Лабораторная работа №2	M3136	2022
Моделирование схем в Verilog	Корнилович Михиал Антонович	

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

Инструментарий: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 10 и новее (полезные материалы: Verilog.docx). В отчёте нужно указать, какой версией вы пользовались (можно также приложить ссылку на онлайн-платформу). Использовать SystemVerilog допустимо, главное, чтобы код компилился под Icarus 10, 11 или 12. Далее в этом документе Verilog+SystemVerilog обозначается как Verilog. В работе использовался Icarus Verilog 12.

Описание

Необходимо построить систему “процессор-кэш-память”. Система должна обладать следующими параметрами (см. таблицу 1).

<table>
<tr><td colspan="4">CPU</td></tr>
<tr><td rowspan="2">Команды</td><td>CPU → Cache</td><td>0 – C1_NOP
1 – C1_READ8
2 – C1_READ16
3 – C1_READ32
4 – C1_INVALIDATE_LINE
5 – C1_WRITE8
6 – C1_WRITE16
7 – C1_WRITE32</td><td rowspan="2">Команда 4 означает инвалидацию всей кэш-линии, содержащей указанный адрес.

Число в командах означает кол-во бит данных, запрашиваемое данной командой.

Команды, запрашивающие несколько байт, не могут пересекать кэш-линию.

NOP – no operation.
Response – ответ на команду.</td></tr>
<tr><td>CPU ← Cache</td><td>0 – C1_NOP
7 – C1_RESPONSE</td></tr>
</table>

Кэш (look-through write-back)			
Политика вытеснения		LRU	
Команды	Cache → Mem	0 – **C2_NOP** 2 – **C2_READ_LINE** 3 – **C2_WRITE_LINE**	Команды пишут и читают порциями, равными размеру кэш-линии.
	Cache ← Mem	0 – **C2_NOP** 1 – **C2_RESPONSE**	
Служебные биты		V (valid), D (dirty)	Если valid установлен в 0, то данная кэш-линия свободна и состояние остальных битов не важно. dirty означает, что кэш-линия хранит изменённые данные, которые ещё не записаны в память.

Таблица №1 – общие параметры системы.

Также необходимо решить задачу аналитическим методом, а потом с помощью построенной модели.

Вариант

Предоставленные мне параметры (см. таблицу 2):

Кэш (продолжение)		
Размер кэша	2 Кб – **CACHE_SIZE**	Размер полезных данных.
Размер кэш-линии	16 байта – **CACHE_LINE_SIZE**	Размер полезных данных.
Кол-во бит под тэг адреса	8 бита – **CACHE_TAG_SIZE**	
Память		
Размер памяти	256 Кбайт – **MEM_SIZE**	(старое значение 128 Кбайт – **MEM_SIZE**)

Таблица №2 – параметры (вариант 2)

Расчёт параметров

Найдём недостающие параметры системы (см. таблицу 3).

CACHE_SIZE	16384
CACHE_LINE_SIZE	128
CACHE_TAG_SIZE	8
MEM_SIZE	2097152
CACHE_LINE_COUNT	128
CACHE_WAY	2
CACHE_SETS_COUNT	64
CACHE_SET_SIZE	6
CACHE_OFFSET_SIZE	7
CACHE_ADDR_SIZE	21
DATA1_BUS_SIZE	16
DATA2_BUS_SIZE	16
ADDR1_BUS_SIZE	14
ADDR2_BUS_SIZE	14
CTR1_BUS_SIZE	3
CTR2_BUS_SIZE	2

Таблица №3 – Все параметры системы

Аналитическое решение задачи

Для начала хотелось бы рассказать, как работает 2-ух ассоциативный кэш. Ассоциативность – это количество кэш-линий в одном сете. В моём варианте кэш делится н 64 сета, и в каждом сете находится 2 кэш-линии. Адресация происходит так: нам даны номер сета и тег, с помощью номера можно найти первую кэш-линию в сете, умножив номер на ассоциативность. После находим нужную кэш-линию, пройдясь по всем линиям в сете и сравнивая их теги с данным тегом. Также процессор даёт сдвиг, который показывает с какого бита нужно начинать читать или записывать.

Для решения воспользуемся языком Python. Для начала надо прописать все параметры в глобальные переменные. После чего нужно реализовать контроллер памяти. Для этого создадим класс с двумя методами для чтения линии из памяти и для записи линии в память. (см. листинг 1)

```
class Memory:
    def __init__(self, data=[[0] * CACHE_LINE_SIZE] * CACHE_SIZE):
        self.data = data
        self.tag = [0] * CACHE_SIZE
```

```
        for i in range(CACHE_SIZE):
            self.tag[i] = i
    def read_line(self, tag):
        for i in range(len(self.data)):
            if self.tag[i] == tag:
                return self.data[i]
    def write_line(self, tag, line):
        for i in range(len(self.data)):
            if self.tag[i] == tag:
                self.data[i] = line
                return
```

Листинг №1 – класс контроллера памяти

Теперь можно перейти к основной части задания – к кэшу. Для его реализации необходимо хранить теги линий, бит валидности линий, бит актуальности в памяти (dirty), данные для каждой кэш-линии, а также, так как в условии указана политика вытеснения LRU, нужно для каждой кэш-линии сохранять частоту обращения к ней. Чтобы посчитать количество cache hit-ов и cache miss-ов создадим соответствующие счётчики. Напишем код (см. листинг 2).

```
class Cache:
    def __init__(self, data=[[0] * CACHE_LINE_SIZE] * CACHE_SIZE):
        self.cache_hits = 0
        self.cache_misses = 0
        self.valid = [0] * CACHE_LINE_COUNT
        self.dirty = [0] * CACHE_LINE_COUNT
        self.data = [[0] * CACHE_LINE_SIZE] * CACHE_LINE_COUNT
        self.tag = [0] * CACHE_LINE_COUNT
        self.lru = [0] * CACHE_LINE_COUNT
        self.mem_ctr = Memory(data)
```

Листинг №2 – инициализация кэша

Рассмотрим операцию чтения. Для этого создадим метод, который на вход принимает размер, который нужно прочитать (data_size). Есть несколько случаев, которые нужно обработать:

1. Искомая кэш-линия находится в кэше, то есть происходит cache hit. Значит нужно также увеличить количество запросов (нужно для LRU)

```
        if self.tag[addr_set * CACHE_WAY] == addr_tag:
            self.cache_hits += 1
            self.lru[addr_set * CACHE_WAY] = 1
            self.lru[addr_set * CACHE_WAY + 1] = 0
            return self.data[addr_set * CACHE_WAY][addr_offset:addr_offset +
data_size]
        elif self.tag[addr_set * CACHE_WAY + 1] == addr_tag:
            self.cache_hits += 1
            self.lru[addr_set * CACHE_WAY + 1] = 1
            self.lru[addr_set * CACHE_WAY] = 0
            return self.data[addr_set * CACHE_WAY + 1][addr_offset:addr_offset +
data_size]
```

Листинг №3 – 1 случай чтения

2. Одна из кэша линий в сете имеет состояние invalid. То есть мы можем занять её. Для этого запишем в неё значение искомой кэш-линии из памяти. Здесь произошёл cache miss. (Незабываем про LRU)

```
        elif self.valid[addr_set * CACHE_WAY] == 0:
            self.cache_misses += 1
            self.valid[addr_set * CACHE_WAY] = 1
            self.lru[addr_set * CACHE_WAY] = 1
            self.lru[addr_set * CACHE_WAY + 1] = 0
            self.tag[addr_set * CACHE_WAY] = addr_tag
            self.data[addr_set * CACHE_WAY] = self.mem_ctr.read_line(addr_tag)
            return self.data[addr_set * CACHE_WAY][addr_offset:addr_offset +
data_size]
        elif self.valid[addr_set * CACHE_WAY + 1] == 0:
            self.cache_misses += 1
            self.valid[addr_set * CACHE_WAY + 1] = 1
            self.lru[addr_set * CACHE_WAY + 1] = 1
            self.lru[addr_set * CACHE_WAY] = 0
            self.tag[addr_set * CACHE_WAY + 1] = addr_tag
            self.data[addr_set * CACHE_WAY + 1] = self.mem_ctr.read_line(addr_tag)
            return self.data[addr_set * CACHE_WAY + 1][addr_offset:addr_offset +
data_size]
```

Листинг №4 – 2 случай чтения

3. Так как все линии в сете заняты, необходимо воспользоваться политикой вытеснения LRU. Для этого найдём кэш-линию с минимальным количеством запросов, если она dirty, то мы должны записать её в память. После чего нужно также прочитать из памяти искомую кэш-линию.

```
        elif self.lru[addr_set * CACHE_WAY] < self.lru[addr_set * CACHE_WAY + 1]:
            self.cache_misses += 1
            if self.dirty[addr_set * CACHE_WAY] == 1:
                self.mem_ctr.write_line(self.tag[CACHE_WAY * addr_set],
self.data[addr_set * CACHE_WAY])
            self.dirty[addr_set * CACHE_WAY] = 0
            self.tag[addr_set * CACHE_WAY] = addr_tag
            self.lru[addr_set * CACHE_WAY] = 1
            return self.data[CACHE_WAY * addr_set][addr_offset:addr_offset +
data_size]
        else:
            self.cache_misses += 1
            if self.dirty[addr_set * CACHE_WAY + 1] == 1:
                self.mem_ctr.write_line(self.tag[CACHE_WAY * addr_set + 1],
self.data[CACHE_WAY * addr_set + 1])
            self.dirty[addr_set * CACHE_WAY + 1] = 0
            self.tag[addr_set * CACHE_WAY + 1] = addr_tag
            self.lru[addr_set * CACHE_WAY + 1] = 1
            return self.data[addr_set * CACHE_WAY + 1][addr_offset:addr_offset +
data_size]
```

Листинг №5 – 3 случай чтения

Теперь наш кэш умеет обрабатывать запросы чтения. Запросы на запись практически аналогично, поэтому просто перечислим случаи, которые могут быть:

1. Искомая кэш-линия находится в кэше, то есть происходит cache hit.
2. Искомой кэш-линии нет в кэше, но одна из кэш-линий, которая находится в том же сете, имеет состояние invalid.
3. Искомой кэш-линии нет в кэше, и все кэш-линии в том же сете имеют состояние valid. Как и в read воспользуемся политикой вытеснения LRU.

Теперь можно приступить к написанию задачи, также этот кусок кода будет выполнять роль процессора. Важно обратить внимание на то, что переменные pa, pb, pc, y, x, k, s находятся не в кэше, а в регистрах процессора. Поэтому в кэш нам нужно записать только массивы a, b и c. Для этого подсчитаем сколько кэш-линий будет занимать каждый массив. Для a это будет. 128 линий, для b 240 линий, для c 960 линий.

Теперь можно инициализировать память.

```
    data = [[0] * CACHE_LINE_SIZE] * CACHE_SIZE
    n = 0
    m = 0
    k = 0
    for I in range(M):
        for j in range(K):
            if k >= CACHE_LINE_SIZE:
                m += 1
                k = 0
            number = bin(random.randint(-128, 127) & 127)
            binary_form_of_number = list(map(int, number[2::]))  # int list equals
binary form of random number
            for l in range(0, 8):
                if l < len(binary_form_of_number):
                    data[m][k] = binary_form_of_number[l]
                k += 1
    # a:    values in cache_line = 128 / 8 = 16, cache_lines_count = K * M / 16 = 128
lines
    #       tag = 0...128
    n = 0
    k = 0
    m = 0
    for I in range(K):
        for j in range(N):
            if n >= CACHE_LINE_SIZE:
                k += 1
                n = 0
            number = bin(random.randint(-32768, 32767) & 32767)
            binary_form_of_number = list(map(int, number[2::]))
            for l in range(0, 16):
                if l < len(binary_form_of_number):
                    data[129 + k][n] = binary_form_of_number[l]
                n += 1
    # b:     values in cache_line = 128 / 16 = 8, cache_lines_count = N * K / 8 = 240
lines
    #       tag = 129..369
```

```
    n = 0
    m = 0
    k = 0
    for I in range(M):
        for j in range(N):
            if n >= CACHE_LINE_SIZE:
                m += 1
                n = 0
            number = bin(random.randint(-2147483648, 2147483647) & 2147483647)
            binary_form_of_number = list(map(int, number[2::]))
            for l in range(0, 32):
                if l < len(binary_form_of_number):
                    data[370 + m][n] = binary_form_of_number[l]
                n += 1
    # c:     values in cache_line = 128 / 32 = 4, cache_lines_count = M * N / 4 = 960
lines
    #       tag = 370..1330
```

Листинг №6 – инициализация памяти

Теперь надо просто переписать код из условия с некоторыми изменениями. Теперь надо обращаться к кэшу причём в качестве номера сета указывать pa % CACHE_SETS_COUNT, а в качестве тега pa.

```
    a = 0
    b = 0
    c = 0
    for y in range(M):
        x = 0
        while x < N * 32:
            pb = 0
            s = 0
            k = 0
            m = 0
            while k < K * 8 and m < K * 16:
                #  print(k * 8)
                if a - pa * CACHE_LINE_SIZE >= 0:
                    pa += 1
                if b - pb * CACHE_LINE_SIZE >= 0:
                    pb += 1
                #print(k * 8)
                a_n = ''.join(str(i) for i in cache.read(pa % CACHE_SETS_COUNT, pa, a
% CACHE_LINE_SIZE, 8))
                b_n = ''.join(str(i) for i in cache.read(pb % CACHE_SETS_COUNT, 129 +
pb, b % CACHE_LINE_SIZE, 16))
                s += int(a_n, 2) + int(b_n, 2)
                k += 8
                m += 16
                a += 8
                b += 16
            data = [0] * 32
            binary_form_of_number = list(map(int, bin(s & 2147483647)[2::]))
            for l in range(32):
```

```
                if l < len(binary_form_of_number):
                    data[l] = binary_form_of_number[l]
            if c - pc * CACHE_LINE_SIZE >= 0:
                pc += 1
            cache.write(pc % CACHE_SETS_COUNT, 370 + pc, c % CACHE_LINE_SIZE, data)
            x += 32
            c += 32
    cache.cache_info()
```

Листинг №7 – задача на языке Python

После запуска программы мы получим такой результат:

Requests	249600
Cache hits	240635
Cache misses	8965
Cache hit ratio	0.964082532051282
Cache miss ratio	0.035917467948717946
Tick count	3373598

Таблица №5 – Результаты аналитического решения

Получается наш Cache hit ration = 96.4%. А количество тактов равно 3373598.

Моделирование заданной системы на Verilog

Сначала нужно определить наш модуль cache

```
module cache #(
    parameter MEM_SIZE              = 2097152,
    parameter CACHE_SIZE            = 16384,
    parameter CACHE_LINE_SIZE       = 128,
    parameter CACHE_LINE_COUNT      = 128,
    parameter CACHE_WAY             = 2,
    parameter CACHE_SETS_COUNT      = 64,
    parameter CACHE_TAG_SIZE        = 8,
    parameter CACHE_SET_SIZE        = 6,
    parameter CACHE_OFFSET_SIZE     = 7,
    parameter CACHE_ADDR_SIZE       = 21,

    parameter DATA1_BUS_SIZE        = 16,
    parameter DATA2_BUS_SIZE        = 16,
    parameter ADDR1_BUS_SIZE        = 14,
    parameter ADDR2_BUS_SIZE        = 14,
    parameter CTR1_BUS_SIZE         = 3,
    parameter CTR2_BUS_SIZE         = 2,

    parameter C1_NOP                = 0,
    parameter C1_READ8              = 1,
    parameter C1_READ16             = 2,
    parameter C1_READ32             = 3,
    parameter C1_INVALIDATE_LINE    = 4,
    parameter C1_WRITE8             = 5,
    parameter C1_WRITE16            = 6,
    parameter C1_WRITE32            = 7,
    parameter C1_RESPONSE           = 7,
    parameter C2_NOP                = 0,
    parameter C2_READ_LINE          = 2,
```

```
    parameter C2_WRITE_LINE         = 3,
    parameter C2_RESPONSE           = 1
    ) (
    input                           CLK,
    input                           RESET,
    input                           C_DUMP,
    input                           M_DUMP,
    input wire[ADDR1_BUS_SIZE-1:0]  A1,
    inout wire[DATA1_BUS_SIZE-1:0]  D1,
    inout wire[CTR1_BUS_SIZE-1:0]   C1
    );
```

Листинг №8 – определения модуля cache на Verilog

Так же, как и в аналитическом решении надо инициализировать переменных. Здесь практически всё то же самое, кроме шин и регистров для них.

```
    int cache_hits_count = 0;
    int cache_miss_count = 0;
    bit valid[CACHE_LINE_COUNT];
    bit dirty[CACHE_LINE_COUNT];
    reg[CACHE_LINE_SIZE-1:0] data[CACHE_LINE_COUNT];
    reg[CACHE_TAG_SIZE-1:0] tag[CACHE_LINE_COUNT];
    bit lru[CACHE_LINE_COUNT];

    mem_ctr mem(CLK, RESET, M_DUMP, A2, D2, C2);

    reg[ADDR1_BUS_SIZE-1:0] a1='bz; // регистры, чтобы можно было посылать данные по
шине
    reg[DATA1_BUS_SIZE-1:0] d1='bz;
    reg[CTR1_BUS_SIZE-1:0] c1='bz;
    reg[ADDR2_BUS_SIZE-1:0] a2='bz;
    reg[DATA2_BUS_SIZE-1:0] d2='bz;
    reg[CTR2_BUS_SIZE-1:0] c2='bz;

    assign A1 = a1; // Здесь мы устанавливаем соответствие между регистрами и шинами
    assign D1 = d1;
    assign C1 = c1;
    assign A2 = a2;
    assign D2 = d2;
    assign C2 = c2;

    reg[CACHE_TAG_SIZE-1:0] addr_tag;
    reg[CACHE_SET_SIZE-1:0] addr_set;
    reg[CACHE_OFFSET_SIZE-1:0] addr_offset;

    int one_tick = 2;
    int index1 = 0;
    int index2 = 0;
    int cur_command = 0;
```

Листинг №9 – инициализация переменных на Verilog

Как происходит чтение в Verilog? Процессор посылает запрос по шине, и, как только он был получен кэшом, кэш перехваытвает владение шиной и

смотрит существует ли линия с адресом, который послал процессор по шине A1. Если такая линия есть, то произошёл cache hit, мы должны обновить LRU, И прочитать данные с линии.

Если же линия не нашлась в кэше, то мы ищем в нашем сете линию с флагом valid равным 0. Если такая есть, то мы обновляем информацию об этой кэш линии (valid, tag), и делаем запрос к mem_ctr (контроллер памяти) на получение данных этой кэш-линии.

Если и invalid кэш-линий е оказалось, то мы пользуемся политикой вытеснения LRU. LRU определяет какую кэш-линию надо выкинуть: кэш-линия, к которой мы дольше всего не обращались выкидывается. Пользуясь LRU, мы выбираем линию, в которую мы запишем новую кэш-линию. Для этого, если линия было до этого обновлена в кэше и не было записана в память, то мы записываем её в память (Write back). Потом мы делаем ещё один запрос к памяти, но теперь, чтобы прочитать линию с новым адресом.

В конце мы отправляем процессору RESPONSE и данные вместе с ответом.

Запись происходит аналогичным образом, только теперь в конце мы пишем в кэш линию новые данные, меняем бит dirty на 1 и отправляем процессору RESPONSE.

Воспроизведение задачи на Verilog.

Теперь напишем mem_ctr, который умеет обробатывать два запроса. На C2_READ_LINE мы будем искать по всем кэш-линиям нужную, после чего отправлять данные по шине d2, заранее передав владение ею mem_ctr.

Аналогично происходит запись в память. После любой из этих двух операций мы отправляем C2_RESPONSE.

Также реализуем процессор и саму программу, между каждой командой поставим задержку в 300 тактов (Verilog тактов). Делается процессор аналогично процессору на Python, кроме того, что теперь мы посылаем запросы по шинам, а не через аргументы функций.

Запустив программу, получим такой результат.

Requests	248533
Cache hits	239567
Cache misses	8966
Cache hit ratio	0.963924
Ticks	3467350

Таблица №6 – Результаты на Verilog

Сравнение полученных результатов.

Как видим результат практически сошёлся. Это разность результатов связана с тем, что на Verilog были неправильны проставлены задержки в некоторых местах.

```
`include "mem_ctr.sv"

module cache #(
    parameter MEM_SIZE              = 2097152,
    parameter CACHE_SIZE            = 16384,
    parameter CACHE_LINE_SIZE       = 128,
    parameter CACHE_LINE_COUNT      = 128,
    parameter CACHE_WAY             = 2,
    parameter CACHE_SETS_COUNT      = 64,
    parameter CACHE_TAG_SIZE        = 8,
    parameter CACHE_SET_SIZE        = 6,
    parameter CACHE_OFFSET_SIZE     = 7,
    parameter CACHE_ADDR_SIZE       = 21,

    parameter DATA1_BUS_SIZE        = 16,
    parameter DATA2_BUS_SIZE        = 16,
    parameter ADDR1_BUS_SIZE        = 14,
    parameter ADDR2_BUS_SIZE        = 14,
    parameter CTR1_BUS_SIZE         = 3,
    parameter CTR2_BUS_SIZE         = 2,

    parameter C1_NOP                = 0,
    parameter C1_READ8              = 1,
    parameter C1_READ16             = 2,
    parameter C1_READ32             = 3,
    parameter C1_INVALIDATE_LINE    = 4,
    parameter C1_WRITE8             = 5,
    parameter C1_WRITE16            = 6,
    parameter C1_WRITE32            = 7,
    parameter C1_RESPONSE           = 7,
    parameter C2_NOP                = 0,
    parameter C2_READ_LINE          = 2,
    parameter C2_WRITE_LINE         = 3,
    parameter C2_RESPONSE           = 1
    ) (
    input                           CLK,
    input                           RESET,
    input                           C_DUMP,
    input                           M_DUMP,
    input wire[ADDR1_BUS_SIZE-1:0]  A1,
    inout wire[DATA1_BUS_SIZE-1:0]  D1,
    inout wire[CTR1_BUS_SIZE-1:0]   C1
    );
    output wire[ADDR2_BUS_SIZE-1:0] A2;
    inout wire[DATA2_BUS_SIZE-1:0] D2;
    inout wire[CTR2_BUS_SIZE-1:0] C2;

    int cache_hits_count = 0;
    int cache_miss_count = 0;
    bit valid[CACHE_LINE_COUNT];
    bit dirty[CACHE_LINE_COUNT];
    reg[CACHE_LINE_SIZE-1:0] data[CACHE_LINE_COUNT];
    reg[CACHE_TAG_SIZE-1:0] tag[CACHE_LINE_COUNT];
```

```
bit lru[CACHE_LINE_COUNT];

mem_ctr mem(CLK, RESET, M_DUMP, A2, D2, C2);

reg[ADDR1_BUS_SIZE-1:0] a1='bz;
reg[DATA1_BUS_SIZE-1:0] d1='bz;
reg[CTR1_BUS_SIZE-1:0] c1='bz;
reg[ADDR2_BUS_SIZE-1:0] a2='bz;
reg[DATA2_BUS_SIZE-1:0] d2='bz;
reg[CTR2_BUS_SIZE-1:0] c2='bz;

assign A1 = a1;
assign D1 = d1;
assign C1 = c1;
assign A2 = a2;
assign D2 = d2;
assign C2 = c2;

reg[CACHE_TAG_SIZE-1:0] addr_tag;
reg[CACHE_SET_SIZE-1:0] addr_set;
reg[CACHE_OFFSET_SIZE-1:0] addr_offset;

int one_tick = 2;
int index1 = 0;
int index2 = 0;
int cur_command = 0;
int count = 0;

initial begin
    //$monitor("%b", D1);
    $dumpfile("dump_cache.vcd");
    $dumpvars(0, cache);
    $dumpoff;
end

function int get_count();
    return count;
endfunction

function void reset();
    cache_hits_count = 0;
    cache_miss_count = 0;
    for (int i = 0; i < CACHE_LINE_COUNT; i++) begin
        valid[i] = 0;
        dirty[i] = 0;
        data[i] = 0;
        tag[i] = 'bz;
        lru[i] = 0;
    end
endfunction

function void cache_info();
$display(count);
```

```
        $display("Requests = %d", cache_miss_count + cache_hits_count);
        $display("Cache hits = %d", cache_hits_count);
        $display("Cache misses = %d", cache_miss_count);
        $display("Cache hit ratio = %f", (1.0 * cache_hits_count) / (1.0 *
(cache_hits_count + cache_miss_count)));
    endfunction

    task read_from_cache(int index, int data_size);
    //$display(data_size);
        case(data_size)
            8: begin
                d1[7:0] = data[index][addr_offset +: 8];
                //$display(D1);
            end
            16: begin
                d1 = data[index][addr_offset +: 16];
            end
            32: begin
                d1 = data[index][addr_offset +: 16];
                #one_tick d1 = data[index][addr_offset+16+:16];
            end
        endcase
    endtask

    task write_in_cache(int index, int data_size);
        case(data_size)
            8: begin
                data[index][addr_offset +: 8] = D2[7:0];
                //$display(123);
            end
            16: begin
                data[index][addr_offset +: 16] = D2;
            end
            32: begin
                data[index][addr_offset +: 16] = D2;
                #one_tick data[index][addr_offset+16 +: 16] = D2;
            end
        endcase
        dirty[index] = 1;
    endtask

    task write_data_in_mem(int index);
        c2 = C2_WRITE_LINE;
        a2 = {addr_tag, addr_set};
        //wait (C2 == C2_RESPONSE);
        for (int i = 0; i < CACHE_LINE_SIZE; i += DATA2_BUS_SIZE) begin
            #(i > 0 ? one_tick : 0) d2 = data[index][i +: DATA2_BUS_SIZE];
        end
        #200 c2 = 'bz;
        dirty[index] = 0;
        d2 = 'bz;
        count += 100;
    endtask
```

```
task read_data_from_mem(int index);
    if (valid[index] == 1 && dirty[index] == 1) begin
        write_data_in_mem(index);
    end
    c2 = C2_READ_LINE;
    a2 = {addr_tag, addr_set};
    #200 c2 = 'bz;
  //  $display("ok");
    //wait (C2 == C2_RESPONSE);
  //  $display("ok");
    valid[index] = 1;
    tag[index] = addr_tag;
    lru[index] = 1;
    for (int i = 0; i < CACHE_LINE_SIZE; i += DATA2_BUS_SIZE) begin
       // $display("Take = %0b", D2);
        #(i > 0 ? one_tick : 0) data[index][i +: DATA2_BUS_SIZE] = D2;
    end
    count += 100;
    //$display("%b", data[index]);
endtask



task read(int data_size);
    index1 = addr_set * CACHE_WAY;
    index2 = addr_set * CACHE_WAY + 1;
    if (valid[index1] == 1 && tag[index1] == addr_tag) begin
        lru[index1] = 1;
        lru[index2] = 0;
        cache_hits_count++;
        c1 = C1_RESPONSE;
        count += 6;
        #12 read_from_cache(index1, data_size);
    end
    else if (valid[index2] == 1 && tag[index2] == addr_tag) begin
        lru[index2] = 1;
        lru[index1] = 0;
        cache_hits_count++;
        c1 = C1_RESPONSE;
        count += 6;
        #12 read_from_cache(index2, data_size);
    end
    else if (valid[index1] == 0) begin
        count += 4;
        cache_miss_count++;
        lru[index2] = 0;
        #8 read_data_from_mem(index1);
        //$display("1234");
        c1 = C1_RESPONSE;
        read_from_cache(index1, data_size);
    end
    else if (valid[index2] == 0) begin
        count += 4;
```

```
            cache_miss_count++;
            lru[index1] = 0;
            #8 read_data_from_mem(index2);
            c1 = C1_RESPONSE;
            read_from_cache(index2, data_size);
        end
        else if (lru[index1] < lru[index2]) begin
            count += 4;
            cache_miss_count++;
            #8 read_data_from_mem(index1);
            c1 = C1_RESPONSE;
            read_from_cache(index1, data_size);
        end
        else begin
            count += 4;
            cache_miss_count++;
            #8 read_data_from_mem(index2);
            c1 = C1_RESPONSE;
            read_from_cache(index2, data_size);
        end
        #2 c1 = 'bz;
    endtask

    task write(int data_size);
        index1 = addr_set * CACHE_WAY;
        index2 = addr_set * CACHE_WAY + 1;
        if (valid[index1] == 1 && tag[index1] == addr_tag) begin
            count += 6;
            #12 cache_hits_count++;
            lru[index1] = 1;
            lru[index2] = 0;
            c1 = C1_RESPONSE;
            #1 write_in_cache(index1, data_size);
        end
        else if (valid[index2] == 1 && tag[index2] == addr_tag) begin
            count += 6;
            #12 cache_hits_count++;
            lru[index2] = 1;
            lru[index1] = 0;
            c1 = C1_RESPONSE;
            #1 write_in_cache(index2, data_size);
        end
        else if (valid[index1] == 0) begin
            count += 4;
            #8 cache_miss_count++;
            lru[index2] = 0;
            read_data_from_mem(index1);
            c1 = C1_RESPONSE;
            #1 write_in_cache(index1, data_size);
        end
        else if (valid[index2] == 0) begin
            count += 4;
            #8 cache_miss_count++;
```

```
            lru[index1] = 0;
            read_data_from_mem(index2);
            c1 = C1_RESPONSE;
            #1 write_in_cache(index2, data_size);
        end
        else if (lru[index1] < lru[index2]) begin
            count += 4;
            #8 cache_miss_count++;
            read_data_from_mem(index1);
            c1 = C1_RESPONSE;
            #1 write_in_cache(index1, data_size);
        end
        else begin
            count += 4;
            #8 cache_miss_count++;
            read_data_from_mem(index2);
            c1 = C1_RESPONSE;
            #1 write_in_cache(index2, data_size);
        end
        #2 c1 = 'bz;
    endtask

    task invalidate_line;
        static int index1 = addr_set * CACHE_WAY;
        static int index2 = addr_set * CACHE_WAY + 1;
        if (tag[index1] == addr_tag) begin
            if (dirty[index1] == 1) begin
                write_data_in_mem(index1);
            end
            c1 = C1_RESPONSE;
            valid[index1] = 0;
            tag[index1] = 0;
            data[index1] = 0;
        end
        else if (tag[index2] == addr_tag) begin
            if (dirty[index2] == 1) begin
                write_data_in_mem(index2);
            end
            c1 = C1_RESPONSE;
            valid[index2] = 0;
            tag[index2] = 0;
            data[index2] = 0;
        end
        #2 c1 = 'bz;
    endtask

    always @(posedge CLK or posedge RESET or posedge C_DUMP) begin
     //y(123);
     //$display("%d", C1);
     //$display(C1);
        if (RESET) begin
            reset();
        end
```

```
        if (C_DUMP) begin
            $dumpon;
        end
        else begin
            $dumpoff;
        end
        if (C1 != 0) begin
            cur_command = C1;
         //   $display("time = %d C1 = %d", $time(), C1);
            addr_tag = A1[7:0];
            addr_set = A1[13:8];
            #one_tick addr_offset = A1[6:0];
            case (cur_command)
                C1_READ8: read(8);
                C1_READ16: read(16);
                C1_READ32: read(32);
                C1_INVALIDATE_LINE: invalidate_line();
                C1_WRITE8: write(8);
                C1_WRITE16: write(16);
                C1_WRITE32: write(32);
            endcase
          //  $display("%0d %0d", $time(), C1);
            //$display($time());
        end
    end
endmodule : cache
```

Листинг №10 – another_cache.sv

```
module mem_ctr #(
    parameter _SEED             = 225526,
    parameter MEM_SIZE          = 2097152,
    parameter CACHE_LINE_SIZE   = 128,
    parameter CACHE_LINE_COUNT  = 16384,
    parameter MEM_TAG_SIZE      = 14,
    parameter MEM_OFFSET_SIZE   = 7,
    parameter ADDR2_BUS_SIZE    = 14,
    parameter DATA2_BUS_SIZE    = 16,
    parameter CTR2_BUS_SIZE     = 2,

    parameter C2_NOP            = 0,
    parameter C2_READ_LINE      = 2,
    parameter C2_WRITE_LINE     = 3,
    parameter C2_RESPONSE       = 1
) (
    input                           CLK,
    input                           RESET,
    input                           M_DUMP,
    input wire[ADDR2_BUS_SIZE-1:0]  A2,
    inout wire[DATA2_BUS_SIZE-1:0]  D2,
    inout wire[CTR2_BUS_SIZE-1:0]   C2
);
    int SEED = 225526;
```

```
reg[DATA2_BUS_SIZE-1:0] d2='bz;
reg[CTR2_BUS_SIZE-1:0]  c2='bz;


reg[MEM_TAG_SIZE-1:0]       tag[CACHE_LINE_COUNT];
bit[CACHE_LINE_SIZE-1:0]    data[CACHE_LINE_COUNT];

assign D2 =  d2;
assign C2 =  c2;

byte                        command;
reg[MEM_TAG_SIZE-1:0]       addr_tag;

// program variables
localparam M = 64;
localparam N = 60;
localparam K = 32;
reg[7:0] a;
reg[15:0] b;
reg[31:0] c;

function void random_init();
    for (int i = 0; i < CACHE_LINE_COUNT; i++) begin
        data[i] = $random(SEED);
        tag[i] = i;
    end
endfunction

function void init();
    int n = 0;
    int m = 0;
    int k = 0;
    for (int i = 0; i < M; i++) begin
        for (int j = 0; j < K; j++) begin
            if (k >= 128) begin
                m++;
                k = 0;
            end
            a = $random(SEED);
            data[m][k +: 8] = a;
            k += 8;
        end
    end
    n = 0;
    k = 0;
    m = 0;
    for (int i = 0; i < K; i++) begin
        for (int j = 0; j < N; j++) begin
            if (m >= 128) begin
                k++;
                m = 0;
            end
            b = $random(SEED);
            data[k][m +: 16] = b;
```

```
                m += 16;
            end
        end
        n = 0;
        k = 0;
        m = 0;
        for (int i = 0; i < M; i++) begin
            for (int j = 0; j < N; j++) begin
                if (n >= 128) begin
                    m++;
                    n = 0;
                end
                c = $random(SEED);
                data[m][n +: 32] = c;
                n += 32;
            end
        end
    endfunction

    initial begin // generate random memory
        random_init();
        $dumpfile("dump_mem_ctr.vcd");
        $dumpvars(0, mem_ctr);
        $dumpoff;
    end

    always @(posedge CLK or posedge RESET) begin
        if (M_DUMP) begin
            $dumpon;
        end
        else begin
            $dumpoff;
        end
        if (RESET) begin
            random_init();
        end
        else begin
            command = C2;
            //$display(command);
            if (command == C2_READ_LINE) begin
                addr_tag = A2;
                #200
                c2 = C2_RESPONSE;
                for (int i = 0; i < CACHE_LINE_COUNT; i++) begin
                    if (tag[i] == addr_tag) begin
                        for (int j = 0; j < CACHE_LINE_SIZE; j += DATA2_BUS_SIZE)
begin

                            //$display("Sent: %0d", $time());
                            //$display("%d %d", i, j);
                            #(j > 0 ? 2 : 0) d2 = data[i][j +:DATA2_BUS_SIZE];
                        end
                    end
                end
```

```
                #2 c2 = 'bz;
                d2 = 'bz;
            end
        end
    end

    always @(negedge CLK) begin
        if (command == C2_WRITE_LINE) begin
            //$display(1234);
            addr_tag = A2;
            for (int i = 0; i < CACHE_LINE_COUNT; i++) begin
                    if (tag[i] == addr_tag) begin
                        for (int j = 0; j < CACHE_LINE_SIZE; j += DATA2_BUS_SIZE)
begin
                            //$display("Sent: %0d", $time());
                            //$display("%d %d", i, j);
                        //  #(j > 0 ? 2 : 0) data[i][j +:DATA2_BUS_SIZE] = D2;
                        end
                    end
            end
            #200 c2 = C2_RESPONSE;
            #2 c2 = 'bz;
            d2 = 'bz;
            //$display(123);
        end
    end

endmodule : mem_ctr
```

Листинг №11 – mem_ctr.sv

```
`include "another_cache.sv"

module cache_tb #(parameter _SEED = 225526) ();
    input wire                          CLK;
    input wire                          RESET;
    input wire[13:0]    A1;
    inout wire[15:0]    D1;
    inout wire[2:0]     C1;
    input wire C_DUMP;
    input wire M_DUMP;

    cache ch(CLK, RESET, C_DUMP, M_DUMP, A1, D1, C1);
    int SEED = _SEED;

    reg clk=0;
    reg[2:0] c1='bz;
    reg[13:0] a1='bz;
    reg[15:0] d1='bz;
    assign CLK=clk;
    assign C1=c1;
    assign A1=a1;
    assign D1=d1;
    localparam M = 64;
```

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

int pa = 0;
int pb = 0;
int pc = 0;
int s = 0;
reg[7:0] a;
reg[15:0] b;
reg[31:0] c;
int i = 0;
int a_i = 0;
int b_i = 0;
int c_i = 0;
int count = 0;

initial begin
    $display("Start");
    for (int y = 0; y < M; y++) begin
        for (int x = 0; x < N; x++) begin
            pb = 0;
            s = 0;
            for (int k = 0; k < K; k++) begin
                if (a_i - pa * 128 >= 0) begin
                    pa++;
                end
                if (b_i - pb * 128 >= 0) begin
                    pb++;
                end
                d1 = 'bz;
                c1 = 1;
                a1[7:0] = pa;
                a1[13:8] = pa % 64;
                #2 a1[6:0] = a_i % 128;
                c1 = 'bz;
                //count += 1;
                #300
                a = D1[7:0];
                #2 d1 = 'bz;
                c1 = 2;
                a1[7:0] = pb + 129;
                a1[13:8] = pb % 64;
                #2 a1[6:0] = b_i % 128;
                c1 = 'bz;
               // count += 1;
                #300
                b = D1[15:0];
                #2 d1 = 'bz;
                s += a + b;
                a_i += 8;
                b_i += 16;
                count += 5 + 1 + 1 + 1;
            end
```

```
                c = s;
                if (c_i - pc * 128 >= 0) begin
                    pc++;
                end
                #300 c1 = 7;
                 a1[7:0] = pc + 370;
                 a1[13:8] = pc % 64;
                 #2 a1[6:0] = c_i % 128;
                 c1 = 'bz;
                 #300
                 c_i += 32;
                 count += 2;
            end
            count += 1;
            $display("$d", y);
        end
        $display(count + ch.get_count());
        ch.cache_info();
        $finish;
    end

    always #1 if ($time() < 100000000) begin
        clk = ~clk;
    end
endmodule
```

Листинг №12 – testbench.sv