Лабораторная работа №2	M3136	2022
Моделирование схем в Verilog	Тарасевич Артем Сергеевич	

Цель работы: построение кэша и моделирование системы «процессор-кэш-память» на языке аппаратуры Verilog.

Инструментарий: язык Verilog, компиляция и симуляция – Icarus Verilog 11. Для аналитического решения использовался Python 3.10.8.

Формулировка задачи.

Имеется следующее определение глобальных переменных и функций:

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

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

Сложение, инициализация переменных и переход на новую итерацию цикла, выход из функции занимают 1 такт. Умножение – 5 тактов. Обращение к памяти вида pc[x] считается за одну команду.

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

Все локальные переменные лежат в регистрах процессора.

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

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

Устройство кэша.

В задаче используется кэш – устройство, которое выполняет роль «умного» буфера между процессором и памятью, способствующее ускорению работы процессора за счет уменьшения количества обращений к памяти, которые являются дорогими относительно операций процессора (если процессор занимает не больше 1-5 тактов на операцию, то память, в свою очередь, требует не менее 100 тактов).

Существует две политики чтения: look-through и look-aside. Если кэш читает look-through способом, то любой запрос обрабатывается кэшем (и если в кэше нет нужной информации, то кэш посылает запрос памяти). В свою-очередь, при политике look-aside процессор посылает запрос и в кэш, и в память. В условии задания требуется реализовать **look-through** кэш.

Помимо политик чтения, бывают политики записи: write-through и write-back. Первая устроена элементарно – при каждой записи в кэш, данные дублируются в память (данная реализация отличается большим временем работы, так как запись данных используется довольно часто), а вторая пишет при надобности – при вытеснении какой-либо линии, если приходят новые данные. В условии задания требуется реализовать **write-back** кэш.

Любой кэш состоит из кэш-линий – элементарных единиц информации, которые хранят несколько подряд идущих байт памяти.

Также, кэш определяется политикой размещения (cache-placement policy), которые делятся на три типа:

1. Полная ассоциативность (fully associative) – любая кэш-линия может быть на любом месте в кэше.
2. Прямое отображение (direct-mapped cache) – место каждой кэш-линии определено однозначно, но в силу ограниченности размера кэша, несколько линий отображаются в одно место.

3. Ассоциативность по блокам (set-associative) – компромисс между первыми двумя типами: каждая кэш-линия отображается в свое множество (размер которого равен ассоциативности).

Так как по условию задания требуется реализовать кэш с ассоциативностью 2, далее под кэшем будет подразумеваться именно он.

Касательно того, как происходит вытеснение кэш-линии: существует несколько политик вытеснения. В условии требуется реализовать **LRU** политику: если внутри блока необходимо вытеснить какую-либо строку, то вытесняется та, что не использовалась дольше всего.

Исходя из этого, любой адрес байта в памяти можно представить в следующем виде:

Tag (бит)	Set (бит)	Offset (бит)
CACHE_TAG_SIZE	CACHE_SET_SIZE	CACHE_OFFSET_SIZE

Табл. 1 Представление адреса

где Set – номер блока, в котором находится кэш-линия, Offset – нужный байт в кэш-линии, Tag – оставшаяся часть, по которой вместе с set можно однозначно определить кэш-линию.

Каждая кэш-линия хранит следующие данные:

- Valid – бит, отвечающий за состояние кэш-линии: хранится ли в ней «мусор» или настоящие данные
- Dirty – бит, отвечающий за согласованность данных, хранящихся в кэш-линии, и в оперативной памяти.
- Tag – см. представление адреса
- Data – полезные данные

Valid	Dirty	Tag	Data
1 бит	1 бит	CACHE_TAG_SIZE бит	CACHE_LINE_SIZE бит

Табл. 2 Устройство кэш-линии

Исходя из всего вышесказанного, любой кэш определяется следующими характеристиками:

- Ассоциативность – CACHE_WAY – количество кэш-линий в одном блоке
- Размер кэша – CACHE_SIZE

- Размер кэш-линии – CACHE_LINE_SIZE – количество бит полезных данных
- Количество кэш-линий – CACHE_LINE_COUNT
- Размер тэга адреса – CACHE_TAG_SIZE
- Размер индекса блока – CACHE_SET_SIZE
- Размер смещения – CACHE_OFFSET_SIZE
- Размер адреса – CACHE_ADDR_SIZE
- Размерность шин адресов, данных, команд

Вычисление недостающих параметров системы.

Исходные данные:

Параметр	Значение
MEM_SIZE	256 Кбайт = 2^18 байт
CACHE_SIZE	2 Кбайт = 2^11 байт
CACHE_LINE_SIZE	16 байт = 2^4 байт
CACHE_TAG_SIZE	8 бит

Табл. 3 Исходные данные

1. Исходя из того, что в памяти помещается 2^18 байт, то на отдельный байт требуется 18 бит – CACHE_ADDR_SIZE.
2. Из CACHE_SIZE и CACHE_LINE_SIZE можно найти количество кэш-линий в кэше = 2^11 / 2^4 = 2^7 – CACHE_LINE_COUNT.
3. Так как в одной кэш-линии помещается 2^4 байт, то для поиска конкретного байта среди кэш-линии требуется отводить 4 бита – CACHE_OFFSET_SIZE
4. CACHE_TAG_SIZE + CACHE_SET_SIZE + CACHE_OFFSET_SIZE = CACHE_ADDR_SIZE (см. табл. 1), из чего следует CACHE_SET_SIZE = 18 - 4 - 8 = 6 бит
5. Тогда, с помощью 6 бит можно различать 64 = 2^6 блоков – CACHE_SETS_COUNT.
6. Ассоциативность есть количество кэш-линий в одном блоке: CACHE_WAY = CACHE_LINE_COUNT / CACHE_SETS_COUNT = 2.

Касательно шин:

- A1 – на неё требуется уметь передавать в один такт max(CACHE_TAG_SIZE + CACHE_SET_SIZE, CACHE_OFFSET_SIZE) = 14 бит;

- A2 требуется уметь передавать CACHE_TAG_SIZE + CACHE_SET_SIZE = 14 бит;
- Так как по шине C1 передается 8 различных команд, то C1 = log(8) = 3 бит
- Аналогично, C2 = log(4) = 2 бит

Итого,

Параметр	Значение
MEM_SIZE	256 Кбайт = 2^18 байт
CACHE_SIZE	2 Кбайт = 2^11 байт
CACHE_LINE_SIZE	16 байт = 2^4 байт
CACHE_LINE_COUNT	128 = 2^7
CACHE_WAY	2
CACHE_SETS_COUNT	64 = 2^6
CACHE_TAG_SIZE	8 бит
CACHE_SET_SIZE	6 бит
CACHE_OFFSET_SIZE	4 бит
CACHE_ADDR_SIZE	18 бит
A1, A2	14 бит
D1, D2	16 бит
C1	3 бит
C2	2 бит

Табл. 4 Итоговые характеристики кэша

Аналитическое решение задачи.

Рассмотрим принцип общения системы «процессор-кэш-память». При каждом запросе CPU передает адрес и команду (и данные, если запрос – это запись). Далее, кэш ищет эти данные внутри, и если находит, то выполняет необходимую операцию (записывает изменения, либо возвращает данные). Данная ситуация, при которой внутри кэша существует требуемая информация, называется **кэш-попаданием**.

Но если таковой информации не обнаружилось, то случается **кэш-промах**. И в этом случае кэш вначале пытается найти место, куда он будет записывать данные из памяти. Если же такого места не нашлось, то согласно политике LRU вытесняется (= записывается в память) кэш-линия, к которой доступ был самым поздним. После чего в выбранное место

записывается линия из памяти и кэш выполняет над ней необходимую операцию.

Так как мною был выбран способ решать задачу на языке высокого уровня, то было решено пренебречь данными (полезной информацией) и протоколом общения между устройствами (вместо этого будет просто-напросто прибавляться задержка на общение).

Исходно была создана структура CacheLine, которая хранит состояние кэш-линии:

```
class CacheLine:
    Valid = bool; Dirty = bool; Tag = int

    def __init__(self) -> None:
        pass

    def __init__(self, Valid = 0, Dirty = 0, Tag = 0):
        self.Valid = Valid
        self.Dirty = Dirty
        self.Tag = Tag
```

Далее, класс Cache, где хранились линии и массив displace – массив, который определяет, какая из строк использовалась раньше всех (нужен для политики LRU.

```
class Cache:
    lines = list[list[CacheLine]]
    displace = list[bool]
```

Так как запись от чтения идейно никак не отличается (при записи бит Dirty заменяется на 1), то был написан метод readOrWrite. Изначально он ищет нужную кэш-линию:

```
for i in range(constants.CACHE_WAY):
    if self.lines[set][i].Tag == tag and self.lines[set][i].Valid == True:
        HITS += 1
        BEATS += 6 # задержка на попадание
        if dirtyBit:                self.lines[set][i].Dirty = 1
        self.displace[set] = 1 - i
        return
BEATS += 4 # задержка на промах
MISSES += 1
```

Если же он найти не смог, то ищет незанятое место:

```
for i in range(constants.CACHE_WAY):
```

```
    if self.lines[set][i].Valid == False:
        BEATS += 100 + constants.CACHE_LINE_SIZE // 2   # читаем из памяти
        self.lines[set][i] = CacheLine(1, dirtyBit, tag)
        self.displace[set] = 1 - i
        return
```

И если не нашел, то вытесняет другую кэш-линию (если это требуется):

```
if self.lines[set][self.displace[set]].Dirty == 1:
    BEATS += 101                #пишем
BEATS += 100 + constants.CACHE_LINE_SIZE // 2           # читаем из памяти
self.lines[set][self.displace[set]] = CacheLine(1, dirtyBit, tag)
self.displace[set] = 1 - self.displace[set]
return
```

Задержка в 101, а не в 100 обусловлена тем, что на передачу управления от памяти к кэшу требуется дополнительный такт.

Тогда, с учетом структуры, исходный код был написан следующим способом:

```
cache = Cache();

M, N, K = 64, 60, 32

start_array_a = 0
start_array_b = start_array_a + M * K
start_array_c = start_array_b + K * N * 2 # учитываем, что массив b состоял из
int16

pa = start_array_a;  TICKS += 1
pc = start_array_c;  TICKS += 1

TICKS += 1 # y = 0
for y in range(M):
    TICKS += 1 # y < M
    TICKS += 1 # x = 0
    for x in range(N):
        TICKS += 1 # X < N
        pb = start_array_b;     TICKS += 1
        TICKS += 1 # s = 0
        TICKS += 1 # k = 0
        for k in range(K):
            TICKS += 1 # k < K
            cache.readOrWrite(pa + k, False) # C1_READ8
            TICKS += 1 # 1 такт на возврат данных
            cache.readOrWrite(pb + x * 2, False) # C1_READ16
            TICKS += 1 # 1 такт на возврат данных
```

```
            TICKS += 5 + 1 # умножение + сложение
            pb += N * 2;        TICKS += 1
            TICKS += 1 # k++
        cache.readOrWrite(pc + x * 4, True) # C1_WRITE32
        TICKS += 1 # 1 такт на возврат
        TICKS += 1 # x++
    pa += K;      TICKS += 1
    pc += N * 4;  TICKS += 1
    TICKS += 1 # y++

TICKS += 1 # выход из функции
```

Итого, получаем, что

Параметр	Значение
Кэш-попадания	230698
Кэш-промахи	18902
Такты	5009172

Табл. 5 Аналитическое решение

Моделирование заданной системы на Verilog.

Было выделено 5 файлов:

- constants.sv – файл, где хранятся все константы
- memory.sv – обертка памяти и контроллера памяти
- cache.sv – модуль кэша
- cpu.sv – обертка процессора
- testbench.sv – файл, где все 3 модуля связываются друг с другом

Так как по D1, D2 и C1, C2 общение двунаправленное (память может как считывать с этих шин, так и писать в них), то требовалось создать регистры, к которым «подвязывается» шина:

```
memory.sv, cache.sv
reg[16-1:0] D2_buffer; reg[2-1:0] C2_buffer;
assign D2 = D2_buffer; assign C2 = C2_buffer;

cpu.sv, cache.sv
reg[16 - 1:0] D1_buffer; assign D1 = D1_buffer;
reg[3 - 1:0]  C1_buffer; assign C1 = C1_buffer;
```

т.е. решено это было механизмом непрерывного присваивания.

Общение по таким шинам происходит по следующим правилам:

1. Любое чтение должно происходить на моменте clk = 1 (либо posedge clk); любая запись должна быть выполнена при clk = 0 (либо negedge clk)
2. Владельцем шины является тот модуль, который в текущий момент на неё пишет. В отношении «процессор-кэш» шиной преимущественно владеет процессор (он отдает владение только на время выполнения операций кэшем). Аналогично, в отношении «кэш-память» кэш имеет преимущество на владение.
3. После того, как отвечающий отослал все данные, он должен в следующем такте (при clk = 0) передать владение (если требуется).
4. Тот, кто отправляет запрос, должен слушать линию C* и ждать, пока на ней будет значения C*_RESPONSE.

Рассмотрим, как происходит запрос данных от CPU (на примере команды C1_READ8):

1. Процессор пишет в шину при clk = 0 в C1 - 1, в A1 – конкатенацию Tag и Set адреса (см. табл. 1). Кэш на второй половине такта (clk = 1) считывает C1, понимает, что ему пришла команда C1, поэтому считывает только адрес (tag + set).
2. На втором такте процессор продолжает передавать данные: в A1 передает offset (C1 продолжает висеть в положении 1). Кэш, также на второй половине такта считывает оставшиеся данные, готовясь отобрать владение шиной.
3. Далее, при clk = 0 процессор отдает владение шинами C1, D1 (то есть пишет в свои регистры z – высокоимпедансное состояние), а кэш, в этот же момент, забирает владение, начав писать в свой регистр (C1_buffer) значение 0.
4. Потом, кэш ищет необходимую линию (при необходимости, запрашивает данные с памяти). После того, как он это сделал, он пишет при clk = 0 в C1 = C1_RESPONSE = 7, D1 – необходимый байт. В этот же такт, при clk = 1, процессор, увидев на шине C1 ответ, считывает оттуда байт, готовясь перехватить управление шиной C1.
5. Кэш при clk = 0 отдает владение шинами (пишет в них z), а процессор в тот же момент забирает их, присваивая значение регистрам 0.

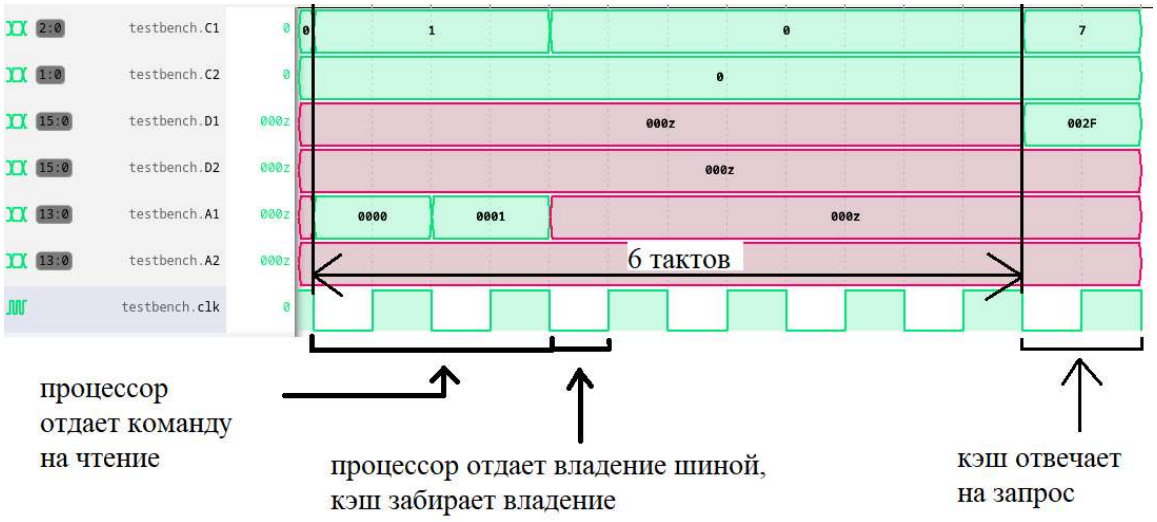

Рис. 1 Кэш-попадание

Для пущей наглядности, чтобы показать кэш-промахи, в следующих двух временных диаграммах время отклика памяти – 10 тактов (в программной реализации 100):

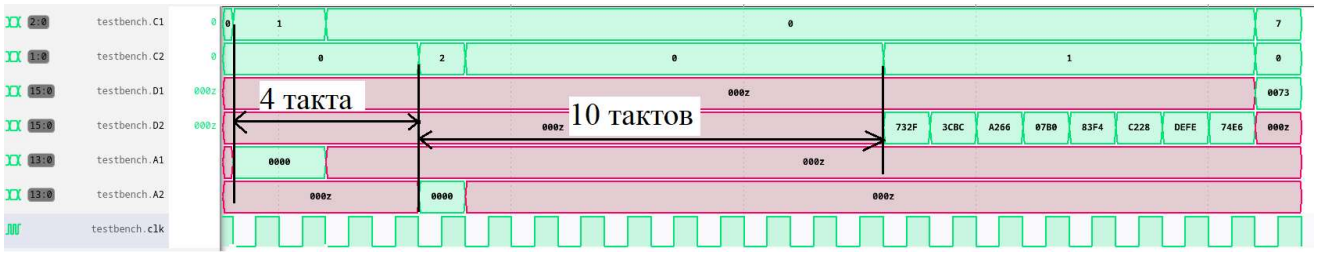

Рис. 2 Кэш-промах без вытеснения

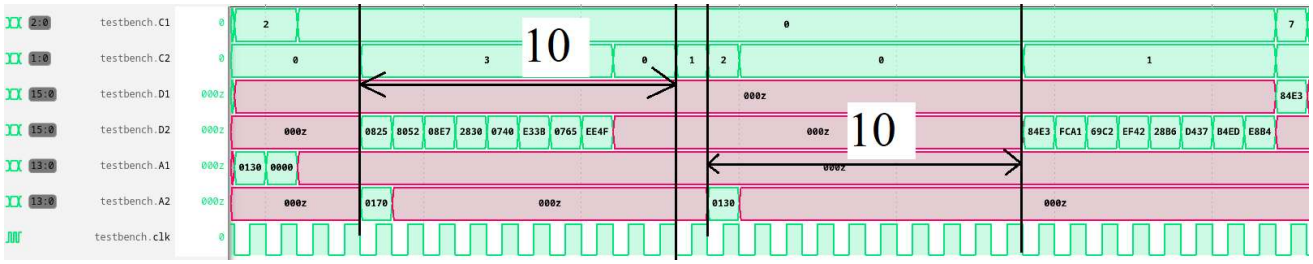

Рис. 3 Кэш-промах с вытеснением

Теперь рассмотрим внутреннее устройство кэша. Каждая кэш-линия представляется в виде 4 массивов:

cache.sv

```
reg Valid[CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
reg Dirty[CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
reg[CACHE_TAG_SIZE-1:0] Tag[CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];
```

```
reg[7:0] Data [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0][CACHE_LINE_SIZE-1:0];
```

Данные в массиве Data нумеруются с младших байт по возрастанию. Тогда, исходя из того, что по протоколу общения байты начинают передаваться с младших, общение по D2 (по D1 аналогично) происходит следующим образом:

memory.sv
```
for (integer i = 0; i < CACHE_LINE_SIZE; i += 2) begin
    D2_buffer = {data[address + i], data[address + i + 1]};
    @(negedge clk);
end
```

cache.sv
```
for (integer j = 0; j < CACHE_LINE_SIZE; j += 2) begin
    Data[set][way][j] = D2 >> 8; Data[set][way][j + 1] = D2 % (1 << 8);
    @(negedge clk);
    if (j == CACHE_LINE_SIZE - 2) begin
        C2_buffer = 0; # перехват управления
    end else @(posedge clk);
end
```

Обращение к данным (включая служебным) конкретной кэш-линии происходит через индекс Set и номер в блоке Way.

Команды, которые кэш умеет выполнять:

- C1_NOP – отсутствие команды
- C1_READ8 – прочитать 1 байт
- C1_READ16 – прочитать 2 байта
- C1_READ32 – прочитать 4 байта
- C1_INVALIDATE_LINE – инвалидировать линию (= выгрузить в память, если линия по данному адресу представлена в кэше)
- C1_WRITE8 – записать 1 байт
- C1_WRITE16 – записать 2 байта
- C1_WRITE32 – записать 4 байта

Так как все методы (за исключением C1_INVALIDATE_LINE) требуют найти кэш-линию, чтобы с ней делать операции, то решено выделить поиск кэш-линии в отдельный таск find_cache_line. Этот метод найдет\запишет линию в блок с индексом Set и присвоит в переменную way место в этом блоке, где оказалась линия. Сравнение на строчку выглядит следующим образом:

cache.sv task find_cache_line()

```
if (Valid[set][__way] == 1 && Tag[set][__way] == tag && cache_hit == 0) ...
```

Если же нужной линии найдено не будет, пойдет вначале поиск пустого места:

cache.sv task find_cache_line()

```
    for (integer __way = 0; __way < CACHE_WAY; __way += 1) begin
        if (Valid[set][__way] == 0 && found_empty == 0) begin
            way = __way;
            ...
        end
    end
```

Если же и пустой линии не было найдено, то вытесняется линия с номером Displace[set]. Displace – массив, который хранит для каждого набора номер линии, который будет вытеснен.

cache.sv task find_cache_line()

```
if (found_empty == 0) begin
    way = Displacing[set];
    ...
end
```

И тогда, каждый вызов команды занимается следующим:

1. Считать входные данные
2. Вызвать find_cache_line()
3. Обработать нужные данные

Только C1_INVALIDATE_LINE не вызывает find_cache_line (в силу того, что find_cache_line загружает нужную линию, даже если её не было). Вместо этого он бежит по нужному блоку и ищет кэш-линию, при необходимости её вытесняя:

cache.sv task C1_INVALIDATE_LINE()

```
for (integer __way = 0; __way < CACHE_WAY; ++__way) begin
    if (Valid[set][__way] == 1 && Tag[set][__way] == tag && cache_hit == 0)
    begin
        cache_hit = 1; way = __way;
        if (Dirty[set][way] == 1) begin
            C2_WRITELINE(); // вытесняем при надобности
        end
    end
end
```

Также, необходимо было доопределить время отклика команды C1_INVALIDATE_LINE – 6 тактов, при наличии данной строки в кэше, и 4 такта, если данной строки не было найдено.

Также были введены таски DUMP(), который полностью выводит состояние модуля в файл, и RESET(), который сбрасывает состояние модуля до исходного. Данные таски есть как в кэше, так и в модуле памяти.

По итогу, кэш понимает, какую команду ему нужно было вызвать следующим образом:

cache.sv

```
always @(posedge clk) begin
            if (C_DUMP == 1)       begin DUMP();                  end
            if (RESET == 1)        begin RESET_TASK();            end
            if (C1 == 0)           begin ;                end // C1_NOP
       else if (C1 == 1)           begin C1_READ8();              end
       else if (C1 == 2)           begin C1_READ16();             end
       else if (C1 == 3)           begin C1_READ32();             end
       else if (C1 == 4)           begin C1_INVALIDATE_LINE();    end
       else if (C1 == 5)           begin C1_WRITE8();             end
       else if (C1 == 6)           begin C1_WRITE16();            end
       else if (C1 == 7)           begin C1_WRITE32();            end
   end
```

Как можно понять, все команды были выделены в таски. В модуле памяти устроено аналогично.

Сам модуль памяти – это объявленный массив регистров байт, способный принимать команды C2_READ_LINE и C2_WRITE_LINE.

Воспроизведение задачи на Verilog.

Исходная задача была воспроизведена в модуле CPU аналогично аналитическому решению. Были введены указатели на начала массивов (с сохранением информации о весе типа):

```
localparam M = 64;
localparam N = 60;
localparam K = 32;

integer a = 0;
integer b = a + M * K;
integer c = b + K * N * 2;
```

Так как в каждом такте clk успевает принимать значения 0 и 1, то такты считаются следующим способом:

```
    always @(negedge clk)   ticks += 1;
```

И соответственно, переход на следующий такт сопровождается командой `@(negedge clk)`.

Обращу внимание, что переход на новую итерацию цикла и сравнение if я считаю за один такт, а также, что инициализация внутренних итераторов каждый раз занимает такт.

cpu.sv

```
        pa = a; @(negedge clk);
        pc = c; @(negedge clk);
        @(negedge clk);  // integer y = 0
        for (integer y = 0; y < M; y += 1) begin
            @(negedge clk); // y < M
            @(negedge clk);// integer x = 0
            for (integer x = 0; x < N; x += 1) begin
                @(negedge clk); // x < N
                pb = b; @(negedge clk);
                s = 0;  @(negedge clk);
                @(negedge clk); // integer k = 0;
                for (integer k = 0; k < K; k += 1) begin
                    @(negedge clk); // k < K
                    address = pa + k; C1_READ8();
                    pa_k = buffer[0];
                    address = pb + x * 2; C1_READ16();
                    pb_x = {buffer[0], buffer[1]};
                    repeat (6) @(negedge clk);  s += pa_k * pb_x;
                    pb += N * 2; @(negedge clk);
                    @(negedge clk);  // k++
                end
                buffer[0] = (s >> 24); buffer[1] = (s >> 16) % (1 << 8);
buffer[2] = (s >> 8) % (1 << 8); buffer[3] = s % (1 << 8);
                address = pc + (x * 4); C1_WRITE32();
                @(negedge clk);     // x++
            end
            @(negedge clk); pa += K;
            @(negedge clk); pc += N * 4;
            @(negedge clk);  // y++
        end

        @(negedge clk); // выход из функции

        @(posedge clk) $display("HITS: %d MISSES: %d TOTAL: %d TICKS: %d",
cache.HITS, cache.MISSES, cache.HITS + cache.MISSES, ticks);

        $finish();
```

Заметим, что такт заканчивается только на posedge clk, потому необходимо выводить результат только при clk = 1.

Итого, ответ, который выводит программа:

Параметр	Значение
Кэш-попадания	230698
Кэш-промахи	18902
Такты	5009172

Табл. 6 Решение на Verilog

Сравнение полученных результатов

Результаты аналитического и смоделированного на Verilog решений совпали. Значит система построена верно.

Параметр	Значение (Python)	Значение (Verilog)
Кэш-попадания	230698	230698
Кэш-промахи	18902	18902
Такты	5009172	5009172

Листинг кода

cache.py

```python
import enum

HITS, MISSES, TICKS = 0, 0, 0

# 3 var
class constants(enum.IntEnum):
    MEM_SIZE = 2 ** 18
    CACHE_SIZE = 2 ** 11
    CACHE_LINE_SIZE = 2 ** 4
    CACHE_LINE_COUNT = 2 ** 7
    CACHE_WAY = 2
    CACHE_SETS_COUNT = 2 ** 6
    CACHE_TAG_SIZE = 8
    CACHE_SET_SIZE = 6
    CACHE_OFFSET_SIZE = 4
    CACHE_ADDR_SIZE = 18

class CacheLine:
    Valid = bool; Dirty = bool; Tag = int

    def __init__(self) -> None:
        pass
```

```
    def __init__(self, Valid = 0, Dirty = 0, Tag = 0):
        self.Valid = Valid
        self.Dirty = Dirty
        self.Tag = Tag

class Cache:
    lines = list[list[CacheLine]]
    displace = list[bool]

    def __init__(self) -> None:
        self.lines = [[CacheLine() for i in range(constants.CACHE_WAY)] for
j in range(constants.CACHE_SETS_COUNT)]
        self.displace = [0 for i in range(constants.CACHE_SETS_COUNT)]

    def readOrWrite(self, address, dirtyBit):
        global HITS, MISSES, TICKS
        tag = getTag(address)
        set = getSet(address)

        for i in range(constants.CACHE_WAY):
            if self.lines[set][i].Tag == tag and self.lines[set][i].Valid ==
True: # CACHE HIT
                HITS += 1
                TICKS += 6 # CACHE_HIT_TIME
                if dirtyBit:                self.lines[set][i].Dirty = 1
                self.displace[set] = 1 - i
                return

        TICKS += 4  # CACHE_MISS_TIME
        MISSES += 1
        for i in range(constants.CACHE_WAY):
            if self.lines[set][i].Valid == False:
                TICKS += 100 + constants.CACHE_LINE_SIZE // 2   #
C2_READLINE
                self.lines[set][i] = CacheLine(1, dirtyBit, tag)
                self.displace[set] = 1 - i
                return

        if self.lines[set][self.displace[set]].Dirty == 1: # LRU-politics
            TICKS += 101 #write to memory
        TICKS += 100 + constants.CACHE_LINE_SIZE // 2           #read from
memory
        self.lines[set][self.displace[set]] = CacheLine(1, dirtyBit, tag)
        self.displace[set] = 1 - self.displace[set]
        return

def getTag(address):
```

```
    return address >> (constants.CACHE_OFFSET_SIZE +
constants.CACHE_SET_SIZE)

def getSet(address):
    return (address >> (constants.CACHE_OFFSET_SIZE)) % (1 <<
constants.CACHE_SET_SIZE)

def getOffset(address):
    return address % (1 << constants.CACHE_OFFSET_SIZE)

# ====MAIN==== #
cache = Cache();

M, N, K = 64, 60, 32

start_array_a = 0
start_array_b = start_array_a + M * K
start_array_c = start_array_b + K * N * 2 # учитываем, что массив b состоял
из int16

pa = start_array_a;  TICKS += 1
pc = start_array_c;  TICKS += 1

TICKS += 1 # y = 0
for y in range(M):
    TICKS += 1 # y < M
    TICKS += 1 # x = 0
    for x in range(N):
        TICKS += 1 # X < N
        pb = start_array_b;     TICKS += 1
        TICKS += 1 # s = 0
        TICKS += 1 # k = 0
        for k in range(K):
            TICKS += 1 # k < K
            cache.readOrWrite(pa + k, False) # C1_READ8
            TICKS += 1 # 1 такт на возврат данных
            cache.readOrWrite(pb + x * 2, False) # C1_READ16
            TICKS += 1 # 1 такт на возврат данных
            TICKS += 5 + 1 # умножение + сложение
            pb += N * 2;        TICKS += 1
            TICKS += 1 # k++
        cache.readOrWrite(pc + x * 4, True) # C1_WRITE32
        TICKS += 1 # 1 такт на возврат
        TICKS += 1 # x++
    pa += K;      TICKS += 1
    pc += N * 4;  TICKS += 1
    TICKS += 1 # y++
```

```
TICKS += 1 # выход из функции
# ====MAIN==== #

print(f"HITS: {HITS} MISSES: {MISSES} TOTAL: {HITS + MISSES} TICKS:
{TICKS}")
```

constants.sv

```
`ifndef CONSTANTS_H
`define CONSTANTS_H
parameter MEM_SIZE = (1 << 18);
parameter CACHE_SIZE = (1 << 11);
parameter CACHE_LINE_SIZE = (1 << 4);
parameter CACHE_LINE_COUNT = (1 << 7);
parameter CACHE_WAY = (2);
parameter CACHE_SETS_COUNT = (1 << 6);
parameter CACHE_TAG_SIZE = (8);
parameter CACHE_SET_SIZE = (6);
parameter CACHE_OFFSET_SIZE = (4);
parameter CACHE_ADDR_SIZE = (18);
parameter _SEED = (225526);

parameter MEM_ACCESS_SPEED_READ = 100;
parameter MEM_ACCESS_SPEED_WRITE = 100 - CACHE_LINE_SIZE / 2;  // учитываем
parameter CACHE_HIT_TIME   = 6;
parameter CACHE_MISS_TIME  = 4;
`endif
```

memory.sv

```
`ifndef MEMORY_H
`define MEMORY_H

`include "constants.sv"

module Memory(
    inout wire [16-1:0] D2,
    inout wire [2-1:0] C2,
    input wire [CACHE_TAG_SIZE + CACHE_SET_SIZE-1:0] A2,
    input wire clk,
    input wire RESET,
    input wire M_DUMP
);

reg[8-1:0] data[MEM_SIZE-1:0]; // непосредственно память

// =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
reg[16-1:0] D2_buffer; reg[2-1:0] C2_buffer;
assign D2 = D2_buffer; assign C2 = C2_buffer;
```

```
// =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======

initial begin
    C2_buffer = 'z; D2_buffer = 'z;
    RESET_TASK();
end

// =======МЕТОДЫ ДЛЯ ОБЩЕНИЯ=========
reg[CACHE_TAG_SIZE + CACHE_OFFSET_SIZE-1:0] address;
task C2_READLINE();
    address = A2;
    @(negedge clk); C2_buffer = 0; // забираем управление
    repeat (MEM_ACCESS_SPEED_READ - 1) @(negedge clk); // вычитаем один
такт, так как потратили уже один такт на перехват управления

    C2_buffer = 1;  // C2_RESPONSE
    for (integer i = 0; i < CACHE_LINE_SIZE; i += 2) begin
        D2_buffer = {data[address + i], data[address + i + 1]};
        @(negedge clk);
    end
    D2_buffer = 'z; C2_buffer = 'z; // передаем управление
endtask

task C2_WRITELINE();
    address = A2;
    for (integer i = 0; i < CACHE_LINE_SIZE; i += 2) begin
        data[address + i] = (D2 >> 8);
        data[address + i + 1] = D2 % (1 << 8);
        @(negedge clk);
        if (i == CACHE_LINE_SIZE - 2) begin // перехватываем управление
            C2_buffer = 0;
        end
        @(posedge clk);
    end
    repeat (MEM_ACCESS_SPEED_WRITE) begin @(negedge clk); end
    C2_buffer = 1; // C2_RESPONSE
    @(negedge clk)   C2_buffer = 'z; // передаем управление
endtask
// =======МЕТОДЫ ДЛЯ ОБЩЕНИЯ=========

integer SEED;
task RESET_TASK();
    SEED = _SEED;
    for (int i = 0; i < MEM_SIZE; i++) begin
        data[i] = $random(SEED)>>16;
    end
endtask
```

```
always @(posedge clk) begin
         if (M_DUMP == 1)   begin DUMP();          end
         if (RESET == 1)    begin RESET_TASK();    end
         if (C2 == 0)       begin      ;           end
    else if (C2 == 2)       begin C2_READLINE();   end
    else if (C2 == 3)       begin C2_WRITELINE();  end
end

integer file;
task DUMP();
    file = $fopen("memory_dump.txt", "w");
    for (integer cur_line = 0; cur_line < MEM_SIZE / CACHE_LINE_SIZE;
cur_line += 1) begin
        $fwrite(file, "ADDR: %x || ", (cur_line << CACHE_OFFSET_SIZE));
        for (integer cur_byte = 0; cur_byte < CACHE_LINE_SIZE; cur_byte +=
1) begin
            $fwrite(file, "%x", data[(cur_line << CACHE_OFFSET_SIZE) +
cur_byte]);
        end
        $fdisplay(file);
    end
    $fclose(file);
endtask

endmodule
`endif
```

cache.sv

```
`include "constants.sv"
`include "memory.sv"

module Cache(
    input wire[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : 0] A1,
    inout wire[16 - 1 : 0] D1,
    inout wire[3 - 1 : 0] C1,
    output reg[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : 0] A2,
    inout wire[16 - 1 : 0] D2,
    inout wire[2 - 1 : 0] C2,

    input wire clk,
    input wire RESET,
    input wire C_DUMP
);

    // =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
    reg[16 - 1:0] D1_buffer; assign D1 = D1_buffer;
```

```
    reg[3 - 1:0]  C1_buffer; assign C1 = C1_buffer;
    reg[16 - 1:0] D2_buffer; assign D2 = D2_buffer;
    reg[2 - 1:0]  C2_buffer; assign C2 = C2_buffer;
    // =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
    // =======МЕТОДЫ ДЛЯ ОБЩЕНИЯ С ПАМЯТЬЮ=======
    task C2_READLINE();
        if (clk == 1)       @(negedge clk);
        C2_buffer = 2;
        A2 = {tag, set}; // ??????????????????????
        D2_buffer = 'z;

        @(negedge clk) C2_buffer = 'z; A2 = 'z;
        do
            @(posedge clk);
        while (C2 != 1);
        // дождались ответа

        // $display("set %x way %x", set, way);

        for (integer j = 0; j < CACHE_LINE_SIZE; j += 2) begin
            Data[set][way][j] = D2 >> 8; Data[set][way][j + 1] = D2 % (1 <<
8);
            @(negedge clk);
            if (j == CACHE_LINE_SIZE - 2) begin
                C2_buffer = 0;
            end else @(posedge clk);
        end
    endtask

    task C2_WRITELINE();
        if (clk == 1) @(negedge clk);
        C2_buffer = 3;
        A2 = {Tag[set][way], set};
        for (integer j = 0; j < CACHE_LINE_SIZE; j += 2) begin
            D2_buffer = {Data[set][way][j], Data[set][way][j + 1]};
            @(negedge clk);
            A2 = 'z;
        end
        D2_buffer = 'z; C2_buffer = 'z;
        do
            @(posedge clk);
        while (C2 != 1);
        // мы дождались ответа
       @(negedge clk)   C2_buffer = 0; // забираем управление
    endtask
    // =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
    // =======РЕГИСТРЫ, НУЖНЫЕ ДЛЯ ПОИСКА ЛИНИИ В КЭШЕ=========
```

```
    reg[CACHE_TAG_SIZE - 1 : 0] tag; reg[CACHE_SET_SIZE - 1 : 0] set;
reg[CACHE_OFFSET_SIZE - 1 : 0] offset; reg[2 - 1 : 0] way;
    reg[8 - 1: 0]   buffer[4 - 1:0];

    reg cache_hit, found_empty;
    // =======РЕГИСТРЫ, НУЖНЫЕ ДЛЯ ПОИСКА ЛИНИИ В КЭШЕ=========
    // ===========МЕТОДЫ ДЛЯ ОБЩЕНИЯ С ПРОЦЕССОРОМ=============

    // ищет линию с поданными tag и set
    // если найти линию не сможет, то будет считывать из памяти (вытесняя
при необходимости)
    // по итогу присваивает в регистр way место в блоке, где лежит
запрашиваемая линия
    task find_cache_line();
        cache_hit = 0;

        for (integer __way = 0; __way < CACHE_WAY; ++__way) begin
            if (Valid[set][__way] == 1 && Tag[set][__way] == tag &&
cache_hit == 0) begin
                cache_hit = 1;
                HITS += 1;

                way = __way;
                Displacing[set] = 1 - way;
                repeat (CACHE_HIT_TIME - 2) @(negedge clk); // было исходно
потрачено 2 такта на
                // передачу адресов\данных\команды => здесь их вычитаем
            end
        end

        if (cache_hit == 0) begin
            found_empty = 0;
            MISSES += 1;
            repeat (CACHE_MISS_TIME - 2) @(negedge clk);

            for (integer __way = 0; __way < CACHE_WAY; __way += 1) begin
                if (Valid[set][__way] == 0 && found_empty == 0) begin
                    found_empty = 1;

                    way = __way;
                    Valid[set][way] = 1;
                    Tag[set][way] = tag;
                    Dirty[set][way] = 0;
                    Displacing[set] = 1 - way;
                    C2_READLINE();
                end
            end
```

```
            if (found_empty == 0) begin
                way = Displacing[set];
                if (Dirty[set][way] == 1) begin
                    C2_WRITELINE();
                end
                C2_READLINE();
                Displacing[set] = 1 - way;
                Valid[set][way] = 1;
                Tag[set][way] = tag;
                Dirty[set][way] = 0;
            end
        end
    endtask

    task C1_READ8();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        find_cache_line();
        C1_buffer = 7;
        D1_buffer = Data[set][way][offset];
        @(negedge clk)  C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task C1_READ16();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        find_cache_line();
        C1_buffer = 7;
        D1_buffer = {Data[set][way][offset], Data[set][way][offset + 1]};
        @(negedge clk)  C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task C1_READ32();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        find_cache_line();
        C1_buffer = 7;
        D1_buffer = {Data[set][way][offset], Data[set][way][offset + 1]};
```

```
        @(negedge clk)  D1_buffer = {Data[set][way][offset + 2],
Data[set][way][offset + 3]};
        @(negedge clk)  C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task C1_WRITE8();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        buffer[0] = D1 % (1 << 8);

        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        find_cache_line();
        Dirty[set][way] = 1;

        Data[set][way][offset] = buffer[0];

        C1_buffer = 7;
        @(negedge clk) C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task C1_WRITE16();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        buffer[0] = D1 >> 8; buffer[1] = D1 % (1 << 8);

        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        find_cache_line();
        Dirty[set][way] = 1;

        Data[set][way][offset] = buffer[0]; Data[set][way][offset + 1] =
buffer[1];

        C1_buffer = 7;
        @(negedge clk) C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task C1_WRITE32();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        buffer[0] = D1 >> 8; buffer[1] = D1 % (1 << 8);
        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
                        buffer[2] = D1 >> 8; buffer[3] = D1 % (1 << 8);
        @(negedge clk)  C1_buffer = 0;
```

```
        find_cache_line();

        Dirty[set][way] = 1;

        Data[set][way][offset] = buffer[0]; Data[set][way][offset + 1] =
buffer[1];
        Data[set][way][offset + 2] = buffer[2]; Data[set][way][offset + 3] =
buffer[3];

        C1_buffer = 7;
        @(negedge clk) C1_buffer = 'z; D1_buffer = 'z;
    endtask

    task RESET_TASK();
        for (integer i = 0; i < CACHE_SETS_COUNT; i += 1) begin
            Displacing         [i]    = 0;
            for (integer j = 0; j < CACHE_WAY; j += 1) begin
                Valid          [i][j] = 0;
                Dirty          [i][j] = 0;
                Tag            [i][j] = 0;
                for (integer k = 0; k < CACHE_LINE_SIZE; k += 1)
                    Data[i][j][k] = 0;
            end
        end
    endtask

    task C1_INVALIDATE_LINE();
        tag = A1 >> CACHE_SET_SIZE;
        set = A1 % (1 << CACHE_SET_SIZE);
        @(posedge clk)  offset = A1 % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  C1_buffer = 0;

        cache_hit = 0;
        for (integer __way = 0; __way < CACHE_WAY; ++__way) begin
            if (Valid[set][__way] == 1 && Tag[set][__way] == tag &&
cache_hit == 0) begin
                cache_hit = 1; way = __way;
                if (Dirty[set][way] == 1) begin
                    C2_WRITELINE(); // вытесняем при надобности
                end
                repeat (CACHE_HIT_TIME) @(negedge clk);
            end
        end

        if (cache_hit == 0) begin
            repeat (CACHE_MISS_TIME) @(negedge clk);
        end
```

```
        Valid[set][way] = 0;
        C1_buffer = 7;
        @(negedge clk) C1_buffer = 'z;
    endtask

    // ===========МЕТОДЫ ДЛЯ ОБЩЕНИЯ С ПРОЦЕССОРОМ=============
    // ==================КЭШ ЛИНИИ============================
    reg                         Valid       [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0];
    reg                         Dirty       [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0];
    reg[CACHE_TAG_SIZE-1:0]     Tag         [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0];
    reg                         Displacing  [CACHE_SETS_COUNT-1:0];
    reg[7:0]                    Data        [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0][CACHE_LINE_SIZE-1:0];
    // ==================КЭШ ЛИНИИ============================
    //
    integer HITS, MISSES, TOTAL_REQUESTS;

    initial begin
        HITS = 0; MISSES = 0; TOTAL_REQUESTS = 0;
        D1_buffer = 'z; C1_buffer = 'z; D2_buffer = 'z;
        A2 = 'z;    C2_buffer = 0; // изначально кэш владеет C2

        buffer[0] = 0; buffer[1] = 0; buffer[2] = 0; buffer[3] = 0;
        RESET_TASK();
    end

    always @(posedge clk) begin
             if (C_DUMP == 1)       begin DUMP();                end
             if (RESET == 1)        begin RESET_TASK();          end
             if (C1 == 0)           begin ;                      end //
C1_NOP
        else if (C1 == 1)           begin C1_READ8();            end
        else if (C1 == 2)           begin C1_READ16();           end
        else if (C1 == 3)           begin C1_READ32();           end
        else if (C1 == 4)           begin C1_INVALIDATE_LINE();  end
        else if (C1 == 5)           begin C1_WRITE8();           end
        else if (C1 == 6)           begin C1_WRITE16();          end
        else if (C1 == 7)           begin C1_WRITE32();          end
    end

    integer file;
    task DUMP();
        file = $fopen("cache_dump.txt", "w");
        for (integer set_ind = 0; set_ind < CACHE_SETS_COUNT; set_ind += 1)
begin
```

```
            for (integer set_way = 0; set_way < CACHE_WAY; set_way += 1)
begin
                $fwrite(file, "SET %x WAY %x || Valid %1x Dirty %1x Tag %x
Data ",
                                    set_ind, set_way, Valid[set_ind][set_way],
Dirty[set_ind][set_way],
                                    Tag[set_ind][set_way]);
                for (integer cur_byte = 0; cur_byte < CACHE_LINE_SIZE;
cur_byte += 1) begin
                    $fwrite(file, "%x", Data[set_ind][set_way][cur_byte]);
                end
                $fdisplay(file);
            end
        end
        $fclose(file);
    endtask
endmodule
```

cpu.sv

```
`include "constants.sv"

module CPU(
    output reg[CACHE_TAG_SIZE + CACHE_SET_SIZE - 1: 0] A1,
    inout wire[16-1:0] D1,
    inout wire[3-1:0] C1,
    input wire clk,

    output reg C_DUMP,
    output reg M_DUMP,
    output reg RESET
);

    // =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
    reg[16-1:0]   D1_buffer; assign D1 = D1_buffer;
    reg[3 - 1:0]    C1_buffer; assign C1 = C1_buffer;
    // =======РЕГИСТРЫ ДЛЯ ОБЩЕНИЯ=======
    // =======МЕТОДЫ ДЛЯ ОБЩЕНИЯ=========
    reg[8-1:0] buffer[CACHE_LINE_SIZE - 1:0];
    reg[CACHE_ADDR_SIZE - 1:0] address;

    task waitToWrite();
        if (clk == 1)   @(negedge clk);
    endtask

    task C1_READ8();
        waitToWrite();
```

```
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 1;
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        buffer[0] = D1 % (1 << 8);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_READ16();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 2;
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        buffer[0] = D1 >> 8; buffer[1] = D1 % (1 << 8);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_READ32();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 3;
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        buffer[0] = D1 >> 8; buffer[1] = D1 % (1 << 8);
        @(posedge clk)  buffer[2] = D1 >> 8; buffer[3] = D1 % (1 << 8);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_WRITE8();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 5;
        D1_buffer = buffer[0];
```

```
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z; D1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_WRITE16();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 6;
        D1_buffer = {buffer[0], buffer[1]};
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z; D1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_WRITE32();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 7;
        D1_buffer = {buffer[0], buffer[1]};
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        D1_buffer = {buffer[2], buffer[3]};
        @(negedge clk)  A1 = 'z; C1_buffer = 'z; D1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);

        @(negedge clk)  C1_buffer = 0;
    endtask

    task C1_INVALIDATE_LINE();
        waitToWrite();
        A1 = address >> CACHE_OFFSET_SIZE; C1_buffer = 1;
        @(negedge clk)  A1 = address % (1 << CACHE_OFFSET_SIZE);
        @(negedge clk)  A1 = 'z; C1_buffer = 'z;

        do
            @(posedge clk);
        while (C1 != 7);
```

```
    @(negedge clk)  C1_buffer = 0;
endtask
// =======МЕТОДЫ ДЛЯ ОБЩЕНИЯ=========
// ======================
localparam M = 64;
localparam N = 60;
localparam K = 32;

integer a = 0; // указатели на начала массивов
integer b = a + M * K;
integer c = b + K * N * 2;

integer pa; integer pa_k;
integer pb; integer pb_x;
integer pc; integer pc_x;

integer s;

integer ticks;

always @(negedge clk)   ticks += 1;

initial begin
    C1_buffer = 0; D1_buffer = 'z; A1 = 'z; ticks = 0;
    // ============================MAIN================================
    pa = a; @(negedge clk);
    pc = c; @(negedge clk);
    @(negedge clk);  // integer y = 0
    for (integer y = 0; y < M; y += 1) begin
        @(negedge clk); // y < M
        @(negedge clk);// integer x = 0
        for (integer x = 0; x < N; x += 1) begin
            @(negedge clk); // x < N
            pb = b; @(negedge clk);
            s = 0;  @(negedge clk);
            @(negedge clk); // integer k = 0;
            for (integer k = 0; k < K; k += 1) begin
                @(negedge clk); // k < K
                address = pa + k; C1_READ8();
                pa_k = buffer[0];
                address = pb + x * 2; C1_READ16();
                pb_x = {buffer[0], buffer[1]};
                repeat (6) @(negedge clk);  s += pa_k * pb_x;
                pb += N * 2; @(negedge clk);
                @(negedge clk);  // k++
            end
```

```
                buffer[0] = (s >> 24); buffer[1] = (s >> 16) % (1 << 8);
buffer[2] = (s >> 8) % (1 << 8); buffer[3] = s % (1 << 8);
                address = pc + (x * 4); C1_WRITE32();
                @(negedge clk);     // x++
            end
            @(negedge clk); pa += K;
            @(negedge clk); pc += N * 4;
            @(negedge clk);  // y++
        end

        @(negedge clk); // выход из функции

        @(posedge clk) $display("HITS: %d MISSES: %d TOTAL: %d TICKS: %d",
cache.HITS, cache.MISSES, cache.HITS + cache.MISSES, ticks);

        $finish();
        // ============================MAIN================================
    end

endmodule
```

testbench.sv

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

module testbench();
    wire [16-1:0] D2;
    wire [2-1:0] C2;
    wire [CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : 0]  A2;

    wire [16-1:0] D1;
    wire [3-1:0] C1;
    reg [CACHE_TAG_SIZE + CACHE_SET_SIZE - 1 : 0]  A1;

    reg clk; wire RESET; wire M_DUMP; wire C_DUMP;

    Memory mem(D2, C2, A2, clk, RESET, M_DUMP);
    Cache cache(A1, D1, C1, A2, D2, C2, clk, RESET, C_DUMP);
    CPU cpu(A1, D1, C1, clk, C_DUMP, M_DUMP, RESET);

    always #1 clk = ~clk;

    initial begin
        clk = 0;
    end
endmodule
```