

ЛАБОРАТОРНАЯ РАБОТА №2	М3136	2022
Моделирование схем в Verilog	АРХАНГЕЛЬСКИЙ АНДРЕЙ АНДРЕЕВИЧ	

## 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 11.

## Описание

Требуется решить следующую задачу сначала аналитически, а затем на языке Verilog, реализуя систему «процессор-кэш-память»

### ЗАДАЧА:

Имеется следующее определение глобальных переменных и функций:

```
#define M 64
#define N 60
#define K 32
int8 a[M][K];
int16 b[K][N];
int32 c[M][N];

void mmul()
{
    int8 *pa = a;
    int32 *pc = c;
    for (int y = 0; y < M; y++)
    {
        for (int x = 0; x < N; x++)
        {
            int16 *pb = b;
            int32 s = 0;
            for (int k = 0; k < K; k++)
            {
                s += pa[k] * pb[x];
                pb += N;
            }
        }
    }
}
```

```
    pc[x] = s;
  }
  pa += K;
  pc += N;
}
```

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.

Массивы последовательно хранятся в памяти, и первый из них начинается с 0.

Все локальные переменные лежат в регистрах процессора.

По моделируемой шине происходит только обмен данными (не командами).

Определите процент попаданий (число попаданий к общему числу обращений) для кэша и общее время (в тактах), затраченное на выполнение этой функции.

Вариант

Параметры системы (общие)

CPU			
Команды	CPU → Cache	0 – C1_NOP 1 – C1_READ8 2 – C1_READ16 3 – C1_READ32 4 – C1_INVALIDATE_LINE 5 – C1_WRITE8 6 – C1_WRITE16 7 – C1_WRITE32	Команда 4 означает инвалидацию всей кэш-линии, содержащей указанный адрес.  Число в командах означает кол-во бит данных, запрашиваемое данной командой.  Команды, запрашивающие несколько байт, не могут пересекать кэш-линию.
	Cache ← CPU	0 – C1_NOP 7 – C1_RESPONSE	NOP – no operation. Response – ответ на команду.
Кэш (look-through write-back)			
Политика вытеснения		LRU	

Команды	Cache Mem →	0 – <b>C2_NOP</b> 2 – <b>C2_READ_LINE</b> 3 – <b>C2_WRITE_LINE</b>	Команды пишут и читают порциями, равными размеру кэш-линии.
	Cache Mem ←	0 – <b>C2_NOP</b> 1 – <b>C2_RESPONSE</b>	
Служебные биты		V (valid), D (dirty)	Если valid установлен в 0, то данная кэш-линия свободна и состояние остальных битов не важно. dirty означает, что кэш-линия хранит изменённые данные, которые ещё не записаны в память.

Параметры (вариант 1)

<b>Кэш (продолжение)</b>		
Ассоциативность	2 – <b>CACHE_WAY</b>	
Размер тэга адреса	10 бит – <b>CACHE_TAG_SIZE</b>	
Размер кэш-линии	16 байт – <b>CACHE_LINE_SIZE</b>	Размер полезных данных.
Кол-во кэш-линий	64 – <b>CACHE_LINE_COUNT</b>	
<b>Память</b>		
Размер памяти	512 Кбайт – <b>MEM_SIZE</b>	

Размерность шин

Шина	Обозначение	Размерность
D1, D2	<b>DATA1_BUS_SIZE, DATA2_BUS_SIZE</b>	16 бит

Недостающие параметры системы вычисляются следующим образом:

- **CACHE\_SIZE** = **CACHE\_LINE\_SIZE** \* **CACHE\_LINE\_COUNT** = 1024 байта
- **CACHE\_SETS\_COUNT** = **CACHE\_LINE\_COUNT** / **CACHE\_WAY** = 64 / 2 = 32
- **CACHE\_SET\_SIZE** =  $\log_2(\text{CACHE\_SETS\_COUNT})$  = 5 бит
- **CACHE\_OFFSET\_SIZE** =  $\log_2(\text{CACHE\_LINE\_SIZE})$  = 4 бит

- $\text{CACHE\_ADDR\_SIZE} = \text{CACHE\_TAG\_SIZE} + \text{CACHE\_SET\_SIZE} + \text{CACHE\_OFFSET\_SIZE} = 19 \text{ бит}$
- $\text{ADDR1\_BUS\_SIZE} = \text{ADDR2\_BUS\_SIZE} = \text{CACHE\_TAG\_SIZE} + \text{CACHE\_SET\_SIZE} = 15$
- $\text{CTR1\_BUS\_SIZE} = \log_2 8 = 3 \text{ бит}$
- $\text{CTR2\_BUS\_SIZE} = \log_2 3 = 2 \text{ бит}$

## Принципы работы системы

### CXEMA

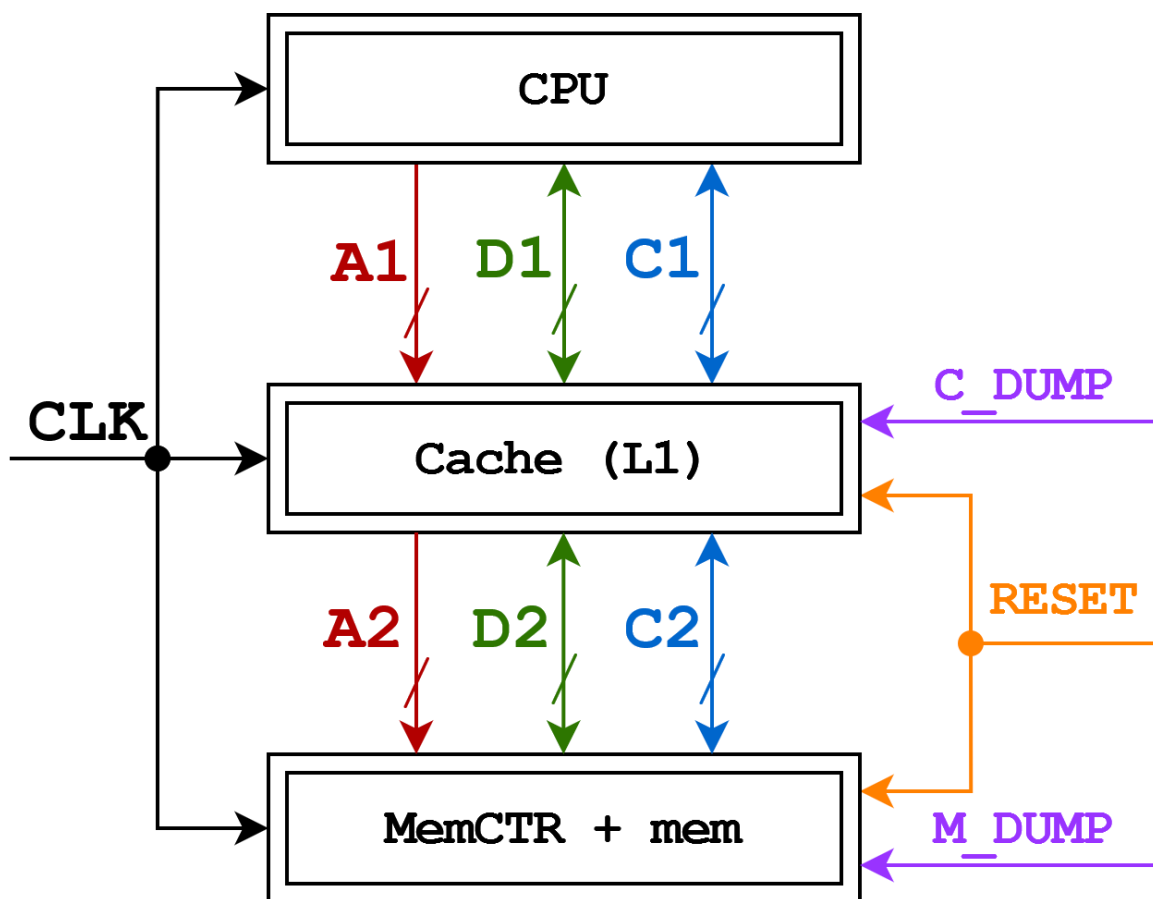


Рисунок 1 – Схема моделируемой системы

Сигналы:

- **CLK** – синхронизация всей схемы
- **RESET** – сброс в начальное состояние
- **\*\_DUMP** – сохранение текущего состояния в файл/вывод в консоль для отладки

Модули:

- **CPU** – модель процессора для верификации работы кэша
- **Cache** – одноуровневый кэш
- **MemCTR (+ mem)** – модель контроллера памяти + модулей памяти для верификации работы кэша

## ПАМЯТЬ

Память получает адрес по шине A2, данные по шине D2 и команду по шине C2.

При этом сама она может отвечать по шине C2 (либо 0, если ещё не готова, либо 7, если готова дать ответ) и использовать для данных шину D2.

Память должна уметь записывать в себя линии и выдавать линию по адресу, если её просят. При этом адрес в память передаётся за один такт, т.к. не передаётся offset (передаётся только 11 бит, а последние 4 – всегда нули).

По условию память начинает отвечать через 100 тактов, после получение команды (первого такта команды).

Также память получает во владение шину, только после запросу к ней. И должна отдать, как только закончит ответ.

## КЭШ

Кэш получает запросы от процессора по шине 1 (A1, D1, C1 аналогично памяти). И отвечает на них по этой же шине (но A1 он может только читать). С памятью кэш общается по шине 2.

Кэш по умолчанию владеет шиной 2, и не владеет шиной 1. То есть он должен забрать владение шиной 2 у памяти, когда та закончила ответ, а также отдать владение шиной 1 процессору, когда сам закончит отвечать.

Данные в кэше хранятся в кэш-линиях (непрерывных кусках памяти) по 16 байт (по условию). Также кэш-линии собраны в кэш-сеты по 2. В какой кэш-сет должен попасть кусок памяти с определённым адресом можно узнать по следующей таблице

tag	set	offset
CACHE_TAG_SIZE	CACHE_SET_SIZE	CACHE_OFFSET_SIZE

Рисунок 2 – Интерпретация адреса кэшем

Set – это номер сета, в который попадёт данные с таким адресом, offset – то, начиная с какого байта в кэш-линии будут лежать данные, tag нужен для проверки того, какие именно данные там лежат (set и offset знаем, значит, хранить нужно только tag, чтобы понять полный адрес данных).

Так же у каждой линии есть флаги **valid** и **dirty**.

- **valid** говорит, актуальна (валидна) ли та информация, которая хранится в кэш-линии (1, если актуальна)

- **dirty** говорит, записаны ли данные, хранящиеся в линии в память (0, если записаны)

Когда кэш приходит запрос, он смотрит, есть ли в соответствующем сете линия, хранящая нужный тэг. Если она есть, то это называется **кэш-попаданием** (иногда в комментариях к коду у меня можно встретить также термин **кэш-хит**), тогда кэш работает с теми данными, которые у него есть. Иначе случается **кэш-промах (кэш-мисс)** и кэшу надо запросить нужную линию у памяти.

В случае кэш-промаха может произойти такое, что у кэша нет свободных линий, чтобы записать. Тогда кэшу нужно выкинуть какую-то кэш-линию, чтобы прочесть на её место нужную. Какую линию заместить подсказывает принцип **LRU**. При таком принципе замещения кэш выкидывает линию, к которой обращались наиболее давно (из всех линий сета). Для этого он хранит вспомогательную структуру, которая позволяет узнать наиболее старую линию.

Кэш по ТЗ должен отвечать через 6 тактов в случае кэш-попадания, и отправлять запрос в память через 4 такта после кэш-промаха.

Так же у кэша политика **look through**, что значит, что процессор обращается только к нему, а он сам решает, нужно ли вызывать память или нет. (Для сравнения при **look aside** процессор обращается сразу и к памяти, и к кэшу, а кэш, если надо, убивает обращение к памяти)

Так же кэш реализует политику **write back**, что значит, что линия в память будет записываться в память только в том случае, когда её понадобится заместить другой (при **write through** все линии всегда записываются и в кэш и в память)

## ПРОЦЕССОР

Процессор по умолчанию владеет шиной 1. По ней он общается с кэшем. Он должен отдать владение ей кэшу после того, как отправил запрос, и забрать его назад, когда получил ответ.

## ПРОТОКОЛ ОБМЕНА ДАННЫМИ ПО ШИНЕ

Команды и ответы на них передаются по шинам за несколько тактов подряд. Но между командой и ответом может быть произвольное кол-во тактов бездействия.

По шине A1 адрес передаётся за 2 такта: в первый такт **tag+set**, во второй - **offset**. По шине A2 передаётся адреса без части **offset** за 1 такт.

По шинам D (D1 и D2) в каждый такт передаётся по 16 бит данных, начиная с младших, little endian.

На линиях команд C (C1 и C2) значение держится всё время передачи команды или ответа.

## Аналитическое решение

Аналитическое решение я выполнил на языке C++ (файл analytics.cpp; запускал с компилятором от Visual Studio 19, но должен сработать любой; стандарт языка – C++11 и выше).

Это аналитическое решение моделирует работу системы, но не учитывает сами данные, его цель – только узнать процент кэш-попаданий и общее количество затраченных на исполнение тактов.

В коде есть подробные комментарии, помогающие его понять, поэтому тут только кратко опишу структуру

### КЛАССЫ (и структуры)

**Memory** – это память. Так как данные в аналитическом решении не важны, память лишь добавляет к глобальному счётчику тактов, сколько она потратит на операции чтения и записи

**CacheAddress** – не более чем удобное представление адреса, в конструкторе разбивающее число на set, offset и tag (согласно Рис. 2)

**CacheLine** – структура кэш-линии. Она хранит только valid, dirty и tag, т.к. данные нам не важны. У неё есть функция reset(), сбрасывающая все поля в 0

**CacheSet** – сет. Внутри имеет 2 кэш-линии и массив последнего их использования. Имеет функции для работы с этим массивом (обновление и получения самой старой линии), а также функцию сброса линий

**Cache** – сам кэш, внутри себя имеет массив кэш-сетов. Также имеет функции чтения, записи и инвалидации, возвращающие то, было ли кэш-попадание или нет. Есть вспомогательная функция, которая возвращает линию, в которую можно будет записать нужную в случае кэш-промаха. Так же в его функциях изменяется глобальный счётчик тактов. Также в нём есть экземпляр Memory, для обращения к последней, когда это потребуется

### MAIN()

В этой функции адаптированный код из задания. Она так же взаимодействует с глобальным счётчиком тактов.

Немного про формирование указателей на элемент массива:

1) Сначала посчитаем размеры массивов (в байтах):

-  $A\_SIZE = M * K$ ;

-  $B\_SIZE = K * N * 2$ ; (2 т.к. он хранит 16-битные числа)

-  $C\_SIZE = M * N * 4$ ; (4, потому что он хранит 32-битные числа)

2) Т.к. массивы лежат последовательно, то указатели первые элементы можно посчитать так:

$pa = 0$

$pb = A\_SIZE$

$pc = A\_SIZE + B\_SIZE$

3) Тогда k-ый элемент массива:

a:  $pa + k$

b:  $pb + 2 * k$

c:  $pc + 4 * k$

## РЕЗУЛЬТАТ АНАЛИТИЧЕСКОГО РЕШЕНИЯ

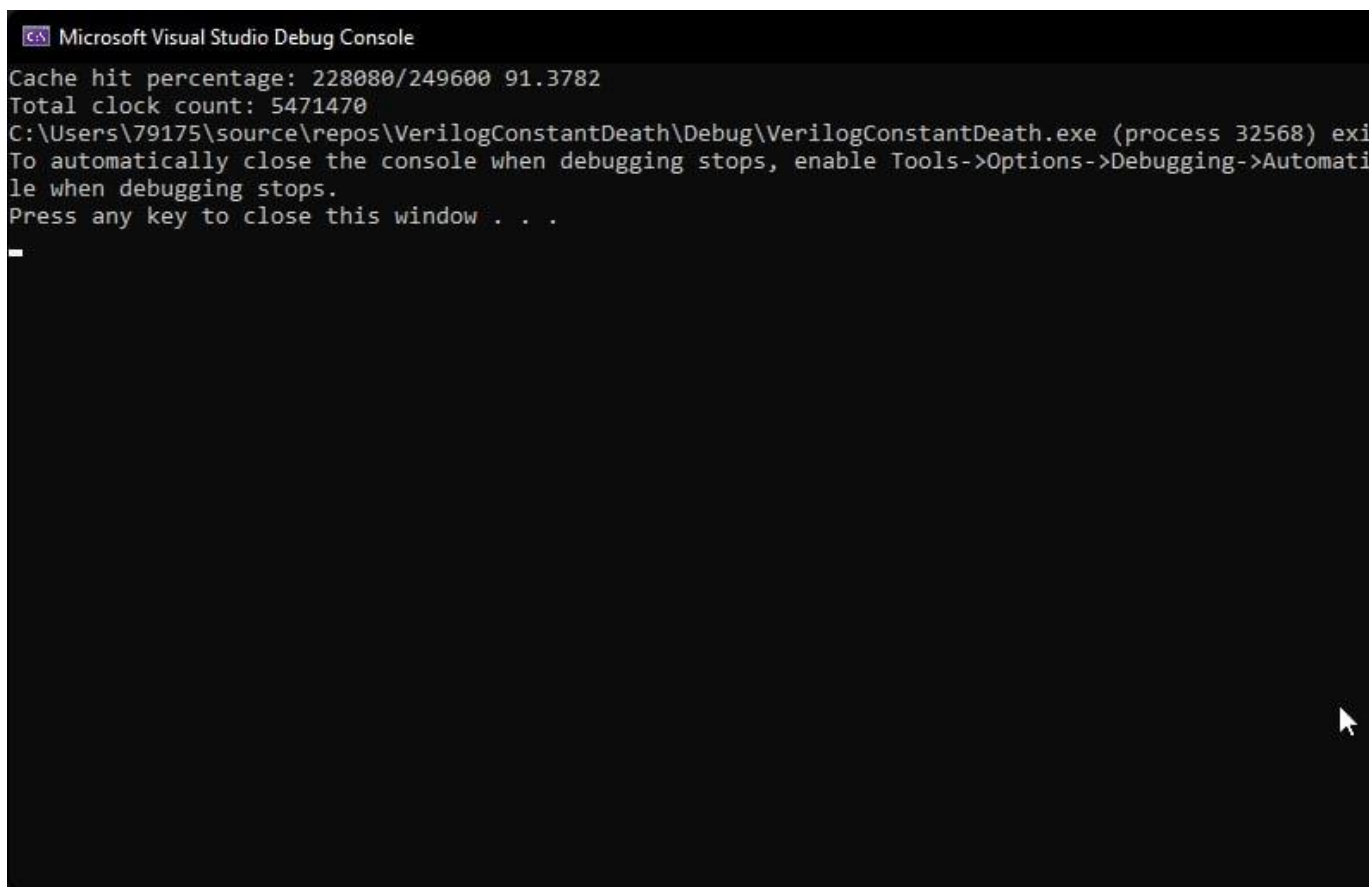


Рисунок 3. – Результат аналитического решения



# Моделирование заданной системы на Verilog

В моей системе по любой шине происходит запись данных только при изменении синхронизации из 1 в 0 (negedge, отрицательный фронт), а чтение – только из 0 в 1 (posedge, положительный фронт).

Когда тот, кто сейчас владеет шиной, хочет передать её другому, то он вешает на неё высокоимпедансное состояние (z). Если это шина команд, то тот, кто хочет завладеть этой шиной в этот момент начинает передавать по ней сигнал 0. Если же это шина данных, то на ней остаётся высокоимпедансное состояние до момента, пока кто-нибудь не захочет по ней что-нибудь передать. На шине адреса между запросами так же высокоимпедансное состояние.

В коде достаточно подробные комментарии, поэтому опишу только общую структуру.

Проект состоит из модулей кэша, памяти и процессора (cache.sv, memory.sv и CPU.sv). А также testbench.sv, связывающего их воедино

## Testbench.sv

Это тестовое окружение, в котором просто проводами соединены остальные модули. А также вычисляется общий CLK системы. CLK вычисляется до тех пор, пока процессор по специальному проводу ответа не вернёт, что он закончил своё задание

## Memory.sv

Память. Основная логика находится в блоке always, который срабатывает по положительному фронту (т.к. память начинает действовать только тогда, когда получит какой-то запрос).

Хранение данных реализовано одномерным массивом восьмибитных регистров.

## Cache.sv

Кэш. Основная логика также срабатывает по положительному фронту по тем же причинам, что и у памяти

Хранение данных происходит в двумерных массивах регистров нужной длины (для тэга, линий, валидности и т.д. для каждого – свой массив)

## CPU.sv

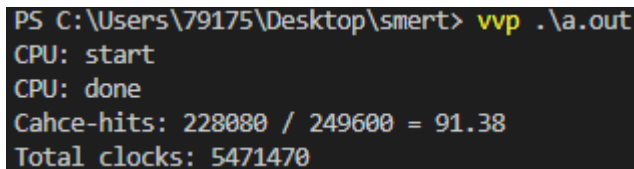
Основной код задачи из ТЗ. Здесь только initial блок, в котором перед самой первой попыткой спросить что-то у кэша я жду отрицательный фронт.

Также все задержки при подсчёте (сложение, умножение и т.д.) реализованы ожиданиями отрицательного фронта, т.к. при выходе из функций (тасок) чтения и записи у нас именно он, ибо мы ждали полтакта, дабы забрать себе шину

## Воспроизведение задачи на Verilog

Сама задача воспроизведена в файле CPU.sv

Во многом, это копия функции main() аналитического решения, за исключением чуть более сложных функций чтения и записи (реализованных в виде task). И более сложного учёта тактов.



```
PS C:\Users\79175\Desktop\smert> vvp .\a.out
CPU: start
CPU: done
Cache-hits: 228080 / 249600 = 91.38
Total clocks: 5471470
```

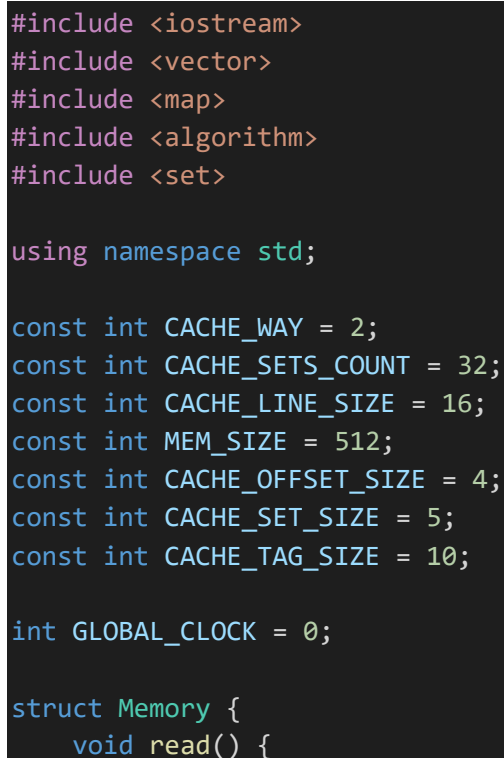
Рисунок 4. – Результат решения на Verilog

## Сравнение полученных результатов

Результаты аналитического решения и решения на Verilog совпали (смотри Рис. 3 и Рис. 4)

## Листинг кода

Аналитическое решение (файл analytics.cpp):



```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <set>

using namespace std;

const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 32;
const int CACHE_LINE_SIZE = 16;
const int MEM_SIZE = 512;
const int CACHE_OFFSET_SIZE = 4;
const int CACHE_SET_SIZE = 5;
const int CACHE_TAG_SIZE = 10;

int GLOBAL_CLOCK = 0;

struct Memory {
    void read() {
```

```

        GLOBAL_CLOCK++; // такт, чтобы получить адрес и саму команду, и начать
что - то делать
        GLOBAL_CLOCK += 100; // какие-то действия в памяти
        GLOBAL_CLOCK += (CACHE_LINE_SIZE * 8) / 16; // ответ
        // ответ занимает количество тактов, равное количеству битов в линии,
делённому на 16 (т.к. шина шириной 16 бит)
    }

    void write() {
        GLOBAL_CLOCK++; // такт, чтобы получить первую порцию данных и саму
команду, и начать что-то делать
        GLOBAL_CLOCK += 100; // какие-то действия в памяти
        GLOBAL_CLOCK++; // такт на ответ
    }
};

struct CacheAddress {
    int tag, set;

    CacheAddress(int tag, int set): tag(tag), set(set) {}
    CacheAddress(int address) {
        tag = address >> (CACHE_OFFSET_SIZE + CACHE_SET_SIZE);
        set = (address >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    }
};

struct CacheLine {
    int valid, dirty, tag;

    CacheLine() {
        reset();
    }

    void reset() {
        valid = 0;
        dirty = 0;
        tag = 0;
    }
};

struct CacheSet {
    vector <CacheLine> lines;
    vector <int> lru;

    CacheSet() {
        lines.assign(CACHE_WAY, CacheLine());
        lru.assign(CACHE_WAY, 1);
        reset();
    }

    void reset() {

```

```

        for (int i = 0; i < lines.size(); ++i) {
            lines[i].reset();
            lru[i] = 1;
        }
    }

    int get_lru_id() {
        if (lru[0] > lru[1]) {
            return 0;
        }
        return 1;
    }

    void update_lru(int id) {
        lru[id] = 0;
        lru[(id + 1) % 2] = 1;
    }
};

class Cache {
private:
    vector <CacheSet> sets;
    Memory memory;

    // эта функция возвращает номер "пустой" линии, в которую можно записать
    // линию из памяти
    int get_empty_line(int set_id) {
        CacheSet& set = sets[set_id];
        for (int i = 0; i < set.lines.size(); ++i) {
            if (!set.lines[i].valid) {
                // есть невалидная строка, значит можно вернуть её
                return i;
            }
        }
    }

    // берём самую старую строку и, если она грязная, то записываем
    // информацию из неё в память
    int lru_line_id = set.get_lru_id();
    CacheLine& line = set.lines[lru_line_id];
    if (line.dirty) {
        memory.write();
        line.dirty = 0;
    }

    return lru_line_id;
}

public:
    Cache() {
        sets.assign(CACHE_SETS_COUNT, CacheSet());
        reset();
    }

```

```

}

void reset() {
    for (int i = 0; i < sets.size(); ++i) {
        sets[i].reset();
    }
}

bool invalidate_line(CacheAddress address) {
    GLOBAL_CLOCK++; // получаем первый такт команды, чтоб начать что-то
    делать

    for (int i = 0; i < sets[address.set].lines.size(); ++i) {
        CacheLine& line = sets[address.set].lines[i];
        if (line.tag == address.tag && line.valid == 1) {
            // кеш-попадание, добавляем к общему количеству 6 тактов отклика
            GLOBAL_CLOCK += 6;

            // если линия грязная, то её надо записать
            if (line.dirty == 1) {
                memory.write();
            }

            // инвалидируем
            line.valid = 0;
            return true;
        }
    }

    // кеш-промах, линии, содержащей такой адрес и нет в кеше
    GLOBAL_CLOCK += 4;
    return false;
}

bool read(CacheAddress address, int size) {
    GLOBAL_CLOCK++; // получаем первый такт команды, чтоб начать что-то
    делать

    bool cache_hit = false;

    for (int i = 0; i < sets[address.set].lines.size(); ++i) {
        CacheLine& line = sets[address.set].lines[i];
        if (line.tag == address.tag && line.valid == 1) {
            // кеш-попадание, добавляем к общему количеству 6 тактов отклика
            cache_hit = true;
            GLOBAL_CLOCK += 6;

            // обновляем массив старости
            sets[address.set].update_lru(i);
        }
    }
}

```

```

    if (!cache_hit) {
        // кеш-промах

        // получаем линию, в которую запишем нужную нам из памяти
        int line_id = get_empty_line(address.set);
        CacheLine& line = sets[address.set].lines[line_id];

        // идём в память
        GLOBAL_CLOCK += 4;
        memory.read();

        // линия теперь валидная и не грязная, т.к. мы её только что считали
        line.valid = 1;
        line.dirty = 0;

        // присваиваем линии соответствующий тег и обновляем массив
последнего использования
        line.tag = address.tag;
        sets[address.set].update_lru(line_id);
    }

    GLOBAL_CLOCK += max(size / 16, 1); // передача данных по 16 бит за такт

    return cache_hit;
}

bool write(CacheAddress address, int size) {
    GLOBAL_CLOCK++; // получаем первый такт команды, чтоб начать что-то
делать

    bool cache_hit = false;

    for (int i = 0; i < sets[address.set].lines.size(); ++i) {
        CacheLine& line = sets[address.set].lines[i];
        if (line.tag == address.tag && line.valid == 1) {
            // кеш-попадание, добавляем к общему количеству 6 тактов отклика
            cache_hit = true;
            GLOBAL_CLOCK += 6;

            // линия теперь грязная, т.к. мы только что записали в неё новую
информацию
            line.dirty = 1;

            // обновляем массив последнего использования
            sets[address.set].update_lru(i);
        }
    }

    if (!cache_hit) {
        int line_id = get_empty_line(address.set);

```

```

        CacheLine& line = sets[address.set].lines[line_id];

        GLOBAL_CLOCK += 4;
        memory.read();

        // линия теперь грязная, т.к. мы только что записали в неё новую
информацию
        line.dirty = 1;
        line.valid = 1;

        // присваиваем нужный тэг и обновляем массив последнего использования
        line.tag = address.tag;
        sets[address.set].update_lru(line_id);
    }

    GLOBAL_CLOCK++; // ответ

    return cache_hit;
}
};

int main() {
    Cache cache;
    cache.reset();

    const int M = 64;
    const int N = 60;
    const int K = 32;

    const int A_SIZE = M * K;
    const int B_SIZE = K * N * 2;
    const int C_SIZE = M * N * 4;

    GLOBAL_CLOCK = 0;

    int cache_calls = 0;
    int cache_hits = 0;

    int pa = 0;
    GLOBAL_CLOCK++; // инициализация переменной

    int pc = A_SIZE + B_SIZE;
    GLOBAL_CLOCK++; // инициализация переменной

    for (int y = 0; y < M; y++) {
        GLOBAL_CLOCK++; // переход на новую итерацию цикла (а на первой итерации
инициализация переменной y)

        for (int x = 0; x < N; x++) {
            GLOBAL_CLOCK++; // переход на новую итерацию цикла (а на первой
итерации инициализация переменной x)

```

```

    int pb = A_SIZE;
    GLOBAL_CLOCK++; // инициализация переменной

    int s = 0;
    GLOBAL_CLOCK++; // инициализация переменной

    for (int k = 0; k < K; k++) {
        GLOBAL_CLOCK++; // переход на новую итерацию цикла (а на первой
// итерации инициализация переменной k)

        int cache_hit_pa = cache.read(CacheAddress(pa + k), 8);
        int cache_hit_pb = cache.read(CacheAddress(pb + 2 * x), 16);
        cache_calls += 2;
        cache_hits += cache_hit_pa + cache_hit_pb;

        //s += pa[k] * pb[x];
        GLOBAL_CLOCK += 6; // сложение и умножение

        pb += 2 * N;
        GLOBAL_CLOCK++; // сложение
    }
    //pc[x] = s;
    int cache_hit_pc = cache.write(CacheAddress(pc + 4 * x), 32);
    cache_calls++;
    cache_hits += cache_hit_pc;
}
pa += K;
GLOBAL_CLOCK++; // сложение

pc += 4 * N;
GLOBAL_CLOCK++; // сложение
}

cout << "Cache hit percentage: " << cache_hits << '/' << cache_calls << ' '
<< (double)cache_hits / (double)cache_calls * 100 << ' ' << endl;
cout << "Total clock count: " << GLOBAL_CLOCK;
}

```

Файл testbench.sv

```

`include "memory.sv"
`include "cache.sv"
`include "CPU.sv"

module testbench
#(
    parameter ADDR1_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,

```



```

    parameter CTR1_BUS_SIZE = 3,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR2_BUS_SIZE = 2
);

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 clk, reset, m_dump, c_dump;
int clk_counter;

reg is_cpu_done;

memory memory_inst(clk, reset, m_dump, a2, d2, c2);
cache cache_inst(clk, reset, c_dump, a1, d1, c1, a2, d2, c2);
CPU cpu_inst(clk, a1, d1, c1, is_cpu_done);

initial begin

    reset = 0;
    m_dump = 0;
    c_dump = 0;
    clk = 0;
    clk_counter = 0;

    while (is_cpu_done == 0 && clk_counter < 100000000) begin
        clk = (clk + 1) % 2;
        if (is_cpu_done < 1 && clk == 1) begin
            clk_counter++;
        end
        #1;
    end

    $display("Total clocks: %0d", clk_counter);

end

endmodule

```

Файл memory.sv

```

module memory
#(

```

```

parameter CACHE_LINE_SIZE = 16,
parameter CACHE_OFFSET_SIZE = 4,
parameter ADDR2_BUS_SIZE = 15,
parameter DATA2_BUS_SIZE = 16,
parameter CTR2_BUS_SIZE = 2,
parameter MEM_SIZE = 512,
parameter MEMORY_RESPONSE_TIME = 100,
parameter _SEED = 225526
)
(
    input CLK,
    input RESET,
    input M_DUMP,
    inout[(ADDR2_BUS_SIZE - 1):0] A2,
    inout[(DATA2_BUS_SIZE - 1):0] D2,
    inout[(CTR2_BUS_SIZE - 1):0] C2
);

reg[7:0] data[(MEM_SIZE * 1024 - 1):0];

// Регистр, привязанный к шине, имеет такое же имя, как и соответствующая ему
шина, но маленькими буквами
reg[(CTR2_BUS_SIZE - 1):0] c2;
reg[(DATA2_BUS_SIZE - 1):0] d2;

assign D2 = d2;
assign C2 = c2;

reg[(ADDR2_BUS_SIZE - 1):0] address_buffer;

int SEED;

int file;

task automatic dump();
begin

    file = $fopen("m_dump.txt", "w");
    for (int line = 0; line < MEM_SIZE * 1024 / CACHE_LINE_SIZE; line++) begin
        $fwrite(file, "address %h: ", (line << CACHE_OFFSET_SIZE));
        for (int i = 0; i < CACHE_LINE_SIZE; i++) begin
            $fwrite(file, "%h ", data[(line << CACHE_OFFSET_SIZE) + i]);
        end
        $fdisplay(file);
    end
    $fclose(file);

end
endtask

task automatic wait_posedge (input int times);

```

```

begin
    repeat (times) begin
        @ (posedge CLK);
    end
end
endtask

task automatic wait_negedge (input int times);
begin
    repeat (times) begin
        @ (negedge CLK);
    end
end
endtask

// отдать шину
task automatic give_away_bus();
begin

    c2 = 'z;
    d2 = 'z;

end
endtask

task automatic reset();
begin

    SEED = _SEED;

    give_away_bus();

    for (int i = 0; i < MEM_SIZE * 1024; i++) begin
        data[i] = $random(SEED) >> 16;
    end

end
endtask

initial begin
    reset();
end

always @ (posedge CLK) begin

    if (RESET == 1) begin
        reset();
    end

    if (M_DUMP == 1) begin
        dump();
    end
end

```

```

end

if (C2 == 2) begin // C2_READ_LINE

    // Читаем адрес в буфер
    address_buffer = A2;

    // забираем владение шиной
    @ (negedge CLK);
    c2 = 0;

    // Ждём нужное количество тактов и начинаем отправлять ответ
    wait_negedge(MEMORY_RESPONSE_TIME);
    c2 = 1;

    for (int i = CACHE_LINE_SIZE - 1; i >= 0; i -= 2) begin
        d2 = (data[(address_buffer << CACHE_OFFSET_SIZE) + i - 1] << 8) +
data[(address_buffer << CACHE_OFFSET_SIZE) + i];

        // ждём следующего такта
        @ (negedge CLK);
    end

    // отдаём владение шиной (такт уже подождали)
    give_away_bus();

end

else if (C2 == 3) begin // C2_WRITE_LINE

    // Адрес держится всё время передачи, можем спокойно записать всё, что надо
    for (int i = CACHE_LINE_SIZE - 1; i >= 0; i -= 2) begin
        data[(A2 << CACHE_OFFSET_SIZE) + i - 1] = (D2 >> 8);
        data[(A2 << CACHE_OFFSET_SIZE) + i] = D2 % (1 << 8);

        // ждём следующий такт
        @ (negedge CLK);
    end

    // т.к. в конце цикла мы всё равно уже подождали такт, можем забрать
управление шиной
    c2 = 0;

    // ждём нужное количество тактов
    wait_negedge(MEMORY_RESPONSE_TIME - (CACHE_LINE_SIZE / 2 - 1));

    // отвечаем, что всё прошло успешно
    c2 = 1;

    // отдаём владение шиной
    @ (negedge CLK);
    give_away_bus();

```

```

        end
    end
endmodule

```

Файл cache.sv

```

module cache
#(
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 32,
    parameter ADDR1_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CACHE_LINE_SIZE = 16,
    parameter CACHE_TAG_SIZE = 10,
    parameter CACHE_SET_SIZE = 5,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR2_BUS_SIZE = 2,
    parameter CACHE_HIT_RESPONSE_TIME = 5,
    parameter CACHE_MISS_RESPONSE_TIME = 3
)
(
    input CLK,
    input RESET,
    input C_DUMP,
    inout[(ADDR1_BUS_SIZE - 1):0] A1,
    inout[(DATA1_BUS_SIZE - 1):0] D1,
    inout[(CTR1_BUS_SIZE - 1):0] C1,
    inout[(ADDR2_BUS_SIZE - 1):0] A2,
    inout[(DATA2_BUS_SIZE - 1):0] D2,
    inout[(CTR2_BUS_SIZE - 1):0] C2
);

    reg[7:0] sets[(CACHE_SETS_COUNT - 1):0][(CACHE_WAY - 1):0][(CACHE_LINE_SIZE - 1):0];
    reg[(CACHE_TAG_SIZE - 1):0] tags[(CACHE_SETS_COUNT - 1):0][(CACHE_WAY - 1):0];
    reg valid[(CACHE_SETS_COUNT - 1):0][(CACHE_WAY - 1):0];
    reg dirty[(CACHE_SETS_COUNT - 1):0][(CACHE_WAY - 1):0];
    reg lru[(CACHE_SETS_COUNT - 1):0][(CACHE_WAY - 1):0];

    // Регистр, привязанный к шине, имеет такое же имя, как и соответствующая ему
    // шина, но маленькими буквами
    reg[(CTR1_BUS_SIZE - 1):0] c1;

```

```

reg[(DATA1_BUS_SIZE - 1):0] d1;

reg[(CTR2_BUS_SIZE - 1):0] c2;
reg[(DATA2_BUS_SIZE - 1):0] d2;
reg[(DATA2_BUS_SIZE - 1):0] a2;

reg[(CACHE_TAG_SIZE - 1):0] tag;
reg[(CACHE_SET_SIZE - 1):0] set;
reg[(CACHE_OFFSET_SIZE - 1):0] offset;

assign D1 = d1;
assign C1 = c1;

assign A2 = a2;
assign D2 = d2;
assign C2 = c2;

reg[31:0] data_buffer;

int c1_buffer;

int cache_hit, empty_line;

int file;

task automatic dump();
begin

    file = $fopen("c_dump.txt", "w");

    for (int cur_set = 0; cur_set < CACHE_SETS_COUNT; cur_set++) begin
        $fdisplay(file, "set %h: ", cur_set);

        for (int cur_line = 0; cur_line < CACHE_WAY; cur_line++) begin
            $fwrite(file, "        line %h: ", cur_line);

            for (int i = 0; i < CACHE_LINE_SIZE; i++) begin
                $fwrite(file, "%h ", sets[cur_set][cur_line][i]);
            end

            $fdisplay(file);
        end

        $fdisplay(file);
    end

    $fclose(file);

end
endtask

```

```

task automatic wait_posedge(input int times);
begin
    repeat (times) begin
        @ (posedge CLK);
    end
end
endtask

task automatic wait_negedge(input int times);
begin
    repeat (times) begin
        @ (negedge CLK);
    end
end
endtask

task automatic read();
begin

    // запоминаем, какая именно команда нам пришла
    c1_buffer = C1;

    if (c1_buffer >= 5 && c1_buffer <= 7) begin // C1_WRITE
        // если это команда на запись, то сохраняем пришедшие данные
        data_buffer = D1;
    end

    // читаем первую часть адреса
    tag = A1 >> CACHE_SET_SIZE;
    set = A1 % (1 << CACHE_SET_SIZE);

    // читаем offset в следующий такт
    @ (posedge CLK);
    offset = A1;

    if (c1_buffer == 7) begin // C1_WRITE32
        // если это команда на запись 32 бит, то надо запомнить ещё 16 бит
        data_buffer += D1 << 16;
    end

    // забираем владение шиной
    @ (negedge CLK);
    c1 = 0;
end
endtask

// отдать шину
task automatic give_away_bus(input bus_id);
begin

    if (bus_id == 1) begin

```

```

        c1 = 'z';
        d1 = 'z';

    end
    else begin

        c2 = 'z';
        d2 = 'z';
        a2 = 'z';

    end

end

endtask

task automatic write_dirty_line_to_memory(input int line_id);
begin

    c2 = 3; // C2_WRITE_LINE
    a2 = (tag << CACHE_SET_SIZE) + set;
    for (int i = CACHE_LINE_SIZE - 1; i >= 0; i -= 2) begin
        d2 = (sets[set][line_id][i - 1] << 8) + sets[set][line_id][i];

        @ (negedge CLK); // отправили первую порцию данных и ждём такт
    end

    // в конце цикла уже подождали такт, поэтому можем сразу отдать владение
    шиной
    give_away_bus(2);

    // Ждём от памяти ответа, что всё записалось
    do begin
        @ (posedge CLK);
    end while (C2 !== 1); // C2_RESPONSE

    // т.к. мы записали эту линию, она больше не хранит незаписанные данные
    dirty[set][line_id] = 0;

end
endtask

// эта задача обрабатывает случай кэш-промаха (просит у памяти нужную линию,
предварительно вымещая самую старую, если это требуется)
task automatic get_line();
begin

    /*
        Далее надо обращаться в память, сразу подождём нужное количество тактов
    */

```



Ждём на один такт меньше указанного, т.к. если линия грязная и её нужно записать в память,  
после того, как мы дождёмся ответа, будет posedge, и надо будет подождать, перед тем,  
как просить память прочитать ещё одну линию.

Но, если мы не записывали грязную линию, то перед запросом на чтение подождём лишний такт

```
*/
wait_negedge(CACHE_MISS_RESPONSE_TIME - 1);

empty_line = -1;

for (int i = 0; i < CACHE_WAY; i++) begin
    if (valid[set][i] == 0 && empty_line == -1) begin
        // Нашли невалидную линию
        empty_line = i;
    end
end

if (empty_line == -1) begin
    // Не нашли невалидную линию, а, значит, надо освобождать (замещать) одну
линию

    for (int i = 0; i < CACHE_WAY; i++) begin
        if (lru[set][i] == 1) begin
            // Нашли наиболее старую линию (ту, которую надо заместить)
            empty_line = i;
        end
    end

    if (dirty[set][empty_line] == 1) begin
        // В линии, которую хотим заместить, хранятся незаписанные в память
данные

        /*
            Ждём один "потерянный" такт и отправляем запрос на чтение

            Этими шинами по умолчанию владеет кэш, а память забирает только на
время ответа на запрос,
            так что можно спокойно передавать по ним данные
        */
        @ (negedge CLK);
        write_dirty_line_to_memory(empty_line);
    end
end

// т.к. до этого мог быть запрос на запись, и, соответственно, сейчас
posedge, надо подождать
@ (negedge CLK);
c2 = 2; // C2_READ_LINE
```

```

a2 = (tag << CACHE_SET_SIZE) + set;

// отдаём владение шиной
@ (negedge CLK);
give_away_bus(2);

// Ждём от памяти ответа
do begin
    @ (posedge CLK);
end while (C2 != 1); // C2_RESPONSE

for (int i = CACHE_LINE_SIZE - 1; i >= 0; i -= 2) begin
    sets[set][empty_line][i - 1] = (D2 >> 8);
    sets[set][empty_line][i] = D2 % (1 << 8);

    // если есть следующая порция данных, то ждём такт
    if (i > 1) begin
        @ (posedge CLK);
    end
end

// забираем владение шиной
@ (negedge CLK);
c2 = 0;

// обновляем массив последнего использования линий
lru[set][empty_line] = 0;
lru[set][(empty_line + 1) % 2] = 1;

tags[set][empty_line] = tag;
valid[set][empty_line] = 1;

end
endtask

task automatic reset();
begin

    give_away_bus(1);

    give_away_bus(2);
    c2 = 0;

    for (int i = 0; i < CACHE_SETS_COUNT; i++) begin
        for (int j = 0; j < CACHE_WAY; j++) begin
            valid[i][j] = 0;
            dirty[i][j] = 0;
            lru[i][j] = 1;
            tags[i][j] = 0;
        end
    end
end

```

```

        end
    endtask

    task automatic send_reading_response(input int line_id);
        begin

            // эта функция не вызывается в posedge, так что не надо ждать лишний такт
            c1 = 7; // C1_RESPONSE

            case (c1_buffer)
                1: // C1_READ8
                    begin
                        d1 = sets[set][line_id][offset];
                    end

                2: // C1_READ16
                    begin
                        d1 = (sets[set][line_id][offset] << 8) + sets[set][line_id][offset +
1];
                    end

                3: // C1_READ32
                    begin
                        // передаём сначала младшие 16 бит
                        d1 = (sets[set][line_id][offset + 2] << 8) +
sets[set][line_id][offset + 3];

                        // через такт старшие
                        @ (negedge CLK);
                        d1 = (sets[set][line_id][offset] << 8) + sets[set][line_id][offset +
1];
                    end
            endcase

        end
    endtask

    task automatic write_data_from_data_buffer(input int line_id);
        begin

            case (c1_buffer)
                5: // C1_WRITE8
                    begin
                        sets[set][line_id][offset] = data_buffer;
                    end

                6: // C1_WRITE16
                    begin
                        sets[set][line_id][offset + 1] = data_buffer % (1 << 8);
                        sets[set][line_id][offset] = data_buffer >> 8;
                    end
            endcase

        end
    endtask

```

```

        end

        7: // C1_WRITE32
        begin
            sets[set][line_id][offset + 3] = data_buffer % (1 << 8);
            sets[set][line_id][offset + 2] = (data_buffer >> 8) % (1 << 8);
            sets[set][line_id][offset + 1] = (data_buffer >> 16) % (1 << 8);
            sets[set][line_id][offset] = data_buffer >> 24;
        end
    endcase

    // теперь эта линия хранит незаписанные в память данные
    dirty[set][line_id] = 1;

end
endtask

initial begin
    reset();
end

always @ (posedge CLK) begin

    if (RESET == 1) begin
        reset();
    end

    if (C_DUMP == 1) begin
        dump();
    end

    if (C1 >= 1 && C1 <= 3) begin // C1_READ

        // читаем адрес и забираем владение шиной
        read();

        cache_hit = 0;

        for (int i = 0; i < CACHE_WAY; i++) begin
            if (tags[set][i] == tag && valid[set][i] == 1) begin
                // Кэш-хит!
                cache_hit = 1;

                // Надо обновить массив, использующийся для определения наиболее
                старой кэш-линии:
                // теперь эта линия - самая свежая, а вторая - самая старая в сети
                (т.к. в сети по ТЗ всего 2 линии)
                lru[set][i] = 0;
                lru[set][(i + 1) % 2] = 1;

                // Ждём положенное количество тактов

```

```

        wait_negedge(CACHE_HIT_RESPONSE_TIME);

        // т.к. мы дождались negedge, можем отправлять данные по шине
        send_reading_response(i);

        // отдаём управление
        @ (negedge CLK);
        give_away_bus(1);

    end
end

if (cache_hit == 0) begin
    // Кэш-промах
    get_line();

    // отправляем ответ
    send_reading_response(empty_line);

    // в конце отдаём управление шиной
    @ (negedge CLK);
    give_away_bus(1);

end

end

else if (C1 == 4) begin // C1_INVALIDATE_LINE
    // читаем адрес и забираем владение шиной
    read();

    cache_hit = 0;

    for (int i = 0; i < CACHE_WAY; i++) begin
        if (tags[set][i] == tag && valid[set][i] == 1) begin
            // нашли линию с таким тегом, значит, это кэш-хит, ждём нужное
            // количество тактов и инвалидируем
            wait_negedge(CACHE_HIT_RESPONSE_TIME);

            // если линия грязная, то отправляем её в память
            if (dirty[set][i] == 1) begin
                write_dirty_line_to_memory(i);
            end

            valid[set][i] = 0;
            cache_hit = 1;
        end
    end

    if (cache_hit == 0) begin
        // это кэш-промах, значит, просто ждём нужное количество тактов
        wait_negedge(CACHE_MISS_RESPONSE_TIME);
    end
end

```

```

end

// уже дождались negedge, отправляем ответ
c1 = 7;

// отдаём управление шиной
@ (negedge CLK);
give_away_bus(1);

end

else if (C1 >= 5 && C1 <= 7) begin // C1_WRITE

// читаем адрес, данные и забираем управление шиной
read();

cache_hit = 0;

for (int i = 0; i < CACHE_WAY; i++) begin
    if (tags[set][i] == tag && valid[set][i] == 1) begin
        // Кэш-хит!

        cache_hit = 1;

        // записываем в нужную линию данные
        write_data_from_data_buffer(i);

        // обновляем массив последнего использования линий
        lru[set][i] = 0;
        lru[set][(i + 1) % 2] = 1;

        // Ждём нужное количество тактов (negedge, т.к. собираемся отправлять
ответ)
        wait_negedge(CACHE_HIT_RESPONSE_TIME);

        // negedge, можем отвечать, что всё хорошо
        c1 = 7; // C1_RESPONSE;

        // отдаём владение шиной
        @ (negedge CLK);
        give_away_bus(1);

    end
end

if (cache_hit == 0) begin
    // Кэш-мисс

    get_line();

    // записываем данные, которые нам прислал процессор
    write_data_from_data_buffer(empty_line);

```

```

        // можем отвечать, что всё хорошо, уже negedge, т.к. мы ждали, чтоб
забрать владение данными
        c1 = 7; // C1_RESPONSE;

        // отдаём владение шиной
        @ (negedge CLK);
        give_away_bus(1);

    end

end

end
endmodule

```

## Файл CPU.sv

```

module CPU
#(
    parameter ADDR1_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CACHE_TAG_SIZE = 10,
    parameter CACHE_SET_SIZE = 5,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter M = 64,
    parameter N = 60,
    parameter K = 32
)
(
    input CLK,
    inout[(ADDR1_BUS_SIZE - 1):0] A1,
    inout[(DATA1_BUS_SIZE - 1):0] D1,
    inout[(CTR1_BUS_SIZE - 1):0] C1,
    output ANS
);

    // Регистр, привязанный к шине, имеет такое же имя, как и соответствующая ему
шина, но маленькими буквами
    reg[(CTR1_BUS_SIZE - 1):0] c1;
    reg[(DATA1_BUS_SIZE - 1):0] d1;
    reg[(ADDR1_BUS_SIZE - 1):0] a1;

    assign C1 = c1;
    assign D1 = d1;
    assign A1 = a1;

```

```

reg ans;
assign ANS = ans;

int A_SIZE, B_SIZE, C_SIZE;
int pa, pb, pc;
int s;
int pa_data, pb_data, pc_data;
int waiting_couter;

int cache_hits, cache_requests;

// отдать шину
task automatic give_away_bus();
begin

    a1 = 'z;
    c1 = 'z;
    d1 = 'z;

end
endtask

task automatic wait_posedge(input int times);
begin
    repeat (times) begin
        @ (posedge CLK);
    end
end
endtask

task automatic wait_negedge(input int times);
begin
    repeat (times) begin
        @ (negedge CLK);
    end
end
endtask

// задача чтения (читаем на выходе int)
task automatic read(input int address, input int command_id, output int out);
begin
    cache_requests++;

    // не надо ждать negedge, т.к. мы никогда не придём сюда в posedge

    // отправляем запрос на запись
    c1 = command_id;
    a1 = address >> CACHE_OFFSET_SIZE;

    // через такт отправляем offset
    @ (negedge CLK);

```



```

a1 = address % (1 << CACHE_OFFSET_SIZE);

// отдаём управление шиной
@ (negedge CLK);
give_away_bus();

waiting_couter = 0;

// ждём ответа от кэша
do begin
    @ (posedge CLK);
    waiting_couter++;
end while (C1 !== 7); // C1_RESPONSE

// если кэш ответил меньше, чем за 100 тактов, то это кэш-попадание
if (waiting_couter < 100) begin
    cache_hits++;
end

// записываем, что нам отправил кэш
out = D1;

if (command_id == 3) begin // C1_READ32
    // если мы запросили 32 бита, что через такт считаем ещё 16 (старших)
    out += D1 << 16;
end

// забираем управление шиной
@ (negedge CLK);
c1 = 0;

end
endtask

// такса записи (на вход данные подаются в виде int)
task automatic write(input int address, input int command_id, input int data);
begin
    cache_requests++;

    // не надо ждать negedge, т.к. мы никогда не придём сюда в posedge

    // отправляем запрос на запись
    c1 = command_id;
    a1 = address >> CACHE_OFFSET_SIZE;
    d1 = data % (1 << 16);

    // через такт отправляем offset
    @ (negedge CLK);
    a1 = address % (1 << CACHE_OFFSET_SIZE);

    if (command_id == 7) begin // C1_WRITING32

```

```

        // если хотим записать 32 бита, то в такт отправки offset отправляем и их
        d1 = data >> 16;
    end

    // отдаём управление шиной
    @ (negedge CLK);
    give_away_bus();

    waiting_couter = 0;

    // ждём ответа от кэша
    do begin
        @ (posedge CLK);
        waiting_couter++;
    end while (C1 !== 7); // C1_RESPONSE

    // если кэш ответил меньше, чем за 100 тактов, то это кэш-попадание
    if (waiting_couter < 100) begin
        cache_hits++;
    end

    // забираем управление шиной
    @ (negedge CLK);
    c1 = 0;

end
endtask

initial begin

    give_away_bus();
    c1 = 0;

    A_SIZE = M * K;
    B_SIZE = K * N * 2;
    C_SIZE = M * N * 4;

    cache_hits = 0;
    cache_requests = 0;
    ans = 0;

    /*
        Вот отсюда начинается функция, которую нужно промоделировать.
        Пусть у CPU синхронизация по negedge
        (т.к. он только отправляет запросы и принимает ответы)
    */
    @ (negedge CLK);

    $display("CPU: start");

    pa = 0;

```

```

wait_negedge(1); // инициализация переменной

pc = A_SIZE + B_SIZE;
wait_negedge(1); // инициализация переменной

for (int y = 0; y < M; y++) begin
    wait_negedge(1); // переход на новую итерацию цикла (а на первой итерации
инициализация переменной y)

    for (int x = 0; x < N; x++) begin
        wait_negedge(1); // переход на новую итерацию цикла (а на первой итерации
инициализация переменной x)

        pb = A_SIZE;
        wait_negedge(1); // инициализация переменной

        s = 0;
        wait_negedge(1); // инициализация переменной

        for (int k = 0; k < K; k++) begin
            wait_negedge(1); // переход на новую итерацию цикла (а на первой
итерации инициализация переменной k)

            read(pa + k, 1, pa_data); // C1_READ8
            read(pb + 2 * x, 2, pb_data); // C1_READ16

            s += pa_data + pb_data;
            wait_negedge(6); // сложение и умножение

            pb += 2 * N;
            wait_negedge(1); // сложение
        end

        write(pc + 4 * x, 7, s); // C1_WRITE32
    end

    pa += K;
    wait_negedge(1); // сложение

    pc += 4 * N;
    wait_negedge(1); // сложение

end

$display("CPU: done");
$display("Cache-hits: %0d / %0d = %.2f", cache_hits, cache_requests,
real'(cache_hits) / real'(cache_requests) * 100);
ans = 1;

end
endmodule

```

