Лабораторная работа №2	**M3136**	*2022*
Моделирование схем в Verilog	**Артемьев Иван Вадимович**	

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

Инструментарий: весь код пишется на языке *Verilog*, компиляция и симуляция – **Icarus Verilog 12**. Далее в этом документе *Verilog+SystemVerilog* обозначается как *Verilog*.

Описание: Дана функция “*mmul*”, необходимо сперва аналитически рассчитать процент кэш попаданий при выполнении этой функции, и время(в тактах процессора) затраченное на ее выполнение. После этого необходимо просимулировать работу этой функции в система процессор-кэш-память на языке *Verilog*.

Вариант 3: Мне досталось конфигурация кэша в которой заданы следующие параметры кэша *CACHE_SIZE* = 2Кб, *CACHE_LINE_SIZE* = 16 байт, *CACHE_TAG_SIZE* = 8 бит. Также размер памяти: MEM_SIZE = 256Кб.

Вычисление недостающих параметров системы:

- *MEM_SIZE = 256 Кб* = 2^{18} *байт*
- *CACHE_SIZE = 2 Кб* = 2^{11} *байт*
- *CACHE_LINE_SIZE = 16 байт* = 2^{4} *байт*
- *CACHE_TAG_SIZE = 8 бит*
- *CACHE_LINE_COUNT* = $\frac{CACHE\_SIZE}{CACHE\_LINE\_SIZE} = 2^{7}$
- *CACHE_ADDR_SIZE* = $log_2 MEM\_SIZE$ = *18 бит*

- *CACHE_OFFSET_SIZE =* $log_2 CACHE\_LINE\_SIZE$ *= 4 бита*
- *CACHE_SET_SIZE =*

= CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - CACHE_TAG_SIZE =
= 18 - 4 - 8 = 6 бит

- $CACHE\_SETS\_COUNT = 2^{CACHE\_SET\_SIZE} = 2^6 = 64$
- $CACHE\_WAY = \frac{CACHE\_LINE\_COUNT}{CACHE\_SETS\_COUNT} = 2$

Также посчитаем размерности шин:

- *ADDR1_BUS_SIZE =*

= max{CACHE_SET_SIZE + CACHE_TAG_SIZE; CACHE_OFFSET_SIZE} =
= max{8 + 6, 4} = 14 бит

- *ADDR2_BUS_SIZE =*

= CACHE_SET_SIZE + CACHE_TAG_SIZE = 14 бит

- *DATA1_BUS_SIZE = 16 бит*
- *DATA2_BUS_SIZE = 16 бит*
- *CTR1_BUS_SIZE =* log_2 *(кол-во комманд) =* $log_2 8$ *= 3 бита*
- *CTR2_BUS_SIZE =* log_2 *(кол-во комманд) =* $log_2 4$ *= 2 бита*

Аналитическое решение задачи:

Для проведения необходимых расчетов, я решил написать код на языке *Java*. Проект содержит четыре класса: **CacheLine.java**, **Address.java**, **Cache.java**, **Main.java**.

Для подсчета количества кэш попаданий и затраченного времени на выполнение нам не нужно хранить память и поэтому будем хранить в кэш линии только бит *valid*, *dirty*, также флаг для реализации *LRU*, если он установлен в *true*, значит эта кэш линия была использована позже чем вторая хранящаяся в этом же сете. Я мог быть хранить номера по которым определять очередность использования кэш линий в сете, но так как у меня ассоциативность-2, мне достаточно флага типа *boolean*.

Теперь попробуем подсчитать время затрачиваемое в тех или иных случаях. Чтобы далее при реализации не задумываться сколько тактов необходимо.

1. *C1_READ8* или *C1_READ16*:
    a. Кэш - попадание. Тогда 2 такта на передачу адреса по шине *A1*, далее через 6 тактов кэш начинает отвечать, и ещё один такт необходим на передачу данных по шине *D1*, так как у нас 1 или 2 байта, что помещается в один такт. Суммарно 9 тактов.
    b. Кэш - промах.
        i. *dirty* = 0. Тогда нам надо: 2 такта на передачу адреса по шине *A1*, через 4 такта кэш начинает отвечать, далее за 1 такт передаем адрес(тег и сет) по шине *A2* для памяти, через 100 тактов она начинает отвечать. Далее за *CACHE_LINE_SIZE* / 2 = 8 тактов по шине *D2* мы передаем новую кэш линию в кэш. И ещё один такт необходим, чтобы вернуть процессору по шине *D1* ответ. Суммарно 116 тактов.
        ii. *dirty* = 1. Тогда нам надо: 2 такта на передачу адреса по шине *A1*, через 4 такта кэш начинает отвечать, далее за 8 тактов передаем по *D2* данные кэш линии из кэша и параллельно с этим передаем по *A2* адрес этой кэш линии. Через 100 тактов память начнет отвечать и за 1 такт отправит *C2_RESPONSE*. После чего за 1 такт передаем адрес(тег и сет) по шине *A2* для памяти через 100 тактов она начинает отвечать. Далее за *CACHE_LINE_SIZE* / 2 = 8 тактов по шине *D2* мы передаем новую кэш линию в кэш. И ещё один такт необходим, чтобы вернуть процессору по шине *D1* ответ.

При этом кэш линия становится *dirty* = 0. Суммарно 225 тактов.

2. *C1_WRITE32*
    a. Кэш попадание. В данном случае нам не важно значение *dirty*. За 2 такта из процессора передается адрес по шине *A1* и данные по шине *D1*. Через 6 тактов кэш начинает отвечать, и за 1 такт отправляем *C1_RESPONSE*. Кэш линия становится *dirty*. Суммарно 9 тактов.
    b. Кэш промах
        i. *dirty* = 0. В этом случае за 2 такта получаем данные и адрес по шинам *D1* и *A1* соответственно. Через 4 такта кэш начинает отвечать, далее за 1 такт передаем адрес(тег и сет) по шине *A2* для памяти, через 100 тактов она начинает отвечать. Далее за *CACHE_LINE_SIZE* / 2 = 8 тактов по шине *D2* мы передаем новую кэш линию в кэш. Далее уже в кэше изменяем кэш линию, как нам надо. И за 1 такт отправляем *C1_RESPONSE*. Суммарно 116 тактов.
        ii. *dirty* = 1. Помимо того, что происходит при *dirty* = 0. Необходимо еще 109 тактов на запись грязной кэш линии в память. (100 на работу памяти, 8 на передачу данных кэш линии, 1 на отправку ответа *C2_RESPONSE*). Суммарно 225 тактов.

Для удобства использования кэш-линии создадим класс **CacheLine.java**. С помощью которого просто будем создавать кэш линии. Имеется конструктор, создающий пустую кэш линию, то есть *valid* = 0.

```
public class CacheLine {
   public int tag;
   public boolean valid;
   public boolean dirty;
```

```
    public boolean lru; //true => later used

    public CacheLine() {
        this.tag = 0;
        valid = false;
        dirty = false;
        lru = false;
    }
}
```

Для удобства использования *CacheLine* я сделал её поля *public*, иначе пришлось бы писать *геттеры* и *сеттеры* и код был бы не таким красивым.

Также для удобства использования я создал класс **Address.java**, с помощью которого можно удобно пользоваться адресом, состоящим из тега и сета. Офсет в данном случае мне не нужен, так как с помощью офсета мы ищем определенный байт в памяти кэш линии.

```
public class Address {
    private final int tag;
    private final int set;

    public Address(int set, int tag) {
        this.tag = tag;
        this.set = set;
    }

    public int getTag() {
        return tag;
    }

    public int getSet() {
        return set;
    }
}
```

Теперь перейдем к классу **Cache.java**. Этот класс хранит в своих полях все кэш линии, также в нём содержатся константы, которые используются для подсчетов. Также имеется конструктор, заполняющий кэш пустыми кэш линиями.

```
public class Cache {
    private static final int CACHE_LINE_SIZE = 16;
```

```
    private static final int CACHE_SETS_COUNT = 64;
    private static final int CACHE_WAY = 2;
    CacheLine[][] cacheLines;

    public Cache() {
        cacheLines = new CacheLine[CACHE_SETS_COUNT][CACHE_WAY];
        for (int i = 0; i < CACHE_SETS_COUNT; i++) {
            for (int j = 0; j < CACHE_WAY; j++) {
                cacheLines[i][j] = new CacheLine();
            }
        }
    }
}
```

Также в кэше есть следующие методы:

“*cacheRequest*” вызывается каждый раз когда используется кэш. Ему передается адрес “addr” в памяти и тип операции *type*(‘r’ - *read* или ‘w’ - *write*). Метод возвращает время которое кэш затратил на выполнение операции. В методе по адресу *addr* находим *set* и *tag*, если кэш в данном сете содержит наш тэг, то это кэш попадание и на него затрачивается 9 тактов. Если же нет, то это это кэш промах и мы возвращаем значение из метода “*add*”.

```
public int cacheRequest(int addr, char type) {
    int set = (addr / CACHE_LINE_SIZE) % CACHE_SETS_COUNT;
    int tag = (addr / CACHE_LINE_SIZE) / CACHE_SETS_COUNT;
    Address address = new Address(set, tag);
    if(this.containsAddress(address, type)) {
        return 9;
    } else {
        return this.add(address, type);
    }
}
```

“*containsAddress*” вызывается для проверки содержится ли данный тэг в сете. Ему передается адрес *address* и тип операции *type*. Кэш линия содержит данный тэг, если она *valid* и ее тэг равен тэгу адреса. При этом

при нахождении нужного тэга, меняем флаг *lru* и если *type* = ‘w’, то кэш линия становится *dirty*, так как мы ее изменили.

```
public boolean containsAddress(Address address, char type) {
    for(int i = 0; i < 2; i++) {
        CacheLine cacheLine = cacheLines[address.getSet()][i];
        if (cacheLine.valid && cacheLine.tag == address.getTag()) {
            // cache hit
            if (type == 'w') {
                cacheLine.dirty = true;
            }
            cacheLines[address.getSet()][1-i].lru = false;
            cacheLines[address.getSet()][i].lru = true;
            return true;
        }
    }
    return false;
}
```

“*add*” вызывается только в случае кэш промаха, этот метод добавляет кэш линию в кэш из памяти. Для этого передаем ей адрес *address* и тип операции *type*. Сперва если линия не *valid*, то мы её заполним из памяти за 116 тактов, также выставим *lru* и тэг, также по типу операции можно однозначно понять будет она *dirty* или нет. Так как при read мы вытащили её из памяти и ничего не поменяли => *dirty* = 0, а если мы выполняем *write*, то вытащим из памяти мы ее поменяем и dirty станет равным 1. Если же обе линии *valid*, то мы вытесняем ту, у которой *lru* = 0. После чего заменяем *lru*, тэг. Если она была dirty то будет затрачено 225 тактов. И так как мы только что вытащили кэш линию из памяти мы однозначно по типу операции определяем её *dirty*. Если линия была не *dirty* и операция ‘w’, то она станет *dirty*.

```
public int add(Address address, char type) {
    for(int i = 0; i < 2; i++) {
        CacheLine cacheLine = cacheLines[address.getSet()][i];
        if (!cacheLine.valid) {
            cacheLines[address.getSet()][i].lru = true;
```

```
                cacheLines[address.getSet()][1-i].lru = false;
                cacheLines[address.getSet()][i].valid = true;
                cacheLines[address.getSet()][i].tag = address.getTag();
                if (type == 'r')
                    cacheLines[address.getSet()][i].dirty = false;
                else
                    cacheLines[address.getSet()][i].dirty = true;
                //data from memory
                return 116;
            }
        }

        if (!cacheLines[address.getSet()][1].lru) {
            cacheLines[address.getSet()][1].lru = true;
            cacheLines[address.getSet()][0].lru = false;
            cacheLines[address.getSet()][1].tag = address.getTag();
            if (cacheLines[address.getSet()][1].dirty) {
                if (type == 'r') {
                    cacheLines[address.getSet()][1].dirty = false;
                } //else if (type == 'w')
                  //  cacheLines[address.getSet()][1].dirty = true;
                  // dirty = true в этом случае
                return 225;
            }
            if (type == 'w')
                cacheLines[address.getSet()][1].dirty = true;
            return 116;
        } else {
            cacheLines[address.getSet()][0].lru = true;
            cacheLines[address.getSet()][1].lru = false;
            cacheLines[address.getSet()][0].tag = address.getTag();
            if (cacheLines[address.getSet()][0].dirty) {
                if (type == 'r') {
                    cacheLines[address.getSet()][0].dirty = false;
                }
                return 225;
            }

            if (type == 'w') {
                cacheLines[address.getSet()][0].dirty = true;
            }
            return 116;
        }
    }
```

Наконец-таки перейдем к классу Main, в котором реализован метод main, в котором создаем *M*, *N*, *K*, пустой Cache, создаем счетчики *time* - считает суммарное время работы, *cacheUsages*, *cacheHits*. При создании pa, pb и pc, затрачивается по одному такту. Чтобы выполнить `s += pa[k] * pb[x];` необходимо достать с помощью кэша *pa[k]* и *pb[x]*, зная то, что массивы последовательно хранятся в памяти мы можем с легкостью находить адрес.

При этом когда мы обращаемся за 2-х или 4-х байтовым числом мы не можем попасть в ситуацию, когда начало числа в одной кэш линии, а конец а другой. Потому что первый массив заканчивается на ячейке с номером *M*K - 1 = 3840 - 1,* поэтому второй начинается с ячейки *M*K = 3840* при этом *3840 % CACHE_LINE_SIZE = 0*, то есть мы попали в начало кэш линии, каждая кэш линия заканчивается на ячейке с нечетным номером, а 2-х байтовое число имеет вид в памяти: *чет, нечет*. Массив *c* начинается с ячейки *M*N + K*N = 5760, 5760 % CACHE_LINE_SIZE = 0* проведя аналогичные рассуждения только не с четностью а с остатком по модулю 4 понимаем что я прав.

Функция *cacheRequest* возвращает 9 только при кэш попадании. Поэтому можем с легкостью считать количество кэш попаданий. Также добавляем увеличение *time* в конце каждой итерации циклов, при прибавлении к переменным значений и при выходе из функции “*mmul*”. При умножении прибавляем 5 к *time*.

```
public class Main {
   public static void main(String[] args) {
       int M = 64;
       int N = 60;
       int K = 32;
       Cache cache = new Cache();

       int time = 0;
```

```
int cacheUsages = 0;
int cacheHits = 0;
int pa = 0;
int pc = 0;
time += 2;
for (int y = 0; y < M; y++)
{
    for (int x = 0; x < N; x++)
    {
        int pb = 0;
        time += 2;
        for (int k = 0; k < K; k++)
        {
            //обращение к pa[k]
            //a - 8bit => addr = pa + k
            int addr = pa + k;
            int addTime = cache.cacheRequest(addr, 'r');
            if (addTime == 9){
                cacheHits++;
            }
            time += addTime;
            cacheUsages++;

            //обращение к pb[x]: b[pb][x]
            //a - 16bit => addr = (pb + x)*2
            addr = M*K + (pb + x)*2;
            addTime = cache.cacheRequest(addr, 'r');
            if (addTime == 9){
                cacheHits++;
            }
            time += addTime;
            cacheUsages++;

            //s += pa[k] * pb[x];
            time += 8; //add, mult, mem x2

            pb+=N;
            time += 2; //add, for
        }
        //pc[x] = s;
        time += 1;
        //обращение к c[pc][x] для записи
        int addr = M*K + 2*K*N + (pc + x)*4;
        int addTime = cache.cacheRequest(addr, 'w');
        if (addTime == 9){
```

```
                    cacheHits++;
                }
                time += addTime;
                cacheUsages++;

                time++;
            }
            pa+=K;
            pc+=N;
            time += 3;
        }

        time++; //exit mmul
        double percentOfCacheHits = cacheHits * 100.0 / cacheUsages;
       System.out.println("Percent of cache hits: "
                           + percentOfCacheHits + "%");
        System.out.println("Percent of cache misses: "
                           + (100 - percentOfCacheHits) + "%");
        System.out.println("Estimated time "
                           + time + " (in processor tact's)");
    }
}
```

В итоге получаем следующий ответ:

Процент *кэш попаданий* = ***92.427083333333333%***,

Процент *кэш промахов* = ***7.572916666666671%***,

Затраченное *время* = ***5 656 713*** тактов.

Моделирование системы Процессор-Кэш-Память на языке Verilog

Кэш хранит в себе некоторые данные из памяти, чтобы ускорить работу компьютера, затрачивая меньше времени на перенос данных по

проводам. Кэш состоит из кэш линий, которые содержат в себе: данные из памяти, тэг, три флага(*valid, dirty, lru*).

Кэш линии разбиты на сеты, каждый из которых содержит по 2 кэш линии, потому что по условию нам задана ассоциативность 2. Таким образом, зная в каком мы сете находимся и зная тег, мы можем понять каким данным из памяти соответствует данная кэш линия. С помощью офсета, можно находить конкретный байт данных в кэше.

Так как размер кэша меньше размера памяти, одному сету из кэша, соответствует несколько линий данных из памяти, поэтому когда от процессора приходит запрос на получение данных их может не оказаться в кэше, данная ситуация называется кэш промахом, а если тэг переданный кэшу для поиска данных содержался в данном сете, то такая ситуация будет называться кэш попаданием.

От общих слов о кэше, перейдем к моей реализации данной системы на языке *SystemVerilog*. Сперва я создал 4 файла расширения *.sv*, и назвал их *testbench.sv, CPU.sv, CACHE.sv, MEM.sv* в которых находятся соответствующие модули. После создания модулей и подключения к ним входов и выходов.

testbench.sv здесь добавил инклюды с файлами реализующими части нашей системы. Также я добавил модуль main, в котором создал шины соединяющие входы и выходы модулей процессора, кэша и памяти. В блоке *always* каждый такт верилога я инвертирую значение регистра *CLK*. Тогда получается, что один такт процессора - это две задержки в верилоге.

CPU.sv - модуль описывающий работу процессора. С помощью *define* я добавил константы, чтобы ими пользоваться. При этом у всех констант я вычел единицу, потому что при создании регистров необходимо

указывать не размер, а номер максимального и минимального битов(начиная с нуля). В модуле *CPU* я создаю регистры *data1, addr1, com1* и с помощью *assign* соединяю их с соответствующими выходами модуля. Также я создаю регистры *set, tag u offset*. В initial я написал основной код, который выполняет задачу данную мне в тз. Но *pa, pb, pc* у меня являются не указателями а просто числами, с помощью которых находится массив в соответствующем списке массивов. Также у меня имеется глобальная переменная *cache_hit_count*, которая считает количество кэш попадании в модуле *CACHE*.

Всего идет три обращения к кэшу: получить значение *pa[k], pb[x]* и записать значение в *pc[x]*. Все обращения примерно одинаковые, рассмотрим одно из них - например запись в *pc[x]*.

Адрес *addr*, я нахожу также как и в аналитическом решении, а далее с помощью него тэг, сет и оффсет, используя двоичные сдвиги. Сперва идет задержка 2 - один такт, чтобы подождать, пока после предыдущего запроса кэш освободит шину. После этого идет задержка 1 - пол такта, когда *CLK* переходит из состояния 0 в состояние 1. Это нужно, чтобы posedge кэша успел уловить изменение клока и начал выполнение соответствующей функции. В этот момент на шину подается номер команды - 7 (*WRITE32*), также подается первая часть адреса: тег и сет, и подается первая половина данных(16 байт), которые необходимо сохранить в кэш. На следующем такте подается оффсет и вторая половина данных. На следующем такте освобождается шина, с помощью отдельного *task*, который устанавливает значения регистров, подключенных к выходам на значение высокоимпедансного состояния *Z*. И перед тем как продолжение выполнение программы, необходимо дождаться ответа от кэша с помощью команды *wait*.

MEM.sv файл в котором содержится модуль *MEM*, описывающий работу памяти в рамках нашей системы. Все данные хранятся в одном больше массиве регистров *data*. Не знаю зачем, но почему-то я сделал его разделенным по кэш линиям(мне показалось это удобным). В первом *initial*, я заполняю данные псевдо рандомными числами, с помощью *SEED*а, который мне был дан в тз.

В блоке *always* с помощью *posedge*, мы обрабатываем каждое изменение клока. Проверяем если нам подали 1 на *M_DUMP*, то мы делаем *dumpfile*. А если 1 на *RESET*, то просто заново заполняем data нашими псевдо рандомными числами. Также с помощью *case* и проверяем:

Если подали команду с номером 0 - *C2_NOP*, то мы ничего не делаем. Также у меня есть глобальная переменная *log*, объявленная в кэше, и когда её значение установлено в 1, то программа выводит что-либо, а когда в 0, то ничего, её удобно использовать для дебага программы. Так вот при значении 1 я вывожу *"Memory: nothing is done"* и больше ничего не делаю.

Если подали команду с номером 2 - *C2_READ_LINE*, то значит на шине адреса уже находится значение тэга и сета, которые мы сохраняем. Потому что на следующем такте, кэш освободит шину и данные исчезнут. После задержки 100, обозначающей время работы кэша, мы подаем на выход команду 1 - *C1_RESPONSE* и первую часть данных, и потом в цикле каждый такт передаем по 16 бит данных. После этого освобождаем шину.

Если подали команду с номером 3 - *C2_WRITE_LINE*, то значит то значит на шине адреса уже находится значение тэга и сета, которые мы сохраняем. А также сохраняем первую переданную часть данных и каждый такт сохраняем по два байта данных. После задержки в 101 такт,

отправляем на *C2* значение *2'b01*, обозначающий *C2_RESPONSE*. На этом функционал память окончен.

CACHE.sv содержит в себе модуль *CACHE*, описывающий работу кэша. Данные кэша мы будем хранить в трехмерном массиве байтовых регистров. Тэги буду хранить в двумерном массиве регистров размера тэга. Ну и три двумерных массива битовых регисторов *valid, dirty, lru*. В первом *initial* установим значения *valid* - 0 на всех кэш линиях(изначально кэш пустой). Аналогично с процессором и памятью создаем регистры, которые соединяем со входами и выходами. Помимо этого есть регистры обозначающие поданные на данную команду тэг, сет, офсет, данные и номер команды. *cur_data1* размера в два раза больше, потому что при команде *write32* нам надо все 4 байта данных сохранять.

В блоке *always*, которые работает также как и в память (по каждому такту процессора) идет проверка какая команда нам сейчас подана. Если дана *C_DUMP*, то также как и в память делаем *dumpfile*. При подаче нам *RESET*, все биты *valid* устанавливаем в 0.

В *case(C1)* если подан 0 - *C1_NOP*, то ничего не делаем. Если подана 4, то тогда сначала считываем данные с помощью таска *get_input*, далее смотрим есть ли в сете кэш линия с заданным тэгом, если есть то ставим у неё значит *valid* - 0, если не то ничего на этом шаге не делаем а после этого отправляем ответ с помощью таска *return_response*.

Если же в на *C1* подана команда *READ* или *WRITE*, то попадаем в третий блок в *case*. С помощью таска *get_input* считываем данные. Потом выжидаем 4 такта - время работы кэша. Потом таском *check_cache_hit*, проверяем есть ли данный тэг. Потом делаем *do_if_cache_hit*, который в случае когда у нас имеется тэг и при этом значение *valid* = 1, выполняет нужную функцию. Сперва выставляет значение *lru*. То есть в той кэш

линии, в которой нашелся тег, мы устанавливаем значение *lru* = 1, а во второй 0. Таким образом всегда чем больше значение *lru* тем более поздно использовалась кэш линия.

Если команда <= 3, то есть *READx*, то мы выполняем таск *return_read*, который возвращает команду - 7 на шину *C1*, то есть *C1_RESPONSE* и возвращает необходимые данные из кэш линии, которые легко находятся по офсету. При этом если команда, когда надо вернуться 4 байта данных, то на втором такте возвращаем ещё 2 байта данных. В конце таск освобождает шину.

Иначе (команда >3) мы выполняем таск *write_procedure*, в котором сначала просто записываем данные в кэш линию и не забываем установить значение *dirty* = 1 и возвращаем по шине *C1 response*. Это делается с помощью тасков *write* и *return_response*.

Далее если не было кэш попадания, то выполняем таск *do_if_cache_miss*.

В нём сперва в таске *check_valid*, проверяем биты *valid*, если находим 0, то начинаем взаимодействовать с памятью с помощью таска *cache_miss_tail_tasks*, о котором расскажу чуть позже.

Далее если пустых кэш линий не оказалось, то выполняем *cache_miss_lru*. В нём мы в сете выбираем ту кэш линию у которой значение *lru* установлено в 0, потому что она дольше не использовалась и её мы будем заменять. При этом если у неё значение *dirty* = 1, то перед тем, как считать новую кэш линию из памяти надо сохранить эту с помощью таска *make_memory_write_request*. После этого выполняем таск *cache_miss_tail_tasks*.

cache_miss_tail_tasks в зависимости от команды выполняет разные вещи. Если команда типа *READ*, то сперва мы посылаем памяти

информацию о том, что мы хотим считать у нее кэш линию по заданному адресу (тег и сет), то есть команда *C2_READ_LINE*. Это выполняется с помощью таска *make_memory_read_request*. После этого с помощью таска *get_data_from_memory*, сохраняем данные которые нам вернула память. В таске *set_info* выставляем выставляем *valid*, *dirty*, *lru* и записываем тэг в *cache_line_tag*. И после этого мы можем вернуть данные процессору *return_read*. Если команда типа *WRITE*, то выполняется таск *write_procedure*.

В итоге получаем следующий ответ:

Процент *кэш попаданий* = ***91.701923 %***,

Процент *кэш промахов* = ***8.298077 %***,

Затраченное *время* = ***4 469 378*** тактов.

Сравнение полученных результатов

92.42708333333333 / 91.701923 = 1.0079078

5656713 / 4469378 = 1.26566

Получается что расхождение в числе попаданий меньше процента, а вот затраченное время отличается на 25%.

Листинг кода

testbench.sv

```
`include "CPU.sv"
`include "MEM.sv"
`include "CACHE.sv"
`define ADDR1_BUS_SIZE 13
`define ADDR2_BUS_SIZE 14 - 1
`define DATA1_BUS_SIZE 16 - 1
`define DATA2_BUS_SIZE 16 - 1
`define CTR1_BUS_SIZE 3 - 1
`define CTR2_BUS_SIZE 2 - 1

module main;
    wire[`ADDR1_BUS_SIZE : 0] A1 = 14'bzzzzzzzzzzzzzz;
    wire[`ADDR2_BUS_SIZE : 0] A2 = 14'bzzzzzzzzzzzzzz;
    wire[`DATA1_BUS_SIZE : 0] D1 = 16'bzzzzzzzzzzzzzzzz;
    wire[`DATA2_BUS_SIZE : 0] D2 = 16'bzzzzzzzzzzzzzzzz;
    wire[`CTR1_BUS_SIZE : 0] C1 = 3'bzzz;
    wire[`CTR2_BUS_SIZE : 0] C2 = 2'bzz;

    reg CLK = 1'b0;
    reg C_DUMP = 1'b0;
    reg M_DUMP = 1'b0;
    reg RESET = 1'b0;

    CPU cpu(.CLK(CLK), .A1(A1), .D1(D1), .C1(C1));

    CACHE cache(.CLK(CLK), .A1(A1), .A2(A2), .D1(D1),
                .D2(D2), .C1(C1), .C2(C2),
                .C_DUMP(C_DUMP), .RESET(RESET));

    MEM mem(.CLK(CLK), .A2(A2), .D2(D2), .C2(C2),
                .M_DUMP(M_DUMP), .RESET(RESET));

    always begin
        #1 CLK = ~CLK;
    end
endmodule
```

CPU.sv

```
`define CACHE_SIZE 2048 - 1
`define CACHE_LINE_SIZE 16 - 1
`define CACHE_LINE_COUNT 128 - 1
`define CACHE_SETS_COUNT 64 - 1
`define CACHE_TAG_SIZE 8 - 1
`define CACHE_SET_SIZE 6 - 1
`define CACHE_OFFSET_SIZE 4 - 1
`define CACHE_ADDR_SIZE 18 - 1

`define ADDR1_BUS_SIZE 14 - 1
`define ADDR2_BUS_SIZE 14 - 1
`define DATA1_BUS_SIZE 16 - 1
`define DATA2_BUS_SIZE 16 - 1
`define CTR1_BUS_SIZE 3 - 1
`define CTR2_BUS_SIZE 2 - 1

int cache_hit_count = 0;
int cache_miss_count = 0;
int cache_usage_count = 0;


module CPU(input CLK, output [`ADDR1_BUS_SIZE : 0] A1, inout [`DATA1_BUS_SIZE
: 0] D1, inout [`CTR1_BUS_SIZE : 0] C1);

    reg[`DATA1_BUS_SIZE : 0] data1 = 16'bzzzzzzzzzzzzzzzz;
    assign D1 = data1;

    reg[`ADDR1_BUS_SIZE : 0] addr1 = 14'bzzzzzzzzzzzzzz;
    assign A1 = addr1;

    reg[`CTR1_BUS_SIZE : 0] com1 = 3'bzzz;
    assign C1 = com1;

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


    reg[`CACHE_SET_SIZE : 0] set = 0;
    reg[`CACHE_TAG_SIZE : 0] tag = 0;
    reg[`CACHE_OFFSET_SIZE : 0] offset = 0;

    task leave_bus_cpu_cache;
        com1 = 3'bzzz;
        data1 = 16'bzzzzzzzzzzzzzzzz;
```

```
    addr1 = 14'bzzzzzzzzzzzzzz;
endtask

int pa = 0;
int pc = 0;
int pb = 0;
int s = 0;
int addr;
int pak;
int pbx;

initial begin
    #4;
    for (int y = 0; y < M; y+=1) begin
        for (int x = 0; x < N; x+=1) begin
            pb = 0;
            s = 0;
            #4;
            for (int k = 0; k < K; k+=1) begin
                //обращение к pa[k]: a[pa][k]
                //a - 8bit => addr = pa + k
                #2;
                addr = pa + k;
                set = addr >> (`CACHE_OFFSET_SIZE + 1);
                tag = addr >> (`CACHE_OFFSET_SIZE + `CACHE_SET_SIZE + 2);
                offset = addr;
                #1;
                com1 = 1;
                addr1[`CACHE_TAG_SIZE : 0] = tag;
                addr1[`ADDR1_BUS_SIZE : `CACHE_TAG_SIZE + 1] = set;
                #2;
                addr1[`CACHE_OFFSET_SIZE : 0] = offset;
                #2;
                leave_bus_cpu_cache();

                wait(C1 == 7);
                cache_usage_count++;


                pak = D1[7:0];

                //обращение к pb[x]: b[pb][x]
                //a - 16bit => addr = (pb + x)*2
                #2;
                addr = M*K + (pb + x)*2;
                set = addr >> (`CACHE_OFFSET_SIZE + 1);
```

```
            tag = addr >> (`CACHE_OFFSET_SIZE + `CACHE_SET_SIZE + 2);
              offset = addr;
              #1;
              com1 = 2;
              addr1[`CACHE_TAG_SIZE : 0] = tag;
              addr1[`ADDR1_BUS_SIZE : `CACHE_TAG_SIZE + 1] = set;
              #2;
              addr1[`CACHE_OFFSET_SIZE : 0] = offset;
              #2;
              leave_bus_cpu_cache();

              wait(C1 == 7);
              cache_usage_count++;

              pbx = D1;

              #20 s += pak * pbx; //время работы +, *

              pb+=N;
          end
          //pc[x] = s;
          //обращение к c[pc][x] для записи
          #2;
          addr = M*K + 2*K*N + (pc + x)*4;
          set = addr >> (`CACHE_OFFSET_SIZE + 1);
          tag = addr >> (`CACHE_OFFSET_SIZE + `CACHE_SET_SIZE + 2);
          offset = addr;
          #1;
          com1 = 7;
          addr1[`CACHE_TAG_SIZE : 0] = tag;
          addr1[`ADDR1_BUS_SIZE : `CACHE_TAG_SIZE + 1] = set;
          data1 = s % 65536;
          #2;
          addr1[`CACHE_OFFSET_SIZE : 0] = offset;
          data1 = s / 65536;
          #2;
          leave_bus_cpu_cache();

          wait(C1 == 7);
          cache_usage_count++;
          #4;
      end
      pa+=K;
      pc+=N;
      #6;
  end
```

```
        #2;
        $display("Value of cache hits: %d", cache_hit_count);
        $display("Value of cache misses: %d", cache_miss_count);
        $display("Value of cache usages: %d", cache_usage_count);
                $display("Percent of cache hits: %f", cache_hit_count * 
100.0/cache_usage_count);
            $display("Percent of cache misses: %f", (100 - cache_hit_count * 
100.0/cache_usage_count));
        $display("Estimated time %d (in processor tact's)", $time / 2);
        $finish;
    end
endmodule
```

MEM.sv

```
`define MEM_SIZE 262144 - 1
`define MEM_LINE_COUNT 16384 - 1
`define CACHE_SIZE 2048 - 1
`define CACHE_LINE_SIZE 16 - 1
`define CACHE_LINE_COUNT 128 - 1
`define CACHE_SETS_COUNT 64 - 1
`define CACHE_TAG_SIZE 8 - 1
`define CACHE_SET_SIZE 6 - 1
`define CACHE_OFFSET_SIZE 4 - 1
`define CACHE_ADDR_SIZE 18 - 1

`define ADDR1_BUS_SIZE 14 - 1
`define ADDR2_BUS_SIZE 14 - 1
`define DATA1_BUS_SIZE 16 - 1
`define DATA2_BUS_SIZE 16 - 1
`define CTR1_BUS_SIZE 3 - 1
`define CTR2_BUS_SIZE 2 - 1

module MEM(input CLK, input [`ADDR2_BUS_SIZE : 0] A2,
            inout [`DATA2_BUS_SIZE : 0] D2, inout [`CTR2_BUS_SIZE : 0] C2,
            input M_DUMP, input RESET);

    reg [`DATA2_BUS_SIZE : 0] retData2 = 16'bzzzzzzzzzzzzzzzz;
    reg [`CTR2_BUS_SIZE : 0] com2 = 2'bzz;

    assign D2 = retData2;
    assign C2 = com2;

    int M = 64;
    int N = 60;
```

```
    int K = 32;

    integer SEED = 225526;

    reg [7:0] data[0 : `MEM_LINE_COUNT][0 : `CACHE_LINE_SIZE];

    initial begin
        for(int i = 0; i < M*K + N*K + M*N; i+=1) begin
            data[ i / (`CACHE_LINE_SIZE + 1) ][ i % (`CACHE_LINE_SIZE + 1) ] =
$random(SEED)>>16;
        end
    end

    reg [`CACHE_TAG_SIZE : 0] cur_tag;
    reg [`CACHE_SET_SIZE : 0] cur_set;

    task leave_bus_mem_cache;
        if(log) begin
            $display("leave_bus_mem_cache");
        end
        #1;
        retData2 = 16'bzzzzzzzzzzzzzzzz;
        com2 = 2'bzz;
    endtask

    always @(posedge CLK) begin
        case (C2)
            0: begin
                //C2_NOP
                if(log) begin
                    $display("Memory: nothing is done");
                end
            end
            2: begin
                //C2_READ_LINE
                if(log) begin
                    $display("Memory: read line");
                end
                cur_tag = A2[`CACHE_TAG_SIZE : 0];
                cur_set = A2[`ADDR2_BUS_SIZE : `CACHE_TAG_SIZE + 1];

                //TAG * set_size + SET
                #100; //memory work time
                if(log) begin
                    $display("Memory: line is read");
                end
```

```
                #1;
                com2 = 1;
                for(int i = 0; i < 8; i+=1) begin
                     retData2 [7:0] = data[cur_tag * (`CACHE_SETS_COUNT + 1) +
cur_set][2*i];
                    retData2 [15:8] = data[cur_tag * (`CACHE_SETS_COUNT + 1) +
cur_set][2*i + 1];
                    #2;
                end
                leave_bus_mem_cache();
            end
            3: begin
                //C2_WRITE_LINE
                if(log) begin
                    $display("Memory: read line");
                end
                cur_tag = A2[`CACHE_TAG_SIZE : 0];
                cur_set = A2[`ADDR1_BUS_SIZE : `CACHE_TAG_SIZE + 1];

                for(int i = 0; i < 8; i+=1) begin
                       data[cur_tag * (`CACHE_SET_SIZE + 1) + cur_set][2*i] =
retData2 [7:0];
                    data[cur_tag * (`CACHE_SET_SIZE + 1) + cur_set][2*i + 1] =
retData2 [15:8];
                    #2;
                end


                #100;
                #1;
                com2 = 1;
                if(log) begin
                    $display("Memory: line is written");
                end
                #2;
                leave_bus_mem_cache();
            end
        endcase

        if(M_DUMP) begin
            if(log) $display("M_DUMP");

            $dumpfile("MEM_DUMP.vcd");
            $dumpvars(1, MEM);
        end
```

```
        if(RESET) begin
            if(log) $display("RESET");

            for(int i = 0; i < M*K + N*K + M*N; i+=1) begin
                data[ i / (`CACHE_LINE_SIZE + 1) ][ i % (`CACHE_LINE_SIZE + 1)
] = $random(SEED)>>16;
            end

        end
    end

endmodule
```

CACHE.sv

```
`define CACHE_SIZE 2048 - 1
`define CACHE_LINE_SIZE 16 - 1
`define CACHE_LINE_COUNT 128 - 1
`define CACHE_SETS_COUNT 64 - 1
`define CACHE_TAG_SIZE 8 - 1
`define CACHE_SET_SIZE 6 - 1
`define CACHE_OFFSET_SIZE 4 - 1
`define CACHE_ADDR_SIZE 18 - 1

`define ADDR1_BUS_SIZE 14 - 1
`define ADDR2_BUS_SIZE 14 - 1
`define DATA1_BUS_SIZE 16 - 1
`define DATA2_BUS_SIZE 16 - 1
`define CTR1_BUS_SIZE 3 - 1
`define CTR2_BUS_SIZE 2 - 1

int log = 0;

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

    reg [1:0] valid[0 : `CACHE_SETS_COUNT][0:1];
    reg [1:0] dirty[0 : `CACHE_SETS_COUNT][0:1];
    reg [1:0] lru[0 : `CACHE_SETS_COUNT][0:1];
    reg [`CACHE_TAG_SIZE : 0] cache_line_tag[0 : `CACHE_SETS_COUNT][0:1];
```

```
reg [7:0] data[0 : `CACHE_SETS_COUNT][0 : `CACHE_LINE_SIZE][0:1];

initial begin
    for(int i = 0; i <= `CACHE_SETS_COUNT; i += 1) begin
        for(int j = 0; j < 2; j += 1) begin
            valid[i][j] = 0;
        end
    end
end

reg[`CACHE_TAG_SIZE : 0] cur_tag;
reg[`CACHE_SET_SIZE : 0] cur_set;
reg[`CACHE_OFFSET_SIZE : 0] cur_offset;
reg[2 * `DATA1_BUS_SIZE + 1: 0] cur_data1;
reg[`DATA2_BUS_SIZE : 0] cur_data2;
reg[`CTR1_BUS_SIZE : 0] cur_com;



reg [`ADDR2_BUS_SIZE : 0] addr2 = 14'bzzzzzzzzzzzzzz;
reg [`DATA1_BUS_SIZE : 0] retData1 = 16'bzzzzzzzzzzzzzzzz;
reg [`DATA2_BUS_SIZE : 0] retData2 = 16'bzzzzzzzzzzzzzzzz;
reg [`CTR1_BUS_SIZE : 0] com1 = 3'bzzz;
reg [`CTR2_BUS_SIZE : 0] com2 = 2'bzz;

reg[1:0] cache_hit;
int done = 0;

assign A2 = addr2;
assign D1 = retData1;
assign D2 = retData2;
assign C1 = com1;
assign C2 = com2;

task leave_bus_cache_mem;
    if(log) begin
        $display("leave_bus_cache_mem");
    end
    #1;
    addr2 = 14'bzzzzzzzzzzzzzz;
    retData2 = 16'bzzzzzzzzzzzzzzzz;
    com2 = 2'bzz;
endtask

task leave_bus_cache_cpu;
    if(log) begin
```

```
        $display("leave_bus_cache_cpu");
    end
    #1;
    retData1 = 16'bzzzzzzzzzzzzzzzz;
    com1 = 3'bzzz;
endtask

task get_input;
    if(log) begin
        $display("get_input");
    end
    cur_tag = A1[`CACHE_TAG_SIZE : 0];
    cur_set = A1[`ADDR1_BUS_SIZE : `CACHE_TAG_SIZE + 1];
    cur_data1[15 : 0] = D1;
    cur_com = C1;
    #2; //wait new part of address and data
    cur_data1[31 : 16] = D1;
    cur_offset = A1;
    #1;
endtask

task check_cache_hit;
    if(log) begin
        $display("check_cache_hit");
    end
    cache_hit = 2'b11;

    for(int j = 0; j < 2; j += 1) begin
        if (cache_line_tag[cur_set][j] != cur_tag) begin
            cache_hit[j] = 0;
        end
    end
endtask

task return_read(int i);
    if(log) begin
        $display("return_read");
    end
    #1;
    com1 = 7;
    retData1[7:0] = data[cur_set][cur_offset][i];
    retData1[15:8] = data[cur_set][cur_offset + 1][i];

    if(cur_com == 3) begin
        #2;
        retData1[7:0] = data[cur_set][cur_offset + 2][i];
```

```
        retData1[15:8] = data[cur_set][cur_offset + 3][i];
    end


    #2;
    leave_bus_cache_cpu();

endtask

task return_response;
    if(log) begin
        $display("return_response");
    end
    #1;
    com1 = 7;

    #2;
    leave_bus_cache_cpu();

endtask

task write(int i);
    if(log) begin
        $display("write");
    end
    data[cur_set][cur_offset][i] = cur_data1[7:0]; //C1_WRITE8
    if(cur_com >= 6) begin  //C1_WRITE16
        data[cur_set][cur_offset + 1][i] = cur_data1[15:8];
    end

    if(cur_com == 7) begin //C1_WRITE32
        data[cur_set][cur_offset + 2][i] = cur_data1[23:16];
        data[cur_set][cur_offset + 3][i] = cur_data1[31:24];
    end
endtask

task set_lru(int i);
    if(log) begin
        $display("set_lru");
    end
    lru[cur_set][i] = 1;
    lru[cur_set][1 - i] = 0;
endtask

task write_procedure(int i);
    if(log) begin
```

```
            $display("write_procedure");
        end
        write(i);
        dirty[cur_set][i] = 1;
        return_response();
    endtask;

    task do_if_cache_hit;
        if(log) begin
            $display("do_if_cache_hit");
        end
        for(int i = 0; i < 2; i += 1) begin
            if (valid[cur_set][i] == 1 && cache_hit[i] == 1 && done == 0)
begin
                // CACHE попадание
                cache_hit_count += 1;

                set_lru(i);
                if(cur_com <= 3) begin
                    return_read(i);
                end else begin
                    write_procedure(i);
                end

                done = 1;
            end
        end
    endtask

    task make_memory_read_request;
        if(log) begin
            $display("make_memory_read_request");
        end

        #1;
        com2 = 2;
        addr2 [`CACHE_TAG_SIZE : 0] = cur_tag;
        addr2 [`ADDR2_BUS_SIZE : `CACHE_TAG_SIZE + 1] = cur_set;
        #2;
        leave_bus_cache_mem();
        #1;

        wait(C2 == 1); //wait for response
    endtask

    task make_memory_write_request(int i);
        if(log) begin
```

```
        $display("make_memory_write_request");
    end
    #2;
    com2 = 3;
    addr2 [`CACHE_TAG_SIZE : 0] = cache_line_tag[cur_set][i];
    addr2 [`ADDR2_BUS_SIZE : `CACHE_TAG_SIZE + 1] = cur_set;

    for(int j = 0; j < 8; j+=1) begin
        retData2[7:0] = data[cur_set][2*j][i];
        retData2[15:8] = data[cur_set][2*j+1][i];
        #2; //wait to send new portion of data
    end

    leave_bus_cache_mem();
    #1;

    //wait for response(to be sure data is written)
    wait(C2 == 1);
    #2;
endtask

task set_info(int i);
    if(log) begin
        $display("set_info");
    end
    valid[cur_set][i] = 1;
    dirty[cur_set][i] = 0;
    set_lru(i);
    cache_line_tag[cur_set][i] = cur_tag;
endtask

task get_data_from_memory(int i);
    if(log) begin
        $display("get_data_from_memory");
    end
    data[cur_set][0][i] = D2[7:0];
    data[cur_set][1][i] = D2[15:8];
    for(int j = 1; j < 8; j += 1) begin
        #2; //wait new data;
        data[cur_set][2*j][i] = D2[7:0];
        data[cur_set][2*j + 1][i] = D2[15:8];
    end
    #2;
endtask

task cache_miss_tail_tasks(int i);
```

```
    if(log) begin
        $display("cache_miss_tail_tasks");
    end
    if(cur_com <= 3) begin

        make_memory_read_request();

        get_data_from_memory(i);

        set_info(i);

        return_read(i);

    end else begin

        write_procedure(i);

    end
endtask

task check_valid;
    if(log) begin
        $display("check_valid");
    end
    for(int i = 0; i < 2; i += 1) begin
        if (valid[cur_set][i] == 0 && done == 0) begin

            cache_miss_tail_tasks(i);

            done = 1;
        end
    end
endtask

task cache_miss_lru;
    if(log) begin
        $display("cache_miss_lru");
    end
    for(int i = 0; i < 2; i += 1) begin

        if (lru[cur_set][i] == 0 && done == 0) begin

            if(dirty[cur_set][i] == 1) begin
                make_memory_write_request(i);
            end
```

```
            cache_miss_tail_tasks(i);
            done = 1;
        end

    end
endtask

task do_if_cache_miss;
    if(log) begin
        $display("do_if_cache_miss");
    end

    cache_miss_count++;

    check_valid();

    if(done == 1) begin
        done = 0;
    end else begin
        cache_miss_lru();
        done = 0;
    end

endtask


always @(posedge CLK) begin
    case (C1)
        0: begin
            //C1_NOP
            if(log) begin
                $display("Cache: command C1_NOP. Nothing is done");
            end
        end
        1, 2, 3, 5, 6, 7: begin

            get_input();

            //C1_READ8 C1_READ16 C1_READ32
            if(log == 1 && cur_com <= 3) begin
                $display("Cache: command C1_READ");
            end

            //C1_WRITE8 C1_WRITE16 C1_WRITE32
            if(log == 1 && cur_com >= 5) begin
                $display("Cache: command C1_WRITE");
```

```
                end

                #8; //cache hit or cache miss wait

                check_cache_hit();

                do_if_cache_hit();

                if(done == 1) begin
                    done = 0;
                end else begin
                    do_if_cache_miss();
                end

            end
            4: begin
                //C1_INVALIDATE_LINE
                if(log) begin
                    $display("Cache: command C1_INVALIDATE_LINE");
                end
                get_input();
                #2;
                for(int i = 0; i < 2; i += 1) begin
                    if (cache_line_tag[cur_set][i] == cur_tag) begin
                        valid[cur_set][i] = 0;
                    end
                end
                return_response();
            end
        endcase
        if(C_DUMP) begin
            $dumpfile("CACHE_DUMP.vcd");
            $dumpvars(1, CACHE);
        end
        if(RESET) begin
            for(int i = 0; i <= `CACHE_SETS_COUNT; i++) begin
                for(int j = 0; j < 2; j++) begin
                    valid[i][j] = 0;
                end
            end
        end
    end
endmodule
```