitCounter += 1

                    return
                }
            }
        }

        cache_miss(set, tag)

        tickCounter += 1 //отправка данных по шине d1
    }

    fun write_data(address: Int, bytes: Int) {
        val address = address.toUInt()
        val offset = address.shl(32 - cache_offset_bits).shr(32 - cache_offset_bits) //last cache_offset_bits bits
        val set = address.shr(cache_offset_bits).shl(32 - cache_set_bits).shr(32 - cache_set_bits) //last cache_set_bits bits
        val tag = address.shr(cache_offset_bits + cache_set_bits).shl(32 - tag_bits).shr(32 - tag_bits) //last tag_bits bits

        val left = set.toInt() * cache_way
        val right = (set.toInt()+1) * cache_way - 1

        for (i in left..right) {
            if (cache_lines[i].tag == tag) {
                if (cache_lines[i].valid) {
                    // нашли в кэше
                    tickCounter += 6 // время, через которое в результате кэш попадания, кэш начинает отвечать
                    cacheHitCounter += 1

                    cache_lines[i].lru = false
                    cache_lines[i xor 1].lru = true

                    cache_lines[i].dirty = true

                    return
                }
            }
        }

        for (i in left..right) {
```

```
            if (cache_lines[i].lru) {
                cache_lines[i].dirty = true
            }
        }

        cache_miss(set, tag)
    }

    private fun cache_miss(set: UInt, tag: UInt) {
        tickCounter += 4 // время, через которое в результате кэш промаха, кэш посылает запрос к памяти.
        tickCounter += 100 // MemCTR обработка

        cacheMissCounter += 1

        val left = set.toInt() * cache_way
        val right = (set.toInt()+1) * cache_way - 1

        for (i in left..right) {
            if (cache_lines[i].lru) {
                cache_lines[i].valid = true

                if (cache_lines[i].dirty) {
                    tickCounter += 100
                }

                cache_lines[i].dirty = false
                cache_lines[i].tag = tag

                cache_lines[i].lru = false
                cache_lines[i xor 1].lru = true

                return
            }
        }
    }
}

fun main() {
    val cache = Cache()
    //Сложение, инициализация переменных и переход на новую итерацию цикла, выход из функции занимают 1 такт.
    // Умножение – 5 тактов. Обращение к памяти вида pc[x] считается за одну команду.
    val M = 64
    val N = 60
    val K = 32

    var pa = 0 //указатель на массив a
    tickCounter += 1 //инициализация

    val b = M * K //адрес начала массива b
```

```
    var pc = b + K * N * 2 //указатель на массив c
    tickCounter += 1 //инициализация

    repeat(M) {
        repeat(N) { x ->
            var pb = b //указатель на массив b
            tickCounter += 1 //инициализация
            tickCounter += 1 //инициализация переменной s

            repeat(K) { k ->
                cache.read_data(pa + k, 8/8)

                cache.read_data(pb + x * 2, 16/8)

                tickCounter += 5 //умножение
                tickCounter += 1 //сложение

                pb += N * 2
                tickCounter += 1 //сложение
                tickCounter += 1 //итерация цикла
            }

            cache.write_data(pc + x * 4, 32/8)
            tickCounter += 1 //итерация цикла
        }

        pa += K
        tickCounter += 1 //сложение

        pc += N * 4
        tickCounter += 1 //сложение
        tickCounter += 1 //итерация цикла
    }

    tickCounter += 1 //выход из функции

    println("Total ticks: $tickCounter")
    println("Total accesses: ${cache.cacheMissCounter + cache.cacheHitCounter}")
    println("Cache hits: ${cache.cacheHitCounter}")
    println("Cache misses: ${cache.cacheMissCounter}")
    println("Part of hits: ${cache.cacheHitCounter.toDouble() / (cache.cacheMissCounter + cache.cacheHitCounter)}")
}
```

Реализация на языке Verilog:

```
//delay фиксит возможную гонку при чтении значения clk, придуман Артемом Пешковым и Тимофеем Маловым
`define delay(TIME, CLOCK) \
    for (int i = 0; i < TIME; i++) begin \
        wait(clk == (i + !CLOCK) % 2); \
    end
```

```
`define BYTE 8

`define C1_NOP 0
`define C1_READ8 1
`define C1_READ16 2
`define C1_READ32 3
`define C1_INVALIDATE_LINE 4
`define C1_WRITE8 5
`define C1_WRITE16 6
`define C1_WRITE32_OR_RESPONSE 7

`define C2_NOP 0
`define C2_RESPONSE 1
`define C2_READ_LINE 2
`define C2_WRITE_LINE 3

`define VALID 1
`define DIRTY 1

`define CACHE_WAY 2
`define CACHE_TAG_SIZE 10
`define SET_SIZE 5
`define OFFSET_SIZE 4

`define MEM_SIZE 524288 //2^19
`define CACHE_LINE_SIZE 16
`define CACHE_LINE_COUNT 64
`define CACHE_SETS_COUNT (`CACHE_LINE_COUNT / `CACHE_WAY)
`define CACHE_LINE_SIZE_IN_BITS (`CACHE_LINE_SIZE * `BYTE)
`define WHOLE_CACHE_LINE_SIZE_IN_BITS (`VALID + `DIRTY + `CACHE_TAG_SIZE +
`CACHE_LINE_SIZE_IN_BITS)

`define MEMCTR_RESPONSE_TIME 100
`define CACHE_HIT_RESPONSE_TIME 6
`define CACHE_MIS_RESPONSE_TIME 4

`define MAX_POSSIBLE_SIZE_OF_REQUESTED_DATA 32
`define SEND_FROM_MEM (`CACHE_LINE_SIZE/`CACHE_WAY)

`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15

`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16

`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2

module test;
   reg clk = 0;

   wire [`ADDR1_BUS_SIZE-1:0] a1;
   wire [`DATA1_BUS_SIZE-1:0] d1;
```

```
    wire [`CTR1_BUS_SIZE-1:0] c1;
    wire [`ADDR2_BUS_SIZE-1:0] a2;
    wire [`DATA2_BUS_SIZE-1:0] d2;
    wire [`CTR2_BUS_SIZE-1:0] c2;

    reg c_dump = 0;
    reg m_dump = 0;
    reg reset = 0;

    integer mdump_file = 0;
    integer cdump_file = 0;
    integer log_file = 0;

    cpu test_cpu(clk, a1, d1, c1);
    cache test_cache(clk, a1, d1, c1, a2, d2, c2, c_dump, reset);
    memory test_memory(clk, a2, d2, c2, m_dump, reset);

    initial begin
        for (int i = 0; i < 14000000; i++) begin
            #1;
            clk = 1-clk;
        end
    end
endmodule

module cpu(input clk, output wire [`ADDR1_BUS_SIZE-1 : 0] a1, inout wire [`DATA1_BUS_SIZE-1 : 0] d1, inout wire
[`CTR1_BUS_SIZE-1 : 0] c1);
    reg [`ADDR1_BUS_SIZE -1:0] inner_a1 = 'z;
    reg [`DATA1_BUS_SIZE - 1 : 0] inner_d1 = 'z;
    reg [`CTR1_BUS_SIZE - 1 : 0] inner_c1 = 'z;

    assign a1 = inner_a1;
    assign d1 = inner_d1;
    assign c1 = inner_c1;

    reg [7:0] result_for_reading_8 = 'z;
    bit reading_8 = 0;

    reg [15:0] result_for_reading_16 = 'z;
    bit reading_16 = 0;

    reg [31:0] result_for_writing_32 = 'z;
    bit writing_32 = 0;

    //начало симуляции задачи
    int M = 64;
    int N = 60;
    int K = 32;

    int pa = 0;
    int b = M * K;
    int pb = 0;
    int pc = b + K * N * 2;
```

```
    int s = 0;
    int additional_tick_counter = 0;

    initial begin
        test.log_file = $fopen("log.txt", "w");

        task_simulation();

        $display("Total ticks: %0t", $time/2 + additional_tick_counter);
        $display("Total memory accesses: %0d", test.test_cache.cacheMissCounter + test.test_cache.cacheHitCounter);
        $display("Cache hits: %0d", test.test_cache.cacheHitCounter);
        $display("Cache misses: %0d", test.test_cache.cacheMissCounter);
        $display("Part of hits: %0f", test.test_cache.cacheHitCounter*1.0 / (test.test_cache.cacheMissCounter +
test.test_cache.cacheHitCounter));

        test.reset = 1;

        `delay(2,1);

        $fclose(test.mdump_file);
        $fclose(test.cdump_file);
        $fclose(test.log_file);
    end

    always @(negedge clk) begin
        if (c1 == `C1_WRITE32_OR_RESPONSE) begin
            if (reading_8 == 1) begin
                $fdisplay(test.log_file, "END OF READING 8, time = %0d", $time/2);
                $fdisplay(test.log_file, "----------------------------------------");
                $fdisplay(test.log_file, "");

                result_for_reading_8 = d1[7:0];
                reading_8 = 0;
            end else if (reading_16 == 1) begin
                $fdisplay(test.log_file, "END OF READING 16, time = %0d", $time/2);
                $fdisplay(test.log_file, "----------------------------------------");
                $fdisplay(test.log_file, "");

                result_for_reading_8 = d1[15:0];
                reading_16 = 0;
            end else if (writing_32 == 1) begin
                $fdisplay(test.log_file, "END OF WRITING 32, time = %0d", $time/2);
                $fdisplay(test.log_file, "----------------------------------------");
                $fdisplay(test.log_file, "");

                inner_d1 = 'z;
                writing_32 = 0;
            end
        end
    end

    task read_data_8(reg[`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : 0] from);
```

```
    $fdisplay(test.log_file, "");
    $fdisplay(test.log_file, "-----------------------------------------");
    $fdisplay(test.log_file, "START OF READING 8, time = %0d", $time/2);

    wait(clk == 1);

    inner_c1 = `C1_READ8;
    inner_a1 = from[`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : `OFFSET_SIZE];

    `delay(2,1);

    inner_c1 = 'z;
    inner_d1 = 'z;
    inner_a1 = from[`OFFSET_SIZE-1:0];

    reading_8 = 1;
    wait(reading_8 == 0);
endtask

task read_data_16(reg [`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : 0] from);
    $fdisplay(test.log_file, "");
    $fdisplay(test.log_file, "-----------------------------------------");
    $fdisplay(test.log_file, "START OF READING 16, time = %0d", $time/2);

    wait(clk == 1);

    inner_a1 = from[`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : `OFFSET_SIZE];
    inner_c1 = `C1_READ16;

    `delay(2,1);

    inner_a1 = from[`OFFSET_SIZE-1:0];
    inner_c1 = 'z;
    inner_d1 = 'z;

    reading_16 = 1;
    wait(reading_16 == 0);
endtask

task write_data_32(reg [`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : 0] to, reg[31:0] data);
    $fdisplay(test.log_file, "");
    $fdisplay(test.log_file, "-----------------------------------------");
    $fdisplay(test.log_file, "START OF WRITING 32, time = %0d", $time/2);

    wait(clk == 1);

    inner_a1 = to[`ADDR1_BUS_SIZE + `OFFSET_SIZE -1 : `OFFSET_SIZE];
    inner_c1 = `C1_WRITE32_OR_RESPONSE;
    inner_d1 = data[`DATA1_BUS_SIZE - 1:0];

    `delay(2,1);

    inner_a1 = to[`OFFSET_SIZE-1:0];
```

```
        inner_c1 = 'z;
        inner_d1 = data[`DATA1_BUS_SIZE*2-1:`DATA1_BUS_SIZE];

        writing_32 = 1;
        wait(writing_32 == 0);
    endtask

    task task_simulation;
        $fdisplay(test.log_file, "START SIMULATION, time = %0d", $time/2);

        additional_tick_counter += 2; //инициализация pa, pc

        for (int y = 0; y < M; y++) begin
            for (int x = 0; x < N; x++) begin
                pb = b;
                s = 0;

                additional_tick_counter += 2; //инициализация b, s

                for (int k = 0; k < K; k++) begin
                    read_data_8(pa + k);
                    read_data_16(pb + 2*x);

                    s += result_for_reading_8 * result_for_reading_16;

                    additional_tick_counter += 5; //умножение
                    additional_tick_counter += 1; //сложение

                    pb += N * 2;

                    additional_tick_counter += 1; //сложение
                    additional_tick_counter += 1; //итерация цикла
                end

                write_data_32(pc + x * 4, s);

                additional_tick_counter += 1; //итерация цикла
            end

            pa += K;
            additional_tick_counter += 1; //сложение

            pc += N * 4;
            additional_tick_counter += 1; //сложение

            additional_tick_counter += 1; //итерация цикла
        end

        additional_tick_counter += 1; //выход из функции

        $fdisplay(test.log_file, "END SIMULATION, time = %0d", $time/2);
    endtask
endmodule
```

```
module cache(input clk, input wire [`ADDR1_BUS_SIZE-1 : 0] a1, inout wire [`DATA1_BUS_SIZE-1 : 0] d1, inout wire [`CTR1_BUS_SIZE-1 : 0] c1, output wire [`ADDR2_BUS_SIZE-1 : 0] a2, inout wire [`DATA2_BUS_SIZE-1 : 0] d2, inout wire [`CTR2_BUS_SIZE-1 : 0] c2, input c_dump, input reset);
  reg [`WHOLE_CACHE_LINE_SIZE_IN_BITS - 1 : 0] cache_data [`CACHE_SETS_COUNT - 1 : 0][`CACHE_WAY-1:0];
  bit lru [`CACHE_SETS_COUNT - 1: 0];
  bit[`MAX_POSSIBLE_SIZE_OF_REQUESTED_DATA-1:0] inner_data;

  integer hit = -1;
  integer command = 0;
  integer writing = 0;
  integer reading = 0;

  bit [`CACHE_TAG_SIZE -1:0] tag;
  bit [`SET_SIZE -1:0] set;
  bit [`OFFSET_SIZE-1:0] offset;

  reg [`CTR2_BUS_SIZE - 1 : 0] inner_c2 = 'z;
  reg [`DATA2_BUS_SIZE - 1 : 0] inner_d2 = 'z;
  reg [`ADDR2_BUS_SIZE -1:0] inner_a2 = 'z;
  reg [`DATA1_BUS_SIZE - 1 : 0] inner_d1 = 'z;
  reg [`CTR1_BUS_SIZE - 1 : 0] inner_c1 = 'z;

  assign c2 = inner_c2;
  assign d2 = inner_d2;
  assign a2 = inner_a2;
  assign d1 = inner_d1;
  assign c1 = inner_c1;

  integer cacheHitCounter = 0;
  integer cacheMissCounter = 0;

  int i = 0;

  initial begin
    $fdisplay(test.log_file, "");
    $fdisplay(test.log_file, "---------------------------------");
    $fdisplay(test.log_file, "INIT OF CACHE, t = %0d", $time/2);
    for (int outer = 0; outer < `CACHE_SETS_COUNT; outer++) begin
      for (int inner = 0; inner < `CACHE_WAY; inner++) begin
        cache_data[outer][inner] = 0;
      end
      lru[outer] = 0;
    end
    $fdisplay(test.log_file, "---------------------------------");
  end

  always @(clk) begin
    if (clk == 0 && reading == 1 && c2==`C2_RESPONSE) begin
      for (int i = 0; i < `SEND_FROM_MEM; i++) begin
        cache_data[set][hit][`DATA2_BUS_SIZE * i +: `DATA2_BUS_SIZE] = d2;
```

```
            `delay(2,0);
        end
        reading = 0;
    end else if (writing == 1 && clk == 1) begin
        for (int i = 0; i < `SEND_FROM_MEM; i++) begin
            inner_d2 = cache_data[set][hit][`DATA2_BUS_SIZE * i +: `DATA2_BUS_SIZE];

            `delay(2,1);

            inner_c2 = 'z;
            inner_a2 = 'z;
        end
        writing = 2;
    end
end

always @(negedge clk) begin
    if (writing == 2 && c2==`C2_RESPONSE) begin
        writing = 0;
    end
end

always @(posedge c_dump) begin
    $fdisplay(test.log_file, "");
    $fdisplay(test.log_file, "C_DUMP, time = %0d", $time/2);

    for (int i = 0; i < `CACHE_SETS_COUNT; i++) begin
        $fdisplay(test.cdump_file, "line %d: %d %d %d %d", 2*i, cache_data[i][0][127:96], cache_data[i][0][95:64],
cache_data[i][0][63:32], cache_data[i][0][31:0]);
        $fdisplay(test.cdump_file, "line %d: %d %d %d %d", 2*i + 1, cache_data[i][1][127:96], cache_data[i][1][95:64],
cache_data[i][1][63:32], cache_data[i][1][31:0]);
    end
end

always @(posedge reset) begin
    $fdisplay(test.log_file, "RESET, time = %0d", $time/2);
    for (int outer = 0; outer < `CACHE_SETS_COUNT; outer++) begin
        for (int inner = 0; inner < `CACHE_WAY; inner++) begin
            cache_data[outer][inner] = 0;
        end
        lru[outer] = 0;
    end
end

always @(negedge clk) begin
    case (c1)
    `C1_READ8, `C1_READ16, `C1_READ32: begin
        command = c1;
        set[`SET_SIZE-1:0] = a1[`SET_SIZE-1:0];
        tag[`CACHE_TAG_SIZE-1:0] = a1[`ADDR1_BUS_SIZE-1:`SET_SIZE];

        `delay(2,0);
```

```
        offset[`OFFSET_SIZE-1:0] = a1[`OFFSET_SIZE-1:0];

        `delay(1,0);

        hit = -1;
        inner_c1 = `C1_NOP;

        i = 0;

        while (hit == -1 && i < `CACHE_WAY) begin
          if (cache_data[set][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_IN_BITS -
1:`CACHE_LINE_SIZE_IN_BITS] == tag) begin
            if (cache_data[set][i][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID]) begin
              $fdisplay(test.log_file, "");
              $fdisplay(test.log_file, "HIT CACHE, time = %0d", $time/2);

              lru[set] = i;

              cacheHitCounter++;

              `delay((`CACHE_HIT_RESPONSE_TIME - 2)* 2, 1);
              hit = i;
            end
          end
          i++;
        end

        //miss
        if (hit == -1) begin
          $fdisplay(test.log_file, "");
          $fdisplay(test.log_file, "CACHE MISS, time = %0d", $time/2);

          cacheMissCounter++;

          hit = 1 - lru[set];

          `delay((`CACHE_MIS_RESPONSE_TIME - 2)* 2, 0);

          if (cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID] &&
cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID - `DIRTY]) begin
            inner_a2[`SET_SIZE - 1: 0] = set;
            inner_a2[`ADDR2_BUS_SIZE-1:`SET_SIZE] = cache_data[set][hit][`CACHE_LINE_SIZE_IN_BITS +
`CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_IN_BITS];
            inner_c2 = `C2_WRITE_LINE;

            writing = 1;
            wait(writing == 0);

            `delay(1,0);
          end

          inner_a2[`SET_SIZE - 1: 0] = set;
          inner_a2[`ADDR2_BUS_SIZE-1:`SET_SIZE] = tag;
```

```
            inner_c2 = `C2_READ_LINE;

            `delay(2,1);

            inner_c2 = 'z;
            inner_a2 = 'z;
            inner_d2 = 'z;

            reading = 1;
            wait(reading == 0);

            cache_data[set][hit][`CACHE_LINE_SIZE_IN_BITS + `CACHE_TAG_SIZE] = 0;
            cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS-`VALID] = 1;
            cache_data[set][hit][`CACHE_TAG_SIZE+`CACHE_LINE_SIZE_IN_BITS-1:`CACHE_LINE_SIZE_IN_BITS] =
tag;

            lru[set] = hit;

            `delay(1,0);
          end

          inner_c1 = `C1_WRITE32_OR_RESPONSE;

          case (command)
            `C1_READ8: begin
              inner_d1[7:0] = cache_data[set][hit][offset * `BYTE +: 8];
            end
            `C1_READ16: begin
              inner_d1[15:0] = cache_data[set][hit][offset * `BYTE +: 16];
            end
            `C1_READ32: begin
              inner_d1[15:0] = cache_data[set][hit][offset * `BYTE +: 16];
              `delay(2,1);
              inner_d1[15:0] = cache_data[set][hit][16 + offset +: 16];
            end
          endcase

          `delay(2,1);

          inner_c1 = 'z;
          inner_d1 = 'z;
        end
        `C1_WRITE8, `C1_WRITE16, `C1_WRITE32_OR_RESPONSE: begin
          command = c1;
          set[`SET_SIZE-1:0] = a1[`SET_SIZE-1:0];
          tag[`CACHE_TAG_SIZE-1:0] = a1[`ADDR1_BUS_SIZE-1:`SET_SIZE];

          inner  data[15:0] = d1;

          `delay(2,0);

          offset[`OFFSET  SIZE-1:0] = a1[`OFFSET  SIZE-1:0];
```

```
        if (command == `C1_WRITE32_OR_RESPONSE) begin
            inner_data[31:16] = d1;
        end

        `delay(1,0);

        hit = -1;
        inner_c1 = `C1_NOP;

        i = 0;

        while (hit == -1 && i < `CACHE_WAY) begin
            if (cache_data[set][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_IN_BITS -
1:`CACHE_LINE_SIZE_IN_BITS] == tag) begin
                if (cache_data[set][i][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID]) begin
                    $fdisplay(test.log_file, "");
                    $fdisplay(test.log_file, "HIT CACHE, time = %0d", $time/2);

                    lru[set] = i;

                    cacheHitCounter++;

                    `delay((`CACHE_HIT_RESPONSE_TIME-2) * 2, 0);
                    hit = i;
                end
            end
            i++;
        end

        //miss
        if (hit == -1) begin
            $fdisplay(test.log_file, "");
            $fdisplay(test.log_file, "CACHE MISS, time = %0d", $time/2);

            cacheMissCounter++;

            hit = 1 - lru[set];

            `delay((`CACHE_MIS_RESPONSE_TIME - 2) * 2, 0);

            if (cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID] &&
cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID - `DIRTY]) begin
                inner_a2[`SET_SIZE - 1: 0] = set;
                inner_a2[`ADDR2_BUS_SIZE-1:`SET_SIZE] = cache_data[set][hit][`CACHE_LINE_SIZE_IN_BITS +
`CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_IN_BITS];
                inner_c2 = `C2_WRITE_LINE;

                writing = 1;
                wait(writing == 0);

                `delay(1,0);
            end
```

```
            inner_a2[`SET_SIZE - 1: 0] = set;
            inner_a2[`ADDR2_BUS_SIZE-1:`SET_SIZE] = tag;

            inner_c2 = `C2_READ_LINE;

            `delay(2,1);

            inner_a2 = 'z;
            inner_c2 = 'z;
            inner_d2 = 'z;

            reading = 1;
            wait(reading == 0);

            cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS-`VALID] = 1;
            cache_data[set][hit][`CACHE_TAG_SIZE+`CACHE_LINE_SIZE_IN_BITS-1:`CACHE_LINE_SIZE_IN_BITS] =
tag;

            lru[set] = hit;

            `delay(1,0);
        end

        cache_data[set][hit][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_IN_BITS] = 0;

        if (command == `C1_WRITE8) begin
            cache_data[set][hit][offset * 8 +: 8] = inner_data[7:0];
        end else if (command == `C1_WRITE16) begin
            cache_data[set][hit][offset * 8 +: 16] = inner_data[15:0];
        end else if (command == `C1_WRITE32_OR_RESPONSE) begin
            cache_data[set][hit][offset * 8 +: 32] = inner_data[31:0];
        end

        inner_c1 = `C1_WRITE32_OR_RESPONSE;

        `delay(2,1);

        inner_c1 = 'z;
        inner_d1 = 'z;
    end
    `C1_INVALIDATE_LINE: begin
        $fdisplay(test.log_file, "");
        $fdisplay(test.log_file, "INVELIDATE LINE, time = %0d", $time/2);

        command = inner_c1;

        set[`SET_SIZE-1:0] = a1[`SET_SIZE-1:0];
        tag[`CACHE_TAG_SIZE-1:0] = a1[`ADDR1_BUS_SIZE-1:`SET_SIZE];

        `delay(2,0);

        offset[`OFFSET_SIZE-1:0] = a1[`OFFSET_SIZE-1:0];
```

```
        `delay(1,0);

        hit = -1;
        inner_c1 = `C1_NOP;

        for (int i = 0; i < `CACHE_WAY && (hit==-1); i++) begin
          if (cache_data[set][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_IN_BITS -
1:`CACHE_LINE_SIZE_IN_BITS] == tag) begin
            if (cache_data[set][i][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID]) begin
              lru[set] = i;

              cacheHitCounter++;

              `delay(`CACHE_HIT_RESPONSE_TIME * 2 - 4, 0);

              hit = i;

              if (cache_data[set][hit][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID - `DIRTY]) begin
                inner_a2[`SET_SIZE - 1: 0] = set;
                inner_a2[`ADDR2_BUS_SIZE-1:`SET_SIZE] = tag;
                inner_c2 = `C2_WRITE_LINE;

                writing = 1;
                wait(writing == 0);

                `delay(1,0);
              end

              cache_data[set][hit][`CACHE_LINE_SIZE_IN_BITS + `CACHE_TAG_SIZE] = 0;
              cache_data[set][i][`WHOLE_CACHE_LINE_SIZE_IN_BITS - `VALID] = 0;
            end
          end
        end
      end
      endcase
    end
endmodule

module memory #(parameter _SEED = 225526) (input clk, input wire [`ADDR2_BUS_SIZE-1 : 0] a2, inout wire
[`DATA2_BUS_SIZE-1 : 0] d2, inout wire [`CTR2_BUS_SIZE-1 : 0] c2, input m_dump, input reset);
    integer SEED = _SEED;

    logic [`BYTE -1:0] memory_data[`MEM_SIZE -1:0];

    reg [`DATA2_BUS_SIZE - 1 : 0] inner_d2 = 'z;
    reg [`CTR2_BUS_SIZE - 1 : 0] inner_c2 = 'z;

    assign c2 = inner  c2;
    assign d2 = inner_d2;

    bit [`ADDR2_BUS_SIZE-1:0] inner_a = 'z;

    initial begin
```

```
    $fdisplay(test.log_file, "---------------------------------");
    $fdisplay(test.log_file, "INIT MEMORY, t = %0d", $time/2);

    for (int i = 0; i < (1 << `MEM_SIZE); i++) begin
        memory_data[i] = $random(SEED)>>16;
    end
end

int i;
//прописать еще reset, dump

always @(posedge m_dump) begin
    $fdisplay(test.log_file, "MDUMP, time = %0d", $time/2);

    for (i = 0; i < `MEM_SIZE; i++) begin
        $fdisplay(test.mdump_file, "line %d: %d", i, memory_data[i]);
    end
end

always @(posedge reset) begin
    $fdisplay(test.log_file, "RESET MEMORY, time = %0d", $time/2);

    SEED = _SEED;
    for (i = 0; i < `MEM_SIZE; i++) begin
        memory_data[i] = $random(SEED)>>16;
    end
end

always @(negedge clk) begin
    case (c2)
        `C2_READ_LINE: begin
        $fdisplay(test.log_file, "", $time/2);
        $fdisplay(test.log_file, "READ FROM MEM, time = %0d", $time/2);

        inner_a = a2;

        `delay(1,0);

        inner_c2 = `C2_NOP;

        `delay(`MEMCTR_RESPONSE_TIME*2, 1);

        inner_c2 = `C2_RESPONSE;

        for (int i = 0; i < `SEND_FROM_MEM; i++) begin
            inner_d2[`BYTE-1:0] = memory_data[inner_a * (1<<`OFFSET_SIZE) + 2 * i];
            inner_d2[`DATA2_BUS_SIZE-1:`BYTE] = memory_data[inner_a * (1<<`OFFSET_SIZE) + 2 * i + 1];

            `delay(2,1);
        end

        inner  c2 = 'z;
        inner_d2 = 'z;
```

```
        inner_a = 'z;
        end
        `C2_WRITE_LINE: begin
        $fdisplay(test.log_file, "");
        $fdisplay(test.log_file, "WRITE TO MEM, time = %0d", $time/2);

        inner_a = a2;

        for (int i = 0; i < `SEND_FROM_MEM; i++) begin
           memory_data[inner_a * (1<<`OFFSET_SIZE) + 2 * i] = inner_d2[`BYTE-1:0];
           memory_data[inner_a * (1<<`OFFSET_SIZE) + 2 * i + 1] = inner_d2[`DATA2_BUS_SIZE-1:`BYTE];
           `delay(2,0);
        end

        `delay(`MEMCTR_RESPONSE_TIME*2 - `SEND_FROM_MEM*2,1);

        inner_c2 = `C2_RESPONSE;

        `delay(2,1);

        inner_c2 = 'z;
        inner_a = 'z;
      end
      endcase
   end
endmodule
```