Лабораторная Работа №2	M3100	2022
Моделирование Схем в Verilog	Ерёмин Владимир Ильич	

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

Инструментарий и требования к работе: весь код пишется на языках Verilog и C++, компиляция и симуляция – Icarus Verilog 11, Clang 15.

Описание: реализовать систему «Процессор-Кэш-Память» и решить предложенную задачу сначала аналитически *(на языке высокого уровня)*, а затем используя язык описания аппаратуры – Verilog.

Задача: определить процент попаданий (число попаданий к общему числу обращений) для кэша и общее время (в тактах), затраченное на выполнение заданной функции.

Вариант 3: Основные параметры системы и дополнительно параметры, заданные вариантом. Схема моделируемой системы. Протокол обмена данными и задержки модулей.

Размер кэша	2 Кбайт
Размер кэш-линии	16 байта
Кол-во бит под тэг адреса	8 бита
Размер памяти	256 Кбайт

Таблица 1 – Базовые параметры системы.

Вычисление параметров системы.

Исходя из общих и заданных вариантом параметров требуется вывести остальные, чтобы иметь возможность реализовать систему.

MEM_SIZE	256 Кб
CACHE_SIZE	2 Кб
CACHE_LINE_SIZE	16 байт
CACHE_LINE_COUNT	128
CACHE_WAY	2
CACHE_SETS_COUNT	64
CACHE_TAG_SIZE	8 бит
CACHE_SET_SIZE	6 бит
CACHE_OFFSET_SIZE	4 бит
CACHE_ADDR_SIZE	18 бит
ADDR1_BUS_SIZE	14 бит
ADDR2_BUS_SIZE	14 бит
DATA1_BUS_SIZE	16 бит
DATA2_BUS_SIZE	16 бит
CTR1_BUS_SIZE	3 бит
CTR2_BUS_SIZE	2 бит

Таблица 2 – Финальные параметры системы.

(Отмечены начальные параметры)

CACHE_LINE_COUNT: Отношение общего размера кэша к размеру одной линии:

$$CACHE\_LINE\_COUNT = \frac{CACHE\_SIZE \cdot 2^{10}}{CACHE\_LINE\_SIZE}$$

(конвертируем Кб в байты)

CACHE_WAY: В одном канале содержится **CACHE_SETS_COUNT** линий, следовательно:

$$CACHE\_WAY = \frac{CACHE\_LINE\_COUNT}{CACHE\_SETS\_COUNT}$$

CACHE_SETS_COUNT: Исходя из количества бит, выделенных под set в адресе, можно получить их количество:

$$\text{CACHE\_SETS\_COUNT} = 2^{CACHE\_SET\_SIZE}$$

CACHE_SET_SIZE: Зная количество бит, выделенных под tag и offset в адресе, можно узнать сколько отводится set: *(ТЗ - Рисунок 3)*

$$\text{CACHE\_SET\_SIZE} = CACHE\_ADDR\_SIZE - CACHE\_TAG\_SIZE - \\ -CACHE\_OFFSET\_SIZE$$

CACHE_OFFSET_SIZE: offset отвечает за выбор байта на линии, следовательно значение ограничено длиной линии. Задает кол-во бит под offset в адресе:

$$CACHE\_OFFSET\_SIZE = \log_2 CACHE\_LINE\_SIZE$$

CACHE_ADDR_SIZE: Состоит из последовательно записанных tag, set, offset. Сам по себе является указателем на ячейку в памяти:

$$CACHE\_ADDR\_SIZE = \log_2(MEM\_SIZE \cdot 2^{13})$$

(не забываем перевести в биты)

ADDR1_BUS_SIZE: за 1 такт может передаваться либо tag+set, либо offset. Очевидно:

$$ADDR1\_BUS\_SIZE = \max(CACHE\_TAG\_SIZE + \\ +CACHE\_SET\_SIZE, CACHE\_OFFSET\_SIZE)$$

ADDR2_BUS_SIZE: здесь только tag+set:

$$ADDR2\_BUS\_SIZE = CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE$$

CTR1_BUS_SIZE: по данной шине передается 9 различных команд, 2-е из которых кодируются одинакого.

$$CTR1\_BUS\_SIZE = \log_2 8$$

CTR2_BUS_SIZE: по данной – 4 команды:

$$CTR2\_BUS\_SIZE = \log_2 4$$

Принцип работы наборно-ассоциативного кэша.

Прежде, чем писать какую-либо реализацию нужно понять его устройство...

Начнем с адреса, по которому процессор запрашивает данные у кэша, а тот у памяти. Именно так и работает кэш **look-through** - все запросы на чтение будут осуществляется непосредственно через кэш, который будет записывать в себя то, что требуется процессору от памяти. Симметрично, при **write-back** – запись будет вестись в кэш, а только за тем данные будут возвращены обратно в память.

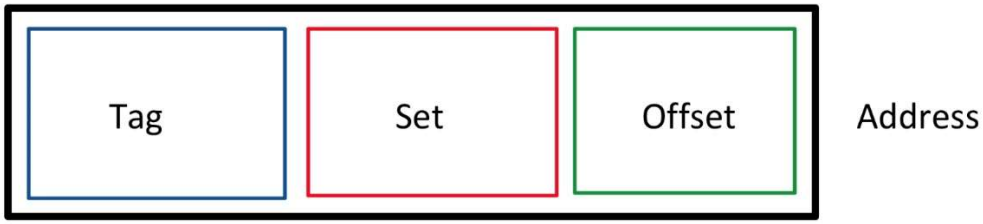

Рисунок 1 – Устройство адреса

Наш адрес можно «поделить» на 3 компоненты. Заметим, что заменив offset на (0) в адресе, мы получим адрес первой ячейки соответствующей кэш-линии состоящей в неком set нашего кэша, с конкретным tag.

Теперь разберемся, что такое set и tag.

Рассмотрим устройство кэша на примере иллюстрации:

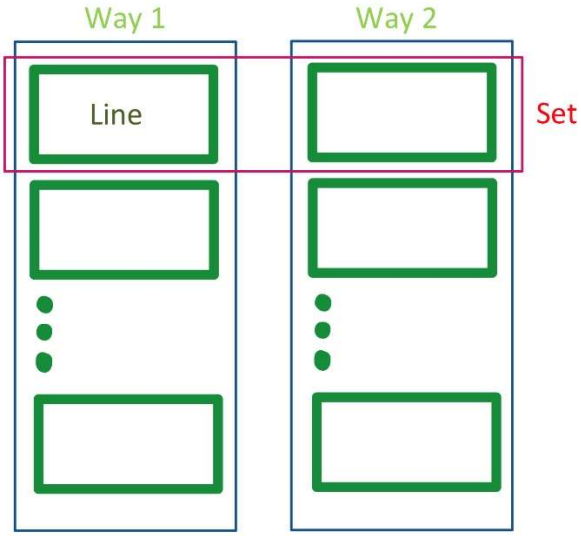

Рисунок 2 – Устройство Наборно-Ассоциативного Кэша

Здесь каждый way является кэшем прямого отображения. Как мы разобрались, зная адрес можно однозначно определить линию в way то, есть set. Однако в каком именно way находится линия мы сказывать сразу не можем… Имея возможность хранить 2 линии с одинаковым set, процесс получения линии чуть усложнился. Но решение довольно простое.

Разберемся с устройством линии:

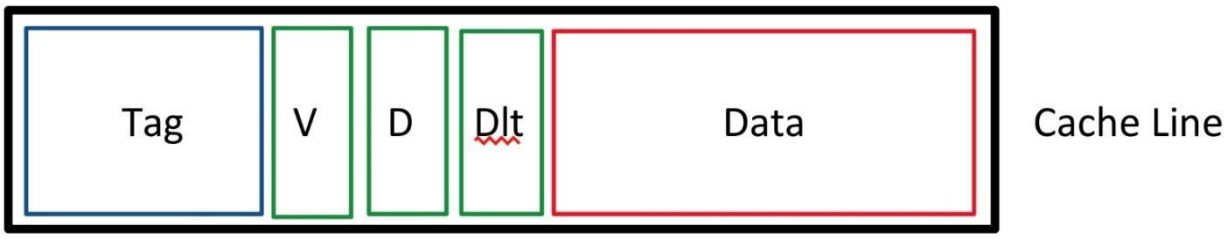

Рисунок 3 – Устройство Кэш-Линии

По мимо «полезных» данных будем хранить еще дополнительную информацию такую, как:

Tag – тогда мы однозначно можем разрешить адрес на линии set некого way, с offset по data.

V (valid) – Подтверждает валидность данной линии, если 1 – то, эта линия соответствует какому-то адресу памяти.

D (dirty) – Показывает статус линии, как объекта записи. Если 1 – то, эта линия была изменена и считается более не соответствующей находящимся в памяти данным.

Dlt (delta) – Отображает востребованность линии по сравнению с остальными в этом-же *set.*

Имея данную информацию в линии, можно пройтись по каждому way, сравнить *tag* линии *set* с искомым. Если процесс удачный – **Кэш-Попадание,** иначе – **Кэш-Промах.**

Очевидно, что при кэш-попадании мы можем просто вернуть требуемые данные.

В нашем устройстве кэша при промахе, по идее нужно просто запросить у памяти нужный нам участок, записать в кэш и вернуть, как при попадании.

Но встает вопрос куда именно записывать новую линию в множество линий *set.* Здесь бы будем применять так называемую – «политику замещения» *(в нашем случае lru).*

Для начала проверим, а есть ли в *set* не валидная линия, если такая есть, то мы можем просто записать новые данные туда. В противном случае, используем бит *Dlt* из нашей линии, чтобы определить самую не востребованную из всех. И заместить её новой запрашиваемой линией.

Получается при обращении к конкретной линии, нам нужно поставить у нее *Dlt* в 0, а у другой 1 *(двух-ассоциативный случай).*

Сам процесс замещения должен помимо обновления линии, вернуть заменимую обратно в память в том случае, если в нее происходили изменения *(D = 1).* Иначе, мы гарантируем ее соответствие данным в памяти.

Аналитическое решение.

В качестве аналитического решения, про эмулируем работу кэша для конкретной задачи:

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

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. Дополнительно: Время отклика:

6 тактов – время, через которое в результате кэш попадания, кэш начинает отвечать.

4 такта – время, через которое в результате кэш промаха, кэш посылает запрос к памяти.

MemCTR – 100 тактов.

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).

Уточнения:

+= - 1 такт.

Само решение с задачей:

Файл в репозитории: ***mockup.cpp***

```
#include <algorithm>
#include <cmath>
#include <cstdint>
#include <cstddef>
#include <array>
#include <ctime>
#include <iostream>
#include <cstring>

using std::array;

// Глобальные переменные для подсчета тактов, попаданий, промахи.
uint64_t clocks = 0;
uint32_t hits = 0;
uint32_t misses = 0;

// Структура реализующая кэш.
// Базовые константы взяты из параметров системы.
struct Cache {
    static constexpr size_t CACHE_SETS_COUNT = 64;
    static constexpr size_t CACHE_LINE_SIZE = 16;
    static constexpr uint32_t CACHE_SET_SIZE = 6;
    static constexpr uint32_t CACHE_OFFSET_SIZE = 4;

    // Структура задающая кэш-линию.
    // Содержит tag, биты V, D, Dlt.
    // Информация - "Мнимая".
    struct Line {
        uint32_t tag;
```

```
30.
31.         bool dirty = false;
32.         bool valid = false;
33.
34.         // Data.
35.
36.         uint32_t delta = 1;
37.     };
38.
39.     // way описывается, как статический массив линий.
40.     using Way = array<Line, CACHE_SETS_COUNT>;
41.
42.     // В данной реализции way разделенны в отдельные переменные.
43.     Way way1{};
44.     Way way2{};
45.
46.     // LRU возвращает итератор на замещаймую линию.
47.     Way::iterator lru(uint32_t set) {
48.         if (way1[set].valid && way2[set].valid) {
49.             return way1[set].delta > way2[set].delta ? way1.begin() + set
   : way2.begin() + set;
50.         }
51.
52.         return way1[set].valid ? way2.begin() + set : way1.begin() + set;
53.     }
54.
55.     // write_back
56.     // Пишет линию обратно в память в соответсвии с описанием.
57.     void write_back(Way::iterator it) {
58.         if (!it->valid || !it->dirty) {
59.             return;
60.         }
61.
62.         // Do move.
63.         ++clocks; // Address pass.
64.
65.         clocks += 100; // Memory await.
66.
67.         // линия более не валидна.
68.         it->valid = false;
69.         it->dirty = false;
70.         it->delta = 1;
71.
72.         clocks += 8; // Transmitting.
73.     }
```

```
74.
75.     // Чтение линии из памяти.
76.     Way::iterator read_line(uint32_t addr) {
77.         uint32_t set = addr_set(addr);
78.         uint32_t tag = addr_tag(addr);
79.
80.         Way::iterator it = lru(set);
81.
82.         write_back(it);
83.
84.         // Ставить Dlt для двух в 1, т.к новая линия будет
    инициализированна с 0.
85.
86.         way1[set].delta = 1;
87.         way2[set].delta = 1;
88.
89.         // запись в кэш.
90.         it->tag = tag;
91.         it->delta = 0;
92.         it->valid = true;
93.         it->dirty = false;
94.
95.         // Do read.
96.
97.         ++clocks; // Address pass.
98.         clocks += 100; // Memory await.
99.         clocks += 8; //fetching.
100.
101.          return it;
102.     }
103.
104.     // Поиск линии в set, если не найдет вернет конец второго way.
105.     // Иначе итератор на нужную линию.
106.     Way::iterator find(uint32_t tag, uint32_t set) {
107.         if (way1[set].valid && way1[set].tag == tag) {
108.             // Обновляем Dlt.
109.             if (way2[set].valid) {
110.                 way1[set].delta = 0;
111.                 way2[set].delta = 1;
112.             }
113.             ++hits;
114.
115.             clocks += 6; // Cache hit await.
116.
117.             return way1.begin() + set;
```

```
118.         }
119.         else if (way2[set].valid && way2[set].tag == tag) {
120.             // Обновляем Dlt.
121.             if (way1[set].valid) {
122.                 way2[set].delta = 0;
123.                 way1[set].delta = 1;
124.             }
125.             ++hits;
126.
127.             clocks += 6; // Cache hit await.
128.
129.             return way2.begin() + set;
130.         }
131.         else {
132.             ++misses;
133.
134.             clocks += 4; // Cache miss await.
135.
136.             return way2.end();
137.         }
138.     }
139.
140.     // Функции разбиения адреса.
141.
142.     // Fetch Tag from address.
143.     uint32_t addr_tag(uint32_t addr) {
144.         return (addr >> CACHE_OFFSET_SIZE >> CACHE_SET_SIZE);
145.     }
146.
147.     // Fetch Set from address.
148.     uint32_t addr_set(uint32_t addr) {
149.         return (addr >> CACHE_OFFSET_SIZE & ((1 << CACHE_SET_SIZE) -
    1));
150.     }
151.
152.     // Fetch Offset from address.
153.     uint32_t addr_offset(uint32_t addr) {
154.         return (addr & ((1 << CACHE_OFFSET_SIZE) - 1));
155.     }
156.
157.     // Чтение 8 бит из кэша, проделываем описанные в предыдущем пункте
    операции.
158.     int8_t read_int8(uint32_t addr) {
159.         clocks += 2; // Address receive.
160.
```

```
161.         uint32_t set = addr_set(addr);
162.         uint32_t tag = addr_tag(addr);
163.
164.         // Поиск линии.
165.         Way::iterator it = find(tag, set);
166.
167.         // Если не найдена - читаем из памяти.
168.         if (it == way2.end()) {
169.             read_line(addr);
170.         }
171.
172.         ++clocks; // Response.
173.
174.         return 0;
175.     }
176.
177.     // Аналогично для 16 бит.
178.     int16_t read_int16(uint32_t addr) {
179.         clocks += 2; // Address receive.
180.
181.         uint32_t set = addr_set(addr);
182.         uint32_t tag = addr_tag(addr);
183.
184.         Way::iterator it = find(tag, set);
185.
186.         if (it == way2.end()) {
187.             read_line(addr);
188.         }
189.
190.         ++clocks; // Response.
191.
192.         return 0;
193.     }
194.
195.     // Аналогично, так-как данные "Мнимые" и букально только запрос
    данных.
196.     void write_int32(uint32_t addr, int32_t value) {
197.         clocks += 2; // Address receive.
198.
199.         uint32_t set = addr_set(addr);
200.         uint32_t tag = addr_tag(addr);
201.
202.         Way::iterator it = find(tag, set);
203.
204.         if (it == way2.end()) {
```

```
205.             it = read_line(addr);
206.         }
207.
208.         clocks++; // Responce;
209.
210.         it->dirty = true;
211.     }
212.
213.     // Выписываем все изменения из кэша обратно в память.
214.     void goodbye() {
215.         for (auto it = way1.begin(); it != way1.end(); ++it) {
216.             write_back(it);
217.         }
218.
219.         for (auto it = way2.begin(); it != way2.end(); ++it) {
220.             write_back(it);
221.         }
222.     }
223. };
224.
225. // Экземпляр кэша.
226. Cache cache;
227.
228. // Константы задания.
229. constexpr int32_t M = 64;
230. constexpr int32_t N = 60;
231. constexpr int32_t K = 32;
232.
233. // Заменены базовые опрерации, чтобы встроить счетчик тактов.
234.
235. // Initializer for uint32 with embeded clock tick.
236. uint32_t init_uint32(uint32_t value) {
237.     ++clocks;
238.     return value;
239. }
240.
241. // Initializer for int32 with embeded clock tick.
242. int32_t init_int32(int32_t value) {
243.     ++clocks;
244.     return value;
245. }
246.
247. // Default addition with embeded clock tick.
248. void addition(uint32_t& value, const uint32_t add) {
249.     ++clocks;
```

```
250.     value += add;
251. }
252.
253. // Default signed addition with embeded clock tick.
254. void addition(int32_t& value, const int32_t add) {
255.     ++clocks;
256.     value += add;
257. }
258.
259. // Default signed multiplication embeded clock tick.
260. int32_t multiplication(const int8_t mul_1, const int16_t mul_2) {
261.     clocks += 5;
262.     return mul_1 * mul_2;
263. }
264.
265. // Just pass the tick for next iteration of function exit.
266. void next() {
267.     ++clocks;
268. }
269.
270. // Все опреции заменины на соответствующие со встроенным тактовым
   счетом.
271.
272. void mmul() {
273.     uint32_t pa_addr = init_uint32(0); // a begin.
274.
275.     uint32_t pc_addr = init_uint32(5888); // M * K + 2 bytes * K * N
276.
277.     for (int y = init_int32(0); y < M; addition(y, 1)) {
278.
279.         for (int x = init_int32(0); x < N; addition(x, 1)) {
280.
281.             uint32_t pb_addr = init_uint32(2048); // M * K
282.             int32_t s = init_int32(0);
283.
284.             for (int k = init_int32(0); k < K; addition(k, 1)) {
285.
286.                 addition(
287.                     s,
288.                     multiplication(
289.                         cache.read_int8(pa_addr + k),
290.                         cache.read_int16(pb_addr + x * 2)
291.                     )
292.                 );
293.                 addition(pb_addr, N * 2);
```

```
294.             }
295.             cache.write_int32(pc_addr + x * 4, s);
296.         }
297.         addition(pa_addr, K);
298.         addition(pc_addr, N * 4);
299.     }
300.     next();
301. }
302.
303. int main() {
304.
305.     mmul();
306.
307.     cache.goodbye();
308.
309.     std::cout << "Clocks: " << clocks << '\n';
310.     std::cout << "Hits: " << hits << " | " << static_cast<double>(hits)
   / static_cast<double>(hits + misses) << '\n';
311.     std::cout << "Misses: " << misses << std::endl;
312.
313.     return 0;
314. }
```

В итоге получаем:

```
$ clang++ -g mockup.cpp -o mockup
$ ./mockup
Clocks: 5411454
Hits: 230698 | 0.924271
Misses: 18902
```

Итого**: 92.427%** кэш попаданий за **5411454** такта.

Моделирование заданной системы на Verilog.

Исходя из принципов работы кэша будем проектировать его на Verilog.

Для более ясного понимания приведена схема:

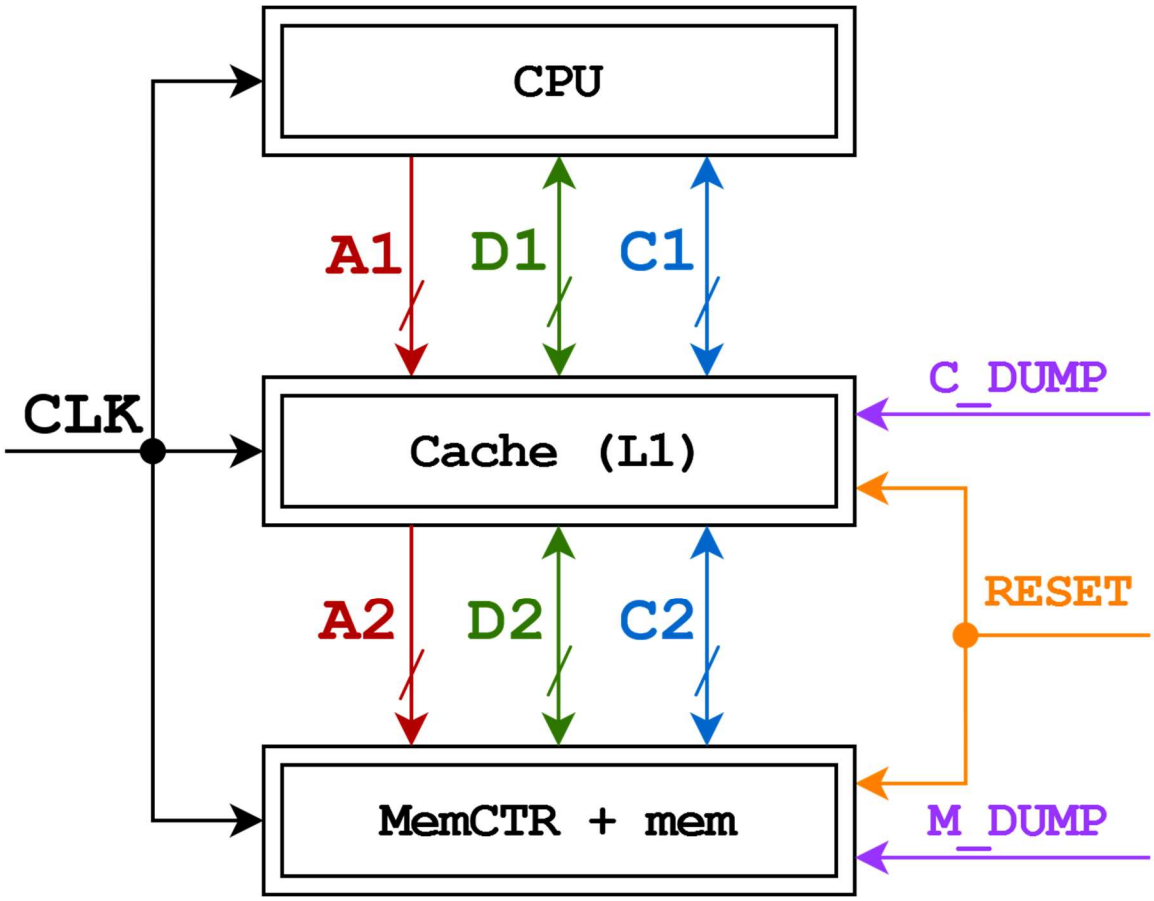

Рисунок 4 – Схема моделируемой системы

Буквально требуется 3 модуля: **cpu, cache** и **mem**. Добавим еще общий модуль testbench, который будет переключать такты, а также задаст эти три модуля, базовые регистры и провода, соединяющие все это.

К базовым регистрам будут относится **clock** для подсчета тактов, **c_dump**, **m_dump** для логирования кэша и памяти соответственно, а также **rst** *(RESET на схеме)*.

По мимо этого, в Verilog код выполняется не так, как в том же C++. Здесь логика строится на «обновлениях и задержках». А именно, нам нужно

придумать протокол взаимодействия модулей такой, который не будет создавать конфликтов.

Одно из замечаний возникает, в шинах вида **inout,** на такие шины писать одновременно с двух сторон нельзя. Чтобы не случалось конфликтов нужно, чтобы, с одной стороны, на регистре было высокоимпедансное состояние *('z)*.

Также не понятна ситуация считывания и записи в один момент времени. Поэтому договоримся считывать на повышение **posedge** такта, а писать на **negedge.** А если требуется написать *(допустим обновить команду на шине)* мы можем легко сдвинуться на полтакта с помощью *#1*, затем, конечно, стоит вернуться в обратно на соответствующий такт.

Логику задания зададим напрямую в **cpu** в блоке **initial,** в общем решение будет схоже с аналитическим, но с дополнениями такими, как:

Удовлетворение протоколу обмена данными.

Инициализацией памяти.

Воспроизведение задачи на Verilog.

Файл: testbench.sv

```
1. `include "memory.sv"
2. `include "cache.sv"
3. `include "cpu.sv"
4.
5. module testbench;
6.
7.     localparam int Addr1BusSize = 14;
8.     localparam int Data1BusSize = 16;
9.     localparam int Ctr1BusSize = 3;
10.
11.      localparam int Addr2BusSize = 14;
12.      localparam int Data2BusSize = 16;
13.      localparam int Ctr2BusSize = 2;
14.
15.      tri [Addr1BusSize-1:0] a1;
16.      tri [Data1BusSize-1:0] d1;
17.      tri [Ctr1BusSize-1:0] c1;
```

```
18.
19.     tri [Addr2BusSize-1:0] a2;
20.     tri [Data2BusSize-1:0] d2;
21.     tri [Ctr2BusSize-1:0] c2;
22.
23.     logic clk;
24.     logic rst;
25.     logic [31:0] cnt = 0;
26.
27.     tri [31:0] misses;
28.     tri [31:0] hits;
29.
30.     cpu cpu_instance(.a1(a1), .d1(d1), .c1(c1), .clk(clk),
   .cnt(cnt), .hits(hits), .misses(misses));
31.
32.     cache cache_instance(
33.         .a1(a1),
34.         .d1(d1),
35.         .c1(c1),
36.         .a2(a2),
37.         .d2(d2),
38.         .c2(c2),
39.         .clk(clk),
40.         .rst(rst),
41.         .hits(hits),
42.         .misses(misses)
43.     );
44.
45.    memory memory_instance(
46.         .a2(a2),
47.         .d2(d2),
48.         .c2(c2),
49.         .clk(clk),
50.         .rst(rst)
51.     );
52.     always #1 begin
53.         clk = ~clk;
54.
55.         cnt++;
56.     end
57. endmodule
```

Файл: cpu.sv

```
1. module cpu(a1, d1, c1, clk, cnt, hits, misses);
2.     localparam int CacheAddrSize = 18;
3.
4.     localparam int Addr1BusSize = 14;
5.     localparam int Data1BusSize = 16;
6.     localparam int Ctr1BusSize = 3;
7.
8.     typedef logic [CacheAddrSize-1:0] address;
9.     typedef bit [31:0] int32;
10.      typedef bit [15:0] int16;
11.      typedef bit [7:0] int8;
12.
13.      output [Addr1BusSize-1:0] a1;
14.      inout [Data1BusSize-1:0] d1;
15.      inout [Ctr1BusSize-1:0] c1;
16.
17.      address addr_tmp;
18.      logic [Data1BusSize-1:0] data_tmp;
19.      logic [Ctr1BusSize-1:0] ctrl_tmp;
20.
21.      assign a1 = addr_tmp;
22.      assign d1 = data_tmp;
23.      assign c1 = ctrl_tmp;
24.
25.      input clk;
26.
27.      input [31:0] cnt;
28.
29.      input [31:0] hits;
30.      input [31:0] misses;
31.
32.      address pa;
33.      address pc;
34.      address pb;
35.      int32 s;
36.
37.      int8 read1;
38.      int16 read2;
39.
40.      logic part;
41.
42.      initial begin
43.          localparam int M = 64;
44.          localparam int N = 60;
```

```
45.         localparam int K = 32;
46.
47.         #1 pa = 0;
48.         #1 pc = 5888;
49.         #1 for (int32 y = 0; y < M; y++) begin
50.             #1 for (int32 x = 0; x < N; x++) begin
51.                 #1 pb = 2048;
52.                 #1 s = 0;
53.                 #1 for (int32 k = 0; k < K; k++) begin
54.                     read_int8(pa + k, read1);
55.                     read_int16(pb + x * 2, read2);
56.                     #6 s += read1 * read2;
57.                     #1 pb += N * 2;
58.                     #1;
59.                 end
60.                 write_int32(pc + x * 4, s);
61.                 #1;
62.             end
63.             #1 pa += K;
64.             #1 pc += N * 4;
65.             #1;
66.         end
67.         #1;
68.
69.         $display("Clk: %d", cnt);
70.
71.         $display("Hits: %d", hits);
72.         $display("Misses: %d", misses);
73.         $display("Efficiency: %d", hits / (hits + misses));
74.         $finish;
75.     end
76.
77.     task static read_int8(input address addr, output int8 data);
78.         ctrl_tmp = 3'b001;
79.
80.         addr_tmp = addr[CacheAddrSize-1:CacheAddrSize-
   Addr1BusSize];
81.         #1 addr_tmp = addr[CacheAddrSize-Addr1BusSize - 1:0];
82.
83.         ctrl_tmp = 'z;
84.
85.         while (ctrl_tmp != 3'b011) begin
86.             #1;
87.         end
88.
```

```
89.          data =  data_tmp[Data1BusSize-1:Data1BusSize-8];
90.      endtask
91.
92.      task static read_int16(input address addr, output int16 data);
93.          ctrl_tmp = 3'b010;
94.
95.          addr_tmp = addr[CacheAddrSize-1:CacheAddrSize-
   Addr1BusSize];
96.          #1 addr_tmp = addr[CacheAddrSize-Addr1BusSize - 1:0];
97.
98.          ctrl_tmp = 'z;
99.
100.         while (ctrl_tmp != 3'b011) begin
101.             #1;
102.         end
103.
104.         data = data_tmp;
105.     endtask
106.
107.     task static write_int32(input address addr, int32 value);
108.         ctrl_tmp = 3'b111;
109.
110.         addr_tmp = addr[CacheAddrSize-1:CacheAddrSize-
   Addr1BusSize];
111.         #1 addr_tmp = addr[CacheAddrSize-Addr1BusSize - 1:0];
112.
113.         ctrl_tmp = 'z;
114.         part = 0;
115.
116.         while (ctrl_tmp != 3'b111) begin
117.             case (part)
118.                 1: data_tmp = value[Data1BusSize-1:0];
119.                 default: data_tmp = value[32-1:32-Data1BusSize];
120.             endcase
121.             #1 part = ~part;
122.         end
123.
124.         data_tmp = 'z;
125.     endtask
126. endmodule
```

Файл: cache.sv

```
1. module cache(a1, d1, c1, a2, d2, c2, clk, rst, c_dump, hits,
   misses);
2.     localparam int Addr1BusSize = 14;
3.     localparam int Data1BusSize = 16;
4.     localparam int Ctr1BusSize = 3;
5.
6.     localparam int Addr2BusSize = 14;
7.     localparam int Data2BusSize = 16;
8.     localparam int Ctr2BusSize = 2;
9.
10.      localparam int CacheSetsCount = 64;
11.      localparam int CacheLineSize = 128;
12.      localparam int CacheSetSize = 6;
13.      localparam int CacheOffsetSize = 4;
14.      localparam int AddressSize = 18;
15.
16.      localparam int CacheTagSize = 8;
17.
18.      localparam int ActualLineSize = CacheTagSize + 3 +
   CacheLineSize;
19.      localparam int ValidBit = ActualLineSize - CacheTagSize - 2;
20.      localparam int DirtyBit = ActualLineSize - CacheTagSize - 3;
21.      localparam int DeltaBit = ActualLineSize - CacheTagSize - 4;
22.      localparam int DataBegin = ActualLineSize - CacheTagSize - 5;
23.
24.      //typedef logic [ActualLineSize-1:0] line;
25.      //typedef line [CacheSetsCount] way;
26.      typedef logic [AddressSize-1:0] address;
27.
28.      input [Addr1BusSize-1:0] a1;
29.      inout [Data1BusSize-1:0] d1;
30.      inout [Ctr1BusSize-1:0] c1;
31.
32.      logic [Addr1BusSize-1:0] addr1_tmp;
33.      logic [Data1BusSize-1:0] data1_tmp;
34.      logic [Ctr1BusSize-1:0] ctrl1_tmp;
35.
36.      assign a1 = addr1_tmp;
37.      assign d1 = data1_tmp;
38.      assign c1 = ctrl1_tmp;
39.
40.      output [Addr2BusSize-1:0] a2;
41.      inout [Data2BusSize-1:0] d2;
42.      inout [Ctr2BusSize-1:0] c2;
```

```
43.
44.     logic [Addr2BusSize-1:0] addr2_tmp;
45.     logic [Data2BusSize-1:0] data2_tmp;
46.     logic [Ctr2BusSize-1:0] ctrl2_tmp;
47.
48.     assign a2 = addr2_tmp;
49.     assign d2 = data2_tmp;
50.     assign c2 = ctrl2_tmp;
51.
52.     input clk;
53.     input rst;
54.     input c_dump;
55.
56.     output [31:0] hits;
57.     output [31:0] misses;
58.
59.     int hits_tmp = 0;
60.     int miss_tmp = 0;
61.
62.     assign hits = hits_tmp;
63.     assign misses = miss_tmp;
64.
65.     logic [ActualLineSize-1:0] way1 [CacheSetsCount];
66.     logic [ActualLineSize-1:0] way2 [CacheSetsCount];
67.
68.     int i;
69.
70.     initial begin
71.         hits_tmp = 0;
72.         miss_tmp = 0;
73.         ctrl1_tmp = 'z;
74.         data1_tmp = 'z;
75.         ctrl2_tmp = 2'b00;
76.         for (i = 0; i < CacheSetsCount; i++) begin
77.             way1[i] = 0;
78.             way2[i] = 0;
79.         end
80.     end
81.
82.     always @(posedge clk) begin
83.         if (rst) begin
84.             hits_tmp = 0;
85.             miss_tmp = 0;
86.             ctrl1_tmp = 'z;
87.             data1_tmp = 'z;
```

```
88.             ctrl2_tmp = 2'b00;
89.             for (i = 0; i < CacheSetsCount; i++) begin
90.                 way1[i] = 0;
91.                 way2[i] = 0;
92.             end
93.         end
94.         if (c_dump) begin
95.             for (int i = 0; i < CacheSetsCount; i++) begin
96.                 $display(way1[i]);
97.                 $display(way2[i]);
98.             end
99.         end
100.    end
101.
102.    logic [Addr1BusSize-1:0] tag_set;
103.    logic [CacheOffsetSize-1:0] offset;
104.
105.    int way_to;
106.
107.    logic [CacheSetSize-1:0] set_to;
108.
109.    always @(posedge clk) begin
110.        case (c1)
111.        3'b001: begin
112.            tag_set = a1;
113.            set_to = tag_set[CacheSetSize:0];
114.            # 2 offset = a1;
115.            # 1 ctrl1_tmp = 3'b000;
116.            find(tag_set, way_to);
117.
118.            if (way_to == 0) begin
119.                read_line(addr_tag_set(tag_set), way_to);
120.            end
121.
122.            ctrl1_tmp = 3'b111;
123.
124.            if (way_to == 1) begin
125.                for (int i = DataBegin-offset*8-1; i >= DataBegin-
  offset*8-8; i--) begin
126.                data1_tmp[DataBegin-offset*8-1 - i]
127.                    = way1[set_to][i];
128.                end
129.            end
130.            else begin
```

```
131.                for (int i = DataBegin-offset*8-1; i >= DataBegin-
   offset*8-8; i--) begin
132.                data1_tmp[DataBegin-offset*8-1 - i]
133.                    = way2[set_to][i];
134.                end
135.            end
136.
137.            #2
138.
139.            ctrl1_tmp = 'z;
140.            data1_tmp = 'z;
141.        end
142.        3'b010: begin
143.            tag_set = a1;
144.            set_to = tag_set[CacheSetSize:0];
145.            # 2 offset = a1;
146.            # 1 ctrl1_tmp = 3'b000;
147.            find(tag_set, way_to);
148.
149.            if (way_to == 0) begin
150.                read_line(addr_tag_set(tag_set), way_to);
151.            end
152.
153.            ctrl1_tmp = 3'b111;
154.
155.            if (way_to == 1) begin
156.                for (int i = DataBegin-offset*8-1; i >= DataBegin-
   offset*8-16; i--) begin
157.                data1_tmp[DataBegin-offset*8-1-i]
158.                    = way1[set_to][i];
159.                end
160.            end
161.            else begin
162.                for (int i = DataBegin-offset*8-1; i >= DataBegin-
   offset*8-16; i--) begin
163.                data1_tmp[DataBegin-offset*8-1-i]
164.                    = way2[set_to][i];
165.                end
166.            end
167.
168.
169.            #2
170.
171.            ctrl1_tmp = 'z;
172.            data1_tmp = 'z;
```

```
173.         end
174.         3'b111: begin
175.             tag_set = a1;
176.             set_to = tag_set[CacheSetSize:0];
177.             # 2 offset = a1;
178.             find(tag_set, way_to);
179.
180.             if (way_to == 0) begin
181.                 #1 read_line(addr_tag_set(tag_set), way_to);
182.                 #1;
183.             end
184.
185.             if (way_to == 1) begin
186.                 for (int i = DataBegin-offset*8-1; i >= DataBegin-
    offset*8-16; i--) begin
187.                     way1[set_to][i] = data1_tmp[DataBegin-offset*8-
    1-i];
188.                 end
189.                 #2;
190.                 for (int i = DataBegin-offset*8-17; i >= DataBegin-
    offset*8-32; i--) begin
191.                     way1[set_to][i] = data1_tmp[DataBegin-offset*8-
    17-i];
192.                 end
193.             end
194.             else begin
195.                 for (int i = DataBegin-offset*8-1; i >= DataBegin-
    offset*8-16; i--) begin
196.                     way2[set_to][i] = data1_tmp[DataBegin-offset*8-
    1-i];
197.                 end
198.                 #2;
199.                 for (int i = DataBegin-offset*8-17; i >= DataBegin-
    offset*8-32; i--) begin
200.                     way2[set_to][i] = data1_tmp[DataBegin-offset*8-
    17-i];
201.                 end
202.             end
203.
204.             #2
205.
206.             ctrl1_tmp = 'z;
207.         end
208.         default: begin
209.         end
```

```
        endcase
    end

    task automatic write_back(input int way_select, address addr);
        logic [CacheSetSize-1:0] set = addr_set(addr);
        int offset = 0;

        if (way_select == 1
        && way1[set][ValidBit] == 1 && way1[set][DirtyBit] == 1) begin
            addr2_tmp = addr[AddressSize-1:CacheOffsetSize];
            ctrl2_tmp = 2'b11;

            #2 ctrl2_tmp = 'z;

            while (ctrl2_tmp != 2'b01) begin
                for (
                    int i = DataBegin*Data2BusSize-1;
                    i >= DataBegin-offset*Data2BusSize-Data2BusSize;
                    i--
                ) begin
                    data2_tmp[DataBegin*Data2BusSize-1-i] = way1[set][i];
                end
                offset++;
            end

            way1[set][ValidBit] = 0;
            way1[set][DirtyBit] = 0;
            way1[set][DeltaBit] = 1;
        end
        else if (way_select == 2
        && way2[set][ValidBit] == 1 && way2[set][DirtyBit] == 1) begin
            addr2_tmp = addr[AddressSize-1:CacheOffsetSize];
            ctrl2_tmp = 2'b11;

            #2 ctrl2_tmp = 'z;

            while (ctrl2_tmp != 2'b01) begin
                for (
                    int i = DataBegin*Data2BusSize-1;
                    i >= DataBegin-offset*Data2BusSize-Data2BusSize;
```

```
250.                     i--
251.                 ) begin
252.                     data2_tmp[DataBegin*Data2BusSize-1-i] =
   way2[set][i];
253.                 end
254.                 offset++;
255.             end
256.
257.             way2[set][ValidBit] = 0;
258.             way2[set][DirtyBit] = 0;
259.             way2[set][DeltaBit] = 1;
260.         end
261.     endtask
262.
263.     task automatic lru(input address addr, output int way_idx);
264.         logic [CacheSetSize-1:0] set = addr_set(addr);
265.
266.         if (way1[set][ValidBit] == 1 && way2[set][ValidBit] == 1)
   begin
267.             way_idx = way1[set][DeltaBit] > way2[set][DeltaBit] ? 1
   : 0;
268.         end
269.         else begin
270.             way_idx = way1[set][ValidBit] == 1 ? 2 : 1;
271.         end
272.     endtask
273.
274.     task automatic read_line(input address addr, output int
   way_idx);
275.         logic [CacheSetSize-1:0] set = addr_set(addr);
276.
277.         int way_select;
278.         lru(addr, way_select);
279.
280.         write_back(way_select, addr);
281.
282.         if (way_select == 1) begin
283.             way1[set][ActualLineSize-1:ActualLineSize-CacheTagSize]
   = addr_tag(addr);
284.             way1[set][DeltaBit] = 0;
285.             way1[set][ValidBit] = 1;
286.             way1[set][DirtyBit] = 0;
287.
288.             way_idx = 1;
289.         end
```

```
290.         else begin
291.             way2[set][ActualLineSize-1:ActualLineSize-CacheTagSize]
   = addr_tag(addr);
292.             way2[set][DeltaBit] = 0;
293.             way2[set][ValidBit] = 1;
294.             way2[set][DirtyBit] = 0;
295.
296.             way_idx = 2;
297.         end
298.     endtask
299.
300.     task automatic find(input logic [Addr1BusSize-1:0] query,
   output int set_idx);
301.         logic [CacheSetSize-1:0] set = query;
302.         logic [CacheTagSize-1:0] tag = query >> CacheSetSize;
303.
304.         if (way1[set][ValidBit] == 1
305.             && way1[set][ActualLineSize-1:ActualLineSize-
   CacheTagSize] == tag) begin
306.             if (way2[set][ValidBit] == 1) begin
307.                 way1[set][DeltaBit] = 0;
308.                 way2[set][DeltaBit] = 1;
309.             end
310.
311.             hits_tmp++;
312.             #6 set_idx = 1;
313.         end
314.         else if (way2[set][ValidBit] == 1
315.             && way2[set][ActualLineSize-1:ActualLineSize-
   CacheTagSize] == tag) begin
316.             if (way1[set][ValidBit] == 1) begin
317.                 way2[set][DeltaBit] = 0;
318.                 way1[set][DeltaBit] = 1;
319.             end
320.
321.             hits_tmp++;
322.             #6 set_idx = 2;
323.         end
324.         else begin
325.
326.             miss_tmp++;
327.             #4 set_idx = 0;
328.         end
329.     endtask
330.
```

```
331.     function automatic address addr_tag_set(input logic
   [Addr1BusSize-1:0] value);
332.         address result = value;
333.         return result << CacheOffsetSize;
334.     endfunction
335.
336.     function static logic [CacheTagSize-1:0] addr_tag(address
   addr);
337.         return addr[AddressSize-1:AddressSize-CacheTagSize];
338.     endfunction
339.
340.     function static logic [CacheSetSize-1:0] addr_set(address
   addr);
341.         return addr[AddressSize-CacheTagSize-1:AddressSize-
   CacheTagSize-CacheSetSize];
342.     endfunction
343.
344.     function static logic [CacheOffsetSize-1:0] addr_offset(address
   addr);
345.         return addr[CacheOffsetSize-1:0];
346.     endfunction
347.
348. endmodule
```

Файл: memory.sv

```
1. module memory(a2, d2, c2, clk, rst, m_dump);
2.
3.     localparam int CacheAddrSize = 18;
4.     localparam int CacheOffsetSize = 4;
5.
6.     localparam int ByteSize = 8;
7.
8.     localparam int Addr2BusSize = 14;
9.     localparam int Data2BusSize = 16;
10.      localparam int Ctr2BusSize = 2;
11.
12.      localparam int MemSize = (1<<CacheAddrSize);
13.
14.      localparam int Seed = 225526;
15.
16.      typedef logic [CacheAddrSize-1:0] address;
17.
18.      input [Addr2BusSize-1:0] a2;
```

```
19.     inout [Data2BusSize-1:0] d2;
20.     inout [Ctr2BusSize-1:0] c2;
21.
22.     input clk;
23.     input rst;
24.     input m_dump;
25.
26.     logic [ByteSize-1:0] mem [MemSize];
27.
28.     logic [Data2BusSize-1:0] data_ray = 'z;
29.     logic [Ctr2BusSize-1:0] control = 'z;
30.     logic [3] line_offset;
31.
32.     assign d2 = data_ray;
33.     assign c2 = control;
34.
35.     initial begin
36.         init(Seed);
37.     end
38.
39.     int i;
40.
41.     address addr;
42.
43.     always @(posedge clk) begin
44.         if (rst) begin
45.             init(Seed);
46.         end
47.
48.         if (m_dump) begin
49.             for (i = 0; i < MemSize; i++) begin
50.                 $display(mem[i]);
51.             end
52.         end
53.     end
54.
55.     always @(posedge clk) begin
56.         case (c2)
57.         2'b10: begin
58.             addr = a2;
59.             addr <<= CacheOffsetSize;
60.
61.             #2;
62.             #1 control = 2'b00;
63.             #180;
```

```
64.             control = 2'b01;
65.
66.             for (i = 0; i < 8; i++) begin
67.                 data_ray[15:8] = mem[addr++];
68.                 data_ray[7:0] = mem[addr++];
69.                 #2;
70.             end
71.
72.             control = 'z;
73.             data_ray = 'z;
74.             #1;
75.         end
76.         2'b11: begin
77.             addr = a2;
78.             addr <<= CacheOffsetSize;
79.
80.             #3 control = 2'b00;
81.
82.             #183;
83.
84.             for (i = 0; i < 8; i++) begin
85.                 mem[addr++] = data_ray[15:8];
86.                 mem[addr++] = data_ray[7:0];
87.                 #2;
88.             end
89.
90.             #1 control = 2'b01;
91.             #2 control = 'z;
92.             #1;
93.         end
94.         default: begin
95.         end
96.         endcase
97.     end
98.
99.     task static init(input int seed);
100.        for (i = 0; i < MemSize; i += 1) begin
101.            mem[i] = $random(seed) >> 16;
102.        end
103.    endtask
104. endmodule
```

Сравнение результатов.

К сожалению не получилось получить удовлетворяющих сравнению результатов, однако выпонение происходит успешно. Скорее всего ошибка в тактовых конфликтах на запись/считывание.

Файл: mockup.cpp

```
1. ss#include <algorithm>
2. #include <cmath>
3. #include <cstdint>
4. #include <cstddef>
5. #include <array>
6. #include <ctime>
7. #include <iostream>
8. #include <cstring>
9.
10.using std::array;
11.
12.// Глобальные переменные для подсчета тактов, попаданий, промахи.
13.uint64_t clocks = 0;
14.uint32_t hits = 0;
15.uint32_t misses = 0;
16.
17.// Структура реализующая кэш.
18.// Базовые константы взяты из параметров системы.
19.struct Cache {
20.    static constexpr size_t CACHE_SETS_COUNT = 64;
21.    static constexpr size_t CACHE_LINE_SIZE = 16;
22.    static constexpr uint32_t CACHE_SET_SIZE = 6;
23.    static constexpr uint32_t CACHE_OFFSET_SIZE = 4;
24.
25.    // Структура задающая кэш-линию.
26.    // Содержит tag, биты V, D, Dlt.
27.    // Информация - "Мнимая".
28.    struct Line {
29.        uint32_t tag;
30.
31.        bool dirty = false;
32.        bool valid = false;
33.
```

```
34.         // Data.
35.
36.         uint32_t delta = 1;
37.     };
38.
39.     // way описывается, как статический массив линий.
40.     using Way = array<Line, CACHE_SETS_COUNT>;
41.
42.     // В данной реализции way разделенны в отдельные переменные.
43.     Way way1{};
44.     Way way2{};
45.
46.     // LRU возвращает итератор на замещаймую линию.
47.     Way::iterator lru(uint32_t set) {
48.         if (way1[set].valid && way2[set].valid) {
49.             return way1[set].delta > way2[set].delta ? way1.begin() + set
    : way2.begin() + set;
50.         }
51.
52.         return way1[set].valid ? way2.begin() + set : way1.begin() + set;
53.     }
54.
55.     // write_back
56.     // Пишет линию обратно в память в соответсвии с описанием.
57.     void write_back(Way::iterator it) {
58.         if (!it->valid || !it->dirty) {
59.             return;
60.         }
61.
62.         // Do move.
63.         ++clocks; // Address pass.
64.
65.         clocks += 100; // Memory await.
66.
67.         // линия более не валидна.
68.         it->valid = false;
69.         it->dirty = false;
70.         it->delta = 1;
71.
72.         clocks += 8; // Transmitting.
73.     }
74.
75.     // Чтение линии из памяти.
76.     Way::iterator read_line(uint32_t addr) {
77.         uint32_t set = addr_set(addr);
```

```
78.         uint32_t tag = addr_tag(addr);
79.
80.         Way::iterator it = lru(set);
81.
82.         write_back(it);
83.
84.         // Ставить Dlt для двух в 1, т.к новая линия будет
    инициализированна с 0.
85.
86.         way1[set].delta = 1;
87.         way2[set].delta = 1;
88.
89.         // запись в кэш.
90.         it->tag = tag;
91.         it->delta = 0;
92.         it->valid = true;
93.         it->dirty = false;
94.
95.         // Do read.
96.
97.         ++clocks; // Address pass.
98.         clocks += 100; // Memory await.
99.         clocks += 8; //fetching.
100.
101.          return it;
102.      }
103.
104.      // Поиск линии в set, если не найдет вернет конец второго way.
105.      // Иначе итератор на нужную линию.
106.      Way::iterator find(uint32_t tag, uint32_t set) {
107.          if (way1[set].valid && way1[set].tag == tag) {
108.              // Обновляем Dlt.
109.              if (way2[set].valid) {
110.                  way1[set].delta = 0;
111.                  way2[set].delta = 1;
112.              }
113.              ++hits;
114.
115.              clocks += 6; // Cache hit await.
116.
117.              return way1.begin() + set;
118.          }
119.          else if (way2[set].valid && way2[set].tag == tag) {
120.              // Обновляем Dlt.
121.              if (way1[set].valid) {
```

```
122.                 way2[set].delta = 0;
123.                 way1[set].delta = 1;
124.             }
125.             ++hits;
126.
127.             clocks += 6; // Cache hit await.
128.
129.             return way2.begin() + set;
130.         }
131.         else {
132.             ++misses;
133.
134.             clocks += 4; // Cache miss await.
135.
136.             return way2.end();
137.         }
138.     }
139.
140.     // Функции разбиения адреса.
141.
142.     // Fetch Tag from address.
143.     uint32_t addr_tag(uint32_t addr) {
144.         return (addr >> CACHE_OFFSET_SIZE >> CACHE_SET_SIZE);
145.     }
146.
147.     // Fetch Set from address.
148.     uint32_t addr_set(uint32_t addr) {
149.         return (addr >> CACHE_OFFSET_SIZE & ((1 << CACHE_SET_SIZE) -
    1));
150.     }
151.
152.     // Fetch Offset from address.
153.     uint32_t addr_offset(uint32_t addr) {
154.         return (addr & ((1 << CACHE_OFFSET_SIZE) - 1));
155.     }
156.
157.     // Чтение 8 бит из кэша, проделываем описанные в предыдущем пункте
    операции.
158.     int8_t read_int8(uint32_t addr) {
159.         clocks += 2; // Address receive.
160.
161.         uint32_t set = addr_set(addr);
162.         uint32_t tag = addr_tag(addr);
163.
164.         // Поиск линии.
```

```
165.         Way::iterator it = find(tag, set);
166.
167.         // Если не найдена - читаем из памяти.
168.         if (it == way2.end()) {
169.             read_line(addr);
170.         }
171.
172.         ++clocks; // Response.
173.
174.         return 0;
175.     }
176.
177.     // Аналогично для 16 бит.
178.     int16_t read_int16(uint32_t addr) {
179.         clocks += 2; // Address receive.
180.
181.         uint32_t set = addr_set(addr);
182.         uint32_t tag = addr_tag(addr);
183.
184.         Way::iterator it = find(tag, set);
185.
186.         if (it == way2.end()) {
187.             read_line(addr);
188.         }
189.
190.         ++clocks; // Response.
191.
192.         return 0;
193.     }
194.
195.     // Аналогично, так-как данные "Мнимые" и букально только запрос
    данных.
196.     void write_int32(uint32_t addr, int32_t value) {
197.         clocks += 2; // Address receive.
198.
199.         uint32_t set = addr_set(addr);
200.         uint32_t tag = addr_tag(addr);
201.
202.         Way::iterator it = find(tag, set);
203.
204.         if (it == way2.end()) {
205.             it = read_line(addr);
206.         }
207.
208.         clocks++; // Responce;
```

```
209.
210.         it->dirty = true;
211.     }
212.
213.     // Выписываем все изменения из кэша обратно в память.
214.     void goodbye() {
215.         for (auto it = way1.begin(); it != way1.end(); ++it) {
216.             write_back(it);
217.         }
218.
219.         for (auto it = way2.begin(); it != way2.end(); ++it) {
220.             write_back(it);
221.         }
222.     }
223. };
224.
225. // Экземпляр кэша.
226. Cache cache;
227.
228. // Константы задания.
229. constexpr int32_t M = 64;
230. constexpr int32_t N = 60;
231. constexpr int32_t K = 32;
232.
233. // Заменены базовые опрерации, чтобы встроить счетчик тактов.
234.
235. // Initializer for uint32 with embeded clock tick.
236. uint32_t init_uint32(uint32_t value) {
237.     ++clocks;
238.     return value;
239. }
240.
241. // Initializer for int32 with embeded clock tick.
242. int32_t init_int32(int32_t value) {
243.     ++clocks;
244.     return value;
245. }
246.
247. // Default addition with embeded clock tick.
248. void addition(uint32_t& value, const uint32_t add) {
249.     ++clocks;
250.     value += add;
251. }
252.
253. // Default signed addition with embeded clock tick.
```

```
254. void addition(int32_t& value, const int32_t add) {
255.     ++clocks;
256.     value += add;
257. }
258.
259. // Default signed multiplication embeded clock tick.
260. int32_t multiplication(const int8_t mul_1, const int16_t mul_2) {
261.     clocks += 5;
262.     return mul_1 * mul_2;
263. }
264.
265. // Just pass the tick for next iteration of function exit.
266. void next() {
267.     ++clocks;
268. }
269.
270. // Все опреции заменины на соответствующие со встроенным тактовым
   счетом.
271.
272. void mmul() {
273.     uint32_t pa_addr = init_uint32(0); // a begin.
274.
275.     uint32_t pc_addr = init_uint32(5888); // M * K + 2 bytes * K * N
276.
277.     for (int y = init_int32(0); y < M; addition(y, 1)) {
278.
279.         for (int x = init_int32(0); x < N; addition(x, 1)) {
280.
281.             uint32_t pb_addr = init_uint32(2048); // M * K
282.             int32_t s = init_int32(0);
283.
284.             for (int k = init_int32(0); k < K; addition(k, 1)) {
285.
286.                 addition(
287.                     s,
288.                     multiplication(
289.                         cache.read_int8(pa_addr + k),
290.                         cache.read_int16(pb_addr + x * 2)
291.                     )
292.                 );
293.                 addition(pb_addr, N * 2);
294.             }
295.             cache.write_int32(pc_addr + x * 4, s);
296.         }
297.         addition(pa_addr, K);
```

```
298.         addition(pc_addr, N * 4);
299.     }
300.     next();
301. }
302.
303. int main() {
304.
305.     mmul();
306.
307.     cache.goodbye();
308.
309.     std::cout << "Clocks: " << clocks << '\n';
310.     std::cout << "Hits: " << hits << " | " << static_cast<double>(hits)
   / static_cast<double>(hits + misses) << '\n';
311.     std::cout << "Misses: " << misses << std::endl;
312.
313.     return 0;
314. }
```