Лабораторная работа №1	M3139	2022
Построение логических схем в среде моделирования	Гоге Анастасия Эдуардовна	

2. Цель работы

Построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog.

3. Инструментарий

Язык Verilog, компиляция и симуляция – Icarus Verilog 11.

4. Формулировка задачи из условия

Задача представляет собой перемножение матриц a размером 64 x 32, в которой каждое число занимает 1 байт, и b размером 32 x 60, в которой каждое число занимает 2 байта. Данные о матрицах записаны в памяти с 0 по 2047 байта – a и с 2048 по 5887 – b. Результат умножения a и b – это матрица с размером 64 x 60, в которой каждое число занимает 4 байта. Она записывается в память с 5888 по 21 247 байт.

5. Вычисление недостающих параметров системы

Вычисленные параметры кэша и пямяти приведены в таблице 1, а размерность шин в таблице 2. (Жирным выделены известные данные)

Кэш	
Размер кэша (CACHE_SIZE)	**2 Кб = 2^{11} байт**
Размер кэш-линии (CACHE_LINE_SIZE)	**16 байта = 2^4 байт**
Кол-во бит под тэг адреса (CACHE_TAG_SIZE)	**8 бит**
Кол-во кэш-линий (CACHE_LINE_COUNT)	2^7 байт
Размер адреса (CACHE_ADDR_SIZE)	18 бит
Размер смещения (CACHE_OFFSET_SIZE)	4 бита
Кол-во бит под хранение индекса набора (CACHE_SET_SIZE)	6 бит
Кол-во наборов кэш-линий (CACHE_SETS_COUNT)	2^6 байт

Ассоциативность (CACHE_WAY)	2
Память	
Размер памяти (MEM_SIZE)	**256 Кбайт = 2^{18} байт**

Таблица 1.Параметры кэша и памяти

Шина	Обозначение	Размерность
A1, A2	ADDR1_BUS_SIZE, ADDR2_BUS_SIZE	14 бит
D1, D2	DATA1_BUS_SIZE, DATA2_BUS_SIZE	**16 бит**
C1, C2	CTR1_BUS_SIZE, CTR2_BUS_SIZE	3 бита, 2 бита

Таблица 2.Размерности шин

Вычисления:

CACHE_LINE_COUNT = CACHE_SIZE / CACHE_LINE_SIZE = 2^{11} / 2^4 = 2^7

CACHE_ADDR_SIZE = $\log_2$(MEM_SIZE) = $\log_2(2^{18})$ = 18 бит

CACHE_OFFSET_SIZE = $\log_2$(CACHE_LINE_SIZE) = $\log_2(2^4)$ = 4 бита

CACHE_SET_SIZE = CACHE_ADDR_SIZE – CACHE_OFFSET_SIZE – CACHE_TAG_SIZE = 18 – 4 – 8 = 6 бит

CACHE_SETS_COUNT = $2^{\text{CACHE\_SETS\_SIZE}}$ = 2^6

CACHE_WAY = CACHE_LINE_COUNT / CACHE_SETS_COUNT = 2^7 / 2^6 = 2

ADDR1_BUS_SIZE = max(CACHE_TAG_SIZE + CACHE_SET_SIZE, CACHE_OFFSET_SIZE) = 14 бит

ADDR2_BUS_SIZE = CACHE_TAG_SIZE + CACHE_SET_SIZE = 14 бит

CTR1_BUS_SIZE = 3 бита (так как вариантов команд 8 = 2^3 и максимальный номер команды 7)

CTR2_BUS_SIZE = 2 бита (так как вариантов команд 3 < 2^2 и максимальный номер команды 3)

6. Аналитическое решение задачи

Решение написано на C++, целиком его можно увидеть в листинге 1. Функция mmul выполняет задачу из условия. В переменной count_tact считаются такты. В коде комментариями указано, чему соответствует каждое ее увеличение. mmul вызывает функцию check_cache, которая проверяет, есть нужная линия в кэше, и соответствующе считает такты, промахи и попадания.

В результате были подучены такие значения:

Количество тактов – 4949975

Количество обращений к кэшу – 249600

Количество кэш-попаданий – 230698

Количество кэш-промахов – 18902

Процент попадания – 92.4271%

```
#include <bits/stdc++.h>
using namespace std;

const int M = 64;
const int N = 60;
const int K = 32;
const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 1 << 6;
const int CACHE_SET_SIZE = 6;
const int CACHE_OFFSET_SIZE = 4;
const int CACHE_LINE_SIZE = (1 << 4);
const int DATA2_BUS_SIZE = 16;

const int SIZE_A = 1;
const int SIZE_B = 2;
const int SIZE_C = 4;

int count_tact = 0;
int count_cache_hit = 0;
int count_cache_miss = 0;

int cache[CACHE_SETS_COUNT][CACHE_WAY];
int cache_addr_use[CACHE_SETS_COUNT][CACHE_WAY];
int cache_valid[CACHE_SETS_COUNT][CACHE_WAY];
int cache_dirty[CACHE_SETS_COUNT][CACHE_WAY];

int pa;
int pb;
int pc;

void read_in_mem(int set, int tag, int count_bytes) {
    if (cache_addr_use[set][0] == 0) {
        if (cache_dirty[set][0] == 1 && cache_valid[set][0] == 1)
{
            count_tact += 100 + 1; //write in mem
```

```
        }
        cache[set][0] = tag;
        cache_valid[set][0] = 1;
        cache_dirty[set][0] = 0;
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
    } else {
        if (cache_dirty[set][1] == 1 && cache_valid[set][1] == 1)
{
            count_tact += 100 + 1; //write in mem
        }
        cache[set][1] = tag;
        cache_valid[set][1] = 1;
        cache_dirty[set][1] = 0;
        cache_addr_use[set][1] = 1;
        cache_addr_use[set][0] = 0;
    }
}

void check_cache(int addr, int count_bytes, bool is_write) {
    int tag = addr >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    int set = (addr >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    if (cache[set][0] == tag && cache_valid[set][0] == 1){
        if (is_write) {
            cache_dirty[set][0] = 1;
        }
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
        count_cache_hit++;
        count_tact += 6 + (count_bytes + 1) / 2;
    } else if (cache[set][1] == tag && cache_valid[set][1] == 1) {
        if (is_write) {
            cache_dirty[set][1] = 1;
        }
        cache_addr_use[set][1] = 1;
        cache_addr_use[set][0] = 0;
        count_cache_hit++;
        count_tact += 6 + (count_bytes + 1) / 2;
    } else {
        count_cache_miss++;
        read_in_mem(set, tag, count_bytes);
        count_tact += 100 + 4 + 1;
        count_tact += CACHE_LINE_SIZE / DATA2_BUS_SIZE;
    }
}

void mmul() {
    count_tact++;  //int8 *pa = a;
    count_tact++;  //int32 *pc = c;
    pa = 0;
    pc = M * K * SIZE_A + N * K * SIZE_B;
    count_tact++; // initialization y
    for (int y = 0; y < M; y++) {
```

```
        count_tact++;  //start of a new loop iteration
        count_tact++; // initialization x
        for (int x = 0; x < N; x++) {
            pb = M * K * SIZE_A;
            count_tact++;  //start of a new loop iteration
            count_tact++;  //int16 *pb = b;
            count_tact++;  //int32 s = 0;
            // count_tact++; // initialization k
            for (int k = 0; k < K; k++) {
                count_tact++;  //start of a new loop iteration
                count_tact+= (1 + 5 + 1 + 1);  //s += pa[k]*pb[x];
                check_cache(pa + k * SIZE_A, SIZE_A, false);
                check_cache(pb + x * SIZE_B, SIZE_B, false);
                count_tact++;  //pb += N;
                pb += N * SIZE_B;
            }
            count_tact++;  //pc[x] = s;
            check_cache(pc + x * SIZE_C, SIZE_C, true);

        }
        pa += K * SIZE_A;
        pc += N * SIZE_C;
        count_tact++;  //pa += K;
        count_tact++;  //pc += N;
    }
    count_tact++;  //exit out of function mmul;
}

int main() {
    mmul();
    cout << "Count tact: " << count_tact << "\n";
    cout << "Cache-hit percentage: ";
    cout << count_cache_hit << " " << count_cache_miss << " " <<
count_cache_miss + count_cache_hit << " ";
    cout << ((float)count_cache_hit / (float)(count_cache_hit +
count_cache_miss) * 100) << "%";
}
```

Листинг 1. Аналитическое решение

7. Моделирование заданной системы на Verilog

В системе присутствуют 3 модуля CPU, Cache и MemCTR. Они связаны проводами во вспомогательном модуле testbench. В нем так же считается синхронизация CLK и подается в остальные модули.

Подключение модулей - look-through. Это значит, что процессор соединен с кэшем, а кэш с памятью и общение процессора с памятью происходит только через кэш.

Все команды, адреса и данные передаются по проводам(если inout) или регистрам(если input или output). Чтобы передать что-то по регистру, нужно просто присвоить ему значение. Чтобы передавать значение про проводу, который является и входом и выходом нужно привязать его к регистру с посощью assign и присваивать значение этому регистру. Если одному проводу с разных регистров будут поданы значения не равные z, то на проводе получится x, поэтому после передачи команды всегда подаем на провод z и передаем команда по очереди, и тогда будет правильно приниматься значение, пришедшее от другого модуля.

Во всех модулях есть функция _wait(count), которая пропускает count тактов.

Модули:

1. CPU

Этот модуль отправляет команды кэшу. Порядок записи little endian, это значит, что байты записываются от младшего к старшему. В initial указываются нужные команды. В нашем случае в нем написано решение данной задачи. Блок always в конце считает количество тактов.

Команды:

Запросы на чтение(READ8, READ16, READ32)(листинг 1)

Эти команды посылают кэшу по шине C1 сигналы 1, 2 или 3 соответственно, сет и тэг в первый такт и оффсет во второй. Потом ждут ответного сигнала от кэша и считывают полученные данные за один или 2 такта в зависимости от количества данных.

```
    task automatic READ8(int x, int k);
        C1_1 = 1;
        A1 = (pa + k * `SIZE_A) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pa + k * `SIZE_A) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_a = D1;
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask

    task automatic READ16(int x, int k);
        C1_1 = 2;
        A1 = (pb + x * `SIZE_B) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pb + x * `SIZE_B) % (1 << `CACHE_OFFSET_SIZE);
```

```
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_b = D1;
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask

    task automatic READ32(int aaaa);
        C1_1 = 3;
        A1 = aaaa >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = aaaa % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_c = D1;
        _wait(1);
        data_c += (D1 << 16);
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask
```

Листинг 2. READ8, READ16, READ32

Запросы на запись(WRITE8, WRITE16, WRITE32)(листинг 3)

Эти команды посылают кэшу по шине C1 сигналы 5, 6 или 7 соответственно, сет и тэг, и первую порцию данных в первый такт и оффсет и вторую порцию данных, если это нужно, во второй. Потом ждут ответного сигнала от кэша.

```
    task automatic WRITE8(int x);
        C1_1 = 7;
        D1_1 = s;
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask

    task automatic WRITE16(int x);
        C1_1 = 7;
```

```
        D1_1 = s;
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask

    task automatic WRITE32(int x);
        C1_1 = 7;
        D1_1 = s % (1 << 16);
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        D1_1 = (s >> 16);
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask
```

Листинг 3. WRITE8, WRITE16, WRITE32

2. Cache

Этот модуль принимает команды от CPU и исполняет их.

Политика замещения – LRU. Это значит, что если все кэш-линии с нужным заняты, а новую кэш-линю записать надо, заменяться будет та, которая была использована раньше.

Политика записи - write-back. Это значит, что при команде записи данные записываются только в кэш, а в память они переписываются только тогда, когда линия вытесняется.(После завершения выполнения программы можно было переписать в память все данные, которые хранятся только в кэше, чтобы матрица хранилась в самой памяти, но в тз это не было указано, поэтому я не стала)

Регистры и переменные(листинг 4):

cache – двумерный регистр, в котором хранятся кэш-линии

cache_addr_use – двумерный регистр, который хранит 2 значения для каждого сета. Строке из сета, которая была записана последней, соответствует значение 1, а второй строке или строке, в которой еще нет данных, соответствует значение 0.

line, data, addr, tag, set, offset – регистры и переменные для хранения кэш-линии, данных, адреса, тэга, сета и оффсета соответственно и взаимодействия разных функций с ними.

fd – для вывода в файл

ans_cpu – регистр для хранения ответа на запрос о чтении для процессора, в ans_cpu[0] хранится то, что будет передаваться в первый такт передачи данных, в ans_cpu[1] – то, что будет передаваться во второй такт передачи данных(для READ32).

cache_req – количество обращений к кэшу

cache_miss – количество кэш-промахов

cur_c1 – для хранения текущего значения сигнала с C1

```
reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0]
cache[`CACHE_LINE_COUNT - 1 : 0];
reg[`CACHE_WAY - 1 : 0] cache_addr_use[`CACHE_SETS_COUNT - 1 : 0];
reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0] line;
reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0] data;
reg[`CACHE_ADDR_SIZE - 1 : 0] addr;
reg[`CACHE_TAG_SIZE - 1 : 0] tag;
reg[(`CACHE_SET_SIZE * 2) - 1: 0] set;
reg[`CACHE_OFFSET_SIZE - 1 : 0] offset;
int fd;
reg[15:0] ans_cpu[1:0];
int cache_req = 0;
int cache_miss = 0;
reg[`CTR1_BUS_SIZE - 1 : 0] cur_c1;
```

Листинг 4. Регистры и переменные

Чтение:

При чтении кэш получает сигнал по шине C1 равный 1, 2 или 3 в зависимости от того, сколько байтов нужно прочитать, тогда срабатывает условие в блоке always(листинг 5), и передается также tag и set по шине A1 в первый такт и offset по шине A1 в следующий.

```
always @(posedge CLK) begin
    if (C1 === 1 || C1 === 2 || C1 === 3) begin
        cur_c1 = C1;
        addr = A1;
        _wait(1);
        addr = (addr << `CACHE_OFFSET_SIZE) + A1;
        _wait(1);
        C1_1 = 0;
        if (cur_c1 === 1) begin
            _READ(1);
        end else if (cur_c1 === 2) begin
```

```
            _READ(2);
        end else begin
            _READ(4);
        end
    end else if (C1 === 5 || C1 === 6 || C1 === 7) begin
        cur_c1 = C1;
        addr = A1;
        data = D1;
        _wait(1);
        addr = (addr << `CACHE_OFFSET_SIZE) + A1;
        if (cur_c1 === 5) begin
            _wait(1);
            C1_1 = 0;
            _WRITE(1);
        end else if (cur_c1 === 6) begin
            _wait(1);
            C1_1 = 0;
            _WRITE(2);
        end else begin
            data = (data << 16) + D1;
            _wait(1);
            C1_1 = 0;
            _WRITE(4);
        end
        D1_1 = 'hz;
    end else if (C1 === 4) begin
        addr = A1;
        _wait(1);
        addr = (addr << `CACHE_OFFSET_SIZE) + A1;
        _wait(1);
        C1_1 = 0;
        _INVALIDATE_LINE();
        C1_1 = 'hz;
    end
end
```

Листинг 5. Always-блок

В этот же такт в модуле запускается task _READ(листинг 6). Который состоит из двух частей read_in_cache и answer_CPU.

```
task automatic _READ(input int count_bytes);
    _wait(4);
    read_in_cache(count_bytes);
    C1_1 = 7;
    _wait(1);
    answer_CPU(count_bytes);
    C1_1 = 'hz;
endtask
```

Листинг 6. _READ

В read_in_cache(листинг 7) проверяется есть ли кэш-линия с нужным тэгом. И если есть, это кеш-попадание. В cache_addr_use[set] отмечается, что последнее действие произведено с соответствующей строкой в кэше, и вызывается fill_ans_cpu(листинг 8), который собирает в ответ для CPU 1, 2 или 4 байта, в зависимости от count_bytes. Если нет, то кэш-промах. Вызывается read_in_mem, который записывает данные из памяти в кэш, а потом снова read_in_cache, который уже не промахнется, так как мы только что записали в кэш нужную линию. Поскольку при любом обращении к кэшу в одном из вызовов функции read_in_cache мы зайдем в одно из двух первых условий, здесь считается именно количество обращений(cache_req), а не попаданий.

```
task automatic read_in_cache(input int count_bytes);
    ans_cpu[0] = 0;
    ans_cpu[1] = 0;
    tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
    set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
    offset = addr % (1 << `CACHE_OFFSET_SIZE);
    if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag
    && cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE + 1] == 1) begin
        cache_req++;
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
        fill_ans_cpu(0, count_bytes);
    end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag
    && cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE + 1] == 1) begin
        cache_req++;
        cache_addr_use[set][0] = 0;
        cache_addr_use[set][1] = 1;
        fill_ans_cpu(1, count_bytes);
    end else begin
        cache_miss++;
        read_in_mem();
        _wait(1);
        read_in_cache(count_bytes);
    end
endtask
```

Листинг 7. read_in_cache

```
task automatic fill_ans_cpu(int num_in_set, int count_bytes);
    if (count_bytes == 1) begin
        for (int v = 0; v < 8; v++) begin
            ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
```

```
            num_in_set][offset * 8 + v]) << v);
        end
    end else if (count_bytes == 2) begin
        for (int v = 0; v < 8; v++) begin
            ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
            num_in_set][(offset + 1) * 8 + v]) << v);
        end
        ans_cpu[0] = ans_cpu[0] << 8;
        for (int v = 0; v < 8; v++) begin
            ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
            num_in_set][offset * 8 + v]) << v);
        end
    end else begin
        int new_set = set;
        new_set = set << 1;
        for (int u = 0; u < 2; u++) begin
            for (int v = 0; v < 8; v++) begin
                ans_cpu[u] = ans_cpu[u] + ((cache[new_set +
                num_in_set][(offset + 1 + u * 2) * 8 + v])
                << v);
            end
            ans_cpu[u] = ans_cpu[u] << 8;
            for (int v = 0; v < 8; v++) begin
                ans_cpu[u] = ans_cpu[u] + ((cache[new_set +
                num_in_set][(offset + u * 2) * 8 + v]) << v);
            end
        end
    end
endtask
```

Листинг 8. fill_ans_cpu

Вернемся к read_in_mem(листинг 9), он отправляет памяти 2 по шине C2 и адрес по шине A2, то есть запрос на чтение данных, ждет ответ и считывает данные, которые отправляет память за 8 тактов, в переменную line. Дальше line записывается в кэш-линию, которая была использована раньше.

```
    task automatic read_in_mem();
    C2_1 = 2;
    A2 = addr >> `CACHE_OFFSET_SIZE;
    _wait(1);
    C2_1 = 'hz;
    while (!(C2 === 1)) _wait(1);
    line = 0;
    for (int a = 0; a < 8; a++) begin
        line = line + (D2 << (16 * a));
        _wait(1);
    end
    if (cache_addr_use[set][0] == 0) begin
        if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
```

```
        `CACHE_TAG_SIZE] == 1 &&
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] == 1) begin
            reg[`CACHE_TAG_SIZE - 1 : 0] new_tag =
            (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
            `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
            write_in_mem(0, set + (new_tag << `CACHE_SET_SIZE));
            _wait(1);
        end
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE - 1 : 0] = (tag <<
        `CACHE_LINE_SIZE_BYTE) + line;
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] = 1;
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] = 0;
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
    end else begin
        if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] == 1 &&
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] == 1) begin
            reg[`CACHE_TAG_SIZE - 1 : 0] new_tag =
           (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
           `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
            write_in_mem(1, set + (new_tag << `CACHE_SET_SIZE));
            _wait(1);
        end
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE - 1 : 0] = (tag <<
        `CACHE_LINE_SIZE_BYTE) + line;
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] = 1;
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] = 0;
        cache_addr_use[set][1] = 1;
        cache_addr_use[set][0] = 0;
    end
endtask
```

Листинг 9. read_in_mem

Но перед этим, если в ней уже были данные, на которых стоял флаг dirty, эти старые данные переписываются в память. Для этого вызывается write_in_mem(листинг 10), который отправляет памяти 3 по шине C2(запрос на запить в память), адрес по шине A2 и в этот же такт начинает пересылать данные по шине D2. И в следующие 7 тактов отправляет остальные порции данных.

```
task automatic write_in_mem(input num_in_set,
```

```
reg[`CACHE_TAG_SIZE + `CACHE_SET_SIZE - 1 : 0] addr);
    A2 = addr;
    C2_1 = 3;
    for (int t = 0; t < (`CACHE_LINE_SIZE / 2); t++) begin
        reg[`DATA2_BUS_SIZE - 1 : 0] data_1 = 0;
        for (int s = 0; s < 16; s++) begin
            data_1 += ((cache[(set << 1) +
            num_in_set][t * 16 + s]) << s);
        end
        D2_1 = data_1;
        _wait(1);
        A2 = 'hz;
        C2_1 = 'hz;
    end
    D2_1 = 'hz;
    while (!(C2 === 1)) _wait(1);
    _wait(1);
    C2_1 = 0;
endtask
```

Листинг 10. write_in_mem

Дальше кеш отвечает памяти в блоке answer_CPU(листинг 11). Он посылает CPU 7 по шине C1 и отпраляет данные.

```
task automatic answer_CPU(input int count_bytes); //+
    C1_1 = 7;
    if (count_bytes == 1 || count_bytes == 2) begin
        D1_1 = ans_cpu[0];
        _wait(1);
    end else begin
        D1_1 = ans_cpu[0];
        _wait(1);
        D1_1 = ans_cpu[1];
        _wait(1);
    end
    D1_1 = 'hz;
    C1_1 = 'hz;
endtask
```

Листинг 11. answer_CPU

Запись:

При записи кэш получает сигнал по шине C1 равный 5, 6 или 7 в зависимости от того, сколько байтов нужно записать, тогда срабатывает условие в блоке always(листинг 5), и передается также tag и set по шине A1 и данные по шине D1 в первый такт и offset по шине A1(и еще порция данных, если команда WRITE32) в следующий.

В этот же так вызывается task _WRITE(листинг 12). Он сначала запускает write_in_cache, и после его выполнения отправляем на один такт 7 по шине C1, чтобы сообщить CPU, что запись закончилась.

```
task automatic _WRITE(input int count_bytes);
    _wait(4);
    write_in_cache(count_bytes);
    C1_1 = 7;
    _wait(1);
    C1_1 = 'hz;
endtask
```

Листинг 12. _WRITE

write_in_cache(листинг 13) проверяет, есть нужная линия в кэше. Если есть записывает туда данные, которые были считаны до этого, если нет запускает read_in_mem, который записывает нужную линию в кэш(его работа была описана раньше), а потом снова запускает write_in_cache, чтобы записать в нужную линию данные.

```
task automatic write_in_cache(input int count_bytes);
    tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
    set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
    offset = addr % (1 <<`CACHE_OFFSET_SIZE);
    if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] === tag
    && cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE
    + 1] == 1) begin
        cache_req++;
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] = 1;
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
        for (int b = 0; b < count_bytes; b++) begin
            for (int c = 0; c < 8; c++) begin
                cache[(set << 1)][(offset + b) * 8 + c]
                = data[b * 8 + c];
            end
        end
    end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] === tag
    && cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE + 1] == 1) begin
        cache_req++;
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] = 1;
        cache_addr_use[set][0] = 0;
        cache_addr_use[set][1] = 1;
        for (int b = 0; b < count_bytes; b++) begin
            for (int c = 0; c < 8; c++) begin
```

```
                cache[(set << 1) + 1][(offset + b) * 8 + c]
                = data[b * 8 + c];
            end
        end
    end else begin
        cache_miss++;
        read_in_mem();
        _wait(1);
        write_in_cache(count_bytes);
    end
endtask
```

Листинг 13. write_in_cache

INVALIDATE_LINE:

В таком случае кэш получает сигнал по шине C1 равный, тогда срабатывает условие в блоке always(листинг 5), и передается также tag и set по шине A1 в первый такт и offset по шине A1 в следующий. В этот же такт запускается task _INVALIDATE_LINE(листинг 14). В нем проверяется, есть ли линия с указанным адресом в кэше, и если есть, она стирается из кэша и записывается в память с помощью write_in_mem.

```
task automatic _INVALIDATE_LINE();
    tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
    set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
    offset = addr % `CACHE_OFFSET_SIZE;
    if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag) begin
        if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] == 1 &&
        cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] == 1) begin
            reg[`CACHE_TAG_SIZE - 1 : 0] new_tag
            = (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE +
            `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
            write_in_mem(0, set + ( new_tag <<
            `CACHE_SET_SIZE));
            _wait(1);
        end
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
        cache[(set << 1)] = 0;
    end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
    `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag) begin
        if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE] == 1 &&
        cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
        `CACHE_TAG_SIZE + 1] == 1) begin
            reg[`CACHE_TAG_SIZE - 1 : 0] new_tag
            = (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
            `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
```

```
            write_in_mem(0, set + ( new_tag <<
            `CACHE_SET_SIZE));
            _wait(1);
        end
        cache_addr_use[set][0] = 0;
        cache_addr_use[set][1] = 1;
        cache[(set << 1) + 1] = 0;
    end
endtask
```

Листинг 14. _INVALIDATE_LINE

Reset(листинг 15):

Все кэш линии и cache_addr_use заполняются нулями. Эта функция запускается каждый раз в самом начале выполнения программы в блоке initial.

```
always @(posedge CLK && RESET == 1) begin
    reset();
end

task automatic reset();
    for (int r = 0; r < `CACHE_LINE_COUNT; r++) begin
        cache[r] = 0;
    end
    for (int r = 0; r < `CACHE_LINE_COUNT; r++) begin
        cache_addr_use[r][0] = 0;
        cache_addr_use[r][1] = 0;
    end
endtask
```

Листинг 15. Reset

C_DUMP(листинг 16):

Кэш выводится в файл DUMP_C.ext и количество кеш-попаданий, кэш-промахов и запросов в кэш в консоль.

```
    always @(posedge CLK && C_DUMP == 1) begin
        fd = $fopen ("DUMP_C.ext", "w");
        for (int j = 0; j < `CACHE_LINE_COUNT; j++)
            $fdisplay (fd, "#%d# %b", j, cache[j]);
        $fclose(fd);
        $display("cache_hit, miss, sum", cache_req - cache_miss,
        cache_miss, cache_req);
    end
```

Листинг 16. C_DUMP

3. MemCTR

Этот модуль принимает запросы от кэша и возвращает ему данные.

Переменные и регистры(листинг 17):

mem – сама память

addr – регистр для хранения адреса

fd – для вывода в файл

SEED – переменная заданная в условии лабораторной для генерации памяти

```
    reg[7:0]  mem[`MEM_SIZE - 1 : 0];
    integer SEED = 225526;
    int fd;
    reg[`CACHE_ADDR_SIZE - 1 : 0] addr;
```

Листинг 17. Переменные и регистры

Чтение:

Когда в память по шине C2 приходит 2(запрос на чтение), в блоке always(листинг 18) срабатывает условие и запускается task _READ_LINE(листинг 19). В нем память принимает тэг и сет, тратит на это такт, ждет еще 99 и после этого начинает отравлять данные.

```
always @(posedge CLK) begin
    if (C2 === 2) begin
        _READ_LINE();
    end else if (C2 === 3) begin
        _WRITE_LINE();
    end
end
```

Листинг 18. Блок always

```
task automatic _READ_LINE();
    addr = A2 << `CACHE_OFFSET_SIZE;
    _wait(1);
    C2_1 = 0;
    _wait(99);
    C2_1 = 1;
    for (int w = 0; w < 8; w++) begin
        D2_1 = (mem[addr + w * 2 + 1] << 8)
        + (mem[addr + w * 2]);
        _wait(1);
    end
    C2_1 = 'hz;
    D2_1 = 'hz;
endtask
```

Листинг 19. _READ_LINE

Запись:

Когда в память по шине C2 приходит 3(запрос на запись), в блоке always(листинг 18) срабатывает условие и запускается task _WRITE_LINE (листинг 20). В нем память принимает тэг и сет и данные за 8 тактов, и ждет еще 92 и после этого посылает по C2 сигнал о том, что запись окончена.

```
task automatic _WRITE_LINE();
    addr = A2 << `CACHE_OFFSET_SIZE;
    for (int d = 0; d < 8; d++) begin
        mem[addr + d * 2] = D2 % (1 << 8);
        mem[addr + d * 2 + 1] = D2 >> 8;
        _wait(1);
        C2_1 = 0;
    end
    _wait(92);
    C2_1 = 1;
    _wait(1);
    C2_1 = 'hz;
endtask
```

Листинг 20. _WRITE_LINE

Reset(листинг 21):

Память заполняется случайными данными. Эта функция запускается каждый раз в самом начале выполнения программы в блоке initial.

```
always @(posedge CLK && RESET == 1) begin
    reset();
end

task automatic reset();
    for (int h = 0; h < `MEM_SIZE; h += 1)
        mem[h] = $random(SEED)>>16;
endtask
```

Листинг 21. Reset

M_DUMP(листинг 22):

Память выводится в файл DUMP_M.ext.

```
always @(posedge CLK && M_DUMP == 1) begin
    fd = $fopen ("DUMP_M.ext", "w");
    for (int e = 0; e < `MEM_SIZE; e++)
```

```
        $fdisplay (fd, "#%d# %b", e, mem[e]);
    $fclose(fd);
end
```

Листинг 22. M_DUMP

8. Воспроизведение задачи на Verilog

Чтобы воспроизвести заданную задачу был написан initial в модуле CPU(листинг 23). В нем исполняется задача, с задержками для операций(сложение, присвоение значения и т.д.). В конце есть закомментированный код, который выводит записанные значения c(для проверки).

В конце выводятся:

Количество тактов – 4952948

Количество обращений к кэшу – 249600

Количество кэш-попаданий – 230698

Количество кэш-промахов – 18902

Процент попадания – 92.4271%

```
initial begin
    C_DUMP = 0;
    M_DUMP = 0;
    pa = 0;
    _wait(1);//int8 *pa = a;
    pc = `M * `K * `SIZE_A + `N * `K * `SIZE_B;
    _wait(1); //int32 *pc = c;
    _wait(1); // initialization y
    for (int y = 0; y < `M; y++) begin
        _wait(1); //start of a new loop iteration
        _wait(1); // initialization x
        for (int x = 0; x < `N; x++) begin
            _wait(1); //start of a new loop iteration
            s = 0;
            _wait(1); //new variable
            pb = `M * `K * `SIZE_A;
            _wait(1); //int16 *pb = b;
            _wait(1); // initialization k
            for (int k = 0; k < `K; k++) begin
                _wait(1); //start of a new loop iteration
                READ8(x, k); // a
                READ16(x, k); // b
                s += data_a * data_b;
                _wait(1 + 5 + 1 + 1); // s += pa[k] * pb[x];
                pb += `N * `SIZE_B;
```

```
                _wait(1); // +
            end
            _wait(1); // pc[x] = s;
            WRITE32(x); // c
        end
        pa += `K * `SIZE_A;
        _wait(1); //pa += K;
        pc += `N * `SIZE_C;
        _wait(1); //pc += N;
    end
    _wait(1); //exit out function mmul;
    $display(count_tact);
    C_DUMP = 1;
    _wait(1);
    C_DUMP = 0;

    // pc = `M * `K * `SIZE_A + `N * `K * `SIZE_B;
    // for (int r = 0; r < `M; r++) begin
    //     for (int y = 0; y < `N; y++) begin
    //         READ32(pc);
    //         $display("c = %0d, addr_c = %d", data_c, pc);
    //         pc += 4;
    //     end
    //     $finish;
    // end

    $finish;
end
```

Листинг 23. Inititial в модуле CPU

9. Сравнение полученных результатов

Количества обращений к кэшу, кэш-попаданий и кэш-промахов, полученные в Verilog и в аналитическом решении, совпадают

Количества тактов отличаются незначительно из-за, того что некоторые задержки при передаче данных в реализации в Verilog были не учтены в аналитическом решении.

10.Листинг кода

```
`define MEM_SIZE (1 << 18)
`define CACHE_SIZE (1 << 11)
`define CACHE_LINE_SIZE (1 << 4)
`define CACHE_LINE_COUNT (1 << 7)
`define CACHE_WAY 2
`define CACHE_SETS_COUNT (1 << 6)
`define CACHE_TAG_SIZE 8
`define CACHE_SET_SIZE 6
`define CACHE_OFFSET_SIZE 4
`define CACHE_ADDR_SIZE 18
```

```
`define ADDR1_BUS_SIZE 14
`define ADDR2_BUS_SIZE 14
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2

`define SIZE_A 1
`define SIZE_B 2
`define SIZE_C 4

`define M 64
`define N 60
`define K 32

module TestRam();
    reg CLK = 0;
    reg C_DUMP;
    reg M_DUMP;
    reg RESET = 0;
    reg[`ADDR1_BUS_SIZE - 1 : 0] A1;
    reg[`ADDR2_BUS_SIZE - 1 : 0] A2;
    wire[`DATA1_BUS_SIZE - 1 : 0] D1;
    wire[`DATA2_BUS_SIZE - 1 : 0] D2;
    wire[`CTR1_BUS_SIZE - 1 : 0] C1;
    wire[`CTR2_BUS_SIZE - 1 : 0] C2;

    CPU CPU(C_DUMP, M_DUMP, A1, D1, C1, CLK);
    Cache Cache(A2, D1, C1, D2, C2, A1, RESET, C_DUMP, CLK);
    MemCTR MemCTR( D2, C2, A2, RESET, M_DUMP, CLK);
    int i;

    task automatic _wait(input int count);
        for (i = 0; i < count; i++) @(posedge CLK);
    endtask

    always #1 CLK = ~CLK;

endmodule

module CPU(output reg C_DUMP,
    output reg M_DUMP,
    output reg[`ADDR1_BUS_SIZE - 1 : 0] A1,
    inout wire[`DATA1_BUS_SIZE - 1 : 0] D1,
    inout wire[`CTR1_BUS_SIZE - 1 : 0] C1,
    input reg CLK
    );
    reg[`DATA1_BUS_SIZE - 1 : 0] D1_1 = 'hz;
    reg[`CTR1_BUS_SIZE - 1 : 0] C1_1 = 'hz;

    assign D1 = D1_1;
    assign C1 = C1_1;

    int count_tact = 0;
    int data_a = 0;
    int data_b = 0;
    int data_c = 0;
    int s = 0;
    int pa;
```

```
    int pb;
    int pc;

    task automatic _wait(input int count);
        for (int i = 0; i < count; i++) @(posedge CLK);
    endtask

    task automatic READ8(int x, int k);
        C1_1 = 1;
        A1 = (pa + k * `SIZE_A) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pa + k * `SIZE_A) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_a = D1;
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask

    task automatic READ16(int x, int k);
        C1_1 = 2;
        A1 = (pb + x * `SIZE_B) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pb + x * `SIZE_B) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_b = D1;
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask

    task automatic READ32(int aaaa);
        C1_1 = 3;
        A1 = aaaa >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = aaaa % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        data_c = D1;
        _wait(1);
        data_c += (D1 << 16);
        _wait(1);
        C1_1 = 0;
        _wait(1);
    endtask

    task automatic WRITE8(int x);
        C1_1 = 7;
        D1_1 = s;
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
```

```
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask

    task automatic WRITE16(int x);
        C1_1 = 7;
        D1_1 = s;
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask

    task automatic WRITE32(int x);
        C1_1 = 7;
        D1_1 = s % (1 << 16);
        A1 = (pc + x * `SIZE_C) >> `CACHE_OFFSET_SIZE;
        _wait(1);
        D1_1 = (s >> 16);
        A1 = (pc + x * `SIZE_C) % (1 << `CACHE_OFFSET_SIZE);
        _wait(1);
        C1_1 = 'hz;
        D1_1 = 'hz;
        while (!(C1 === 7)) _wait(1);
        _wait(1);
        C1_1 = 0;
    endtask

    initial begin
        C_DUMP = 0;
        M_DUMP = 0;
        pa = 0;
        _wait(1);//int8 *pa = a;
        pc = `M * `K * `SIZE_A + `N * `K * `SIZE_B;
        _wait(1); //int32 *pc = c;
        _wait(1); // initialization y
        for (int y = 0; y < `M; y++) begin
            _wait(1); //start of a new loop iteration
            _wait(1); // initialization x
            for (int x = 0; x < `N; x++) begin
                _wait(1); //start of a new loop iteration
                s = 0;
                _wait(1); //new variable
                pb = `M * `K * `SIZE_A;
                _wait(1); //int16 *pb = b;
                _wait(1); // initialization k
                for (int k = 0; k < `K; k++) begin
                    _wait(1); //start of a new loop iteration
                    READ8(x, k); // a
```

```
                    READ16(x, k); // b
                    s += data_a * data_b;
                    _wait(1 + 5 + 1 + 1); // s += pa[k] * pb[x];
                    pb += `N * `SIZE_B;
                    _wait(1); // +
                end
                _wait(1); // pc[x] = s;
                WRITE32(x); // c
            end
            pa += `K * `SIZE_A;
            _wait(1); //pa += K;
            pc += `N * `SIZE_C;
            _wait(1); //pc += N;
        end
        _wait(1); //exit out function mmul;
        $display(count_tact);
        C_DUMP = 1;
        _wait(1);
        C_DUMP = 0;

        // pc = `M * `K * `SIZE_A + `N * `K * `SIZE_B;
        // for (int r = 0; r < `M; r++) begin
        //     for (int y = 0; y < `N; y++) begin
        //         READ32(pc);
        //         $display("c = %0d, addr_c = %d", data_c, pc);
        //         pc += 4;
        //     end
        //     $finish;
        // end

        $finish;
    end

    always @(posedge CLK) count_tact++;

endmodule


module Cache(
    output reg[`ADDR2_BUS_SIZE - 1 : 0] A2,
    inout wire[`DATA1_BUS_SIZE - 1 : 0] D1,
    inout wire[`CTR1_BUS_SIZE - 1 : 0] C1,
    inout wire[`DATA2_BUS_SIZE - 1 : 0] D2,
    inout wire[`CTR2_BUS_SIZE - 1 : 0] C2,
    input reg[`ADDR1_BUS_SIZE - 1 : 0] A1,
    input reg RESET,
    input reg C_DUMP,
    input reg CLK
    );

    reg[`DATA1_BUS_SIZE - 1 : 0] D1_1 = 'hz;
    reg[`CTR1_BUS_SIZE - 1 : 0] C1_1 = 'hz;
    reg[`DATA2_BUS_SIZE - 1 : 0] D2_1 = 'hz;
    reg[`CTR2_BUS_SIZE - 1 : 0] C2_1 = 'hz;

    assign D1 = D1_1;
    assign C1 = C1_1;
    assign D2 = D2_1;
    assign C2 = C2_1;
```

```
    reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0]
cache[`CACHE_LINE_COUNT - 1 : 0];
    reg[`CACHE_WAY - 1 : 0] cache_addr_use[`CACHE_SETS_COUNT - 1 : 0];
    reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0] line;
    reg[2 + `CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BYTE - 1 : 0] data;
    reg[`CACHE_ADDR_SIZE - 1 : 0] addr;
    reg[`CACHE_TAG_SIZE - 1 : 0] tag;
    reg[(`CACHE_SET_SIZE * 2) - 1: 0] set;
    reg[`CACHE_OFFSET_SIZE - 1 : 0] offset;
    int fd;
    reg[15:0] ans_cpu[1:0];
    int cache_req = 0;
    int cache_miss = 0;
    reg[`CTR1_BUS_SIZE - 1 : 0] cur_c1;

    initial begin
        A2 = 0;
        reset();
    end

    always @(posedge CLK && RESET == 1) begin
        reset();
    end

    always @(posedge CLK && C_DUMP == 1) begin
        fd = $fopen ("DUMP_C.ext", "w");
        for (int j = 0; j < `CACHE_LINE_COUNT; j++)
            $fdisplay (fd, "#%d# %b", j, cache[j]);
        $fclose(fd);
        $display("cache_hit, miss, sum",  cache_miss, cache_req - cache_miss,
cache_req);
    end

    always @(posedge CLK) begin
        if (C1 === 1 || C1 === 2 || C1 === 3) begin
            cur_c1 = C1;
            addr = A1;
            _wait(1);
            addr = (addr << `CACHE_OFFSET_SIZE) + A1;
            _wait(1);
            C1_1 = 0;
            if (cur_c1 === 1) begin
                _READ(1);
            end else if (cur_c1 === 2) begin
                _READ(2);
            end else begin
                _READ(4);
            end
        end else if (C1 === 5 || C1 === 6 || C1 === 7) begin
            cur_c1 = C1;
            addr = A1;
            data = D1;
            _wait(1);
            addr = (addr << `CACHE_OFFSET_SIZE) + A1;
            if (cur_c1 === 5) begin
                _wait(1);
                C1_1 = 0;
                _WRITE(1);
```

```
            end else if (cur_c1 === 6) begin
                _wait(1);
                C1_1 = 0;
                _WRITE(2);
            end else begin
                data = (data << 16) + D1;
                _wait(1);
                C1_1 = 0;
                _WRITE(4);
            end
            D1_1 = 'hz;
        end else if (C1 === 4) begin
            addr = A1;
            _wait(1);
            addr = (addr << `CACHE_OFFSET_SIZE) + A1;
            _wait(1);
            C1_1 = 0;
            _INVALIDATE_LINE();
            C1_1 = 'hz;
        end
    end

    task automatic _wait(input count);
        for (int p = 0; p < count; p++) @(posedge CLK);
    endtask

    task automatic reset();
        for (int r = 0; r < `CACHE_LINE_COUNT; r++) begin
            cache[r] = 0;
        end
        for (int r = 0; r < `CACHE_LINE_COUNT; r++) begin
            cache_addr_use[r][0] = 0;
            cache_addr_use[r][1] = 0;
        end
    endtask

    task automatic _READ(input int count_bytes);
        _wait(4);
        read_in_cache(count_bytes);
        C1_1 = 7;
        _wait(1);
        answer_CPU(count_bytes);
        C1_1 = 'hz;
    endtask

    task automatic _WRITE(input int count_bytes);
        _wait(4);
        write_in_cache(count_bytes);
        C1_1 = 7;
        _wait(1);
        C1_1 = 'hz;
    endtask

    task automatic read_in_cache(input int count_bytes);
        ans_cpu[0] = 0;
        ans_cpu[1] = 0;
        tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
        set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
        offset = addr % (1 << `CACHE_OFFSET_SIZE);
```

```
        if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 :
`CACHE_LINE_SIZE_BYTE] == tag
        && cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1] ==
1) begin
            cache_req++;
            cache_addr_use[set][0] = 1;
            cache_addr_use[set][1] = 0;
            fill_ans_cpu(0, count_bytes);
        end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
`CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag
        && cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1]
== 1) begin
            cache_req++;
            cache_addr_use[set][0] = 0;
            cache_addr_use[set][1] = 1;
            fill_ans_cpu(1, count_bytes);
        end else begin
            cache_miss++;
            read_in_mem();
            _wait(1);
            read_in_cache(count_bytes);
        end
    endtask

    task automatic fill_ans_cpu(int num_in_set, int count_bytes);
        if (count_bytes == 1) begin
            for (int v = 0; v < 8; v++) begin
                ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
num_in_set][offset * 8 + v]) << v);
            end
        end else if (count_bytes == 2) begin
            for (int v = 0; v < 8; v++) begin
                ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
num_in_set][(offset + 1) * 8 + v]) << v);
            end
            ans_cpu[0] = ans_cpu[0] << 8;
            for (int v = 0; v < 8; v++) begin
                ans_cpu[0] = ans_cpu[0] + ((cache[(set << 1) +
num_in_set][offset * 8 + v]) << v);
            end
        end else begin
            int new_set = set;
            new_set = set << 1;
            for (int u = 0; u < 2; u++) begin
                for (int v = 0; v < 8; v++) begin
                    ans_cpu[u] = ans_cpu[u] + ((cache[new_set +
num_in_set][(offset + 1 + u * 2) * 8 + v]) << v);
                end
                ans_cpu[u] = ans_cpu[u] << 8;
                for (int v = 0; v < 8; v++) begin
                    ans_cpu[u] = ans_cpu[u] + ((cache[new_set +
num_in_set][(offset + u * 2) * 8 + v]) << v);
                end
            end
        end
    endtask

    task automatic read_in_mem();
        C2_1 = 2;
```

```
        A2 = addr >> `CACHE_OFFSET_SIZE;
        _wait(1);
        C2_1 = 'hz;
        while (!(C2 === 1)) _wait(1);
        line = 0;
        for (int a = 0; a < 8; a++) begin
            line = line + (D2 << (16 * a));
            _wait(1);
        end
        if (cache_addr_use[set][0] == 0) begin
            if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] ==
1 &&
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1] ==
1) begin
                reg[`CACHE_TAG_SIZE - 1 : 0] new_tag = (cache[(set <<
1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
                write_in_mem(0, set + (new_tag << `CACHE_SET_SIZE));
                _wait(1);
            end
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 :
0] = (tag << `CACHE_LINE_SIZE_BYTE) + line;
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1] =
1;
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] = 0;
            cache_addr_use[set][0] = 1;
            cache_addr_use[set][1] = 0;
        end else begin
            if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
`CACHE_TAG_SIZE] == 1 &&
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE +
1] == 1) begin
                reg[`CACHE_TAG_SIZE - 1 : 0] new_tag = (cache[(set << 1) +
1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
                write_in_mem(1, set + (new_tag << `CACHE_SET_SIZE));
                _wait(1);
            end
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1
: 0] = (tag << `CACHE_LINE_SIZE_BYTE) + line;
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE +
1] = 1;
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] =
0;
            cache_addr_use[set][1] = 1;
            cache_addr_use[set][0] = 0;
        end
    endtask

    task automatic answer_CPU(input int count_bytes); //+
        C1_1 = 7;
        if (count_bytes == 1 || count_bytes == 2) begin
            D1_1 = ans_cpu[0];
            _wait(1);
        end else begin
            D1_1 = ans_cpu[0];
            _wait(1);
            D1_1 = ans_cpu[1];
            _wait(1);
        end
        D1_1 = 'hz;
```

```
        C1_1 = 'hz;
    endtask

    task automatic write_in_cache(input int count_bytes);
        tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
        set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
        offset = addr % (1 <<`CACHE_OFFSET_SIZE);
        if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 :
`CACHE_LINE_SIZE_BYTE] === tag
        && cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1] ==
1) begin
            cache_req++;
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] = 1;
            cache_addr_use[set][0] = 1;
            cache_addr_use[set][1] = 0;
            for (int b = 0; b < count_bytes; b++) begin
                for (int c = 0; c < 8; c++) begin
                    cache[(set << 1)][(offset + b) * 8 + c] = data[b * 8 +
c];
                end
            end
        end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
`CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] === tag
        && cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1]
== 1) begin
            cache_req++;
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] =
1;
            cache_addr_use[set][0] = 0;
            cache_addr_use[set][1] = 1;
            for (int b = 0; b < count_bytes; b++) begin
                for (int c = 0; c < 8; c++) begin
                    cache[(set << 1) + 1][(offset + b) * 8 + c] = data[b * 8
+ c];
                end
            end
        end else begin
            cache_miss++;
            read_in_mem();
            _wait(1);
            write_in_cache(count_bytes);
        end
    endtask

    task automatic write_in_mem(input num_in_set, reg[`CACHE_TAG_SIZE +
`CACHE_SET_SIZE - 1 : 0] addr);
        A2 = addr;
        C2_1 = 3;
        for (int t = 0; t < (`CACHE_LINE_SIZE / 2); t++) begin
            reg[`DATA2_BUS_SIZE - 1 : 0] data_1 = 0;
            for (int s = 0; s < 16; s++) begin
                data_1 += ((cache[(set << 1) + num_in_set][t * 16 + s]) <<
s);
            end
            D2_1 = data_1;
            _wait(1);
            A2 = 'hz;
            C2_1 = 'hz;
        end
```

```
        D2_1 = 'hz;
        while (!(C2 === 1)) _wait(1);
        _wait(1);
        C2_1 = 0;
    endtask

    task automatic _INVALIDATE_LINE();
        tag = addr >> (`CACHE_SET_SIZE + `CACHE_OFFSET_SIZE);
        set = (addr >> `CACHE_OFFSET_SIZE) % (1 << `CACHE_SET_SIZE);
        offset = addr % `CACHE_OFFSET_SIZE;
        if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 :
`CACHE_LINE_SIZE_BYTE] == tag) begin
            if (cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE] ==
1 &&
            cache[(set << 1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE + 1] ==
1) begin
                reg[`CACHE_TAG_SIZE - 1 : 0] new_tag = (cache[(set <<
1)][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
                write_in_mem(0, set + ( new_tag << `CACHE_SET_SIZE));
                _wait(1);
            end
            cache_addr_use[set][0] = 1;
            cache_addr_use[set][1] = 0;
            cache[(set << 1)] = 0;
        end else if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
`CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE] == tag) begin
            if (cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE +
`CACHE_TAG_SIZE] == 1 &&
            cache[(set << 1) + 1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE +
1] == 1) begin
                reg[`CACHE_TAG_SIZE - 1 : 0] new_tag = (cache[(set << 1) +
1][`CACHE_LINE_SIZE_BYTE + `CACHE_TAG_SIZE - 1 : `CACHE_LINE_SIZE_BYTE]);
                write_in_mem(0, set + ( new_tag << `CACHE_SET_SIZE));
                _wait(1);
            end
            cache_addr_use[set][0] = 0;
            cache_addr_use[set][1] = 1;
            cache[(set << 1) + 1] = 0;
        end
    endtask

endmodule

module MemCTR(
    inout wire[`DATA2_BUS_SIZE - 1 : 0] D2,
    inout wire[`CTR2_BUS_SIZE - 1 : 0] C2,
    input reg[`ADDR2_BUS_SIZE - 1 : 0] A2,
    input reg RESET,
    input reg M_DUMP,
    input reg CLK
    );

    reg[`DATA2_BUS_SIZE - 1 : 0] D2_1 = 'hz;
    reg[`CTR2_BUS_SIZE - 1 : 0] C2_1 = 'hz;

    assign D2 = D2_1;
    assign C2 = C2_1;

    reg[7:0]  mem[`MEM_SIZE - 1 : 0];
```

```
    integer SEED = 225526;
    int fd;
    reg[`CACHE_ADDR_SIZE - 1 : 0] addr;

    initial reset();

    always @(posedge CLK && RESET == 1) begin
        reset();
    end

    always @(posedge CLK && M_DUMP == 1) begin
        fd = $fopen ("DUMP_M.ext", "w");
        for (int e = 0; e < `MEM_SIZE; e++)
            $fdisplay (fd, "#%d# %b", e, mem[e]);
        $fclose(fd);
    end

    always @(posedge CLK) begin
        if (C2 === 2) begin
            _READ_LINE();
        end else if (C2 === 3) begin
            _WRITE_LINE();
        end
    end

    task automatic _wait(input int count);
        for (int m = 0; m < count; m++) @(posedge CLK);
    endtask

    task automatic reset();
        for (int h = 0; h < `MEM_SIZE; h += 1)
            mem[h] = $random(SEED)>>16;
    endtask

    task automatic _READ_LINE();
        addr = A2 << `CACHE_OFFSET_SIZE;
        _wait(1);
        C2_1 = 0;
        _wait(99);
        C2_1 = 1;
        for (int w = 0; w < 8; w++) begin
            D2_1 = (mem[addr + w * 2 + 1] << 8) + (mem[addr + w * 2]);
            _wait(1);
        end
        C2_1 = 'hz;
        D2_1 = 'hz;
    endtask

    task automatic _WRITE_LINE();
        addr = A2 << `CACHE_OFFSET_SIZE;
        for (int d = 0; d < 8; d++) begin
            mem[addr + d * 2] = D2 % (1 << 8);
            mem[addr + d * 2 + 1] = D2 >> 8;
            _wait(1);
            C2_1 = 0;
        end
        _wait(92);
        C2_1 = 1;
        _wait(1);
```

```
        C2_1 = 'hz;
    endtask

endmodule
```

Листинг 24. Verilog

```
#include <bits/stdc++.h>
using namespace std;

const int M = 64;
const int N = 60;
const int K = 32;
const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 1 << 6;
const int CACHE_SET_SIZE = 6;
const int CACHE_OFFSET_SIZE = 4;
const int CACHE_LINE_SIZE = (1 << 4);
const int DATA2_BUS_SIZE = 16;

const int SIZE_A = 1;
const int SIZE_B = 2;
const int SIZE_C = 4;

int count_tact = 0;
int count_cache_hit = 0;
int count_cache_miss = 0;

int cache[CACHE_SETS_COUNT][CACHE_WAY];
int cache_addr_use[CACHE_SETS_COUNT][CACHE_WAY];
int cache_valid[CACHE_SETS_COUNT][CACHE_WAY];
int cache_dirty[CACHE_SETS_COUNT][CACHE_WAY];

int pa;
int pb;
int pc;

void read_in_mem(int set, int tag, int count_bytes) {
    if (cache_addr_use[set][0] == 0) {
        if (cache_dirty[set][0] == 1 && cache_valid[set][0] ==
1) {
            count_tact += 100 + 1; //write in mem
        }
        cache[set][0] = tag;
        cache_valid[set][0] = 1;
        cache_dirty[set][0] = 0;
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
    } else {
        if (cache_dirty[set][1] == 1 && cache_valid[set][1] ==
1) {
            count_tact += 100 + 1; //write in mem
        }
```

```
        cache[set][1] = tag;
        cache_valid[set][1] = 1;
        cache_dirty[set][1] = 0;
        cache_addr_use[set][1] = 1;
        cache_addr_use[set][0] = 0;
    }
}

void check_cache(int addr, int count_bytes, bool is_write) {
    int tag = addr >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    int set = (addr >> CACHE_OFFSET_SIZE) % (1 <<
CACHE_SET_SIZE);
    if (cache[set][0] == tag && cache_valid[set][0] == 1){
        if (is_write) {
            cache_dirty[set][0] = 1;
        }
        cache_addr_use[set][0] = 1;
        cache_addr_use[set][1] = 0;
        count_cache_hit++;
        count_tact += 6 + (count_bytes + 1) / 2;
    } else if (cache[set][1] == tag && cache_valid[set][1] == 1)
{
        if (is_write) {
            cache_dirty[set][1] = 1;
        }
        cache_addr_use[set][1] = 1;
        cache_addr_use[set][0] = 0;
        count_cache_hit++;
        count_tact += 6 + (count_bytes + 1) / 2;
    } else {
        count_cache_miss++;
        read_in_mem(set, tag, count_bytes);
        count_tact += 100 + 4 + 1;
        count_tact += CACHE_LINE_SIZE / DATA2_BUS_SIZE;
    }
}

void mmul() {
    count_tact++;  //int8 *pa = a;
    count_tact++;  //int32 *pc = c;
    pa = 0;
    pc = M * K * SIZE_A + N * K * SIZE_B;
    count_tact++; // initialization y
    for (int y = 0; y < M; y++) {
        count_tact++;  //start of a new loop iteration
        count_tact++; // initialization x
        for (int x = 0; x < N; x++) {
            pb = M * K * SIZE_A;
            count_tact++;  //start of a new loop iteration
            count_tact++;  //int16 *pb = b;
            count_tact++;  //int32 s = 0;
            // count_tact++; // initialization k
            for (int k = 0; k < K; k++) {
```

```
                count_tact++;  //start of a new loop iteration
                count_tact+= (1 + 5 + 1 + 1);  //s +=
pa[k]*pb[x];
                check_cache(pa + k * SIZE_A, SIZE_A, false);
                check_cache(pb + x * SIZE_B, SIZE_B, false);
                count_tact++;  //pb += N;
                pb += N * SIZE_B;
            }
            count_tact++;  //pc[x] = s;
            check_cache(pc + x * SIZE_C, SIZE_C, true);

        }
        pa += K * SIZE_A;
        pc += N * SIZE_C;
        count_tact++;  //pa += K;
        count_tact++;  //pc += N;
    }
    count_tact++;  //exit out of function mmul;
}

int main() {
    mmul();
    cout << "Count tact: " << count_tact << "\n";
    cout << "Cache-hit percentage: ";
    cout << count_cache_hit << " " << count_cache_miss << " " <<
count_cache_miss + count_cache_hit << " ";
    cout << ((float)count_cache_hit / (float)(count_cache_hit +
count_cache_miss) * 100) << "%";
}
```

Листинг 25. Аналитическое решение