Лабораторная работа №2	М3138	2022
Моделирование схем в Verilog	Селезнев Дмитрий Александрович	

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

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 12.

Задача

Имеется следующее определение глобальных переменных и функций:

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

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

Необходимо определить процент попаданий для кэша и общее время (в тактах), затраченное на выполнение этой функции.

Вычисление недостающих параметров кэша

CACHE_SIZE = CACHE_LINE_COUNT * CACHE_LINE_SIZE

CACHE_SETS_COUNT = CACHE_LINE_COUNT / CACHE_WAY

CACHE_SET_SIZE = log2(CACHE_SETS_COUNT)

CACHE_OFFSET_SIZE = log2(CACHE_LINE_SIZE)

CACHE_ADDR_SIZE = log2(MEM_SIZE)

Параметры	Значение
CACHE_WAY – ассоциативность	2
CACHE_TAG_SIZE – размер тэга адреса	10 бит
CACHE_LINE_SIZE – размер кэш-линии	16 байт
CACHE_LINE_COUNT – кол-во кэш-линий	64
MEM_SIZE – размер памяти	512 Кбайт
CACHE_SIZE – размер кэша	1024 байт
CACHE_SETS_COUNT – кол-во наборов кэш-линий	32
CACHE_SET_SIZE – размер индекса в наборе кэш-линий	5 бит
CACHE_OFFSET_SIZE – размер смещения	4 бит
CACHE_ADDR_SIZE – размер адреса	19 бит

Таблица 1 – параметры (вариант 1)

Шина	Обозначение	Размерность
A1, A2	**ADDR1_BUS_SIZE**, **ADDR2_BUS_SIZE**	15 бит
D1, D2	**DATA1_BUS_SIZE, DATA2_BUS_SIZE**	16 бит
C1, C2	**CTR1_BUS_SIZE**, **CTR2_BUS_SIZE**	3 бита 2 бита

Таблица 2 – размерность шин

Аналитическое решение задачи

Для того чтобы я решить задачу аналитически я решил написать эмуляцию работы системы на языке программирования C++.

```
void mmul() {
    time_++; // int8 *pa = a;
    time_++; // int32 *pc = c;
    time_++; // int y = 0
    for (int y = 0; y < M; y++) {
        time_++; // int x = 0
        for (int x = 0; x < N; x++) {
            time_++; // int16 *pb = b;
            time_++; // int32 s = 0;
            time_++; // int k = 0
            for (int k = 0; k < K; k++) {
                int addr_a = get_addr(0, y, k);
                int addr_b = get_addr(1, k, x);
                get_from_cache(addr_a);
                get_from_cache(addr_b);
                time_ += 9; // s += pa[k] * pb[x]; pb += N;
                time_++; // new iteration
            }
            int addr_c = get_addr(2, y, x);
            write_to_cache(addr_c);
            time_++; // pc[x] = s;
            time_++; // new iteration
        }
        time_++; // pa += K;
        time_++; // pc += N;
        time_++; // new iteration
    }
}
```

Листинг 1 – Функция mmul

По сути это функция из задания, только адаптированная под решение задачи. time_ – это общее время в тактах, в принципе напротив каждого увеличения time_ написано, почему оно происходит (ровно по тем правилам, что даны в задании).

Далее будут приведены куски кода, показывающие реализацию тех, или иных функции, с минимальными пояснениями, так как основные пояснения будут даны в листинге кода на verilog, а здесь лишь эмуляция поведения системы. Все константы соответствуют данным в задании. Также все названия переменных соответствуют переменным из verilog.

Функция get_addr возвращает адрес первого байта ячейки массива в памяти. Вот ее реализация:

```
int get_addr(int m, int k, int n) {
    if (m == 0) return k * K + n;
    if (m == 1) return M * K + 2 * (k * N + n);
    return M * K + K * N * 2 + 4 * (k * N + n);
}
```

Листинг 2 – Функция get_addr

m=0 означает, что это массив a, тогда адрес байта в памяти считается, как (количество строк)*(размер строки) + позиция в строке. Аналогично m=0 означает массив b, m=2 массив c, и считаются по той же формуле, учитывая, что массивы лежат последовательно, поэтому нужно прибавлять размер предыдущих массивов.

Функция get_from_cache эмулирует поведения системы при чтении.

```
void get_from_cache(int n) {
    int set = (n >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    int tag = n >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    auto block = cache_sets[set];
    for (int i = 0; i < CACHE_WAY; i++) {
        if (tag == block[i] && valid[set * CACHE_WAY + i] == 1) {
            cache_hit++;
            time_ += 6; // wait
            if (i == 1) swap_(set);
            return;
        }
    }
    cache_miss++;
    if (dirty[set * CACHE_WAY + 1] == 1) time_ += 101;
    valid[set * CACHE_WAY + 1] = 1;
    dirty[set * CACHE_WAY + 1] = 0;
    cache_sets[set][1] = tag;
    time_ += 104 + CACHE_LINE_SIZE / DATA1_BUS_SIZE;
    swap_(set);
}
```

Листинг 3 – Функция get_from_cache

В начале из адреса выделяются сет и тэг. Затем идет проверка на кэш попадание и соответствующие действия, если оно произошло, иначе считаем кэш промах и выполняем соответствующие действия.

Функция write_to_cache эмулирует поведение системы при чтении.

```
void write_to_cache(int n) {
    int set = (n >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    int tag = n >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    auto block = cache_sets[set];
    for (int i = 0; i < CACHE_WAY; i++) {
        if (tag == block[i] && valid[set * CACHE_WAY + i] == 1) {
            dirty[set * CACHE_WAY + i] = 1;
            cache_hit++;
            time_ += 6; // wait
            if (i == 1) swap_(set);
            return;
        }
    }
    cache_miss++;
    if (dirty[set * CACHE_WAY + 1] == 1) time_ += 101;
    valid[set * CACHE_WAY + 1] = 1;
    dirty[set * CACHE_WAY + 1] = 1;
```

```
    cache_sets[set][1] = tag;
    time_ += 96 + CACHE_LINE_SIZE / DATA2_BUS_SIZE + CACHE_LINE_SIZE /
DATA1_BUS_SIZE;
    swap_(set);
}
```

Листинг 4 – Функция write_to_cache

Функция практически идентична предыдущей, за исключением того, что иногда нужно пометить кэш-линию, что в ней есть данные, которых нет в памяти.

```
void swap_(int idx) {
    swap(cache_sets[idx][0], cache_sets[idx][1]);
    swap(dirty[idx * CACHE_WAY], dirty[idx * CACHE_WAY + 1]);
    swap(valid[idx * CACHE_WAY], valid[idx * CACHE_WAY + 1]);
}
```

Листинг 5 – Функция swap_

Меняет местами линии внутри одного блока.

В итоге программа вывела:

time: 5202517

Cache_hit: 228080 cache_miss: 21520

Percent: 0.913782

Моделирование заданной системы на Verilog

Я использовал system verilog, так как некоторые конструкции отсутствовали в стандартной версии (например, отсутствие int).

В моей модели три основных блока: CPU, cache, MemCTR, также есть testbench, в котором блоки соединяются проводами в соответствии с данной нам схемой, за исключением того что есть еще один провод STATS, который проведен из CPU в cache, и нужен затем, чтобы cache вывел в консоль статистику попаданий, после завершения работы функции из задачи. Также по шинам D1 и D2 все данные посылаются в little endian, то есть начиная со старших битов.

Для начала рассмотрим функцию, которая есть в каждом блоке: wait_(n), она задерживает выполнение программы на n тактов.

```
task automatic wait_(int time_);
    for (int i = 0; i < time_; i++) begin
      @(posedge CLK);
    end
endtask
```

Листинг 6 – Функция wait_

Теперь подробнее о каждом из блоков. Начнем с MemCTR.

Для начала, рассмотрим, как подключаются шины.

```
input wire[`ADDR2_BUS_SIZE-1:0] A2,
inout wire[`DATA2_BUS_SIZE-1:0] D2,
inout wire[`CTR2_BUS_SIZE-1:0] C2,
input wire CLK,
input wire M_DUMP,
input wire RESET
/* some code */
reg[`DATA2_BUS_SIZE-1:0] D2_ = 'bz;
reg[`CTR2_BUS_SIZE-1:0] C2_ = 'bz;
assign D2 = D2_;
assign C2 = C2_;
```

Листинг 7 – Подключение проводов

Как видно переменные, названные как шины, но с нижним подчеркиванием это регистры, к которым подключены шины. По умолчанию, на регистрах высокоимпедансное состояние

Память умеет лишь записывать кэш-линию из кэша по заданному адресу, возвращать кэш-линию кэшу, а также сбрасывать данные в память(DUMP) и заполнять ячейки значениями по умолчанию(RESET).

```
always @(posedge CLK && C2 === 2) begin
    a2 = A2;
    C2_ = 0;
    wait_(100);
    C2_ = 1;
    for (int i = 7; i >= 0; i--) begin
      D2_ = mem[a2 * `CACHE_LINE_SIZE + i * 2] + (mem[a2 * `CACHE_LINE_SIZE +
i * 2 + 1] << 8);
      wait_(1);
    end
    D2_ = 'bz;
    C2_ = 'bz;
  end
```

Листинг 8 – Чтение из памяти

Строка wait_(100) имитирует задержку в 100 тактов, которая требуется памяти на поиск нужных байтов. После этого память должна начать отвечать, и раз в такт, отдавать по 2 байта данных по нужному адресу, записывая данные на регистр, подключенный к шине данных (для того чтобы это работало, нужно быть уверенным, что со стороны кэша на регистре высокоимпедансное состояние, иначе на проводе будет x). То есть в сумме данная функция работает 108 тактов.

```
always @(posedge CLK && C2 === 3) begin
    a2 = A2;
    for (int i = 7; i >= 0; i--) begin
      mem[a2 * `CACHE_LINE_SIZE + i * 2] = D2 % (1 << 8);
      mem[a2 * `CACHE_LINE_SIZE + i * 2 + 1] = D2 >> 8;
```

```
      wait_(1);
    end
    C2_ = 0;
    wait_(92);
    C2_ = 1;
    wait_(1);
    C2_ = 'bz;
end
```

Листинг 9 – Запись в память

Сразу, как только пришел запрос на запись, начинаем записывать, по указанному на шине A адресу, затем имитируем задержку так, чтобы в сумме память начала отвечать через 100 тактов. Также в обеих приведенных выше функциях в конце нужно поставить на регистры высокоимпедансное состояние, чтобы кэш мог послать следующий запрос.

Далее рассмотрим работу кэша(module cache).

В задании дан look-through write-back кэш с политикой вытеснения LRU.

Look-through – это значит, что у нас CPU не общается напрямую с памятью, только с кэшем. Write-back – это значит, что при получении команды на запись, данные для начала записываются в кэш и данная кэш-линия помечается dirty, и если нам поступила команда инвалидации или по какой либо другой причине нам нужно убрать кэш-линию из кэша, то нужно для начала выписать данную кэш-линию в память. При политике вытеснения LRU при кэш промахе вытесняется та линия, к которой не обращались дольше.

Для начала рассмотрим инициализации переменных и подключение шин к регистрам.

```
  reg[`ADDR2_BUS_SIZE-1:0] A2_out = 'bz;
  reg[`DATA1_BUS_SIZE-1:0] D1_ = 'bz;
  reg[`CTR1_BUS_SIZE-1:0] C1_ = 'bz;
  reg[`DATA2_BUS_SIZE-1:0] D2_ = 'bz;
  reg[`CTR2_BUS_SIZE-1:0] C2_ = 0;
  assign A2 = A2_out;
  assign D1 = D1_;
  assign D2 = D2_;
  assign C1 = C1_;
  assign C2 = C2_;


  reg[`DATA1_BUS_SIZE-1:0] save_data1;
  reg[`DATA1_BUS_SIZE-1:0] save_data2;
  reg[`CTR1_BUS_SIZE-1:0] save_c1;
  reg[`CACHE_TAG_SIZE*2-1:0] Cache_Set[0:`CACHE_SETS_COUNT-1];
  reg[7:0] Cache[0:`CACHE_SIZE-1];
  reg valid[0:`CACHE_LINE_COUNT-1];
  reg dirty[0:`CACHE_LINE_COUNT-1];
  int cache_miss = 0;
```

```
int cache_hit = 0;
integer SEED = 225526;
integer fd;
int h;
int a1;
int offset = 0;
int set = 0;
int tag = 0;
int tags_line[0:`CACHE_WAY - 1];
int idx = 0;
int pos;
int flag;
```

Листинг 10 – Инициализация

Так как данные из CPU отсылаются в два такта, то нужно где-то их сохранять, для этого созданы save_data и save_c1, а также a1, set, tag, offset – для хранения адреса. В Cache_set лежат тэги из одного сета, причем с 0 по 9 биты всегда лежит тэг, по которому обращались позднее. Cache – хранит кэш-линии в байтах по порядку. valid и dirty хранят соответственно биты валидности и изменяемости. cahce_miss и cache_hit – счетчики числа промахов и попаданий. В tags_line в некоторые моменты времени записываются тэги из Cache_set (для этого реализована функция get_tags_from_set). Остальные переменные вспомогательные и о них будет чуть позже.

Теперь рассмотрим функции, которые взаимодействуют с MemCTR.

```
task automatic read_mem(int A);
  A2_out = A;
  C2_ = `CTR2_BUS_SIZE'b10;
  wait_(1);
  A2_out = 'bz;
  C2_ = 'bz;
endtask

task automatic write_mem(int A, int tag_in_set);
  get_set(A);
  A2_out = A;
  C2_ = `CTR2_BUS_SIZE'b11;
  pos = (set * `CACHE_WAY + tag_in_set) * `CACHE_LINE_SIZE;
  for (int i = 7; i >= 0; i--) begin
      D2_ = (Cache[pos + i * 2 + 1] << 8) + Cache[pos + i * 2];
      wait_(1);
      A2_out = 'bz;
  end
  D2_ = 'bz;
  C2_ = 'bz;
endtask
```

Листинг 11 – Функции read_mem и write_mem

Как видно read_mem, посылает запрос на чтение данных по адресу, который ей передали, причем адрес сразу без offset, так как при взаимодействии с памятью он не нужен. Что происходит со стороны памяти,

я уже писал выше. Функция write_mem посылает команду на запись, и за 8 тактов отсылает кэш-линию, которую надо записать в память.

Функции get_set, get_tag, get_offset достают из переданного им адреса set, tag, offset соответственно (причем get_set, get_tag работают с учетом того, что им дали адрес с обрезанным offset, а get_offset так, что нет тэга и сета).

```
task automatic swap(int idx);
    reg[`CACHE_TAG_SIZE-1:0] first_tag = Cache_Set[idx] % (1 <<
`CACHE_TAG_SIZE);
    Cache_Set[idx] >>= `CACHE_TAG_SIZE;
    Cache_Set[idx] = Cache_Set[idx] + (first_tag << `CACHE_TAG_SIZE);
    h = valid[idx * `CACHE_WAY];
    valid[idx * `CACHE_WAY] = valid[idx * `CACHE_WAY + 1];
    valid[idx * `CACHE_WAY + 1] = h;
    h =  dirty[idx * `CACHE_WAY];
    dirty[idx * `CACHE_WAY] = dirty[idx * `CACHE_WAY + 1];
    dirty[idx * `CACHE_WAY + 1] = h;
    for (int i = 0; i < `CACHE_LINE_SIZE; i++) begin
      reg[7:0] t = Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i];
      Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i] = Cache[idx * `CACHE_WAY
* `CACHE_LINE_SIZE + i + `CACHE_LINE_SIZE];
      Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i + `CACHE_LINE_SIZE] = t;
    end
  endtask
```

Листинг 12 – Функция swap

Так как мне необходимо всегда держать тэги в определенном порядке (чтобы знать какой был раньше), то иногда нужно менять кэш-линии местами, для этого, и предназначена эта функция. Ей на вход передается индекс сета, где нужно поменять линии местами.

Далее рассмотрим функцию, которая после отправки запроса на чтение из памяти, принимает оттуда ответ.

```
task automatic read_from_mem(int A);
    wait(C2 === 1);
    get_set(A);
    get_tag(A);
    swap(set);
    Cache_Set[set] = ((Cache_Set[set] >> 8) << 8) + tag;
    valid[set * `CACHE_WAY] = 1;
    dirty[set * `CACHE_WAY] = 0;
    for (int i = 7; i >= 0; i--) begin
      Cache[set * `CACHE_LINE_SIZE * `CACHE_WAY + i * 2] = D2 % (1 << 8);
      Cache[set * `CACHE_LINE_SIZE * `CACHE_WAY + i * 2 + 1] = (D2 >> 8);
      wait_(1);
    end
  endtask
```

Листинг 13 – Функция read_from_mem

Ей на вход передается адрес кэш-линии и в самом начале она ждет ответ из памяти. Затем записывает данные вместо той линии, адрес которой

передали. Попутно нужно еще поменять местами кэш линии, так как нужно поддерживать инвариант того, что сначала идет линия, к которой обращались последней.

Теперь рассмотрим функции взаимодействующие с CPU.

Далее несколько частей кода будут взяты из одного always блока, они разбиты на несколько вставок для лучшей читаемости.

```
    if (C1 === 4) begin
      a1 = A1;
      invalid(a1);
      wait_(1);
      C2_ = 0;
      C1_ = 3'b111;
      wait_(1);
      C1_ = 'bz;
    end
```

Листинг 14 – Запрос инвалидации

Когда приходит запрос инвалидации вызывается функция, в которой и реализована инвалидация. Здесь же она лишь вызывается, а потом отсылает CPU ответ, что инвалидация успешно завершилась.

Реализация этой функции.

```
  task automatic invalid(int A);
    get_set(A);
    get_tag(A);
    get_tags_from_set(set);
    if (tags_line[0] === tag) begin
        if (dirty[set * `CACHE_WAY] == 1 && valid[set * `CACHE_WAY] == 1)
begin write_mem(A, 0); @(posedge C2); end
        valid[set * `CACHE_WAY] = 0;
        swap(set);
    end else if (tags_line[1] === tag) begin
        if (dirty[set * `CACHE_WAY + 1] == 1 && valid[set * `CACHE_WAY + 1]
== 1) begin write_mem(A, 1); @(posedge C2); end
        valid[set * `CACHE_WAY + 1] = 0;
    end
  endtask
```

Листинг 15 – Функция invalid

По указанному адресу находит кэш-линию, и если она помечена dirty (то есть ее нет в mem), то она записывается в память. Затем обнуляется валидность и на этом функция заканчивает работу.

Далее рассмотрим часть кода, отвечающую за регистрацию кэш-попадания.

```
save_c1 = C1;
a1 = A1;
save_data1 = D1;
get_set(a1);
```

```
get_tag(a1);
get_tags_from_set(set);
flag = 0;
```

Листинг 16 – Сохранение значений с шин

Для начала сохраним все переданные нам значения с проводов во вспомогательные регистры.

```
        for (int i = 0; i < `CACHE_WAY; i++) begin
           if (tags_line[i] === tag && valid[set * `CACHE_WAY + i] === 1
&& flag == 0) begin
               flag = 1;
               cache_hit++;
               wait_(1);
               save_data2 = D1;
               get_offset(A1);
               C1_ = 0;
               idx = (set * `CACHE_WAY + i) * `CACHE_LINE_SIZE + offset;
               /* some code */
           end
       end
```

Листинг 17 – Проверка на кэш попадание

Затем идет перебор тэгов из соответствующего сета, если находим совпадения, то считаем кэш-попадание, принимаем вторую половину адреса и данных и выставляем в ответ на шину команд NOP.

```
if (save_c1 > 0 && save_c1 < 4) begin
  wait_(5);
  C1_ = `CTR1_BUS_SIZE'b111;
  if (save_c1 === 1) begin
    D1_ = Cache[idx];
  end else if (save_c1 === 2) begin
    D1_ = (Cache[idx + 1] << 8) + Cache[idx];
  end else if (save_c1 === 3) begin
    D1_ = (Cache[idx + 3] << 8) + Cache[idx + 2];
    wait_(1);
    D1_ = (Cache[idx + 1] << 8) + Cache[idx];
  end
  if (i == 1) swap(set);
  wait_(1);
  C1_ = 'bz;
  D1_ = 'bz;
end
```

Листинг 18 – Запрос на чтение

Далее, если у нас команда на чтение, то записываем данные с шины данных, имитируем задержку до 6 тактов до ответа и отсылаем запрошенные данные. После меняем местами линии, если мы записывали во вторую, так как нужно сохранить инвариант.

```
if (save_c1 > 4 && save_c1 < 8) begin
```

```
  dirty[set * `CACHE_WAY + i] = 1;
  if (save_c1 === 5) begin
    Cache[idx] = save_data1 % (1 << 8);
  end else if (save_c1 === 6) begin
    Cache[idx] = save_data1 % (1 << 8);
    Cache[idx + 1] = save_data1 >> 8;
  end else if (save_c1 === 7) begin
    Cache[idx] = save_data2 % (1 << 8);
    Cache[idx + 1] = save_data2 >> 8;
    Cache[idx + 2] = save_data1 % (1 << 8);
    Cache[idx + 3] = save_data1 >> 8;
  end
  if (i == 1) begin swap(set); end
  wait_(5);
  C1_ = `CTR1_BUS_SIZE'b111;
  wait_(1);
  C1_ = 'bz;
end
```

Листинг 19 – Запрос на запись

Если же пришел запрос на запись, то записываем данные и ждем до 6 тактов, чтобы ответить успешным завершением записи.

Если же кэш-попадания не было, то происходит следующее.

```
      if (flag == 0) begin
        cache_miss++;
        wait_(1);
        get_offset(A1);
        save_data2 = D1;
        C1_ = 0;
        wait_(3);
        if (dirty[set * `CACHE_WAY + 1] === 1 && valid[set * `CACHE_WAY + 1]
=== 1) begin
          invalid((tags_line[1] << `CACHE_SET_SIZE) + set);
          wait_(1);
        end
        read_mem(a1);
        read_from_mem(a1);
        answer_for_CPU(a1);
      end
```

Листинг 20 – Кэш-промах

Увеличиваем счетчик, получаем вторую порцию данных с шины. Делаем проверку, что на кэш-линии, которую мы хотим заменить, данные уже есть в памяти, иначе записываем их в память. Далее читаем из памяти нужную нам линию, и отсылаем CPU ответ о завершении работы.

```
task automatic answer_for_CPU(int A);
    get_set(A);
    get_tag(A);
    idx = set * `CACHE_WAY * `CACHE_LINE_SIZE + offset;
    if (save_c1 === 1) begin
      dirty[set * `CACHE_WAY] = 0;
      C1_ = `CTR1_BUS_SIZE'b111;
```

```
      D1_ = Cache[idx];
    end else if (save_c1 === 2) begin
      dirty[set * `CACHE_WAY] = 0;
      C1_ = `CTR1_BUS_SIZE'b111;
      D1_ = (Cache[idx + 1] << 8) + Cache[idx];
    end else if (save_c1 === 3) begin
      dirty[set * `CACHE_WAY] = 0;
      C1_ = `CTR1_BUS_SIZE'b111;
      D1_ = (Cache[idx + 3] << 8) + Cache[idx + 2];
      wait_(1);
      D1_ = (Cache[idx + 1] << 8) + Cache[idx];
    end
    if (save_c1 === 5) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data1 % (1 << 8);
    end else if (save_c1 === 6) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data1 % (1 << 8);
      Cache[idx + 1] = save_data1 >> 8;
    end else if (save_c1 === 7) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data2 % (1 << 8);
      Cache[idx + 1] = save_data2 >> 8;
      Cache[idx + 2] = save_data1 % (1 << 8);
      Cache[idx + 3] = save_data1 >> 8;
    end
    C1_ = `CTR1_BUS_SIZE'b111;
    wait_(1);
    D1_ = 'bz;
    C1_ = 'bz;
  endtask
```

Листинг 21 – Функция answer_for_CPU

Помимо отправки response, данная функция также обновляет данные в кэше и помечает эту линию, если был запрос на запись, и отсылает данные, если был запрос на чтение.

Теперь рассмотрим модуль CPU.

```
reg[`ADDR1_BUS_SIZE-1:0] A1_out = 'bz;
  reg[`DATA1_BUS_SIZE-1:0] D1_ = 'bz;
  reg[`CTR1_BUS_SIZE-1:0] C1_ = 3'b0;
  reg stats = 0;
  assign STATS = stats;
  assign A1 = A1_out;
  assign D1 = D1_;
  assign C1 = C1_;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_a;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_b;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_c;
  reg[`DATA1_BUS_SIZE - 1 : 0] reg_d;
  int pa = 0;
  int command = 0;
  int s;
  int pb;
  int pc = `M * `K + `K * `N * 2;
  int addr_a;
```

```
  int addr_b;
  int addr_c;
  int a, b, c;
  int all_tic = 0;
  int tacts = 0;
  int fd = $fopen("input.txt", "w");
```

Листинг 22 – Инициализация переменных в CPU

Здесь подключается провод STATS, по которому CPU отправляет запрос к кэшу на вывод кэш-попаданий и промахов. На регистрах reg_a, reg_b, reg_c, сохраняются данные из запросов на чтение 1, 2, 4, байт соответственно. Смысл остальных станет понятен позже.

```
task automatic inval(int A);
    C1_ = 4;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    wait_(1);
    C1_ = 'bz;
    A1_out = 'bz;
    wait(C1 === 7);
    C1_ = 0;
  endtask
```

Листинг 23 – Функция отправки запроса на инвалидацию

Ничего необычного, просто отправка запроса на инвалидацию, с ожиданием ответа от кэша.

```
task write(int A, int cc);
    C1_ = cc;
    command = cc;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    if (cc === 5) begin
      D1_ = reg_a % (1 << 8);
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    end else if (cc === 6) begin
      reg_d = reg_b;
      D1_ = reg_b;
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    end else if (cc === 7) begin
      reg_d = reg_c >> `DATA1_BUS_SIZE;
      D1_ = reg_d;
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
      reg_d = reg_c % (1 << `DATA1_BUS_SIZE);
      D1_ = reg_d;
    end
    wait_(1);
    D1_ = 'bz;
    C1_ = 'bz;
    A1_out = 'bz;
  endtask
```

Листинг 24 – Функция write

Функция запоминает, что был отправлен запрос на запись, затем отправляет данные за 1 или 2 такта, в зависимости от команды. Отмечу, что функция не ждет ответа, так как за это отвечает другая.

```
task read(int A, int cc);
    C1_ = cc;
    command = cc;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    wait_(1);
    A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    wait_(1);
    C1_ = 'bz;
    A1_out = 'bz;
  endtask
```

Листинг 25 – Функция read

Функция также запоминает, что был послан запрос на чтение, затем отсылает сам запрос.

Далее будет приведена функция, которая ждет ответ после отправки запросов, и в зависимости от него делает соответствующие действия. При записи просто идет дальше, при чтении записывает данные в регистры за 1 или 2 такта.

```
task automatic answer_from();
    wait(C1 === 7);
    if (command == 1) begin
      reg_a = D1;
    end else if (command == 2) begin
      reg_b = D1;
    end else if (command == 3) begin
      reg_c = D1;
      wait_(1);
      reg_c <<= 16;
      reg_c = D1 + reg_c;
    end
    C1_ = 0;
    command = 0;
  endtask
```

Листинг 26 – Функция answer_from

```
    wait_(2);
    pa = 0;
    pc = `M * `K + `K * `N * 2;
    wait_(1); // int y = 0;
    for (int y = 0; y < `M; y++) begin
      wait_(1); // int x = 0;
      for (int x = 0; x < `N; x++) begin
        pb = `M * `K;
        s = 0;
        wait_(2);
        wait_(1); // int k = 0;
        for (int k = 0; k < `K; k++) begin
          read(pa + k, 1);
          all_tic++;
```

```
            answer_from();
            all_tic++;
            wait_(1);
            read(pb + x * 2, 2);
            answer_from();
            a = reg_a;
            b = reg_b;
            s += a * b;
            wait_(8);
            pb += `N * 2;
            wait_(1); // new iteration
          end
          addr_c = get_addr(2, y, x);
          reg_c = s;
          all_tic++;
          write(pc + x * 4, 7);
          answer_from();
          wait_(1);
          wait_(1); // new iteration
        end
        pa += `K;
        pc += `N * 4;
        wait_(2);
        if (y % 10 == 0) begin
          $display(y, tacts);
          stats = 1;
        end
        wait_(1); // new iteration
        stats = 0;
      end
      $display("Time: %0d", tacts);
      stats = 1;
```

Листинг 27- Реализация задачи в CPU

Все wait_ имитируют те или иные задержки согласно условию задачи, подробнее о каждой можно увидеть в аналитическом решении. pa, pb, pc – хранят адрес первого байта текущей ячейки памяти в соответствующем массиве. В переменной all_tic считаются все обращения к кэшу. В самом внутреннем for происходит два чтения из памяти и получения ответа, с помощью функций, которые я описал выше, а также подсчет значения, которое в последствии запишется в c. После этого происходит запись в память значения из массива c, также используя приведенные выше функции. Также раз в 10 итераций самого внешнего массива я вывожу логи, чтобы удобнее было смотреть за выполнением программы. В конце я вывожу общее количество тактов, которое потребовалось на выполнение программы (это переменная tacts, она увеличивается в другом always блоке, но он буквально из 1 строки). Также стоит отметить, что в коде в данном always блоке в конце будет несколько for, которые записывают массив c в файл input.txt. Это сделано для проверки работоспособности чтения и записи 4 байтовых чисел.

Все блоки соединяются в еще одном блоке testbench, как это выглядит:

```
  reg clk = 0;
  reg c_dum = 0;
  reg m_dum = 0;
  reg res = 0;
  wire STATS;
  wire C_DUM;
  wire M_DUM;
  wire RES;
  wire CLK;
  wire[`ADDR1_BUS_SIZE-1:0] A1;
  wire[`DATA1_BUS_SIZE-1:0] D1;
  wire[`CTR1_BUS_SIZE-1:0] C1;
  wire[`ADDR2_BUS_SIZE-1:0] A2;
  wire[`DATA2_BUS_SIZE-1:0] D2;
  wire[`CTR2_BUS_SIZE-1:0] C2;
  assign CLK = clk;
  assign C_DUM = c_dum;
  assign M_DUM = m_dum;
  assign RES = res;

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

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

  MemCTR MEM(.A2(A2), .D2(D2), .C2(C2), .CLK(CLK), .M_DUMP(M_DUM),
.RESET(RES));

  always #1 clk = ~clk;

  initial begin
    res = 1;
    #1;
    res = 0;
  end
```

Листинг 28 – Соединение модулей

Также здесь происходит изменение тактов и reset всей памяти с кэшем.

Логи программы:

0 82016

cache_hit: 3552, cache_miss: 348

10 897974

cache_hit: 39161, cache_miss: 3739

20 1712832

cache_hit: 74788, cache_miss: 7112

30 2521400

cache_hit: 110461, cache_miss: 10439

40 3337792

cache_hit: 146064, cache_miss: 13836

50 4149682

cache_hit: 181719, cache_miss: 17181

60 4961218

cache_hit: 217364, cache_miss: 20536

Time: 5202517

cache_hit: 228080, cache_miss: 21520

Эти данные полностью совпадают со значениями, полученными в аналитическом решении, так что, я считаю, что система работает верно.

Листинг кода

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

// const for cache
const int CACHE_WAY = 2;
const int CACHE_TAG_SIZE = 10;
const int CACHE_LINE_SIZE = 16;
const int CACHE_LINE_COUNT = 64;
const int MEM_SIZE = (1 << 19);
const int CACHE_SIZE = (1 << 10);
const int CACHE_SETS_COUNT = 32;
const int CACHE_SET_SIZE = 5;
const int CACHE_OFFSET_SIZE = 4;
const int CACHE_ADDR_SIZE = 19;

// const for bus
const int ADDR1_BUS_SIZE = 15;
const int ADDR2_BUS_SIZE = 15;
const int DATA1_BUS_SIZE = 2;
const int DATA2_BUS_SIZE = 2;
const int CTR1_BUS_SIZE = 3;
const int CTR2_BUS_SIZE = 2;

// const for task
const int M = 64;
const int N = 60;
const int K = 32;

int time_ = 0;
int cache_hit = 0, cache_miss = 0;
// int8 a[M][K];
// int16 b[K][N];
// int32 c[M][N];
int cache_sets[CACHE_SETS_COUNT][CACHE_WAY];
int valid[CACHE_LINE_COUNT];
int dirty[CACHE_LINE_COUNT];
```

```
void swap_(int idx) {
    swap(cache_sets[idx][0], cache_sets[idx][1]);
    swap(dirty[idx * CACHE_WAY], dirty[idx * CACHE_WAY + 1]);
    swap(valid[idx * CACHE_WAY], valid[idx * CACHE_WAY + 1]);
}

void get_from_cache(int n) {
    int set = (n >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    int tag = n >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    auto block = cache_sets[set];
    for (int i = 0; i < CACHE_WAY; i++) {
        if (tag == block[i] && valid[set * CACHE_WAY + i] == 1) {
            cache_hit++;
            time_ += 6; // wait
            if (i == 1) swap_(set);
            return;
        }
    }
    cache_miss++;
    if (dirty[set * CACHE_WAY + 1] == 1) time_ += 101;
    valid[set * CACHE_WAY + 1] = 1;
    dirty[set * CACHE_WAY + 1] = 0;
    cache_sets[set][1] = tag;
    time_ += 104 + CACHE_LINE_SIZE / DATA1_BUS_SIZE;
    swap_(set);
}


void write_to_cache(int n) {
    int set = (n >> CACHE_OFFSET_SIZE) % (1 << CACHE_SET_SIZE);
    int tag = n >> (CACHE_SET_SIZE + CACHE_OFFSET_SIZE);
    auto block = cache_sets[set];
    for (int i = 0; i < CACHE_WAY; i++) {
        if (tag == block[i] && valid[set * CACHE_WAY + i] == 1) {
            dirty[set * CACHE_WAY + i] = 1;
            cache_hit++;
            time_ += 6; // wait
            if (i == 1) swap_(set);
            return;
        }
    }
    cache_miss++;
    if (dirty[set * CACHE_WAY + 1] == 1) time_ += 101;
    valid[set * CACHE_WAY + 1] = 1;
    dirty[set * CACHE_WAY + 1] = 1;
    cache_sets[set][1] = tag;
    time_ += 96 + CACHE_LINE_SIZE / DATA2_BUS_SIZE + CACHE_LINE_SIZE /
DATA1_BUS_SIZE;
    swap_(set);
}


int get_addr(int m, int k, int n) {
    if (m == 0) return k * K + n;
    if (m == 1) return M * K + 2 * (k * N + n);
    return M * K + K * N * 2 + 4 * (k * N + n);
}
```

```
void mmul() {
    time_++; // int8 *pa = a;
    time_++; // int32 *pc = c;
    time_++; // int y = 0
    for (int y = 0; y < M; y++) {
        time_++; // int x = 0
        for (int x = 0; x < N; x++) {
            time_++; // int16 *pb = b;
            time_++; // int32 s = 0;
            time_++; // int k = 0
            for (int k = 0; k < K; k++) {
                int addr_a = get_addr(0, y, k);
                int addr_b = get_addr(1, k, x);
                get_from_cache(addr_a);
                get_from_cache(addr_b);
                time_ += 9; // s += pa[k] * pb[x]; pb += N;
                time_++; // new iteration
            }
            int addr_c = get_addr(2, y, x);
            write_to_cache(addr_c);
            time_++; // pc[x] = s;
            time_++; // new iteration
        }
        time_++; // pa += K;
        time_++; // pc += N;
        time_++; // new iteration
    }
}


int main() {
    for (auto &i : cache_sets) for (auto &j : i) j = -1;
    mmul();
    cout << "time: " << time_ << "\n";
    cout << cache_hit << " " << cache_miss << " " << cache_hit + cache_miss
<< "\n";
    cout << "percent: " << (double)cache_hit / (double)(cache_hit +
cache_miss) << "\n";
}
```

Листинг 29 – Аналитическое решение

```
`define CACHE_WAY 2
`define CACHE_TAG_SIZE 10
`define CACHE_LINE_SIZE 16
`define CACHE_LINE_COUNT 64
`define MEM_SIZE (1 << 19)
`define CACHE_SIZE (1 << 10)
`define CACHE_SETS_COUNT 32
`define CACHE_SET_SIZE 5
`define CACHE_OFFSET_SIZE 4
`define CACHE_ADDR_SIZE 19
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2
`define M 64
```

```
`define N 60
`define K 32

module CPU (
  output wire[`ADDR1_BUS_SIZE-1:0] A1,
  inout wire[`DATA1_BUS_SIZE-1:0] D1,
  inout wire[`CTR1_BUS_SIZE-1:0] C1,
  input wire CLK,
  output wire STATS
  );
  reg[`ADDR1_BUS_SIZE-1:0] A1_out = 'bz;
  reg[`DATA1_BUS_SIZE-1:0] D1_ = 'bz;
  reg[`CTR1_BUS_SIZE-1:0] C1_ = 3'b0;
  reg stats = 0;
  assign STATS = stats;
  assign A1 = A1_out;
  assign D1 = D1_;
  assign C1 = C1_;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_a;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_b;
  reg[`DATA1_BUS_SIZE * 2 - 1 : 0] reg_c;
  reg[`DATA1_BUS_SIZE - 1 : 0] reg_d;
  int pa = 0;
  int command = 0;
  int s;
  int pb;
  int pc = `M * `K + `K * `N * 2;
  int addr_a;
  int addr_b;
  int addr_c;
  int a, b, c;
  int all_tic = 0;
  int tacts = 0;
  int fd = $fopen("input.txt", "w");

  task automatic wait_(int time_);
    for (int i = 0; i < time_; i++) begin
      @(posedge CLK);
    end
  endtask

  function int get_addr(int m, int k, int n);
    if (m == 0) return k * `K + n;
    if (m == 1) return `M * `K + 2 * (k * `N + n);
    return `M * `K + `K * `N * 2 + 4 * (k * `N + n);
  endfunction

  always @(posedge CLK && 1) tacts++;

  always @(posedge CLK) begin
    $display("MAIN------------------------------------------------------------
-----------------------------------------------");
    C1_ = 0;
    wait_(2);
    pa = 0;
    pc = `M * `K + `K * `N * 2;
    wait_(1); // int y = 0;
    for (int y = 0; y < `M; y++) begin
      wait_(1); // int x = 0;
```

```
      for (int x = 0; x < `N; x++) begin
        pb = `M * `K;
        s = 0;
        wait_(2);
        wait_(1); // int k = 0;
        for (int k = 0; k < `K; k++) begin
          read(pa + k, 1);
          all_tic++;
          answer_from();
          all_tic++;
          wait_(1);
          read(pb + x * 2, 2);
          answer_from();
          a = reg_a;
          b = reg_b;
          s += a * b;
          wait_(8);
          pb += `N * 2;
          wait_(1); // new iteration
        end
        addr_c = get_addr(2, y, x);
        reg_c = s;
        all_tic++;
        write(pc + x * 4, 7);
        answer_from();
        wait_(1);
        wait_(1); // new iteration
      end
      pa += `K;
      pc += `N * 4;
      wait_(2);
      if (y % 10 == 0) begin
        $display(y, tacts);
        stats = 1;
      end
      wait_(1); // new iteration
      stats = 0;
    end
    $display("Time: %0d", tacts);
    stats = 1;
    wait_(1);
    for (int y = 0; y < `M; y++) begin
      for (int x = 0; x < `N; x++) begin
        addr_c = get_addr(2, y, x);
        read(addr_c, 3);
        answer_from();
        $fdisplay(fd, "%d", reg_c);
        wait_(1);
      end
    end
    $fclose(fd);
    $finish;
  end

  task automatic inval(int A);
    C1_ = 4;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    wait_(1);
    C1_ = 'bz;
```

```
    A1_out = 'bz;
    wait(C1 === 7);
    C1_ = 0;
  endtask

  task automatic answer_from();
    wait(C1 === 7);
    if (command == 1) begin
      reg_a = D1;
    end else if (command == 2) begin
      reg_b = D1;
    end else if (command == 3) begin
      reg_c = D1;
      wait_(1);
      reg_c <<= 16;
      reg_c = D1 + reg_c;
    end
    C1_ = 0;
    command = 0;
  endtask

  task write(int A, int cc);
    C1_ = cc;
    command = cc;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    if (cc === 5) begin
      D1_ = reg_a % (1 << 8);
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    end else if (cc === 6) begin
      reg_d = reg_b;
      D1_ = reg_b;
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    end else if (cc === 7) begin
      reg_d = reg_c >> `DATA1_BUS_SIZE;
      D1_ = reg_d;
      wait_(1);
      A1_out = A % (1 << `CACHE_OFFSET_SIZE);
      reg_d = reg_c % (1 << `DATA1_BUS_SIZE);
      D1_ = reg_d;
    end
    wait_(1);
    D1_ = 'bz;
    C1_ = 'bz;
    A1_out = 'bz;
  endtask

  task read(int A, int cc);
    C1_ = cc;
    command = cc;
    A1_out = A >> `CACHE_OFFSET_SIZE;
    wait_(1);
    A1_out = A % (1 << `CACHE_OFFSET_SIZE);
    wait_(1);
    C1_ = 'bz;
    A1_out = 'bz;
  endtask
endmodule
```

Листинг 30 – CPU.sv

```
`define CACHE_WAY 2
`define CACHE_TAG_SIZE 10
`define CACHE_LINE_SIZE 16
`define CACHE_LINE_COUNT 64
`define MEM_SIZE (1 << 19)
`define CACHE_SIZE (1 << 10)
`define CACHE_SETS_COUNT 32
`define CACHE_SET_SIZE 5
`define CACHE_OFFSET_SIZE 4
`define CACHE_ADDR_SIZE 19
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2
`define M 64
`define N 60
`define K 32

module cache (
  input wire[`ADDR1_BUS_SIZE-1:0] A1,
  output wire[`ADDR2_BUS_SIZE-1:0] A2,
  inout wire[`DATA1_BUS_SIZE-1:0] D1,
  inout wire[`DATA2_BUS_SIZE-1:0] D2,
  inout wire[`CTR1_BUS_SIZE-1:0] C1,
  inout wire[`CTR2_BUS_SIZE-1:0] C2,
  input wire CLK,
  input wire C_DUMP,
  input wire RESET,
  input wire STATS
  );
  reg[`ADDR2_BUS_SIZE-1:0] A2_out = 'bz;
  reg[`DATA1_BUS_SIZE-1:0] D1_ = 'bz;
  reg[`CTR1_BUS_SIZE-1:0] C1_ = 'bz;
  reg[`DATA2_BUS_SIZE-1:0] D2_ = 'bz;
  reg[`CTR2_BUS_SIZE-1:0] C2_ = 0;
  assign A2 = A2_out;
  assign D1 = D1_;
  assign D2 = D2_;
  assign C1 = C1_;
  assign C2 = C2_;


  reg[`DATA1_BUS_SIZE-1:0] save_data1;
  reg[`DATA1_BUS_SIZE-1:0] save_data2;
  reg[`CTR1_BUS_SIZE-1:0] save_c1;
  reg[`CACHE_TAG_SIZE*2-1:0] Cache_Set[0:`CACHE_SETS_COUNT-1];
  reg[7:0] Cache[0:`CACHE_SIZE-1];
  reg valid[0:`CACHE_LINE_COUNT-1];
  reg dirty[0:`CACHE_LINE_COUNT-1];
  int cache_miss = 0;
  int cache_hit = 0;
  integer SEED = 225526;
  integer fd;
  int h;
```

```
  int a1;
  int offset = 0;
  int set = 0;
  int tag = 0;
  int tags_line[0:`CACHE_WAY - 1];
  int idx = 0;
  int pos;
  int flag;

  initial begin
    for (int i = 0; i < `CACHE_LINE_COUNT; i++) Cache_Set[i] = 0;
    for (int i = 0; i < `CACHE_SIZE; i++) Cache[i] = 0;
  end

  task automatic get_set(int A);
    set = A % (1 << `CACHE_SET_SIZE);
  endtask

  task automatic get_offset(int A);
    offset = A % (1 << `CACHE_OFFSET_SIZE);
  endtask

  task automatic get_tag(int A);
    tag = A >> `CACHE_SET_SIZE;
  endtask

  task automatic get_tags_from_set(int set1);
    tags_line[0] = Cache_Set[set1] % (1 << `CACHE_TAG_SIZE);
    tags_line[1] = Cache_Set[set1] >> `CACHE_TAG_SIZE;
  endtask

  always @(posedge RESET) begin
    reset();
  end

  always @(posedge C_DUMP) begin
    // $display("cache_hit: %d, cache_miss1: %d, cache_miss2: %d", cache_hit,
cache_miss1, cache_miss2);
    dump();
  end

  always @(posedge STATS) begin
    $display("cache_hit: %0d, cache_miss: %0d", cache_hit, cache_miss);
  end

  task automatic swap(int idx);
    reg[`CACHE_TAG_SIZE-1:0] first_tag = Cache_Set[idx] % (1 <<
`CACHE_TAG_SIZE);
    Cache_Set[idx] >>= `CACHE_TAG_SIZE;
    Cache_Set[idx] = Cache_Set[idx] + (first_tag << `CACHE_TAG_SIZE);
    h = valid[idx * `CACHE_WAY];
    valid[idx * `CACHE_WAY] = valid[idx * `CACHE_WAY + 1];
    valid[idx * `CACHE_WAY + 1] = h;
    h =  dirty[idx * `CACHE_WAY];
    dirty[idx * `CACHE_WAY] = dirty[idx * `CACHE_WAY + 1];
    dirty[idx * `CACHE_WAY + 1] = h;
    for (int i = 0; i < `CACHE_LINE_SIZE; i++) begin
      reg[7:0] t = Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i];
      Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i] = Cache[idx * `CACHE_WAY
```

```
* `CACHE_LINE_SIZE + i + `CACHE_LINE_SIZE];
      Cache[idx * `CACHE_WAY * `CACHE_LINE_SIZE + i + `CACHE_LINE_SIZE] = t;
    end
  endtask

  always @(posedge CLK && (C1 === 1 || C1 === 2 || C1 === 3 || C1 === 4 || C1
=== 5 || C1 === 6 || C1 === 7)) begin
    if (C1 === 4) begin
      a1 = A1;
      invalid(a1);
      wait_(1);
      C2_ = 0;
      C1_ = 3'b111;
      wait_(1);
      C1_ = 'bz;
    end else begin
      save_c1 = C1;
      a1 = A1;
      save_data1 = D1;
      get_set(a1);
      get_tag(a1);
      get_tags_from_set(set);
      flag = 0;
      for (int i = 0; i < `CACHE_WAY; i++) begin
          if (tags_line[i] === tag && valid[set * `CACHE_WAY + i] === 1 &&
flag == 0) begin
              flag = 1;
              cache_hit++;
              wait_(1);
              save_data2 = D1;
              get_offset(A1);
              C1_ = 0;
              idx = (set * `CACHE_WAY + i) * `CACHE_LINE_SIZE + offset;
              if (save_c1 > 0 && save_c1 < 4) begin
                wait_(5);
                C1_ = `CTR1_BUS_SIZE'b111;
                if (save_c1 === 1) begin
                  D1_ = Cache[idx];
                end else if (save_c1 === 2) begin
                  D1_ = (Cache[idx + 1] << 8) + Cache[idx];
                end else if (save_c1 === 3) begin
                  D1_ = (Cache[idx + 3] << 8) + Cache[idx + 2];
                  wait_(1);
                  D1_ = (Cache[idx + 1] << 8) + Cache[idx];
                end
                if (i == 1) swap(set);
                wait_(1);
                C1_ = 'bz;
                D1_ = 'bz;
              end
              if (save_c1 > 4 && save_c1 < 8) begin
                dirty[set * `CACHE_WAY + i] = 1;
                if (save_c1 === 5) begin
                  Cache[idx] = save_data1 % (1 << 8);
                end else if (save_c1 === 6) begin
                  Cache[idx] = save_data1 % (1 << 8);
                  Cache[idx + 1] = save_data1 >> 8;
                end else if (save_c1 === 7) begin
                  Cache[idx] = save_data2 % (1 << 8);
```

```
                    Cache[idx + 1] = save_data2 >> 8;
                    Cache[idx + 2] = save_data1 % (1 << 8);
                    Cache[idx + 3] = save_data1 >> 8;
                  end
                  if (i == 1) begin swap(set); end
                  wait_(5);
                  C1_ = `CTR1_BUS_SIZE'b111;
                  wait_(1);
                  C1_ = 'bz;
                end
            end
        end
        if (flag == 0) begin
          cache_miss++;
          wait_(1);
          get_offset(A1);
          save_data2 = D1;
          C1_ = 0;
          wait_(3);
          if (dirty[set * `CACHE_WAY + 1] === 1 && valid[set * `CACHE_WAY + 1]
=== 1) begin
            invalid((tags_line[1] << `CACHE_SET_SIZE) + set);
            wait_(1);
          end
          read_mem(a1);
          read_from_mem(a1);
          answer_for_CPU(a1);
        end
      end
    end

    task automatic read_from_mem(int A);
      wait(C2 === 1);
      get_set(A);
      get_tag(A);
      swap(set);
      Cache_Set[set] = ((Cache_Set[set] >> 8) << 8) + tag;
      valid[set * `CACHE_WAY] = 1;
      dirty[set * `CACHE_WAY] = 0;
      for (int i = 7; i >= 0; i--) begin
        Cache[set * `CACHE_LINE_SIZE * `CACHE_WAY + i * 2] = D2 % (1 << 8);
        Cache[set * `CACHE_LINE_SIZE * `CACHE_WAY + i * 2 + 1] = (D2 >> 8);
        wait_(1);
      end
    endtask

    task automatic answer_for_CPU(int A);
      get_set(A);
      get_tag(A);
      idx = set * `CACHE_WAY * `CACHE_LINE_SIZE + offset;
      if (save_c1 === 1) begin
        dirty[set * `CACHE_WAY] = 0;
        C1_ = `CTR1_BUS_SIZE'b111;
        D1_ = Cache[idx];
      end else if (save_c1 === 2) begin
        dirty[set * `CACHE_WAY] = 0;
        C1_ = `CTR1_BUS_SIZE'b111;
        D1_ = (Cache[idx + 1] << 8) + Cache[idx];
      end else if (save_c1 === 3) begin
```

```
      dirty[set * `CACHE_WAY] = 0;
      C1_ = `CTR1_BUS_SIZE'b111;
      D1_ = (Cache[idx + 3] << 8) + Cache[idx + 2];
      wait_(1);
      D1_ = (Cache[idx + 1] << 8) + Cache[idx];
    end
    if (save_c1 === 5) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data1 % (1 << 8);
    end else if (save_c1 === 6) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data1 % (1 << 8);
      Cache[idx + 1] = save_data1 >> 8;
    end else if (save_c1 === 7) begin
      dirty[set * `CACHE_WAY] = 1;
      Cache[idx] = save_data2 % (1 << 8);
      Cache[idx + 1] = save_data2 >> 8;
      Cache[idx + 2] = save_data1 % (1 << 8);
      Cache[idx + 3] = save_data1 >> 8;
    end
    C1_ = `CTR1_BUS_SIZE'b111;
    wait_(1);
    D1_ = 'bz;
    C1_ = 'bz;
  endtask

  task automatic invalid(int A);
    get_set(A);
    get_tag(A);
    get_tags_from_set(set);
    if (tags_line[0] === tag) begin
        if (dirty[set * `CACHE_WAY] == 1 && valid[set * `CACHE_WAY] == 1)
begin write_mem(A, 0); @(posedge C2); end
        valid[set * `CACHE_WAY] = 0;
        swap(set);
    end else if (tags_line[1] === tag) begin
        if (dirty[set * `CACHE_WAY + 1] == 1 && valid[set * `CACHE_WAY + 1]
== 1) begin write_mem(A, 1); @(posedge C2); end
        valid[set * `CACHE_WAY + 1] = 0;
    end
  endtask

  task automatic read_mem(int A);
    A2_out = A;
    C2_ = `CTR2_BUS_SIZE'b10;
    wait_(1);
    A2_out = 'bz;
    C2_ = 'bz;
  endtask

  task automatic write_mem(int A, int tag_in_set);
    get_set(A);
    A2_out = A;
    C2_ = `CTR2_BUS_SIZE'b11;
    pos = (set * `CACHE_WAY + tag_in_set) * `CACHE_LINE_SIZE;
    for (int i = 7; i >= 0; i--) begin
        D2_ = (Cache[pos + i * 2 + 1] << 8) + Cache[pos + i * 2];
        wait_(1);
        A2_out = 'bz;
```

```
    end
    D2_ = 'bz;
    C2_ = 'bz;
  endtask

  task automatic reset();
    for (int i = 0; i < `CACHE_LINE_COUNT; i++) begin
      valid[i] = 0;
    end
  endtask

  task automatic wait_(int time_);
    for (int i = 0; i < time_; i++) begin
      @(posedge CLK);
    end
  endtask

  task automatic dump();
    fd = $fopen("output.txt", "w");
    for (int i = 0; i < `CACHE_LINE_COUNT; i++) begin
      $fdisplay(fd, "%0d", i);
      if (i % 2 == 0) $fdisplay(fd, "%b", (Cache_Set[i/2] % (1 <<
`CACHE_TAG_SIZE)));
      else $fdisplay(fd, "%b", (Cache_Set[i/2] >> `CACHE_TAG_SIZE));
      $fdisplay(fd, "Valid: %b", valid[i]);
      $fdisplay(fd, "Dirty: %b", dirty[i]);
      for (int j = 0; j < `CACHE_LINE_SIZE; j++)
        $fdisplay(fd, Cache[i * `CACHE_LINE_SIZE + j]);
    end
    $fclose(fd);
  endtask
endmodule
```

Листинг 31 – cache.sv

```
`define CACHE_WAY 2
`define CACHE_TAG_SIZE 10
`define CACHE_LINE_SIZE 16
`define CACHE_LINE_COUNT 64
`define MEM_SIZE (1 << 19)
`define CACHE_SIZE (1 << 10)
`define CACHE_SETS_COUNT 32
`define CACHE_SET_SIZE 5
`define CACHE_OFFSET_SIZE 4
`define CACHE_ADDR_SIZE 19
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2
`define M 64
`define N 60
`define K 32

module MemCTR (
  input wire[`ADDR2_BUS_SIZE-1:0] A2,
  inout wire[`DATA2_BUS_SIZE-1:0] D2,
  inout wire[`CTR2_BUS_SIZE-1:0] C2,
```

```
  input wire CLK,
  input wire M_DUMP,
  input wire RESET
  );
  reg[`DATA2_BUS_SIZE-1:0] D2_ = 'bz;
  reg[`CTR2_BUS_SIZE-1:0] C2_ = 'bz;
  assign D2 = D2_;
  assign C2 = C2_;
  reg[7:0] mem[0:`MEM_SIZE-1];
  integer SEED = 225526;
  int fd;
  int a2;

  always @(posedge RESET) begin
    reset();
  end

  always @(posedge M_DUMP) begin
    dump();
  end

  always @(posedge CLK && C2 === 2) begin
    a2 = A2;
    C2_ = 0;
    wait_(100);
    C2_ = 1;
    for (int i = 7; i >= 0; i--) begin
      D2_ = mem[a2 * `CACHE_LINE_SIZE + i * 2] + (mem[a2 * `CACHE_LINE_SIZE +
i * 2 + 1] << 8);
      wait_(1);
    end
    D2_ = 'bz;
    C2_ = 'bz;
  end

  always @(posedge CLK && C2 === 3) begin
    a2 = A2;
    for (int i = 7; i >= 0; i--) begin
      mem[a2 * `CACHE_LINE_SIZE + i * 2] = D2 % (1 << 8);
      mem[a2 * `CACHE_LINE_SIZE + i * 2 + 1] = D2 >> 8;
      wait_(1);
    end
    C2_ = 0;
    wait_(92);
    C2_ = 1;
    wait_(1);
    C2_ = 'bz;
  end

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

  task automatic wait_(int time_);
    for (int i = 0; i < time_; i++) begin
      @(posedge CLK);
    end
```

```
  endtask

  task automatic dump();
    fd = $fopen("output.txt", "w");
    for (int i = 0; i < `MEM_SIZE; i++)
      $fdisplay(fd, "%b", mem[i]);
    $fclose(fd);
  endtask

endmodule
```

Листинг 32 – MemCTR.sv

```
`define CACHE_WAY 2
`define CACHE_TAG_SIZE 10
`define CACHE_LINE_SIZE 16
`define CACHE_LINE_COUNT 64
`define MEM_SIZE (1 << 19)
`define CACHE_SIZE (1 << 10)
`define CACHE_SETS_COUNT 32
`define CACHE_SET_SIZE 5
`define CACHE_OFFSET_SIZE 4
`define CACHE_ADDR_SIZE 19
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15
`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16
`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2
`define M 64
`define N 60
`define K 32

`include "MemCTR.sv"
`include "cache.sv"
`include "CPU.sv"

module test #(parameter _SEED = 225526);
  integer SEED = _SEED;
  byte mem[0:`MEM_SIZE-1];
  integer i = 0;
  initial begin
    for (i = 0; i < `MEM_SIZE; i += 1) begin
      mem[i] = $random(SEED)>>16;
    end
  end
endmodule

module testbench;
  int count = 0;
  reg clk = 0;
  reg c_dum = 0;
  reg m_dum = 0;
  reg res = 0;
  wire STATS;
  wire C_DUM;
  wire M_DUM;
  wire RES;
  wire CLK;
```

```
  wire[`ADDR1_BUS_SIZE-1:0] A1;
  wire[`DATA1_BUS_SIZE-1:0] D1;
  wire[`CTR1_BUS_SIZE-1:0] C1;
  wire[`ADDR2_BUS_SIZE-1:0] A2;
  wire[`DATA2_BUS_SIZE-1:0] D2;
  wire[`CTR2_BUS_SIZE-1:0] C2;
  assign CLK = clk;
  assign C_DUM = c_dum;
  assign M_DUM = m_dum;
  assign RES = res;

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

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

  MemCTR MEM(.A2(A2), .D2(D2), .C2(C2), .CLK(CLK), .M_DUMP(M_DUM),
.RESET(RES));

  always #1 clk = ~clk;

  initial begin
    res = 1;
    #1;
    res = 0;
  end

  always @(posedge CLK) begin
    count++;
  end
endmodule
```

Листинг 33 – testbench.sv