Лабораторная работа №2	M3138	2022
Моделирование схем в Verilog	Мурысин Максим Дмитриевич	

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

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 10. Далее в этом документе Verilog+SystemVerilog обозначается как Verilog.

Описание

Построить кэш, смоделировать систему “процессор-кэш-память” на языке описания Verilog. Решить задачу из условия двумя способами: аналитически и смоделировав в Verilog. Задачей было определить время вычисления в тактах и процент кэш-попаданий для кэша.

Вариант

Вариант 2

Ход работы

Что такое кэш?

Кэш – это память с большой (по сравнению с оперативной памятью) скоростью доступа и сравнительно маленьким объемом. Кэш нужен, чтобы процессор имел быстрый доступ к областям памяти, к которым он часто обращается. Представим, что кэша не было бы. Например, процессору надо сложить два числа. Сложение двух чисел в процессоре работает за 1 такт, а вот скорость доступа до оперативной памяти, может быть, например, 100 тактов. Тогда процессор берет два числа из памяти за 100 тактов, а после этого их складывает за 1 такт. Получается неэффективно. А теперь представим, что у нас есть кэш и мы уже работали с областью памяти, в которой лежали эти два числа (например, мы работаем с массивом, этот массив уже лежит в кэше, а числа, которые мы хотим сложить, лежат в этом самом массиве). Скорость доступа до кэша, например, 5 тактов. Тогда каждое число получим из кэша за 5 тактов, а сложим за 1. Тогда мы получим результат сложения намного быстрее. Для этого и нужен кэш.

Пишу я про кэш процессора, хотя, конечно, кэш используется не только в процессорах.

Почему скорость доступа кэша быстрее?

Большая скорость доступа достигается за счет меньшего объема, того, что кэш находится прямо в процессоре, а также построен на статической памяти, которая быстрее динамической, на которой построена оперативная память.

Что такое уровень кэша?

Кэш часто делят на несколько уровней. Чем ниже уровень кэша, тем больше его скорость и ниже объем. Каждый уровень берет данные с уровня выше. Таким образом, можно кэшировать больше данных, но не сильно терять в скорости доступа. Но мне нужно реализовать систему, в которой только один уровень кэша.

Write-back кэш

Мне нужно было реализовать write-back кэш. Что это такое? Эти слова значат, что кэш записывает данные в память либо тогда, когда его явно попросили, либо тогда, когда у него закончилось место, а ему нужно что-то записать. Тогда он выкидывает что-то из своей памяти согласно политике вытеснения (о которой будет написано позже), но если он выкидывает что-

то, что модифицировалось процессором (то есть, что-то, копии чего нет в оперативной памяти), то он записывает это в оперативную память.

Look-through кэш

Это значит, что процессор имеет доступ к памяти только через кэш. То есть, чтобы получить какие-то данные или что-то записать, процессор дает команду кэшу, а он уже дальше эту команду выполняет. Процессор связан шиной с кэшем, а кэш другой шиной с контроллером памяти.

Бывает еще *look-aside* кэш. Это значит, что процессор, кэш и контроллер памяти находятся на одной шине. Чтобы что-то сделать, кэш дает запрос на шину. А ответить ему уже могут и кэш, и оперативная память.

Мне нужно было реализовать look-through кэш.

Как устроен кэш?

Кэш поделен на одинаковые куски памяти - кэш линии. Запись и чтение в оперативную память/из оперативной памяти происходит порциями по одной кэш линии.

Ассоциативность кэша

Оперативная память тоже делится на непрерывные линии такого же размера. Ассоциативность кэша – это количество линий в кэше, в которые может попасть отдельно взятая линия в памяти. Зачем это нужно? Представим, что любая линия в памяти может попасть в любое место в кэше. Пусть процессор хочет прочитать что-то из линии. Тогда нам придется пробежать весь кэш, чтобы проверить, лежит ли линия в кэше. Это долго. Тогда приходит в голову идея сделать так, чтобы линия могла попасть в ограниченное количество мест в кэше. Тогда по ее адресу в памяти получим те места, где она может быть и их переберем. Делают так: разбивают кэш на наборы. Каждой линии сопоставляется некоторый набор. При этом номер набора, в который попадет линия, можно найти, зная ее адрес в оперативной памяти.

Тогда получается так:

tag	set	offset
CACHE_TAG_SIZE	CACHE_SET_SIZE	CACHE_OFFSET_SIZE

Рисунок 1 – интерпретация адреса кэшем

Пояснение к рисунку 1: мы берем адрес байта в оперативной памяти. Делим его на куски фиксированного размера. Младшие несколько бит считаем offset’ом – смещением внутри линии. Соответственно, размер смещения – это log2(cache_line_length). Следующие считаем номером set’а – набора линий в кэше. Значит, количество бит, из которых состоит номер набора – это log2(cache_set_count). То есть, линия пойдет именно в тот набор, номер которого получается в ее адресе. Всё, что осталось от адреса, считаем tag’ом линии. По tag’у мы будем сравнивать две линии внутри кэша, чтобы понять, ту ли линию, которую нам надо, мы рассматриваем.

В итоге, чтобы найти конкретный байт в кэше, мы берем его адрес в оперативной памяти, делим на куски так, как написано выше. Дальше смотрим в набор, номер которого получился. Внутри набора линейно ищем линию, в которой находится наш байт, сравнивая тэг из адреса с тэгами линий, хранящимися в кэше.

Политика вытеснения

Политика вытеснения – алгоритм, по которому выбирается линия, которая будет выкинута из кэша, когда место в кэше закончилось и что-то надо записать.

Last recently used (LRU) – политика вытеснения, которая состоит в том, что выбрасывается та линия из набора, в который мы пишем, которая дольше не была использована. Использование линии – чтение или запись в нее/из нее данных. Мне нужно было реализовать именно эту политику вытеснения.

Задача

Дан код (код, который перемножает две матрицы), нужно исследовать, сколько тактов будет выполняться это код в системе с заданными параметрами, а также посчитать долю кэш попаданий (относительно всех обращений в память).

Вот данный код:

```
#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;
  }
}
```

Параметры системы и нахождение недостающих

Имеем систему «процессор-кэш-память». Вычисления происходят в процессоре, массивы лежат в памяти последовательно, начиная с первого.

Даны следующие параметры:

Размер кэша (CACHE_SIZE) – 1 килобайт, размер кэш-линии (CACHE_LINE_SIZE) - 32 байта, кол-во бит под хранение индекса набора (BITS_SET) – 4 бита, размер памяти (MEM_SIZE) – 1 Мбайт.

CACHE_SETS_COUNT = 2^(CACHE_SET_SIZE) = 16.

CACHE_LINE_COUNT = CACHE_SIZE / CACHE_LINE_SIZE = 1024B / 32B = 32

CACHE_WAY = CACHE_LINE_COUNT / CACHE_SET_COUNT = 32 / 16 = 2

BITS_ADDR = log2(MEM_SIZE) = 20

BITS_OFFSET = log2(CACHE_LINE_SIZE) = 5

BITS_TAG = BITS_ADDR – BITS_SET – BITS_OFFSET = 20 – 5 – 6 = 11.

Как происходит взаимодействие в системе?

Есть набор команд. Процессор, кэш и память обмениваются командами по шинам. Команды такие:

Команды между процессором и кэшем:

C1_NOP – no operation – отсутствие команды

Команды от процессора кэшу:
C1_READ8 – прочитать 8 бит (1 байт) из заданного адреса в оперативной памяти
C1_READ16 - прочитать 16 бит (2 байт), начиная с заданного адреса в оперативной памяти
C1_READ32 - прочитать 32 бита (4 байта), начиная с заданного адреса в оперативной памяти
C1_INVALIDATE_LINE – инвалидизировать линию – выкинуть линию из кэша по заданному адресу
C1_WRITE8 - записать 8 бит (1 байт) по заданному адресу в оперативной памяти
C1_WRITE16 - записать 16 бит (2 байта), начиная с заданного адреса в оперативной памяти
C1_WRITE32 - записать 32 бит (4 байт), начиная с заданного адреса в оперативной памяти

Всего 9 команд. Чтобы их передавать, нам надо иметь 9 состояний шины. Значит, размер шины (CTR1_BUS_SIZE) 4 бита.

От кэша процессору:

C1_RESPONSE – ответ за запрос процессора

Команды между памятью и кэшем:

C2_NOP – отсутствие команды

От кэша к памяти:

C2_READ_LINE – прочесть линию с заданным адресом

C2_WRITE_LINE – записать в память линию по заданному адресу

От памяти к кэшу

C2_RESPONSE – ответ на запрос кэша

Имеем 4 команды, для передачи достаточно 2 бит (CTR2_BUS_SIZE = 2).

По условию размер обеих шин данных 16 бит, размер шин адреса:

ADDR1_BUS_SIZE должен быть хотя бы MIN(BITS_TAG + BITS_SET, OFFSET). То есть, хотя бы 11 бит + 4 бита = 15 бит. Такой размер и возьмем.

ADDR2_BUS_SIZE нужен BITS_TAG + BITS_SET = 15 бит.

Протокол взаимодействия

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.

В начальный момент времени (или после Reset) шиной 1 владеет CPU, а шиной 2 - Cache. После подачи команды и до окончания отправки ответа владение шиной переходит к Cache и MemCTR соответственно. Владение шиной означает какое устройство задает логические уровни на проводах шины.

Little-endian – данные хранятся и передаются таким образом, что младшие биты идут раньше старших.

Взаимодействие:

Процессор посылает запрос кэшу. Ждет, ничего не делает, пока кэш не отправит C1_RESPONSE. Кэш, в свою очередь, выполняя команду процессора, может отправить запрос памяти. Тоже ждет, пока память не отправит C2_RESPONSE.

Аналитическое решение задачи
Программа на C++.

В конце можно раскомментировать вывод таблицы, чтобы посмотреть результирующую матрицу. Но это есть смысл делать, если заполнить память изначально рандомом из Verilog.

Воспроизведение системы в Verilog и моделирование задачи

Использовался Icarus Verilog 10.

Комментарии в коде поясняют, как работает каждая из команд.

Важно частью реализации является то, что запись в шины я всегда делал на negedge CLK, а чтение на posedge CLK. То есть, на фронте и на спаде. Нужно это для того, чтобы записанные данные можно было однозначно прочитать. Если чтение и запись случились бы одновременно, то произошла бы гонка состояний – то есть, при чтении я бы получил случайное значение – возможно старое, возможно новое.

Получилось 3 модуля: кэш, память, testbench (совмещенный с CPU).

Сравнение результатов

Итак, получились следующие результаты:

Аналитическое решение: 213761 попаданий, 249600 запросов всего. Процент попаданий - ~0,8564%

Тот же результат получился в Verilog.

Количество тактов: аналитика: 7500657, Verilog: 7937484.

Процент попаданий получился одинаковый. Количество тактов достаточно сильно отличается. Скорее всего, связано с тем, что в аналитике учел не все задержки, которые я делал в Verilog при передаче команд и данных.

Листинг кода

Аналитика

```
#include <iostream>

#include <vector>

const int C1_NOP = 0;

const int C1_READ8 = 1;

const int C1_READ16 = 2;

const int C1_READ32 = 3;

const int C1_INVALIDATE_LINE = 4;

const int C1_WRITE8 = 5;

const int C1_WRITE16 = 6;

const int C1_WRITE32 = 7;

const int C1_RESPONSE = 8;


const int C2_NOP = 0;

const int C2_READ_LINE = 1;

const int C2_WRITE_LINE = 2;

const int C2_RESPONSE = 3;


const int MEM_SIZE = (1<<20);

const int CACHE_SIZE = (1<<10);

const int CACHE_LINE_SIZE = 32;

const int CACHE_LINE_COUNT = 32;
```

```
const int CACHE_WAY = 2;

const int CACHE_SETS_COUNT = 16;

const int CACHE_TAG_SIZE = 11;

const int CACHE_SET_SIZE = 4;

const int CACHE_OFFSET_SIZE = 5;

const int CACHE_ADDR_SIZE = 9;

const int ADDR1_BUS_SIZE = 11;

const int ADDR2_BUS_SIZE = 15;

const int DATA1_BUS_SIZE = 16;

const int DATA2_BUS_SIZE = 16;

const int CTR1_BUS_SIZE = 4;

const int CTR2_BUS_SIZE = 2;


const int M = 64;

const int N = 60;

const int K = 32;


using namespace std;


int ticksCount = 0;

int cacheHits = 0;

int totalQueries = 0;
```

```
vector<int> memory(MEM_SIZE, 1);



vector<vector<vector<int>>> cacheMemory(CACHE_SETS_COUNT,
vector<vector<int>> (CACHE_WAY, vector<int>
(CACHE_LINE_SIZE, 0)));



vector<vector<int>> tags(CACHE_SETS_COUNT, vector<int>
(CACHE_WAY, -1));



vector<vector<bool>> isValid(CACHE_SETS_COUNT,
vector<bool> (CACHE_WAY, false));



vector<vector<bool>> isDirty(CACHE_SETS_COUNT,
vector<bool> (CACHE_WAY, false));



vector<int> lastRecentlyUsed(CACHE_SETS_COUNT, 0);



int getTag(int address) {

    return address / (1 << CACHE_SET_SIZE +
CACHE_OFFSET_SIZE);

}



int getSet(int address) {

    return (address / (1 << CACHE_OFFSET_SIZE)) % (1 <<
CACHE_SET_SIZE);
```

```
}


int getOffset(int address) {
    return address  % (1 << CACHE_OFFSET_SIZE);
}




void invalidateLine(int set_, int lineInSet) {
    if (isDirty[set_][lineInSet]) {
        ticksCount += 1; // на передачу команды
        // в линии есть данные, не записанные в память, запишем их
        ticksCount += 100;
        //cout << tags[set_][lineInSet] << endl;
        int startInMemory = (tags[set_][lineInSet]<<(CACHE_OFFSET_SIZE + CACHE_SET_SIZE)) + (set_<<(CACHE_OFFSET_SIZE));
        //cout << startInMemory << endl;
        for (int i = 0; i < CACHE_LINE_SIZE; i++) {
            memory[startInMemory + i] = cacheMemory[set_][lineInSet][i];
        }
    }
```

```
    isValid[set_][lineInSet] = false;

}



void writeLine(int tag, int set_, int lineInSet) {

    ticksCount += 1; // такты на передачу команды

    ticksCount += 100;

    int startInMemory = (tag<<(CACHE_OFFSET_SIZE +
CACHE_SET_SIZE)) + (set_<<(CACHE_OFFSET_SIZE));

    for (int i = 0; i < CACHE_LINE_SIZE; i++) {

        cacheMemory[set_][lineInSet][i] =
memory[startInMemory + i];

    }

}



int read8(int address) {

    ticksCount += 2; // такты на передачу команды

    totalQueries++;

    // проверим, есть ли линия в кэше

    int tag = getTag(address);

    int set_ = getSet(address);

    int offset = getOffset(address);

    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {
```

```
        if (tags[set_][lineInSet] == tag &&
isValid[set_][lineInSet]) {

            // нашли линию, кэш попадание

            ticksCount += 7; // кэш ответит через 6
тактов, передаст всё за 1 такт

            cacheHits++;

            lastRecentlyUsed[set_] = 1 - lineInSet;

            return cacheMemory[set_][lineInSet][offset];

        }

    }

    //не нашли линию, кэш промах

    ticksCount += 4; // кэш посылает запрос в память
через 4 такта

    int lineToWrite = -1;

    // попробуем найти пустую линию (isValid = 0)

    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {

        if (!isValid[set_][lineInSet]) {

            // нашли пустую линию

            lineToWrite = lineInSet;

        }

    }

    if (lineToWrite == -1) {

        // пустых диний нет, нужно выкидывать какую-то
```

```
        // выкидываем самую долго не использованную
        invalidateLine(set_, lastRecentlyUsed[set_]);
        lineToWrite = lastRecentlyUsed[set_];
    }
    // нашли пустую линию, пишем туда данные из памяти
    writeLine(tag, set_, lineToWrite);
    isValid[set_][lineToWrite] = true;
    tags[set_][lineToWrite] = tag;
    isDirty[set_][lineToWrite] = false;
    lastRecentlyUsed[set_] = 1 - lineToWrite;
    // линия записана, можем ответить процессору
    ticksCount++;
    return cacheMemory[set_][lineToWrite][offset];
}

int read16(int address) {
    ticksCount += 2; // на передачу команды
    totalQueries++;
    // проверим, есть ли линия в кэше
    int tag = getTag(address);
    int set_ = getSet(address);
    int offset = getOffset(address);
```

```
    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {

        if (tags[set_][lineInSet] == tag &&
isValid[set_][lineInSet]) {

            // нашли линию, кэш попадание

            ticksCount += 8; // кэш ответит через 6
тактов, передаст всё за 2 такта

            cacheHits++;

            lastRecentlyUsed[set_] = 1 - lineInSet;

            return cacheMemory[set_][lineInSet][offset] +
256 * cacheMemory[set_][lineInSet][offset + 1];

        }

    }

    // не нашли линию, кэш промах

    ticksCount += 4; // кэш посылает запрос в память
через 4 такта

    int lineToWrite = -1;

    // попробуем найти пустую линию (isValid = 0)

    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {

        if (!isValid[set_][lineInSet]) {

            // нашли пустую линию

            lineToWrite = lineInSet;

        }

    }
```

```
    if (lineToWrite == -1) {
        // пустых диний нет, нужно выкидывать какую-то
        // выкидываем самую долго не использованную
        invalidateLine(set_, lastRecentlyUsed[set_]);
        lineToWrite = lastRecentlyUsed[set_];
    }
    // нашли пустую линию, пишем туда данные из памяти
    writeLine(tag, set_, lineToWrite);
    tags[set_][lineToWrite] = tag;
    isValid[set_][lineToWrite] = true;
    isDirty[set_][lineToWrite] = false;
    lastRecentlyUsed[set_] = 1 - lineToWrite;
    // линия записана, можем ответить процессору
    ticksCount++;
    return cacheMemory[set_][lineToWrite][offset] + 256 *
cacheMemory[set_][lineToWrite][offset + 1];
}


void write32(int address, int value) {
    ticksCount += 2; // на передачу команды
    totalQueries++;
    // проверим, есть ли линия в кэше
    int tag = getTag(address);
```

```
    int set_ = getSet(address);

    int offset = getOffset(address);

    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {

        if (tags[set_][lineInSet] == tag &&
isValid[set_][lineInSet]) {

            // нашли линию, кэш попадание

            ticksCount += 6; // кэш ответит через 6
тактов

            cacheHits++;

            cacheMemory[set_][lineInSet][offset] = value
% 256;

            cacheMemory[set_][lineInSet][offset + 1] =
(value/256) % 256;

            cacheMemory[set_][lineInSet][offset + 2] =
(value/(256*256)) % 256;

            cacheMemory[set_][lineInSet][offset + 3] =
(value/(256*256*256)) % 256;

            lastRecentlyUsed[set_] = 1 - lineInSet;

            // отвечаем кэшу

            ticksCount++;

            return;

        }

    }

    // не нашли линию, кэш промах
```

```
    ticksCount += 4; // кэш посылает запрос в память
через 4 такта

    int lineToWrite = -1;

    // попробуем найти пустую линию (isValid = 0)

    for (int lineInSet = 0; lineInSet < CACHE_WAY;
lineInSet++) {

        if (!isValid[set_][lineInSet]) {

            // нашли пустую линию

            lineToWrite = lineInSet;

            break;

        }

    }

    if (lineToWrite == -1) {

        // пустых диний нет, нужно выкидывать какую-то

        // выкидываем самую долго не использованную

        invalidateLine(set_, lastRecentlyUsed[set_]);

        lineToWrite = lastRecentlyUsed[set_];

    }

    // нашли пустую линию, пишем туда данные из памяти

    writeLine(tag, set_, lineToWrite);

    isValid[set_][lineToWrite] = true;

    lastRecentlyUsed[set_] = 1 - lineToWrite;

    // линия перенесена в кэш. можем ее менять
```

```
    cacheMemory[set_][lineToWrite][offset] = value % 256;

    cacheMemory[set_][lineToWrite][offset + 1] =
(value/256) % 256;

    cacheMemory[set_][lineToWrite][offset + 2] =
(value/(256*256)) % 256;

    cacheMemory[set_][lineToWrite][offset + 3] =
(value/(256*256*256)) % 256;

    tags[set_][lineToWrite] = tag;

    isDirty[set_][lineToWrite] = true;

    // линия записана, можем ответить процессору

    ticksCount++;

    return;

}



int main()

{

    vector<vector<int>> c(M, vector<int> (N, -1));

    int pa = 0;

    int pc = M * K + 2 * K * N;

    for (int y = 0; y < M; y++) {

        ticksCount++; // инициализация y

        for (int x = 0; x < N; x++) {

            ticksCount++; // инициализация x

            int pb = M * K;
```

```
            ticksCount++; // присваивание
            int s = 0;
            ticksCount++; //  присваивание
            for (int k = 0; k < K; k++) {
                ticksCount++; // инициализация k

                // s += pa[k] * pb[x]; мы хотим сделать
такое действие

                // pa[k] <=> mem[pa + k]
                // pb[x] <=> mem[pb + 2 * x]

                int q1 = read8(pa + k); // получили pa[k]
                int q2 = read16(pb + 2 * x); // получили
pb[x]
                s += q1 * q2; // s += pa[k] * pb[x]
                ticksCount += 5 + 1; // умножение стоит 5
и сложение 1
                pb += 2 * N;
                ticksCount += 1; // сложение 1 такт

                ticksCount += 2; // увеличение счетчика
цикла и условный переход
            }
```

```
            // pc[x] <=> mem[pc + 4 * x]

            write32(pc + 4 * x, s);

            c[y][x] = s; // это учтено в записи в кэш

            ticksCount += 2; // увеличение счетчика цикла
и условный переход

        }

        pa += K;

        ticksCount += 1; // сложение 1 такт

        pc += 4 * N;

        ticksCount += 1; // сложение 1 такт

        ticksCount += 2; // увеличение счетчика цикла и
условный переход

    }

    cout << "ticks count: " << ticksCount << endl;

    cout << "cache hits: " << cacheHits << " total
queries: " << totalQueries << endl;

    cout << "ratio: " << 1. * cacheHits / totalQueries <<
endl;

    /*for(auto& i : c) {

        for(auto& j : i) {

            cout << j << ' ';

        } cout << endl;

    }*/

    return 0;
```

```
}
```

Verilog

```
`define C1_NOP 4'b0000
`define C1_READ8 4'b0001
`define C1_READ16 4'b0010
`define C1_READ32 4'b0011
`define C1_INVALIDATE_LINE 4'b0100
`define C1_WRITE8 4'b0101
`define C1_WRITE16 4'b0110
`define C1_WRITE32 4'b0111
`define C1_RESPONSE 4'b1000

`define C2_NOP 2'b00
`define C2_READ_LINE 2'b01
`define C2_WRITE_LINE 2'b10
`define C2_RESPONSE 2'b11

`define MEM_SIZE 2**20
`define CACHE_SIZE 2**10
`define CACHE_LINE_SIZE 32
`define CACHE_LINE_COUNT 32
`define CACHE_WAY 2
`define CACHE_SETS_COUNT 16
`define CACHE_TAG_SIZE 11
`define CACHE_SET_SIZE 4
`define CACHE_OFFSET_SIZE 5
`define CACHE_ADDR_SIZE 9
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 4
`define CTR2_BUS_SIZE 2

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

module MemoryController(
        inout [`DATA2_BUS_SIZE-1:0] dataBus,
        inout [`CTR2_BUS_SIZE-1:0] controlBus,
        input CLK,
        input [`ADDR2_BUS_SIZE-1:0] addressBus,
        input M_DUMP,
        input RESET
        );

    wire CLK;
```

```
    wire[`DATA2_BUS_SIZE-1:0] dataBus;
    reg[`DATA2_BUS_SIZE-1:0] dataRegister;
    reg[7:0] dataBuffer[`CACHE_LINE_SIZE-1:0]; // буфер, где будем хранить
данные, которые нужно записать в память

    wire[`ADDR2_BUS_SIZE-1:0] addressBus;
    reg[`ADDR2_BUS_SIZE-1:0] addressBuffer; // буфер для хранения адреса

    wire[`CTR2_BUS_SIZE-1:0] controlBus;
    reg[`CTR2_BUS_SIZE-1:0] controlRegister;

    reg[7:0] memory[`MEM_SIZE-1:0];
    int ticksCount; // количество тактов, прошедших с запуска/reset'a
    int currentCount; // вспомогательная переменная для task'a для ожидания
тактов
    int i; // переменная для циклов

    integer SEED;

    // "подключаем" провода к соответствующим регистрам:
    assign dataBus = dataRegister;
    assign controlBus = controlRegister;

    // task для того, чтобы пропустить/подождать несколько тактов
    task waitTicks(input int TicksToWait);
        currentCount = ticksCount;
        wait(ticksCount == currentCount + TicksToWait);
    endtask

    initial begin
        dataRegister = 16'bz;
        controlRegister = 2'bz;
        ticksCount = 0;
        //$monitor("%d", ticksCount);
    end

    initial begin
        SEED = 225526;

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

        //for (i = 0; i < 100; i += 1) begin
        //$display("[%d] %d", i, memory[i]);
        //end
    end
```

```
    always @(posedge CLK) begin
        if (RESET == 1) begin
            SEED = 225526;
            for (i = 0; i < `MEM_SIZE; i += 1) begin
                memory[i] = $random(SEED)>>16;
            end
            ticksCount = 0;
            dataRegister = 16'bz;
            controlRegister = 2'bz;
        end else begin
            ticksCount++;
        end
        //$display("%d", ticksCount);
    end

    // обрабатываем команды

    always @(posedge CLK) begin
        if (controlBus == `C2_WRITE_LINE) begin
            if (M_DUMP == 1) begin
                $display("WRITE_LINE: \nticks: %d, address: %d, controlBus:
%d", ticksCount, addressBus, controlBus);
            end
            //$display("address: %b, command: %b, dataBus: %b", address,
command, data);
            // здесь запоминаем адрес и данные, чтобы потом писать в память
            addressBuffer = addressBus;

            // сколько порций данных нам нужно получить? нам передают 1 кэш
линию по 16 бит за такт
            // (32B / 2B) = 16  => нужно получить 16 порций данных:
            for (i = 0; i < 16; i++) begin
                // сохраняем в буфер побайтово:
                dataBuffer[2 * i] = dataBus[7:0];
                dataBuffer[2 * i + 1] = dataBus[15:8];
                @(posedge CLK);
            end

            // мы работаем уже 16 тактов, подождем еще 84, чтобы было 100
тактов - время ответа памяти
            waitTicks(84);

            // пишем данные в память
            // добавляем к адресу 5 бит, так как 5 бит - это cache offset
или log2(cache_line_size)
            for (i = {addressBuffer, 5'b00000}; i <= {addressBuffer,
5'b11111}; i++) begin
                memory[i] = dataBuffer[i - {addressBuffer, 5'b00000}];
```

```
            end

            // записали данные, нужно ответить кэшу
            @ (negedge CLK);
            controlRegister = `C2_RESPONSE;

            // теперь, через такт (чтобы кэш успел заметить ответ), отдадим
управление шиной кэшу
            @ (negedge CLK);
            controlRegister = 2'bz;

        end else if (controlBus == `C2_READ_LINE) begin
            if (M_DUMP == 1) begin
                $display("READ_LINE: \nticks: %d, address: %d, controlBus:
%d", ticksCount, addressBus, controlBus);
            end
            // запоминаем адрес
            addressBuffer = addressBus;

            // ждем 100 тактов - время ответа памяти
            waitTicks(100);
            // кэш уже убрал свою команду с шины, поставим команду, что
отвечаем ему
            @ (negedge CLK);
            // отвечаем кэшу
            controlRegister = `C2_RESPONSE;
            //dataRegister = {memory[{addressBuffer, 5'b00001}],
memory[{addressBuffer, 5'b00000}]};
            //$display("%b", dataBus);
            for (i = {addressBuffer, 5'b00000}; i < {addressBuffer,
5'b11111}; i += 2) begin
                dataRegister = {memory[i + 1], memory[i]};
                @ (negedge CLK);
            end

            // выполнили запрос, теперь нужно освободить шины
            controlRegister = 2'bz;
            dataRegister = 16'bz;
        end
    end

endmodule



module Cache(
        output [`ADDR2_BUS_SIZE-1:0] address2Bus,
        inout [`DATA1_BUS_SIZE-1:0] data1Bus,
        inout [`CTR1_BUS_SIZE-1:0] control1Bus,
```

```
        inout [`DATA2_BUS_SIZE-1:0] data2Bus,
        inout [`CTR2_BUS_SIZE-1:0] control2Bus,
        input CLK,
        input [`ADDR1_BUS_SIZE-1:0] address1Bus,
        input C_DUMP,
        input RESET
        );
    wire CLK;

    wire[`DATA1_BUS_SIZE-1:0] data1Bus;
    reg[`DATA1_BUS_SIZE-1:0] data1Register;
    wire[`DATA2_BUS_SIZE-1:0] data2Bus;
    reg[`DATA2_BUS_SIZE-1:0] data2Register;
    reg[7:0] lineBuffer[`CACHE_LINE_SIZE-1:0];

    wire[`ADDR1_BUS_SIZE-1:0] address1Bus;
    reg[`CACHE_TAG_SIZE-1:0] tagBuffer;
    reg[`CACHE_SET_SIZE-1:0] setBuffer;
    reg[`CACHE_OFFSET_SIZE-1:0] offsetBuffer;
    wire[`ADDR2_BUS_SIZE-1:0] address2Bus;
    reg[`ADDR1_BUS_SIZE-1:0] address2Register;

    wire[`CTR1_BUS_SIZE-1:0] control1Bus;
    reg[`CTR1_BUS_SIZE-1:0] control1Register;
    wire[`CTR2_BUS_SIZE-1:0] control2Bus;
    reg[`CTR2_BUS_SIZE-1:0] control2Register;

    reg[7:0] cacheMemory[`CACHE_SETS_COUNT-1:0][`CACHE_WAY-
1:0][`CACHE_LINE_SIZE-1:0]; // собственно, память кэша
    reg[`CACHE_TAG_SIZE-1:0] tags[`CACHE_SETS_COUNT-1:0][`CACHE_WAY-1:0]; //
тэг для каждой линии в кэше
    reg[`CACHE_SETS_COUNT-1:0] leastRecentlyUsed; // номер линии в блоке,
которая была не использована дольше
    reg[`CACHE_WAY-1:0] isValid[`CACHE_SETS_COUNT-1:0]; // служебный бит
valid, 0 или 1 для каждой кэш линии
    reg[`CACHE_WAY-1:0] isDirty[`CACHE_SETS_COUNT-1:0]; // служебный бит
dirty, 0 или 1 для каждой кэш линии

    int ticksCount; // количество тактов, прошедших с запуска/reset'а
    int currentCount; // вспомогательная переменная для task'а для ожидания
тактов
    int i, j; // переменные для циклов

    int hits;
    int sum;

    // переменная, в которой будем хранить номер линии в блоке в случае кэш
попадания
```

```
    // -1 в случае кэш промаха
    int foundLine;

    int bitCount;

    reg[31: 0] writeBuffer;

    // "подключаем" провода к соответствующим регистрам
    assign address2Bus = address2Register;
    assign data1Bus = data1Register;
    assign data2Bus = data2Register;
    assign control1Bus = control1Register;
    assign control2Bus = control2Register;

    always @(posedge CLK) begin
        if (RESET == 1) begin
            hits = 0;
            sum = 0;
            ticksCount = 0;
            data1Register = 16'bz;
            data2Register = 16'bz;
            address2Register = 15'bz;
            control1Register = 4'bz;
            control2Register = 2'bz;
            for (i = 0; i < `CACHE_SETS_COUNT; i++) begin
                leastRecentlyUsed[i] = 0;
                for (j = 0; j < `CACHE_WAY; j++) begin
                    isValid[i][j] = 0;
                    isDirty[i][j] = 0;
                end
            end
        end else begin
            ticksCount++;
        end
        // $display("ticks: %d, control1Bus: %d, control2Bus: %d",
ticksCount, control1Bus, control2Bus);
    end

    initial begin
        hits = 0;
        sum = 0;
        ticksCount = 0;
        data1Register = 16'bz;
        data2Register = 16'bz;
        address2Register = 15'bz;
        control1Register = 4'bz;
        control2Register = 2'bz;
        for (i = 0; i < `CACHE_SETS_COUNT; i++) begin
```

```
            leastRecentlyUsed[i] = 0;
            for (j = 0; j < `CACHE_WAY; j++) begin
                isValid[i][j] = 0;
                isDirty[i][j] = 0;
            end
        end
    end


    always @(C_DUMP) $display("HITS, sum: %d, %d", hits, sum);

    // task для того, чтобы пропустить/подождать несколько тактов
    task waitTicks(input int ticksToWait);
        currentCount = ticksCount;
        wait(ticksCount == currentCount + ticksToWait);
    endtask

    // по номеру блока в кэше (хранится в setBuffer) выкидывает линию в нем,
которая дольше не использовалась
    task makeFreeSlotForLine;
        if (isDirty[setBuffer][leastRecentlyUsed[setBuffer]] == 1) begin
            // линия, которую хотим выкинуть, изменялась, и эти изменения не
были записаны в память
            // запишем изменения в память
            // напрявляем команду памяти
            @ (negedge CLK);
            control2Register = `C2_WRITE_LINE;
            // передаем адрес
            address2Register = {tagBuffer, setBuffer};
            // передаем данные для записи
            // data2Register = {lineBuffer[0], lineBuffer[1]};
            for (j = 0; j < 16; j++) begin
                data2Register = {lineBuffer[2 * i + 1], lineBuffer[2 * i]};
                @ (negedge CLK);;
            end
            // освобождаем шину команд
            control2Register = 2'bz;
            // ждем ответа памяти
            wait(control2Bus == `C2_RESPONSE);
            // память ответила, всё записалось. освобождаем шины
            @ (negedge CLK);
            address2Register = 15'bz;
            data2Register = 16'bz;
            waitTicks(1); // ждем 1 такт, чтобы память освободила шину
        end
        // линия, которую мы хотим выкинуть, сохранена в памяти
        // поставим ей в бит valid 0
        isValid[setBuffer][leastRecentlyUsed[setBuffer]] = 0;
    endtask
```

```
    // достать из памяти линию, адрес которой лежит в буфере
    task getLineFromMemory;
        // заметим, что когда task вызвали, CLK был на posegde
        // нужно сделать запрос в память, чтобы получить линию по адресу
        // передаем команду и адрес
        @(negedge CLK);
        control2Register = `C2_READ_LINE;
        address2Register = {tagBuffer, setBuffer};
        if (C_DUMP == 1) begin
            //$display("tag: %b, set: %b", tagBuffer, setBuffer);
            //$display("getline: ticks: %d, control2: %b, address: %b",
ticksCount, control2Register, address2Register);
        end
        // освобождаем шину
        // хотим писать, а пишем только на negedge:
        @(negedge CLK);
        control2Register = 2'bz;
        // ждем, пока память начнет отвечать
        wait(control2Bus == `C2_RESPONSE);
        @(posedge CLK);
        // память начала отвечать, сохраним данные в LineBuffer
        // дадут нам 16 порций данных (16 порций по 16 бит = 32 байта =
CACHE_LINE_SIZE)
        for (i = 0; i < 16; i++) begin
            lineBuffer[2 * i] = data2Bus[7:0];
            //$display("buffer: %b", lineBuffer[2 * i]);
            lineBuffer[2 * i + 1] = data2Bus[15:8];
            //$display("buffer: %b", lineBuffer[2 * i + 1]);
            @(posedge CLK);
        end
        //$display("got line");
        // взаимодействие с памятью завершено, освободим шину
        @(negedge CLK);
        address2Register = 15'bz;
        waitTicks(1); // ждем 1 такт, чтобы сигнал распространился
    endtask

    always @(posedge CLK) begin
        if (control1Bus == `C1_READ32 || control1Bus == `C1_READ16 ||
control1Bus == `C1_READ8) begin
            sum++;
            // запомним команду
            if (control1Bus == `C1_READ32) begin
                bitCount = 32;
            end else if (control1Bus == `C1_READ16) begin
                bitCount = 16;
            end else begin
```

```
                bitCount = 8;
            end
            if (C_DUMP == 1) begin
                $display("cache read: ticks: %d, control: %d, address: %d",
ticksCount, control1Bus, address1Bus);
            end

            // сохраним передаваемый процессором адрес, он передается в два
такта
            setBuffer = address1Bus[`CACHE_SET_SIZE-1:0];
            tagBuffer = address1Bus[`CACHE_SET_SIZE+`CACHE_TAG_SIZE-
1:`CACHE_SET_SIZE];

            // ждем 1 такт
            @(posedge CLK);
            // сохраняем вторую половину адреса
            offsetBuffer = address1Bus[`CACHE_OFFSET_SIZE-1:0];
            // проверим, кэш попадание у нас или кэш промах
            // перебираем линии в наборе, соответствующем адресу
            foundLine = -1; // пока что считаем, что кэш промах
            for (i = 0; i < `CACHE_WAY; i++) begin
                // проверим, valid ли кэш линия, которую мы смотрим
                if (isValid[setBuffer][i] == 1) begin
                    // сравним tag линий:
                    //$display("tagBuffer: %b, tags[][]: %b", tagBuffer,
tags[setBuffer][i]);
                    if (tagBuffer == tags[setBuffer][i]) foundLine = i;
                end
            end
            if (foundLine == -1) begin
                if (C_DUMP) begin
                    $display("ticks: %d, cache miss", ticksCount);
                end
                // кэш-промах, ждем 4 такта перед тем, как делать запрос в
память
                waitTicks(4);

                // идем в память за данными
                getLineFromMemory;

                // в lineBuffer лежит линия, сохраним ее в
кэш
                // нужно найти место. ищем:
                // сначала попробуем найти линию, в которой не хранятся
данные (isValid 0)
                // тут идем с конца, чтобы в случае, когда invalid обе
линии, взять первую
                for (i = `CACHE_WAY - 1; i >= 0; i--) begin
```

```
                    if (isValid[setBuffer][i] == 0) begin
                        foundLine = i;
                    end
                end
                if (foundLine == -1) begin
                    if (C_DUMP == 1) begin
                        $display("all lines are valid, invalidating a
line");
                    end
                    // все линии заняты
                    // освободим линию
                    // нужно выкинуть из набора линию, которая не
использовалась дольше
                    makeFreeSlotForLine;
                    // теперь в наборе есть свободная линия. это та, которая
дольше не использовалась
                    foundLine =  leastRecentlyUsed[setBuffer];
                end
                // нашли линию, куда можно записать данные. пишем:

                for (j = 0; j < `CACHE_LINE_SIZE; j++) begin
                    cacheMemory[setBuffer][foundLine][j] = lineBuffer[j];
                    //$display("setBuffer: %b, foundLine: %d, j: %d",
setBuffer, foundLine, j);
                    //$display("lineBuffer: %b", lineBuffer[j]);
                    //$display("memory: %b",
cacheMemory[setBuffer][foundLine][j]);
                end
                // добавим служебные значения
                isValid[setBuffer][foundLine] = 1;
                isDirty[setBuffer][foundLine] = 0;
                tags[setBuffer][foundLine] = tagBuffer;
            end else begin
                hits++;
                //$display("wow! cache hit!");
                // кэш-попадание, ждем 6 тактов
                waitTicks(6);
            end

            // линия, откуда нам нужно взять данные, лежит в кэше
            // передаем данные
            @(negedge CLK);
            control1Register = `C1_RESPONSE;

            if (bitCount == 8) begin
                data1Register = {8'b0,
cacheMemory[setBuffer][foundLine][offsetBuffer]};
                @(negedge CLK);
```

```
            end else begin
                for (i = 0; i < bitCount / 16; i++) begin
                    data1Register =
{cacheMemory[setBuffer][foundLine][offsetBuffer + i + 1],
cacheMemory[setBuffer][foundLine][offsetBuffer + i]};
                    @(negedge CLK);
                end
            end
            // команда выполнена, освободим шины
            data1Register = 16'bz;
            control1Register = 4'bz;

            // обратились к кэш линии, значит, дольше не обращались к другой
            leastRecentlyUsed[setBuffer] = ~foundLine;

        end else if (control1Bus == `C1_WRITE32 || control1Bus ==
`C1_WRITE16 || control1Bus == `C1_WRITE8) begin
            sum++;
            // запомним команду
            if (control1Bus == `C1_WRITE32) begin
                bitCount = 32;
            end else if (control1Bus == `C1_WRITE16) begin
                bitCount = 16;
            end else begin
                bitCount = 8;
            end
            if (C_DUMP == 1) begin
                $display("cache write: ticks: %d, control: %d, address: %d",
ticksCount, control1Bus, address1Bus);
            end
            // сохраним адрес и данные, которые надо записать
            writeBuffer = data1Bus;
            setBuffer = address1Bus[`CACHE_SET_SIZE-1:0];
            tagBuffer = address1Bus[`CACHE_SET_SIZE+`CACHE_TAG_SIZE-
1:`CACHE_SET_SIZE];
            // ждем 1 такт
            @(posedge CLK);
            // сохраняем вторую половину адреса
            offsetBuffer = address1Bus[`CACHE_OFFSET_SIZE-1:0];

            // если пишем 32 бита, то сохранить надо еще 16 бит
            if (bitCount == 32) writeBuffer[31:16] = data1Bus;


            // проверим, кэш попадание у нас или кэш промах
            // перебираем линии в наборе, соответствующем адресу
            foundLine = -1; // пока что считаем, что кэш промах
            for (i = 0; i < `CACHE_WAY; i++) begin
                // проверим, valid ли кэш линия, которую мы смотрим
```

```
                if (isValid[setBuffer][i] == 1) begin
                    // сравним tag линий:
                    if (tagBuffer == tags[setBuffer][i]) foundLine = i;
                end
            end
            if (foundLine == -1) begin
                if (C_DUMP) begin
                    $display("ticks: %d, cache miss", ticksCount);
                end
                // кэш-промах, ждем 4 такта перед тем, как делать запрос в
память
                waitTicks(4);

                // идем в память за данными
                getLineFromMemory;

                // в lineBuffer лежит линия, сохраним ее в
кэш
                // нужно найти место. ищем:
                // сначала попробуем найти линию, в которой не хранятся
данные (isValid 0)
                // тут идем с конца, чтобы в случае, когда invalid обе
линии, взять первую
                for (i = `CACHE_WAY - 1; i >= 0; i--) begin
                    if (isValid[setBuffer][i] == 0) begin
                        foundLine = i;
                    end
                end
                if (foundLine == -1) begin
                    if (C_DUMP == 1) begin
                        $display("all lines are valid, invalidating a
line");
                    end
                    // все линии заняты
                    // освободим линию
                    // нужно выкинуть из набора линию, которая не
использовалась дольше
                    makeFreeSlotForLine;
                    // теперь в наборе есть свободная линия. это та, которая
дольше не использовалась
                    foundLine =  leastRecentlyUsed[setBuffer];
                end
                // нашли линию, куда можно записать данные. пишем:

                for (j = 0; j < `CACHE_LINE_SIZE; j++) begin
                    cacheMemory[setBuffer][foundLine][j] = lineBuffer[j];
                    //$display("setBuffer: %b, foundLine: %d, j: %d",
setBuffer, foundLine, j);
```

```
                    //$display("lineBuffer: %b", lineBuffer[j]);
                    //$display("memory: %b",
cacheMemory[setBuffer][foundLine][j]);
                end
                // добавим служебные значения
                isValid[setBuffer][foundLine] = 1;
                isDirty[setBuffer][foundLine] = 0;
                tags[setBuffer][foundLine] = tagBuffer;
            end else begin
                hits++;
                //$display("cache hit in write");
                // кэш-попадание, ждем 6 тактов
                waitTicks(6);
            end
            // линия, в которой мы будем изменять данные, лежит в кэше
            // меняем данные
            if (bitCount == 8) begin
                cacheMemory[setBuffer][foundLine][offsetBuffer] =
writeBuffer[7:0];
            end else if (bitCount == 16) begin
                cacheMemory[setBuffer][foundLine][offsetBuffer] =
writeBuffer[7:0];
                cacheMemory[setBuffer][foundLine][offsetBuffer + 1] =
writeBuffer[15:8];
            end else begin
                cacheMemory[setBuffer][foundLine][offsetBuffer] =
writeBuffer[7:0];
                cacheMemory[setBuffer][foundLine][offsetBuffer + 1] =
writeBuffer[15:8];
                cacheMemory[setBuffer][foundLine][offsetBuffer + 2] =
writeBuffer[23:16];
                cacheMemory[setBuffer][foundLine][offsetBuffer + 3] =
writeBuffer[31:24];
            end

            @ (negedge CLK);
            control1Register = `C1_RESPONSE;
            // команда выполнена, освободим шину
            @ (negedge CLK);

            control1Register = 4'bz;

            // обратились к кэш линии, значит, дольше не обращались к другой
            leastRecentlyUsed[setBuffer] = ~foundLine;

        end else if (control1Register == `C1_INVALIDATE_LINE) begin
            // сохраним передаваемый процессором адрес, он передается в два
такта
```

```
            setBuffer = address1Bus[`CACHE_SET_SIZE-1:0];
            tagBuffer = address1Bus[`CACHE_SET_SIZE+`CACHE_TAG_SIZE-
1:`CACHE_SET_SIZE];

            // ждем 1 такт
            @(posedge CLK);
            // сохраняем вторую половину адреса
            offsetBuffer = address1Bus[`CACHE_OFFSET_SIZE-1:0];

            // нас просят инвалидировать линию по этому адресу
            if (isDirty[setBuffer][leastRecentlyUsed[setBuffer]] == 1) begin
                // линия, которую хотим выкинуть, изменялась, и эти
изменения не были записаны в память
                // запишем изменения в память
                // напрявляем команду памяти
                @ (negedge CLK);
                control2Register = `C2_WRITE_LINE;
                // передаем адрес
                address2Register = {tagBuffer, setBuffer};
                // передаем данные для записи
                //data2Register = {lineBuffer[0], lineBuffer[1]};
                for (j = 0; j < 16; j++) begin
                    data2Register = {lineBuffer[2 * i + 1], lineBuffer[2 *
i]};
                    @ (negedge CLK);;
                end
                // освобождаем шину команд
                control2Register = 2'bz;
                // ждем ответа памяти
                wait(control2Bus == `C2_RESPONSE);
                // память ответила, всё записалось. освобождаем шины
                @ (negedge CLK);
                address2Register = 15'bz;
                data2Register = 16'bz;
                waitTicks(1); // ждем 1 такт, чтобы память освободила шину
            end
            // линия, которую мы хотим выкинуть, сохранена в памяти
            // поставим ей в бит valid 0
            isValid[setBuffer][leastRecentlyUsed[setBuffer]] = 0;
        end
    end


endmodule

/*
    C2_NOP - "00"
    C2_READ_LINE - "01"
    C2_WRITE_LINE - "10"
```

```
    C2_RESPONSE - "11"
*/

module tb;
    reg CLK;
    reg RESET;
    reg M_DUMP;
    reg C_DUMP;
    wire[`ADDR2_BUS_SIZE-1:0] address2Bus;
    wire[`DATA2_BUS_SIZE-1:0] data2Bus;
    wire[`CTR2_BUS_SIZE-1:0] control2Bus;
    MemoryController memctr(data2Bus, control2Bus, CLK, address2Bus, M_DUMP,
RESET);
    wire[`DATA1_BUS_SIZE-1:0] data1Bus;
    wire[`ADDR1_BUS_SIZE-1:0] address1Bus;
    wire[`CTR1_BUS_SIZE-1:0] control1Bus;
    Cache cache(address2Bus, data1Bus, control1Bus, data2Bus, control2Bus,
CLK, address1Bus, C_DUMP, RESET);

    reg[`DATA1_BUS_SIZE-1:0] data1Register;
    reg[`ADDR1_BUS_SIZE-1:0] address1Register;
    reg[`CTR1_BUS_SIZE-1:0] control1Register;


    reg[31:0] dataBuffer;

    int ticksCount;
    int currentCount;

    reg[7:0] a[`M][`K];
    reg[15:0] b[`K][`N];
    reg[31:0] c[`M][`N];

    int pa;
    int pb;
    int pc;

    reg[7:0] q1;
    reg[15:0] q2;

    int y;
    int x;
    int k;
    int s;

    int i, j;

    assign data1Bus = data1Register;
    assign control1Bus = control1Register;
```

```
    assign address1Bus = address1Register;

    always @(posedge CLK) begin
        if (RESET == 1) begin
            CLK = 0;
            M_DUMP = 0;
            C_DUMP = 0;
            data1Register = 16'bz;
            control1Register = 4'bz;
            address1Register = 11'bz;
        end
        ticksCount++;
    end

    task write8(input reg[19:0] address, input reg[7:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_WRITE8;
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        // данные на запись
        data1Register = data;
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        //$display("GIVEN TAG : %b",
address[`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE+`CACHE_TAG_SIZE-
1:`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE]);
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(negedge CLK);
        // все записалось, освобождаем шины
        address1Register = 11'bz;
        data1Register = 16'bz;
        @(posedge CLK);
    endtask

    task write16(input reg[19:0] address, input reg[15:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_WRITE16;
```

```
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        // данные на запись
        data1Register = data;
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(negedge CLK);
        // все записалось, освобождаем шины
        address1Register = 11'bz;
        data1Register = 16'bz;
        @(posedge CLK);
    endtask

    task write32(input reg[19:0] address, input reg[31:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_WRITE32;
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        // первая половина данных на запись
        data1Register = data[15:0];
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        // вторая половина данных на запись
        data1Register = data[31:16];
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(negedge CLK);
        // все записалось, освобождаем шины
        address1Register = 11'bz;
        data1Register = 16'bz;
        @(posedge CLK);
```

```
    endtask

    task read8(input reg[19:0] address, output reg[7:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_READ8;
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(posedge CLK);
        data = data1Bus[7:0];
        @(negedge CLK);
        // освобождаем шины
        address1Register = 11'bz;
        @(posedge CLK);
    endtask

    task read16(input reg[19:0] address, output reg[15:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_READ16;
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(posedge CLK);
        data = data1Bus;
        @(negedge CLK);
        // освобождаем шины
```

```
        address1Register = 11'bz;
        @(posedge CLK);
    endtask

    task read32(input reg[19:0] address, output reg[31:0] data);
        @(negedge CLK);
        // передаем команду кэшу
        control1Register = `C1_READ32;
        //передаем tag и set - это (`CACHE_TAG_SIZE + `CACHE_SET_SIZE)
старших бит
        address1Register =
address[`CACHE_TAG_SIZE+`CACHE_SET_SIZE+`CACHE_OFFSET_SIZE-
1:`CACHE_OFFSET_SIZE];
        @(negedge CLK);
        // передаем вторую часть адреса - оффсет. это `CACHE_OFFSET_SIZE
младших бит
        address1Register = address[`CACHE_OFFSET_SIZE-1:0];
        @(negedge CLK);
        control1Register = 4'bz;
        // передали команду, ждем, пока нам ответят
        wait(control1Bus == `C1_RESPONSE);
        @(posedge CLK);
        data[15:0] = data1Bus;
        @(posedge CLK);
        data[31:16] = data1Bus;
        @(negedge CLK);
        // освобождаем шины
        address1Register = 11'bz;
        @(posedge CLK);
    endtask

    task waitTicks(input int TicksToWait);
        currentCount = ticksCount;
        wait(ticksCount == currentCount + TicksToWait);
    endtask



    initial begin
        CLK = 0;
        M_DUMP = 0;
        C_DUMP = 0;
        data1Register = 16'bz;
        control1Register = 4'bz;
        address1Register = 11'bz;
        waitTicks(1);
        pa = 0;
        pc = `M * `K + 2 * `K * `N;
        for (y = 0; y < `M; y++) begin
```

```
            waitTicks(1);
            for (x = 0; x < `N; x++) begin
                waitTicks(1);
                pb = `M * `K;
                waitTicks(1);
                s = 0;
                waitTicks(1);
                for (k = 0; k < `K; k++) begin
                    waitTicks(1);
                    // s += pa[k] * pb[x];
                    // pa[k] <=> mem[pa + k]
                    // pb[x] <=> mem[pb + 2 * x]
                    read8(pa + k, q1); // q1 - это pa[k]
                    read16(pb + 2 * x, q2); // q2 - это pb[x]
                    s += q1 * q2; // s += pa[k] * pb[x];
                    waitTicks(5 + 1);
                    pb += 2 * `N;
                    waitTicks(1);

                    waitTicks(2);
                end
                // pc[x] <=> mem[pc + 4 * x]
                write32(pc + 4 * x, s);
                c[y][x] = s; // записываем значение, которое положили в
массив для отладки
                waitTicks(2);
            end
            pa += `K;
            waitTicks(1);
            pc += 4 * `N;
            waitTicks(1);
            waitTicks(2);
        end
        @(negedge CLK);
        C_DUMP = 1;
        @(posedge CLK);
        $display("total ticks: %d", ticksCount);
        for (i = 0; i < `M; i++) begin
            for (j = 0; j < `N; j++) begin
                $write("%h ", c[i][j]);
            end
            $write("\n");
        end
        $finish;
    end

    always #1 CLK = ~CLK;
```

```
endmodule
```