Лабораторная работа №2	**M3136**	**2022**
Построение логических схем в среде моделирования	**Шинкарева Алёна Игоревна**	

Цель работы: моделирование логических схем на элементах с памятью.

Инструментарий и требование к работе: Icarus Verilog 10.

Вариант: 3.

Задачи:

1. Вычислить недостающие параметры системы.

2. По заданной функции аналитически определить общее время в тактах, затраченное на выполнение этой функции, и процент попадания в кэш.

3. Смоделировать на языке описание Verilog систему “процессор-кэш-память” и выполнить заданную функцию.

4. Сравнить полученные результаты.

Решение.

1. Вычисление недостающих параметров системы.

Ниже в таблице 1 представлены параметры системы, данные в 3 варианте, а также те параметры, которые необходимо вычислить.

Память		
Размер памяти	**MEM_SIZE**	256 Кбайт = 2^{18} байт
Кэш		
Размер кэша	**CACHE_SIZE**	2 Кбайт = 2^{11} байт
Размер кэш-линии	**CACHE_LINE_SIZE**	16 байт = 2^{4} байт
Количество кэш-линий	**CACHE_LINE_COUNT**	Необходимо вычислить (1)
Ассоциативность	**CACHE_WAY**	Необходимо вычислить (2)
Количество наборов кэш-линий	**CACHE_SETS_COUNT**	Необходимо вычислить (3)
Размер тэга адреса	**CACHE_TAG_SIZE**	8 бит
Размер индекса в наборе кэш-линий	**CACHE_SET_SIZE**	Необходимо вычислить (4)
Размер смещения	**CACHE_OFFSET_SIZE**	Необходимо вычислить (5)
Размер адреса	**CACHE_ADDR_SIZE**	Необходимо вычислить (6)

Таблица 1 - Данные в варианте 3 размерности параметров памяти и кэша.

Теперь нужно вычислить недостающие параметры:

(6) Размер адреса:

$$\textbf{CACHE\_ADDR\_SIZE} = log_2(\textbf{MEM\_SIZE}) = log_2(2^{18}) = 18 \text{ бит};$$

(5) Размер смещения:

$$\textbf{CACHE\_OFFSET\_SIZE} = log_2(\textbf{CACHE\_LINE\_SIZE}) = log_2(2^{4}) = 4 \text{ бита};$$

(4) Размер индекса в наборе кэш-линий:

$$\textbf{CACHE\_SET\_SIZE} = \textbf{ADDR\_SIZE} - \textbf{TAG\_SIZE} - \textbf{OFFSET\_SIZE} = 18 - 8 - 4 = 6 \text{ бит};$$

(3) Количество наборов кэш-линий:

$$\mathbf{CACHE\_SETS\_COUNT} = 2^{CACHE\_SET\_SIZE} = 2^6 = 64;$$

(1) Количество кэш-линий:

$$\mathbf{CACHE\_LINE\_COUNT} = \frac{CACHE\_SIZE}{CACHE\_LINE\_SIZE} = \frac{2^{11}}{2^4} = 2^7 = 128;$$

(2) Ассоциативность:

$$\mathbf{CACHE\_WAY} = \frac{CACHE\_LINE\_COUNT}{CACHE\_SETS\_COUNT} = \frac{2^7}{2^6} = 2;$$

Ниже в таблице 2 представлены все размерности параметров системы.

Память		
Размер памяти	**MEM_SIZE**	256 Кбайт = 2^{18} байт
Кэш		
Размер кэша	**CACHE_SIZE**	2 Кбайт = 2^{11} байт
Размер кэш-линии	**CACHE_LINE_SIZE**	16 байт = 2^4 байт
Количество кэш-линий	**CACHE_LINE_COUNT**	2^7 = 128 кэш-линий
Ассоциативность	**CACHE_WAY**	2
Количество наборов кэш-линий	**CACHE_SETS_COUNT**	2^6 = 64 набора
Размер тэга адреса	**CACHE_TAG_SIZE**	8 бит
Размер индекса в наборе кэш-линий	**CACHE_SET_SIZE**	6 бит
Размер смещения	**CACHE_OFFSET_SIZE**	4 бита
Размер адреса	**CACHE_ADDR_SIZE**	18 бит

Таблица 2 - Итоговые размерности параметров памяти и кэша.

Кроме размерностей параметров системы также необходимо вычислить размерности шин. Ниже в таблице 3 представлены данные в варианте 3 размерности.

Шина	Обозначение	Размерность
A1, A2	**ADDR1_BUS_SIZE**, **ADDR2_BUS_SIZE**	Вычислить самостоятельно (1, 2)
D1, D2	**DATA1_BUS_SIZE, DATA2_BUS_SIZE**	16 бит
C1, C2	**CTR1_BUS_SIZE**, **CTR2_BUS_SIZE**	Вычислить самостоятельно (3, 4)

Таблица 3 - Данные в варианте 3 размерности шин.

Вычисления:

(1) Шина A1 - по ней адрес передается в 2 такта: за первый такт передаются **CACHE_TAG** и **CACHE_SET**, а за второй такт - **CACHE_OFFSET**, поэтому **ADDR1_BUS_SIZE** = max(**CACHE_TAG** + **CACHE_SET**, **CACHE_OFFSET**) = max(8 + 6, 4) = 14 бит.

(2) Шина A2 - по ней передается адрес без **CACHE_OFFSET**, следовательно **ADDR2_BUS_SIZE** = **CACHE_TAG** + **CACHE_SET** = 8 + 6 = 14 бит.

(3) Шиной C1 в начальный момент времени владеет CPU, после подачи команды и до окончания отправки ответа владение переходит к Cache, количество команд **CPU** → **Cache** - 8, а **CPU** ← **Cache** - 2, следовательно

CTR1_BUS_SIZE = $log_2(8)$ = 3 бита.

(4) Шиной C2 в начальный момент времени владеет Cache, после подачи команды и до окончания отправки ответа владение переходит к MemCTR количество команд **Cache** → **Mem** - 3, а **Cache** ← **Mem** - 2, следовательно

CTR1_BUS_SIZE = $\lceil log_2(3) \rceil$ = 2 бита.

Ниже в таблице 4 представлены все размерности шин.

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

Таблица 4 - Итоговые размерности шин.

2. Аналитическое решение задачи.

Аналитическое решение представлено в виде кода на языке высокого уровня **Java**, эмулирующего работу системы. Ниже на рисунке 1 представлена схема моделируемой системы.

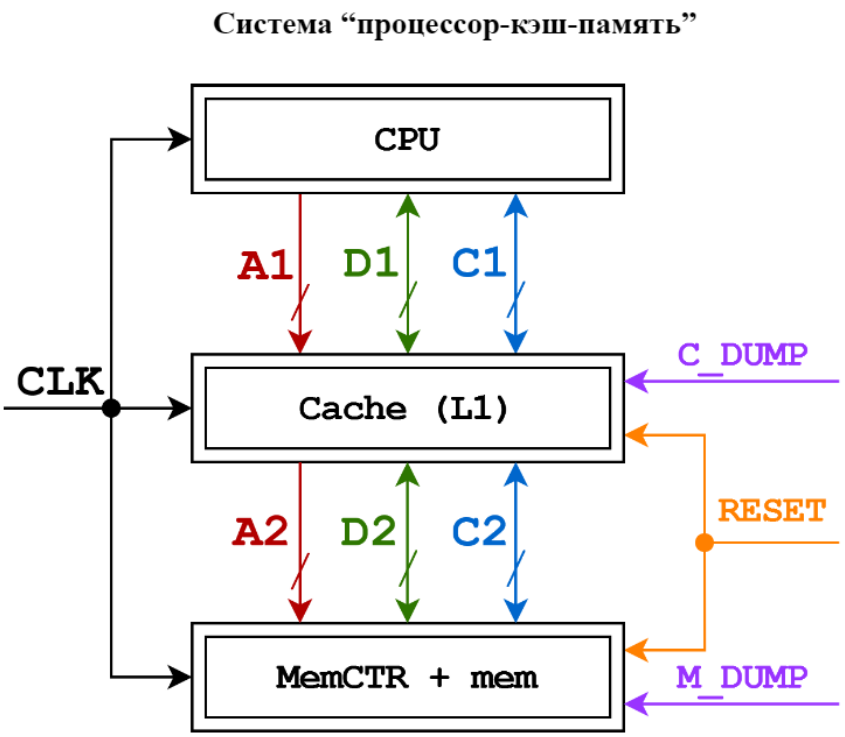

Рисунок 1 - Схема моделируемой системы.

Код написан исключительно для решения поставленной задачи, а именно для определения общего времени (в тактах), затраченного на выполнение заданной функции, и процента попаданий, который вычисляется как отношение числа попаданий к общему числу обращений). В коде реализованы модули **CACHE**, **CACHE_SET**, **CACHE_LINE** в виде соответствующих классов. Разберем работу каждого класса (модуля).

CACHE_LINE

CACHE_LINE (кэш-линия) устроена следующим образом: имеются флаги **flags**, тэг адрес **tag**, полезные данные **data**. Подробнее разберём флаги. По данному условию политика вытеснения **LRU**. **Last Recently Used (LRU)** - это алгоритм, при котором вытесняются элементы, которые дольше всего не запрашивались. Поэтому необходимо хранить дополнительный бит (достаточно

1 бита, так как из ассоциативности **`CACHE_WAY = 2`** следует, что для хранения этой информации для 2 кэш-линий достаточно выделить 1 бит) для каждой кэш-линии, который будет отвечать за её возраст, - флаг **O (old)** (0 - кэш-линия новая, 1 - старая). Кроме этого есть два других служебных бита (флага): флаг **V (valid)** указывает на то, свободна ли кэш-линия (0 - данная кэш-линия свободна, 1 - занята), флаг **D (dirty)** указывает на то, были ли записаны измененные данные в кэш-линии в память (1 - кэш-линия хранит измененные данные, которые еще не были записаны в память). Ниже на рисунке 2 приведено устройство кэш-линии.

old	valid	dirty	tag	data
1	1	1	CACHE_TAG_SIZE	CACHE_LINE_SIZE

Рисунок 2 - Устройство кэш-линии

Ниже на рисунке 3 приведен пример работы кэш-блока по политике вытеснения **Last Recently Used (LRU)**.

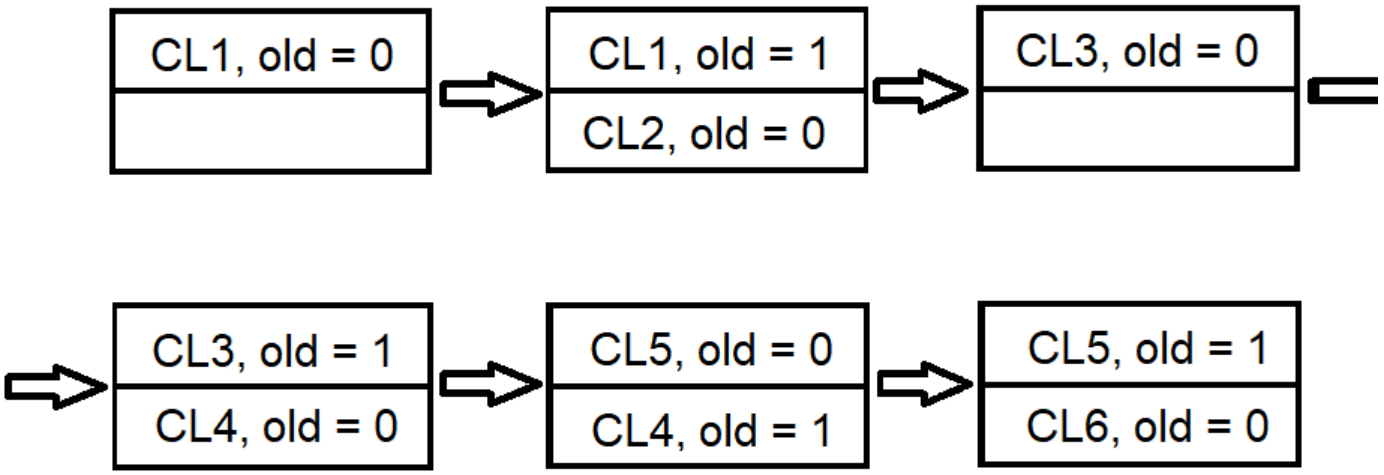

Рисунок 3 - Пример работы по политике вытеснения LRU.

Для аналитического решения необходимо определить общее время (в тактах), затраченное на выполнение заданной функции, и процент попаданий. Из поставленной задачи следует, что для решения не важно, что хранится в **data**, поэтому в реализации на **Java** достаточно хранить информацию о флагах (**old,**

valid, dirty) и **tag**. Так как флаги - **old, valid, dirty** - могут принимать значение 0 или 1 каждый, то 0 можно интерпретировать как **false**, а 1 - как **true**. Поэтому в реализации эти флаги имеют логический тип **boolean**. **Tag** удобно хранить в виде строкового типа **string**. Ниже представлена реализации кэш-линии **CACHE_LINE** в виде класса CacheLine, в коде присутствуют комментарии для понимания того, чем служит каждая переменная и что делают объявленные методы.

```
class CacheLine {
   public boolean old; // указывает какую первой удалять
   public boolean dirty; // 1 (true) => еще не записаны в память
   public boolean valid; // 0 (false) => кэш-линия свободна, 1
(true) => занята
   public String tag; // tag хранится в string
   public CacheLine() {
       this.old = false;
       this.dirty = false;
       this.valid = false;
       this.tag = new String();
   }
}
```

CACHE_SET

Как было вычислено в пункте отчета 1. вычисление недостающих параметров системы ассоциативность **CACHE_WAY = 2**, поэтому кэш разделён на кэш-блоки по 2 кэш-линии каждый. Кэш-блок **CACHE_SET** реализован в классе CacheSet. Ниже приведена часть кода-реализации кэш-блока.

```
class CacheSet {
   public CacheLine line_1; // кэш-линия 1
   public CacheLine line_2; // кэш-линия 2

   public CacheSet() {
       this.line_1 = new CacheLine();
       this.line_2 = new CacheLine();
   }
```

В данной части кода объявляются кэш-линии блока: кэш-линия 1 и кэш-линия 2. Кроме этого в данном классе CacheSet объявлено 2 метода. Ниже представлен первый метод.

```
public boolean isCacheHit(String tag) {
   return (line_1.tag.equals(tag) &&  line_1.valid) ||
          (line_2.tag.equals(tag) && line_2.valid);
}
```

Этот метод определяет, произошло ли кэш-попадание, другими словами, есть ли кэш-линия с таким tag в данном кэш-блоке.

Следующая вспомогательная функция считает число тактов за одно обращение к памяти. Ниже приведена упомянутая функция, после её реализации следует описание работы этой функции.

```
public void memAccess(String tag, int bytes, Number tacts,
                      Number hits, boolean read)
{

//         кэш-попадание:
   if (isCacheHit(tag)) { // такой тег есть => кэш-попадание

       hits.number += 1; // увеличиваем счетчик кэш-попаданий
       tacts.number += 7; // кэш-попадание => кэш отвечает за 6
тактов и на 7 получаем ответ
       if (read) {
           if (bytes <= 2) {// проверка сколько битов передаётся
кэш -> цпу
               tacts.number += 1; // если 1 или 2 байта (8 или 16
бит)
           } else {
               tacts.number += 2; //  4 байта кэш -> цпу
           }
       }

       if (line_1.tag.equals(tag)) {
           line_1.old = false;
           if (line_2.valid) {
               line_2.old = true;
           }
           if (read) {
               line_1.dirty = true;
           }
       } else if (line_2.tag.equals(tag)) {
           line_2.old = false;
           if (line_1.valid) {
               line_1.old = true;
           }
           if (read) {
               line_2.dirty = true;
           }
       }

//             кэш - промах
   } else { // случился кэш-промах

       tacts.number += 4; // кэш посылает запрос к памяти MemCTR
       tacts.number += 100; // MemCTR отвечает за 100 тактов
(здесь учитывается 1 такт - передача адреса по A2
       tacts.number += 8; // передается обратно по A2 по 2 байта
поэтому CACHE_LINE_SIZE / 2 = 8
```

```
        if (line_1.valid && line_2.valid) {
            if (line_1.old) {
                if (line_1.dirty) {
                    tacts.number += 100;
                    line_1.dirty = false;
                }
                line_1.tag = tag;
                line_1.old = false;
                line_2.old = true;
            } else if (line_2.old) {
                if (line_2.dirty) {
                    tacts.number += 100;
                    line_2.dirty = false;
                }
                line_2.tag = tag;
                line_2.old = false;
                line_1.old = true;
            }
        } else if (!line_1.valid) {
            line_1.valid = true;
            line_1.old = false;
            line_1.tag = tag;
            if (read) {
                line_1.dirty = true;
            } else {
                line_1.dirty = false;
            }
            if (line_2.valid) {
                line_2.old = true;
            }
        } else if (!line_2.valid) {
            line_2.valid = true;
            line_2.old = false;
            line_2.tag = tag;
            if (read) {
                line_2.dirty = true;
            } else {
                line_2.dirty = false;
            }
            line_1.old = true;
        }

        if (read) {
            if (bytes <= 2) {
                tacts.number += 1;
            } else {
                tacts.number += 2;
            }
        }
    }
}
```

Теперь опишем работу этой функции. Эта функция симулирует одно обращение к памяти и рассматривает все возможные случаи при этом обращении:

кэш-промахи и кэш-попадания. Разберем, что происходит в каждом из этих случаев. Для начала с помощью ранее описанной функции мы определяем, случилось ли кэш-попадание.

Случай 1: случилось кэш-попадание. Тогда увеличиваем счетчик кэш-попаданий (**hits**) на 1. По условию время отклика кэша в случае кэш-попадания равно 6 тактам, это значит, что за эти 6 тактов передаются по шине **A1 tag + set**, потом **offset** (это происходит за 2 такта), на 7 такт произойдет ответ. В случае, если идёт чтение, то в зависимости от запроса данные передаются обратно по шине D1 (а именно 8 или 16 бит - 1 так, 32 бита - 2 такта) итоговое число тактов увеличится еще на 1 или 2 соответственно. Ниже на рисунке 4 схематично представлен данный алгоритм.

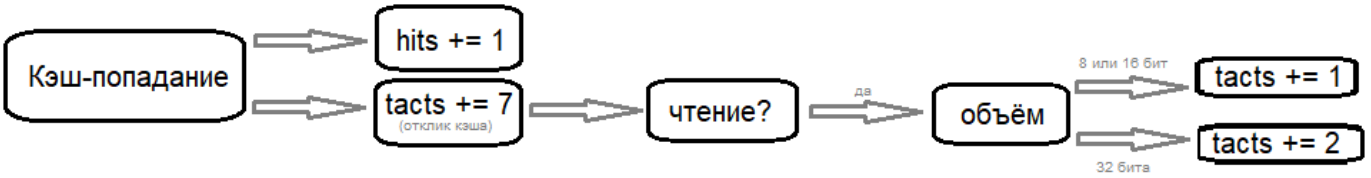

Рисунок 4 - Алгоритм в случае кэш-попадания

Случай 2: случился кэш-промах. В случае кэш-промаха по условию кэш посылает запрос к памяти за 4 такта, после чего проходит еще 100 тактов, за которые происходит отклик от первого такта команды до ответа **MemCTR**. После чего возможны два случая: так как нужно выгрузить данные на место кэш-линии, то важно знать, какой у неё флаг **dirty**. Если **dirty = 0**, то записать кэш-линию в память не требуется, тогда за 8 тактов данные выгрузятся из **Memory** в **Cache**. Если **dirty = 1**, то из **Memory** в **Cache** загрузка данных происходит за 100 + 8 = 108 тактов. Ниже на рисунке 5 схематично представлен данный алгоритм.

Рисунок 5 - Алгоритм в случае кэш-промаха

В представленной выше функции прописаны все эти случаи с помощью блоков **if** и **else**.

CACHE

Кэш **CACHE** состоит из **CACHE_SETS_COUNT = 64** кэш-блоков. Кэш **CACHE** реализован в классе Cache. Ниже представлена данная реализация.

```
class Cache {
   public CacheSet[] sets = new CacheSet[64];

   public Cache() {
       for (int i = 0; i < 64; i++) {
           sets[i] = new CacheSet(); // создание кэша из 64
кэш-блоков
       }
   }

   public void memAccess (String address, int bytes, Number
tacts, Number misses, boolean read_write) {
       int set = Integer.parseInt(address.substring(8, 14), 2);
       String tag = address.substring(0, 8);
       sets[set].memAccess(tag, bytes, tacts, misses,
read_write);
   }
}
```

В данном классе объявляется сам кэш: он состоит из 64 кэш-блоков. Кроме этого реализована вспомогательная функция, которая эмулирует обращение к памяти. В этой функции участвует ранее описанная функция обращения к памяти кэш-блока.

Основное решение MAIN

В классе Main реализована данная функция. Кроме того также реализована вспомогательная функция, которая формирует адрес как двоичное представление соответствующего числа. Ниже представлена её реализация.

```
public static String createAddress(int a) {
   StringBuilder address = new
StringBuilder(Integer.toBinaryString(a));
   while (address.length() < 18) {
       address.insert(0, '0');
   }
   return address.toString();
}
```

В классе объявлен счетчик тактов tacts, счетчик кэш-попаданий hits, счетчик запросов requests. В реализации функции из условия происходит следующее: описывается данный в условии алгоритм, в котором после каждого действия, на которое требуется какое-то количество тактов, увеличивается счетчик тактов tacts на соответствующую величину. В третьем цикле for:

```
for (int k1 = 0; k1 < K; k1++) {
                    tacts.number += 1;
                    requests += 2;
//                      s += pa[k1] * pb[x]
                    cache.memAccess(createAddress(pa + k1), 1,
tacts, hits, false);
                    cache.memAccess(createAddress(pb + 2 * x), 2,
tacts, hits, false);
                    tacts.number += 5; //умножение
                    pb += 2 * N;
                    tacts.number += 1;
                }
```

происходит два обращения к памяти для получения значений pa[k] и pb[x]. После выхода из этого цикла происходит еще одно обращение к памяти для записи значения переменной s:

```
cache.memAccess(createAddress(pc + 4 * x), 4, tacts, hits, true);
```

Выводится количество тактов, процент попаданий, число кэш-попаданий, общее число запросов. Ниже представлена полная реализация класса Main.

```
public class Main {
   public static String createAddress(int a) {
        StringBuilder address = new
StringBuilder(Integer.toBinaryString(a));
        while (address.length() < 18) {
            address.insert(0, '0');
        }
        return address.toString();
   }
   public static void main(String[] args) {
        Cache cache = new Cache();

        Number tacts = new Number(); // счетчик тактов
```

```
        Number hits = new Number(); // счетчик попаданий
        int requests = 0; // счетчик обращений
        int M = 64;
        int N = 60;
        int K = 32;

        tacts.number += 3; // инициализация M, N, K
        int pa = 0;
        int pc = 5888;
        tacts.number += 2;

        for (int y = 0; y < M; y++) {
            tacts.number += 1;
            for (int x = 0; x < N; x++) {
                tacts.number += 1;

                int pb = 2048;
                tacts.number += 1;
                int s = 0;
                tacts.number += 1;

                for (int k1 = 0; k1 < K; k1++) {
                    tacts.number += 1;
                    requests += 2;
//                      s += pa[k1] * pb[x]
                    cache.memAccess(createAddress(pa + k1), 1,
tacts, hits, false);
                    cache.memAccess(createAddress(pb + 2 * x), 2,
tacts, hits, false);
                    tacts.number += 5; //умножение
                    pb += 2 * N;
                    tacts.number += 1;
                }
//                 pc[x] = s;
                requests += 1;
                cache.memAccess(createAddress(pc + 4 * x), 4,
tacts, hits, true);
            }
            pa += K;
            tacts.number += 1;
            pc += 4 * N;
            tacts.number += 1;
        }
        tacts.number += 1;
        System.out.println("tacts: " + tacts.number + " pr hits: "
+ ((double) hits.number / (double) requests) * 100 + "%" + "
hits: " + hits.number + "  requests " + requests);
    }
}
```

Ниже представлена общая реализация всего аналитического решения (всех классов).

```
package testbench;

class Number {
   public int number;
}
class CacheLine {
   public boolean old; // указывает какую первой удалять
   public boolean dirty; // 1 (true) => еще не записаны в память
   public boolean valid; // 0 (false) => кэш-линия свободна, 1
(true) => занята
   public String tag; // tag хранится в string
   public CacheLine() {
       this.old = false;
       this.dirty = false;
       this.valid = false;
       this.tag = new String();
   }
}

class CacheSet {
   public CacheLine line_1; // кэш-линия 1
   public CacheLine line_2; // кэш-линия 2

   public CacheSet() {
       this.line_1 = new CacheLine();
       this.line_2 = new CacheLine();
   }

   public boolean isCacheHit(String tag) {
       return (line_1.tag.equals(tag) &&  line_1.valid) ||
               (line_2.tag.equals(tag) && line_2.valid);
   }

   public void memAccess(String tag, int bytes, Number tacts,
Number hits, boolean read) {

//         кэш-попадание:
       if (isCacheHit(tag)) { // такой тег есть => кэш-попадание

           hits.number += 1; // увеличиваем счетчик кэш-попаданий
           tacts.number += 7; // кэш-попадание => кэш отвечает за
6 тактов и на 7 получаем ответ
           if (read) {
               if (bytes <= 2) {// проверка сколько битов
передаётся кэш -> цпу
                   tacts.number += 1; // если 1 или 2 байта (8
или 16 бит)
               } else {
                   tacts.number += 2; //  4 байта кэш -> цпу
               }
           }

           if (line_1.tag.equals(tag)) {
               line_1.old = false;
               if (line_2.valid) {
```

```
                    line_2.old = true;
                }
                if (read) {
                    line_1.dirty = true;
                }
            } else if (line_2.tag.equals(tag)) {
                line_2.old = false;
                if (line_1.valid) {
                    line_1.old = true;
                }
                if (read) {
                    line_2.dirty = true;
                }
            }

//              кэш - промах
        } else { // случился кэш-промах

            tacts.number += 4; // кэш посылает запрос к памяти
MemCTR
            tacts.number += 100; // MemCTR отвечает за 100 тактов
(здесь учитывается 1 такт - передача адреса по A2
            tacts.number += 8; // передается обратно по A2 по 2
байта поэтому CACHE_LINE_SIZE / 2 = 8

            if (line_1.valid && line_2.valid) {
                if (line_1.old) {
                    if (line_1.dirty) {
                        tacts.number += 100;
                        line_1.dirty = false;
                    }
                    line_1.tag = tag;
                    line_1.old = false;
                    line_2.old = true;
                } else if (line_2.old) {
                    if (line_2.dirty) {
                        tacts.number += 100;
                        line_2.dirty = false;
                    }
                    line_2.tag = tag;
                    line_2.old = false;
                    line_1.old = true;
                }
            } else if (!line_1.valid) {
                line_1.valid = true;
                line_1.old = false;
                line_1.tag = tag;
                if (read) {
                    line_1.dirty = true;
                } else {
                    line_1.dirty = false;
                }
                if (line_2.valid) {
                    line_2.old = true;
                }
```

```
            } else if (!line_2.valid) {
                line_2.valid = true;
                line_2.old = false;
                line_2.tag = tag;
                if (read) {
                    line_2.dirty = true;
                } else {
                    line_2.dirty = false;
                }
                line_1.old = true;
            }

            if (read) {
                if (bytes <= 2) {
                    tacts.number += 1;
                } else {
                    tacts.number += 2;
                }
            }
        }
    }
}

class Cache {
    public CacheSet[] sets = new CacheSet[64];

    public Cache() {
        for (int i = 0; i < 64; i++) {
            sets[i] = new CacheSet(); // создание кэша из 64
кэш-блоков
        }
    }

    public void memAccess (String address, int bytes, Number
tacts, Number misses, boolean read_write) {
        int set = Integer.parseInt(address.substring(8, 14), 2);
        String tag = address.substring(0, 8);
        sets[set].memAccess(tag, bytes, tacts, misses,
read_write);
    }
}
public class Main {
    public static String createAddress(int a) {
        StringBuilder address = new
StringBuilder(Integer.toBinaryString(a));
        while (address.length() < 18) {
            address.insert(0, '0');
        }
        return address.toString();
    }
    public static void main(String[] args) {
        Cache cache = new Cache();

        Number tacts = new Number(); // счетчик тактов
        Number hits = new Number(); // счетчик попаданий
```

```
        int requests = 0; // счетчик обращений
        int M = 64;
        int N = 60;
        int K = 32;

        tacts.number += 3; // инициализация M, N, K
        int pa = 0;
        int pc = 5888;
        tacts.number += 2;

        for (int y = 0; y < M; y++) {
            tacts.number += 1;
            for (int x = 0; x < N; x++) {
                tacts.number += 1;

                int pb = 2048;
                tacts.number += 1;
                int s = 0;
                tacts.number += 1;

                for (int k1 = 0; k1 < K; k1++) {
                    tacts.number += 1;
                    requests += 2;
//                      s += pa[k1] * pb[x]
                    cache.memAccess(createAddress(pa + k1), 1,
tacts, hits, false);
                    cache.memAccess(createAddress(pb + 2 * x), 2,
tacts, hits, false);
                    tacts.number += 5; //умножение
                    pb += 2 * N;
                    tacts.number += 1;
                }
//                  pc[x] = s;
                requests += 1;
                cache.memAccess(createAddress(pc + 4 * x), 4,
tacts, hits, true);
            }
            pa += K;
            tacts.number += 1;
            pc += 4 * N;
            tacts.number += 1;
        }
        tacts.number += 1;
        System.out.println("tacts: " + tacts.number + " pr hits: "
+ ((double) hits.number / (double) requests) * 100 + "%" + "
hits: " + hits.number + "  requests " + requests);
    }
}
```

3. Моделирование заданной системы на Verilog.

Моделирование заданной системы на Verilog идейно схоже с реализацией на языке Java, которая была произведена при аналитическом решении. Схема моделируемой системы представлена ниже на рисунке 6.

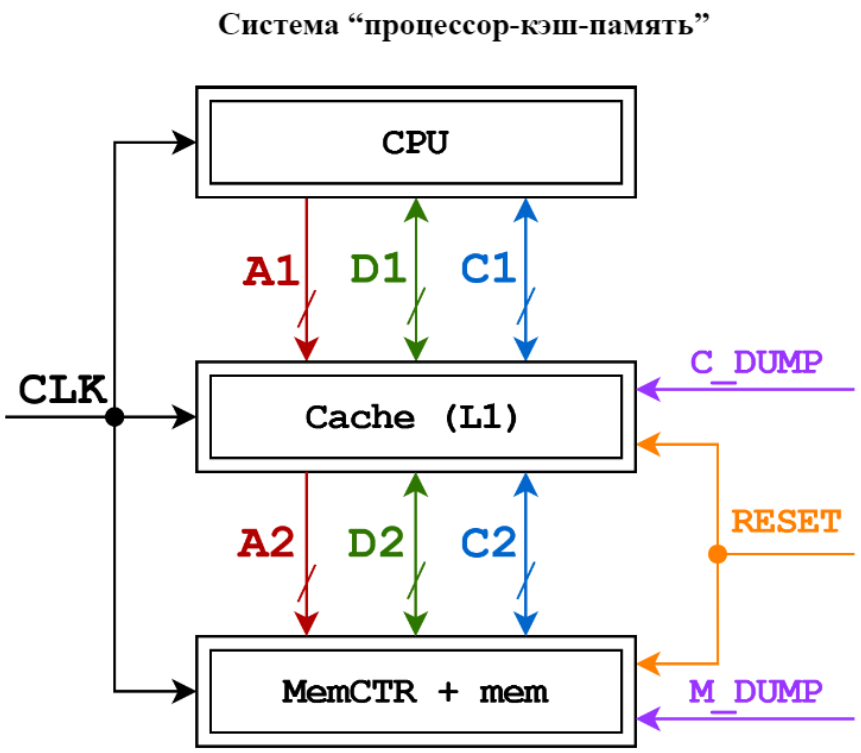

Рисунок 6 - Схема моделируемой системы.

CACHE

В таблице 5 представлены команды которые нужно было реализовать для “общения” **CPU** и **Cache**.

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**
CPU ← Cache	0 – **C1_NOP** 7 – **C1 RESPONSE**

Таблица 5 - Команды “общения” CPU и Cache.

В коде реализовано 5 вспомогательных task, которые служили для избавления от постоянно повторяющихся строчек кода в других реализованных task, в каждом из них уже предусмотрены расстановки тактов. Ниже представлены все эти task.

```
task byte_to_cpu(input firstSecond, input [2 : 0] number_bytes);
    begin
        l_C1 = 1;
        C1_reg = 7;
        writeD1 = 1;
        for (i = 0; i < number_bytes; i += 2)
        begin
            #1;
            D1_reg[7 : 0] = cache_line_data[addr_set * 2 + firstSecond][addr_offset +
i];
            if (number_bytes > 1)
            begin
                D1_reg[15 : 8] = cache_line_data[addr_set * 2 +
firstSecond][addr_offset + i + 1];
            end
            #1;
        end
        C1_reg = 0;
        l_C1 = 0;
        writeD1 = 0;
    end
    endtask
```

```
task byte_from_cpu(input firstSecond, input [2 : 0] number_bytes);
    begin
        l_C1 = 1;
        C1_reg = 7;
        for (i = 0; i < number_bytes; i += 2)
        begin
            #1;
            cache_line_data[addr_set * 2 + firstSecond][addr_offset + i] = D1[7 : 0];
            if (number_bytes > 1)
            begin
                cache_line_data[addr_set * 2 + firstSecond][addr_offset + i + 1] =
D1[15 : 8];
            end
            #1;
        end
```

```
        C1_reg = 0;
        l_C1 = 0;
    end
    endtask
```

```
task read_line_mem(input firstSecond);
    begin
        l_C2 = 1;
        C2_reg = 2;
        #200 l_C2 = 0;
        for (i = 0; i < 8; i++)
        begin
            #1 cache_line_data[addr_set * 2 + firstSecond][i * 2] = D2[7 : 0];
            cache_line_data[addr_set * 2 + firstSecond][i * 2 + 1] = D2[15 : 8];
            #1;
        end
        cache_line_tag[addr_set * 2 + firstSecond] = addr_tag;
        cache_line_dirty[addr_set * 2 + firstSecond] = 0;
    end
    endtask
```

```
task write_line_mem(input firstSecond);
    begin
        l_C2 = 1;
        C2_reg = 3;
        #200 l_C2 = 0;
        for (i = 0; i < 16; i += 2)
        begin
            #1 D2_reg[7 : 0] = cache_line_data[addr_set * 2 + firstSecond][i];
            D2_reg[15 : 8] = cache_line_data[addr_set * 2 + firstSecond][i + 1];
            #1;
        end
    end
    endtask
```

```
task read_addr;
    begin
        addr_set = A1[CACHE_SET_SIZE - 1 : 0];
        addr_tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : CACHE_SET_SIZE];
        #2;
```

```
        addr_offset = A1[CACHE_OFFSET_SIZE - 1 : 0];
        requests += 1;
    end
    endtask
```

Кроме этих task, реализованы 2 основных, которые отвечают за запись и чтение. Каждый из них отвечает за запись определённого количества байт, это количество подается им на вход. Ниже представлена реализации task - чтение байтов.

```
task read_bytes(input [2 : 0] number_bytes);
    begin
        read_addr();
        if (cache_line_tag[addr_set * 2] == addr_tag && cache_line_valid[addr_set *
2] == 1)
        begin
            // кэш-попадание первая кэш-линия
            #8;
            byte_to_cpu(0, number_bytes);
            hits += 1;
        end
        else if (cache_line_tag[addr_set * 2 + 1] == addr_tag &&
cache_line_valid[addr_set * 2 + 1] == 1)
        begin
            // кэш-попадание вторая кэш-линия
            #8;
            byte_to_cpu(1, number_bytes);
            hits += 1;
        end
        else
        begin
            // кэш-промах
            #4;
            if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set * 2
+ 1] == 1)
            begin
                // линия1 и линия2 заняты
                if (cache_line_old[addr_set * 2] == 1)
                begin
                    // линия1 old
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2] == 1)
                    begin
```

```
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                l_C2 = 0;
                byte_to_cpu(0, number_bytes);
            end
            else
            begin
                // линия2 old
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    write_line_mem(1);
                end
                read_line_mem(1);
                cache_line_old[addr_set * 2 + 1] = 0;
                cache_line_old[addr_set * 2] = 1;
                byte_to_cpu(1, number_bytes);
            end
        end
        else if (cache_line_valid[addr_set * 2] == 0 && cache_line_valid[addr_set
* 2 + 1] == 1)
        begin
            A2_reg = {addr_tag, addr_set};
            if (cache_line_dirty[addr_set * 2] == 1)
            begin
                write_line_mem(0);
            end
            read_line_mem(0);
            cache_line_valid[addr_set * 2] = 1;
            cache_line_old[addr_set * 2] = 0;
            cache_line_old[addr_set * 2 + 1] = 1;
            byte_to_cpu(0, number_bytes);
        end
        else if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set
* 2 + 1] == 0)
        begin
            A2_reg = {addr_tag, addr_set};
            if (cache_line_dirty[addr_set * 2 + 1] == 1)
            begin
                write_line_mem(1);
            end
            read_line_mem(1);
```

```
                cache_line_valid[addr_set * 2 + 1] = 1;
                cache_line_old[addr_set * 2 + 1] = 0;
                cache_line_old[addr_set * 2] = 1;
                byte_to_cpu(1, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 0 && cache_line_valid[addr_set
* 2 + 1] == 0)
            begin
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                byte_to_cpu(0, number_bytes);
            end
        end
    end
    endtask
```

Ниже представлена реализация task, отвечающего за запись байтов.

```
task write_bytes(input [1 : 0] number_bytes);
    begin
        if (cache_line_tag[addr_set * 2] == addr_tag && cache_line_valid[addr_set *
2] == 1)
        begin
            // кэш-попадание первая кэш-линия
            #8;
            byte_from_cpu(0, number_bytes);
        end
        else if (cache_line_tag[addr_set * 2 + 1] == addr_tag &&
cache_line_valid[addr_set * 2 + 1] == 1)
        begin
            // кэш-попадание вторая кэш-линия
            #8;
            byte_from_cpu(1, number_bytes);
        end
        else
        begin
```

```
            // кэш-промах
            #4;
            if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set * 2
+ 1] == 1)
            begin
                // линия1 и линия2 заняты
                if (cache_line_old[addr_set * 2] == 1)
                begin
                    // линия1 old
                    l_C2 = 1;
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2] == 1)
                    begin
                        C2_reg = 3;
                        write_line_mem(0);
                    end
                    C2_reg = 2;
                    read_line_mem(0);
                    cache_line_old[addr_set * 2] = 0;
                    cache_line_old[addr_set * 2 + 1] = 1;
                    l_C2 = 0;
                    byte_from_cpu(0, number_bytes);
                end
                else
                begin
                    // линия2 old
                    l_C2 = 1;
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2 + 1] == 1)
                    begin
                        C2_reg = 3;
                        write_line_mem(1);
                    end
                    C2_reg = 2;
                    read_line_mem(1);
                    cache_line_old[addr_set * 2 + 1] = 0;
                    cache_line_old[addr_set * 2] = 1;
                    l_C2 = 0;
                    byte_from_cpu(1, number_bytes);
                end
            end
            else if (cache_line_valid[addr_set * 2] == 0 && cache_line_valid[addr_set
* 2 + 1] == 1)
            begin
                l_C2 = 1;
```

```
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(0);
                end
                C2_reg = 2;
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                l_C2 = 0;
                byte_from_cpu(0, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set
* 2 + 1] == 0)
            begin
                l_C2 = 1;
                A2_reg = {addr_tag, addr_set};
                C2_reg = 2;
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(1);
                end
                read_line_mem(1);
                cache_line_valid[addr_set * 2 + 1] = 1;
                cache_line_old[addr_set * 2 + 1] = 0;
                cache_line_old[addr_set * 2] = 1;
                l_C2 = 0;
                byte_from_cpu(1, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 0 && cache_line_valid[addr_set
* 2 + 1] == 0)
            begin
                l_C2 = 1;
                A2_reg = {addr_tag, addr_set};
                C2_reg = 2;
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
```

```
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                l_C2 = 0;
                byte_from_cpu(0, number_bytes);
            end
        end
    end
    endtask
```

Далее в always описываются команды **CPU → Cache** и **CPU ← Cache**. Ниже представлена реализация.

```
always @(posedge clk)
    begin
        #1;
        case (C1)
        3'b000 :  // NOP
        begin
            $display("nothing");
        end
        3'b001 : // READ8
        begin
            read_bytes(1);
        end
        3'b010 : // READ16
        begin
            read_bytes(2);
        end
        3'b011 : //READ32
        begin
            read_bytes(4); //READ32
        end
        3'b100 : // INVALIDATE_LINE
        begin
            addr_set = A1[5 : 0];
            write_line_mem(A1[6]);
        end
        3'b101 : // WRITE8
        begin
            write_bytes(1);
        end
        3'b110 : // WRITE16
        begin
```

```
            write_bytes(2);
        end
        3'b111 : // WRITE32
        begin
            write_bytes(4);
        end
        endcase
    end
```

MEMORY

В таблице 6 представлены команды, которые нужно было реализовать для общения **Memory** и **Cache**.

Cache → Mem	0 – **C2_NOP** 2 – **C2_READ_LINE** 3 – **C2_WRITE_LINE**
Cache ← Mem	0 – **C2_NOP** 1 – **C2_RESPONSE**

Таблица 6 - Команды для "общения" Memory и Cache.

Реализация этих команд в always представлена ниже.

```
always @(posedge clk)
begin
    #2
    case(C2)
        2'b00 : // NOP
        begin
            $display("nop");
        end
        2'b01 : // RESPONSE
        begin
            $display("response");
        end
        2'b10 : // READ_LINE
        begin
            //reading c.line
            #198;
            // wire = 1;
            l_C2 = 1;
            C2_reg = 1;
```

```
            for (i = A2 * 8; i < A2 * 8 + 16; i += 2)
            begin
                D2_reg = {memory[i + 1], memory[i]};
                #2;
            end
            C2_reg = 0;
            l_C2 = 0;
            write = 0;
        end
        2'b11 : // WRITE_LINE
        begin
            //writing c.line
            #198 write = 1;
            l_C2 = 1;
            C2_reg = 1;
            for (i = A2 * 8; i < A2 * 8 + 16; i += 2)
            begin
                memory[i] = D2[7 : 0];
                memory[i + 1] = D2[15 : 8];
                #2;
            end
            l_C2 = 0;
            C2_reg = 0;
        end
    endcase
end
```

4. Воспроизведение задачи на Verilog.

Воспроизведение задачи на Verilog также схоже идейно с воспроизведением на Java, которое было сделано в аналитическом решении. Также в этом же коде внутри считается количество запросов, а количество кэш-попаданий считается в Cache. При реализации у меня произошла проблема: программа не могла проложить свою работу, когда входила в любой из **wait** (их всего 3 в коде). Ниже представлена реализация.

```
wait(C1 === 7);
```

К выводу о том, что программа не может продолжить свою работу именно в этих местах, я пришла путем нескольких экспериментов (например комментировала их, выводила что-либо на экран внутри каждого из циклов и так далее). К сожалению, я так и не смогла “пофикисить” эту проблему, даже переписав заново весь код для **cache**, я не поняла, где ошибка. Поэтому мне пришлось подобрать на место этих **wait** какое-то количество тактов. Из-за такого “подбора” общее общее время (в тактах), затраченное на выполнение этой функции, и число кэш-попаданий посчитано неточно, с некоторой погрешностью. Но при этом число обращений считается независимо от этого, поэтому оно посчитано без погрешностей. При этом в самом коде те места, где должны стоять **wait**, помечены закомментированным **wait** (*//wait(C1 == 7)*). Ниже представлена реализация заданной функции.

```
initial
begin
    clk = 1;
    M = 64;
    #2;
    N = 60;
    #2;
    K = 32;
    #2;
    pa = 0;
    #2;
    pc = 0;
    #2;
    requests_count = 0;
    for (y = 0; y < M; y += 1)
    begin
        for (x = 0; x < N; x += 1)
        begin
            pb = 0;
            #2;
            s = 0;
            #2;
            for (k = 0; k < K; k += 1)
            begin
                requests_count += 2;
                C1_reg = 1;
                l_C1 = 1;
                A1_reg = (pa * K + k) >> CACHE_OFFSET_SIZE;
```

```
                #2;
                A1_reg = (pa * K + k) % CACHE_LINE_SIZE;
                #2;
                l_C1 = 0;
                #1
                //wait(C1 == 7);
                #7;
                a = D1[7 : 0];
                C1_reg = 2;
                l_C1 = 1;
                A1_reg = (M * K + (pb * N + x) * 2) >> CACHE_OFFSET_SIZE;
                #2;
                A1_reg = (M * K + (pb * N + x) * 2) % CACHE_LINE_SIZE;
                #2;
                l_C1 = 0;
                #1;
                //wait(C1 == 7);
                #50;
                b = D1[7 : 0];
                s += a * b;
                #10;
                pb += 1;
                #2;
                #2;
            end
            requests_count += 1;
            C1_reg = 7;
            l_C1 = 1;
            A1_reg = (M * K + K * N * 2 + (pc * N + x) * 4) >> CACHE_OFFSET_SIZE;
            #2;
            A1_reg = (M * K + K * N * 2 + (pc * N + x) * 4) % CACHE_LINE_SIZE;
            #2;
            l_C1 = 0;
            #2;
            //wait(C1 == 7);
            #3;
            #2;
        end
        pa += 1;
        #2;
        pc += 1;
        #2;
        #2;
    end
    #2;
```

```
    $display("Cache hits: %0d", hits);
    $display("Cache requests: %0d", requests_count);
    $display("Tacts: ", $time);
    #1;
    $finish;
end
```

5. Сравнение полученных результатов.

Как уже было описано в пункте 4 (воспроизведение задачи на Verilog) из-за возникшей проблемы с **wait** произошла погрешность в полученных результатах, но при этом видно, что они схожи (как минимум, если округлять). Ниже приведена таблица сравнения полученных результатов.

Значение	Аналитическое решение на Java	Решение на Verilog
Всего тактов	4695168	5005638
число кэш-попаданий	230698	244994
число обращений	249600	249600
процент попадания	92.427083%	98.154647%

6. Листинг кода.

Аналитическое решение

```
package testbench;

class Number {
    public int number;
}
class CacheLine {
    public boolean old; // указывает какую первой
удалять
    public boolean dirty; // 1 (true) => еще не записаны
в память
    public boolean valid; // 0 (false) => кэш-линия
```

```
свободна, 1 (true) => занята
    public String tag; // tag хранится в string
    public CacheLine() {
        this.old = false;
        this.dirty = false;
        this.valid = false;
        this.tag = new String();
    }
}

class CacheSet {
    public CacheLine line_1; // кэш-линия 1
    public CacheLine line_2; // кэш-линия 2

    public CacheSet() {
        this.line_1 = new CacheLine();
        this.line_2 = new CacheLine();
    }

    public boolean isCacheHit(String tag) {
        return (line_1.tag.equals(tag) &&  line_1.valid)
||
                (line_2.tag.equals(tag) &&
line_2.valid);
    }

    public void memAccess(String tag, int bytes, Number
tacts, Number hits, boolean read) {

//        кэш-попадание:
        if (isCacheHit(tag)) { // такой тег есть =>
кэш-попадание

            hits.number += 1; // увеличиваем счетчик
кэш-попаданий
            tacts.number += 7; // кэш-попадание => кэш
отвечает за 6 тактов и на 7 получаем ответ
            if (read) {
                if (bytes <= 2) {// проверка сколько
битов передаётся кэш -> цпу
                    tacts.number += 1; // если 1 или 2
байта (8 или 16 бит)
                } else {
                    tacts.number += 2; //  4 байта кэш
-> цпу
                }
            }
```

```
            if (line_1.tag.equals(tag)) {
                line_1.old = false;
                if (line_2.valid) {
                    line_2.old = true;
                }
                if (read) {
                    line_1.dirty = true;
                }
            } else if (line_2.tag.equals(tag)) {
                line_2.old = false;
                if (line_1.valid) {
                    line_1.old = true;
                }
                if (read) {
                    line_2.dirty = true;
                }
            }

//              кэш - промах
        } else { // случился кэш-промах

            tacts.number += 4; // кэш посылает запрос к памяти MemCTR
            tacts.number += 100; // MemCTR отвечает за 100 тактов (здесь учитывается 1 такт - передача адреса по A2
            tacts.number += 8; // передается обратно по A2 по 2 байта поэтому CACHE_LINE_SIZE / 2 = 8

            if (line_1.valid && line_2.valid) {
                if (line_1.old) {
                    if (line_1.dirty) {
                        tacts.number += 100;
                        line_1.dirty = false;
                    }
                    line_1.tag = tag;
                    line_1.old = false;
                    line_2.old = true;
                } else if (line_2.old) {
                    if (line_2.dirty) {
                        tacts.number += 100;
                        line_2.dirty = false;
                    }
                    line_2.tag = tag;
                    line_2.old = false;
                    line_1.old = true;
```

```
                }
            } else if (!line_1.valid) {
                line_1.valid = true;
                line_1.old = false;
                line_1.tag = tag;
                if (read) {
                    line_1.dirty = true;
                } else {
                    line_1.dirty = false;
                }
                if (line_2.valid) {
                    line_2.old = true;
                }
            } else if (!line_2.valid) {
                line_2.valid = true;
                line_2.old = false;
                line_2.tag = tag;
                if (read) {
                    line_2.dirty = true;
                } else {
                    line_2.dirty = false;
                }
                line_1.old = true;
            }

            if (read) {
                if (bytes <= 2) {
                    tacts.number += 1;
                } else {
                    tacts.number += 2;
                }
            }
        }
    }
}

class Cache {
    public CacheSet[] sets = new CacheSet[64];

    public Cache() {
        for (int i = 0; i < 64; i++) {
            sets[i] = new CacheSet(); // создание кэша
из 64 кэш-блоков
        }
    }

    public void memAccess (String address, int bytes,
```

```
Number tacts, Number misses, boolean read_write) {
        int set = Integer.parseInt(address.substring(8,
14), 2);
        String tag = address.substring(0, 8);
        sets[set].memAccess(tag, bytes, tacts, misses,
read_write);
    }
}
public class Main {
    public static String createAddress(int a) {
        StringBuilder address = new
StringBuilder(Integer.toBinaryString(a));
        while (address.length() < 18) {
            address.insert(0, '0');
        }
        return address.toString();
    }
    public static void main(String[] args) {
        Cache cache = new Cache();

        Number tacts = new Number(); // счетчик тактов
        Number hits = new Number(); // счетчик попаданий
        int requests = 0; // счетчик обращений
        int M = 64;
        int N = 60;
        int K = 32;

        tacts.number += 3; // инициализация M, N, K
        int pa = 0;
        int pc = 5888;
        tacts.number += 2;

        for (int y = 0; y < M; y++) {
            tacts.number += 1;
            for (int x = 0; x < N; x++) {
                tacts.number += 1;

                int pb = 2048;
                tacts.number += 1;
                int s = 0;
                tacts.number += 1;

                for (int k1 = 0; k1 < K; k1++) {
                    tacts.number += 1;
                    requests += 2;
//                      s += pa[k1] * pb[x]
                    cache.memAccess(createAddress(pa +
```

```
k1), 1, tacts, hits, false);
                    cache.memAccess(createAddress(pb + 2
* x), 2, tacts, hits, false);
                    tacts.number += 5; //умножение
                    pb += 2 * N;
                    tacts.number += 1;
                }
//                  pc[x] = s;
                requests += 1;
                cache.memAccess(createAddress(pc + 4 *
x), 4, tacts, hits, true);
            }
            pa += K;
            tacts.number += 1;
            pc += 4 * N;
            tacts.number += 1;
        }
        tacts.number += 1;
        System.out.println("tacts: " + tacts.number + "
pr hits: " + ((double) hits.number / (double) requests)
* 100 + "%" + " hits: " + hits.number + "  requests " +
requests);
    }
}
```

Система и решение на Verilog.

```
`include "memory.sv"
module cache #(
    parameter MEM_SIZE = 2**18,
    parameter CACHE_SIZE = 2**11,
    parameter CACHE_LINE_SIZE = 16,
    parameter CACHE_LINE_COUNT = 2**7,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 64,
    parameter CACHE_TAG_SIZE = 8,
    parameter CACHE_SET_SIZE = 6,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_ADDR_SIZE = 18,
    parameter ADDR1_BUS_SIZE = 14,
    parameter ADDR2_BUS_SIZE = 14,
    parameter DATA1_BUS_SIZE = 16,
```

```
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
)
(
    input wire clk,
    input wire [ADDR1_BUS_SIZE - 1 : 0] A1,
    inout wire [DATA1_BUS_SIZE - 1 : 0] D1,
    inout wire [CTR1_BUS_SIZE - 1 : 0] C1,
    output int hits,
    output int requests
);
    reg [CACHE_TAG_SIZE - 1 : 0] cache_line_tag [CACHE_LINE_COUNT - 1 : 0];
    byte cache_line_data [CACHE_LINE_COUNT - 1 : 0][CACHE_LINE_SIZE - 1 : 0];
    reg cache_line_old [CACHE_LINE_COUNT - 1 : 0];
    reg cache_line_valid [CACHE_LINE_COUNT - 1 : 0];
    reg cache_line_dirty [CACHE_LINE_COUNT - 1 : 0];
    reg writeD1;
    reg l_C1;
    reg writeD2;
    reg l_C2;
    reg [ADDR2_BUS_SIZE - 1 : 0] A2_reg;
    reg [DATA2_BUS_SIZE - 1 : 0] D2_reg;
    reg [CTR2_BUS_SIZE - 1 : 0] C2_reg;
    wire [DATA2_BUS_SIZE - 1 : 0] D2;
    wire [CTR2_BUS_SIZE - 1 : 0] C2;
    reg [CTR1_BUS_SIZE - 1 : 0] C1_reg;
    reg [DATA1_BUS_SIZE - 1 : 0] D1_reg;
    reg [CACHE_OFFSET_SIZE - 1 : 0] addr_offset;
    reg [CACHE_SET_SIZE - 1 : 0] addr_set;
    reg [CACHE_TAG_SIZE - 1 : 0] addr_tag;
    reg [7:0] byte_byte;
    assign D1 = writeD1 ? D1_reg : 'hz;
    assign C1 = l_C1 ? C1_reg : 'hz;
    assign D2 = writeD2 ? D2_reg : 'hz;
    assign C2 = l_C2 ? C2_reg : 'hz;
    assign byte_byte = cache_line_data[0][0];
    memory memory(.clk(clk), .A2(A2_reg), .D2(D2), .C2(C2));
    integer i;
    integer hits_count;
    integer requests_count;

    initial
    begin
      //  $dumpfile("dump2.vcd");
```

```
        // $dumpvars(0, clk, D1, C1, l_C2, C2, D2, writeD1, writeD2, addr_set,
addr_tag, addr_offset, A2_reg, byte_byte, D1_reg);
        writeD1 = 0;
        l_C1 = 0;
        writeD2 = 0;
        l_C2 = 0;
        for (i = 0; i < CACHE_LINE_COUNT; i++)
        begin
            cache_line_valid[i] = 0;
        end
    end

    task byte_to_cpu(input firstSecond, input [2 : 0] number_bytes);
    begin
        l_C1 = 1;
        C1_reg = 7;
        writeD1 = 1;
        for (i = 0; i < number_bytes; i += 2)
        begin
            #1;
            D1_reg[7 : 0] = cache_line_data[addr_set * 2 +
firstSecond][addr_offset + i];
            if (number_bytes > 1)
            begin
                D1_reg[15 : 8] = cache_line_data[addr_set * 2 +
firstSecond][addr_offset + i + 1];
            end
            #1;
        end
        C1_reg = 0;
        l_C1 = 0;
        writeD1 = 0;
    end
    endtask

    task byte_from_cpu(input firstSecond, input [2 : 0] number_bytes);
    begin
        l_C1 = 1;
        C1_reg = 7;
        for (i = 0; i < number_bytes; i += 2)
        begin
            #1;
            cache_line_data[addr_set * 2 + firstSecond][addr_offset + i] = D1[7 :
0];
            if (number_bytes > 1)
```

```
            begin
                cache_line_data[addr_set * 2 + firstSecond][addr_offset + i + 1]
= D1[15 : 8];
            end
            #1;
        end
        C1_reg = 0;
        l_C1 = 0;
    end
    endtask

    task read_line_mem(input firstSecond);
    begin
        l_C2 = 1;
        C2_reg = 2;
        #200 l_C2 = 0;
        for (i = 0; i < 8; i++)
        begin
            #1 cache_line_data[addr_set * 2 + firstSecond][i * 2] = D2[7 : 0];
            cache_line_data[addr_set * 2 + firstSecond][i * 2 + 1] = D2[15 : 8];
            #1;
        end
        cache_line_tag[addr_set * 2 + firstSecond] = addr_tag;
        cache_line_dirty[addr_set * 2 + firstSecond] = 0;
    end
    endtask

    task write_line_mem(input firstSecond);
    begin
        l_C2 = 1;
        C2_reg = 3;
        #200 l_C2 = 0;
        for (i = 0; i < 16; i += 2)
        begin
            #1 D2_reg[7 : 0] = cache_line_data[addr_set * 2 + firstSecond][i];
            D2_reg[15 : 8] = cache_line_data[addr_set * 2 + firstSecond][i + 1];
            #1;
        end
    end
    endtask

    task read_addr;
    begin
        addr_set = A1[CACHE_SET_SIZE - 1 : 0];
        addr_tag = A1[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : CACHE_SET_SIZE];
```

```
        #2;
        addr_offset = A1[CACHE_OFFSET_SIZE - 1 : 0];
        requests += 1;
    end
    endtask

    task read_bytes(input [2 : 0] number_bytes);
    begin
        read_addr();
        if (cache_line_tag[addr_set * 2] == addr_tag && cache_line_valid[addr_set
* 2] == 1)
        begin
            // кэш-попадание первая кэш-линия
            #8;
            byte_to_cpu(0, number_bytes);
            hits += 1;
        end
        else if (cache_line_tag[addr_set * 2 + 1] == addr_tag &&
cache_line_valid[addr_set * 2 + 1] == 1)
        begin
            // кэш-попадание вторая кэш-линия
            #8;
            byte_to_cpu(1, number_bytes);
            hits += 1;
        end
        else
        begin
            // кэш-промах
            #4;
            if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set
* 2 + 1] == 1)
            begin
                // линия1 и линия2 заняты
                if (cache_line_old[addr_set * 2] == 1)
                begin
                    // линия1 old
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2] == 1)
                    begin
                        write_line_mem(0);
                    end
                    read_line_mem(0);
                    cache_line_old[addr_set * 2] = 0;
                    cache_line_old[addr_set * 2 + 1] = 1;
                    l_C2 = 0;
```

```
                    byte_to_cpu(0, number_bytes);
                end
                else
                begin
                    // линия2 old
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2 + 1] == 1)
                    begin
                        write_line_mem(1);
                    end
                    read_line_mem(1);
                    cache_line_old[addr_set * 2 + 1] = 0;
                    cache_line_old[addr_set * 2] = 1;
                    byte_to_cpu(1, number_bytes);
                end
            end
            else if (cache_line_valid[addr_set * 2] == 0 &&
cache_line_valid[addr_set * 2 + 1] == 1)
            begin
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2] == 1)
                begin
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                byte_to_cpu(0, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 1 &&
cache_line_valid[addr_set * 2 + 1] == 0)
            begin
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    write_line_mem(1);
                end
                read_line_mem(1);
                cache_line_valid[addr_set * 2 + 1] = 1;
                cache_line_old[addr_set * 2 + 1] = 0;
                cache_line_old[addr_set * 2] = 1;
                byte_to_cpu(1, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 0 &&
```

```
cache_line_valid[addr_set * 2 + 1] == 0)
            begin
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                byte_to_cpu(0, number_bytes);
            end
        end
    end
    endtask

    task write_bytes(input [1 : 0] number_bytes);
    begin
        if (cache_line_tag[addr_set * 2] == addr_tag && cache_line_valid[addr_set
* 2] == 1)
        begin
            // кэш-попадание первая кэш-линия
            #8;
            byte_from_cpu(0, number_bytes);
        end
        else if (cache_line_tag[addr_set * 2 + 1] == addr_tag &&
cache_line_valid[addr_set * 2 + 1] == 1)
        begin
            // кэш-попадание вторая кэш-линия
            #8;
            byte_from_cpu(1, number_bytes);
        end
        else
        begin
            // кэш-промах
            #4;
            if (cache_line_valid[addr_set * 2] == 1 && cache_line_valid[addr_set
* 2 + 1] == 1)
            begin
                // линия1 и линия2 заняты
                if (cache_line_old[addr_set * 2] == 1)
                begin
                    // линия1 old
                    l_C2 = 1;
```

```
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2] == 1)
                    begin
                        C2_reg = 3;
                        write_line_mem(0);
                    end
                    C2_reg = 2;
                    read_line_mem(0);
                    cache_line_old[addr_set * 2] = 0;
                    cache_line_old[addr_set * 2 + 1] = 1;
                    l_C2 = 0;
                    byte_from_cpu(0, number_bytes);
                end
                else
                begin
                    // линия2 old
                    l_C2 = 1;
                    A2_reg = {addr_tag, addr_set};
                    if (cache_line_dirty[addr_set * 2 + 1] == 1)
                    begin
                        C2_reg = 3;
                        write_line_mem(1);
                    end
                    C2_reg = 2;
                    read_line_mem(1);
                    cache_line_old[addr_set * 2 + 1] = 0;
                    cache_line_old[addr_set * 2] = 1;
                    l_C2 = 0;
                    byte_from_cpu(1, number_bytes);
                end
            end
            else if (cache_line_valid[addr_set * 2] == 0 &&
cache_line_valid[addr_set * 2 + 1] == 1)
            begin
                l_C2 = 1;
                A2_reg = {addr_tag, addr_set};
                if (cache_line_dirty[addr_set * 2] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(0);
                end
                C2_reg = 2;
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
```

```
                cache_line_old[addr_set * 2 + 1] = 1;
                l_C2 = 0;
                byte_from_cpu(0, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 1 &&
cache_line_valid[addr_set * 2 + 1] == 0)
            begin
                l_C2 = 1;
                A2_reg = {addr_tag, addr_set};
                C2_reg = 2;
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(1);
                end
                read_line_mem(1);
                cache_line_valid[addr_set * 2 + 1] = 1;
                cache_line_old[addr_set * 2 + 1] = 0;
                cache_line_old[addr_set * 2] = 1;
                l_C2 = 0;
                byte_from_cpu(1, number_bytes);
            end
            else if (cache_line_valid[addr_set * 2] == 0 &&
cache_line_valid[addr_set * 2 + 1] == 0)
            begin
                l_C2 = 1;
                A2_reg = {addr_tag, addr_set};
                C2_reg = 2;
                if (cache_line_dirty[addr_set * 2 + 1] == 1)
                begin
                    C2_reg = 3;
                    write_line_mem(0);
                end
                read_line_mem(0);
                cache_line_valid[addr_set * 2] = 1;
                cache_line_old[addr_set * 2] = 0;
                cache_line_old[addr_set * 2 + 1] = 1;
                l_C2 = 0;
                byte_from_cpu(0, number_bytes);
            end
        end
    end
    endtask

    always @(negedge clk)
```

```
    begin
        #1;
    end

    always @(posedge clk)
    begin
        #1;
        case (C1)
        3'b000 :  // NOP
        begin
            $display("nothing");
        end
        3'b001 : // READ8
        begin
            read_bytes(1);
        end
        3'b010 : // READ16
        begin
            read_bytes(2);
        end
        3'b011 : //READ32
        begin
            read_bytes(4); //READ32
        end
        3'b100 : // INVALIDATE_LINE
        begin
            addr_set = A1[5 : 0];
            write_line_mem(A1[6]);
        end
        3'b101 : // WRITE8
        begin
            write_bytes(1);
        end
        3'b110 : // WRITE16
        begin
            write_bytes(2);
        end
        3'b111 : // WRITE32
        begin
            write_bytes(4);
        end
        endcase
    end
endmodule
```

```
module memory #(
    parameter MEM_SIZE = 2**18,
    parameter ADDR2_BUS_SIZE = 14,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR2_BUS_SIZE = 2
)
(
    input wire clk,
    input wire [ADDR2_BUS_SIZE - 1 : 0] A2,
    inout wire [DATA2_BUS_SIZE - 1 : 0] D2,
    inout wire [CTR2_BUS_SIZE - 1 : 0] C2
);
reg [DATA2_BUS_SIZE - 1 : 0] D2_reg;
reg [CTR2_BUS_SIZE - 1 : 0] C2_reg;
reg write;
reg l_C2;
reg [7 : 0] memory [MEM_SIZE - 1 : 0];
integer SEED = 225526;
integer i;

assign D2 = write ? D2_reg : 'hz;
assign C2 = l_C2 ? C2 : 'hz;

initial begin
    $dumpfile("dump3.vcd");
    //$dumpvars(0, clk, l_C2, wire);
    l_C2 = 0;
    write = 0;
    for (i = 0; i < MEM_SIZE; i++)
    begin
        memory[i] = $random(SEED)>>16;
    end
    for (i = 0; i < 8; i += 1)
    begin
        //$display("[%h] %h", i, memory[i]);
    end
end

always @(posedge clk)
begin
    #2
    case(C2)
        2'b00 : // NOP
        begin
            $display("nop");
```

```
        end
        2'b01 : // RESPONSE
        begin
            $display("response");
        end
        2'b10 : // READ_LINE
        begin
            //reading c.line
            #198;
            // wire = 1;
            l_C2 = 1;
            C2_reg = 1;
            for (i = A2 * 8; i < A2 * 8 + 16; i += 2)
            begin
                D2_reg = {memory[i + 1], memory[i]};
                #2;
            end
            C2_reg = 0;
            l_C2 = 0;
            write = 0;
        end
        2'b11 : // WRITE_LINE
        begin
            //writing c.line
            #198 write = 1;
            l_C2 = 1;
            C2_reg = 1;
            for (i = A2 * 8; i < A2 * 8 + 16; i += 2)
            begin
                memory[i] = D2[7 : 0];
                memory[i + 1] = D2[15 : 8];
                #2;
            end
            l_C2 = 0;
            C2_reg = 0;
        end
    endcase
end
endmodule
```

```
`include "cache.sv"
module testbench #(
    parameter MEM_SIZE = 2**18,
```

```
    parameter CACHE_SIZE = 2**11,
    parameter CACHE_LINE_SIZE = 16,
    parameter CACHE_LINE_COUNT = 2**7,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 64,
    parameter CACHE_TAG_SIZE = 8,
    parameter CACHE_SET_SIZE = 6,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_ADDR_SIZE = 18,
    parameter ADDR1_BUS_SIZE = 14,
    parameter ADDR2_BUS_SIZE = 14,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
);
int hits;
int requests;
reg [ADDR1_BUS_SIZE - 1 : 0] A1_reg;
reg [DATA1_BUS_SIZE - 1 : 0] D1_reg;
reg [CTR1_BUS_SIZE - 1 : 0] C1_reg;
reg clk;
inout wire [CTR1_BUS_SIZE - 1 : 0] C1;
inout wire [DATA1_BUS_SIZE - 1 : 0] D1;
integer M;
integer N;
integer K;
integer x;
integer y;
integer k;
integer s;
integer pa;
integer pb;
integer pc;
integer requests_count;
reg [17 : 0] addr;
reg [7 : 0] a;
reg [15 : 0] b;
reg [31 : 0] c;
reg l_C1;
reg writeD1;
cache cache(clk, A1_reg, D1, C1, hits, requests);
assign C1 = l_C1 ? C1_reg : 'hz;
assign D1 = writeD1 ? D1_reg : 'hz;
```

```
always #1 clk = ~clk;

initial
begin
    clk = 1;
    M = 64;
    #2;
    N = 60;
    #2;
    K = 32;
    #2;
    pa = 0;
    #2;
    pc = 0;
    #2;

    requests_count = 0;
    for (y = 0; y < M; y += 1)
    begin
        for (x = 0; x < N; x += 1)
        begin
            pb = 0;
            #2;
            s = 0;
            #2;
            for (k = 0; k < K; k += 1)
            begin
                requests_count += 2;
                C1_reg = 1;
                l_C1 = 1;
                A1_reg = (pa * K + k) >> CACHE_OFFSET_SIZE;
                #2;
                A1_reg = (pa * K + k) % CACHE_LINE_SIZE;
                #2;
                l_C1 = 0;
                #1
                //wait(C1 == 7);
                #7;
                a = D1[7 : 0];
                C1_reg = 2;
                l_C1 = 1;
                A1_reg = (M * K + (pb * N + x) * 2) >> CACHE_OFFSET_SIZE;
                #2;
                A1_reg = (M * K + (pb * N + x) * 2) % CACHE_LINE_SIZE;
                #2;
```

```
                l_C1 = 0;
                #1;
                //wait(C1 == 7);
                #50;
                b = D1[7 : 0];
                s += a * b;
                #10;
                pb += 1;
                #2;
                #2;
            end
            requests_count += 1;
            C1_reg = 7;
            l_C1 = 1;
            A1_reg = (M * K + K * N * 2 + (pc * N + x) * 4) >> CACHE_OFFSET_SIZE;
            #2;
            A1_reg = (M * K + K * N * 2 + (pc * N + x) * 4) % CACHE_LINE_SIZE;
            #2;
            l_C1 = 0;
            #2;
            //wait(C1 == 7);
            #3;
            #2;
        end
        pa += 1;
        #2;
        pc += 1;
        #2;
        #2;
    end

    #2;
    $display("Cache hits: %0d", hits);
    $display("Cache requests: %0d", requests_count);
    $display("Tacts: ", $time);
    #1;
    $finish;

end
endmodule
```