Лабораторная работа №2	M3139	2022
Моделирование схем в Verilog	Гришечкин Павел Аверьянович	

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

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 11

Описание

Задача

Имеется следующее определение глобальных переменных и функций:

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

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

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.

Вариант

2 вариант

Константы и параметры системы

<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>
<tr><td colspan="4">Кэш (look-through write-back)</td></tr>
<tr><td colspan="2">Политика вытеснения</td><td colspan="2">LRU</td></tr>
</table>

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.
Размер кэша		1 Кб – CACHE_SIZE	Размер полезных данных.
Размер кэш-линии		32 байта – CACHE_LINE_SIZE	Размер полезных данных.
Количество кэш-линий		32 – CACHE_LINE_COUNT	
Ассоциативность		2 – CACHE_WAY	
Количество блоков кэш-линий		16 – CACHE_SETS_COUNT	
Размер тэга адреса		11 бит – CACHE_TAG_SIZE	
Кол-во бит под хранение индекса набора		4 бита – CACHE_SET_SIZE	
Размер смещения		5 бит – CACHE_OFFSET_SIZE	
Размер адреса		20 бит – CACHE_ADDR_SIZE	
Память			
Размер памяти		1 Мбайт – MEM_SIZE	
Шина		Обозначение	Размерность

A1, A2	ADDR1_BUS_SIZE, ADDR2_BUS_SIZE	15 бит
D1, D2	DATA1_BUS_SIZE, DATA2_BUS_SIZE	16 бит
C1, C2	CTR1_BUS_SIZE, CTR2_BUS_SIZE	C1 – 3 бита C2 – 2 бита

Таблица 1 – Константы и параметры системы

Способ вычисления

CACHE_LINE_COUNT = CACHE_SIZE / CACHE_LINE_SIZE

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

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

$CACHE\_ADDR\_SIZE = \log_2(MEM\_SIZE)$

CACHE_TAG_SIZE = CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE - CACHE_SET_SIZE

CACHE_WAY = CACHE_LINE_COUNT / CACHE_SETS_COUNT

ADDR1_BUS_SIZE, ADDR2_BUS_SIZE = CACHE_TAG_SIZE + CACHE_SET_SIZE

Аналитическое решение

(В этом пункте будет приведен листинг кода с аналитическим решением и разбор функций. Объяснение того, что происходит будет в следующем. Все фрагменты кода взяты из cache_simulation.cpp)

```
const int A_STEP = 1;
const int B_STEP = 2;
const int C_STEP = 4;

const int A = 0;
const int B = M * K * A_STEP;
const int C = B + K * N * B_STEP;

int timer = 1;

int cache_tags[CACHE_SETS_COUNT][CACHE_WAY];
int cache_times[CACHE_SETS_COUNT][CACHE_WAY];
int cache_valid[CACHE_SETS_COUNT][CACHE_WAY];
int cache_dirty[CACHE_SETS_COUNT][CACHE_WAY];

int tacts_counter = 0;

int cache_miss = 0;
int cache_hit = 0;
```

Листинг кода 1 – Константы и переменные аналитического решения

*_STEP – кол-во байт в числе из *

A, B, C – адреса первой ячейки соответствующего массива

timer – время последнего использования линии(используется для LRU)

cache_times – времена последнего использования для каждой линии

cache_*[][] – данные о линии соответствующие названию

tacts_counter, cache_miss, cache_hit – счетчики соответствующие названию

```
bool check_entry(int set_num, int tag, bool write) {
    for (int i = 0; i < CACHE_WAY; ++i) {
        if (cache_tags[set_num][i] == tag) {
            cache_times[set_num][i] = timer++;
            if (write) cache_dirty[set_num][i] = 1;
            return true;
        }
    }
    return false;
}
```

Листинг кода 2 – проверка вхождения линии в кэш

Функция проверяет вхождение линии в кэш. Если write = true, то функция помечает линию измененной.

```
void memory_request(int addr, int bytes, bool write) {
    addr = addr >> CACHE_OFFSET_SIZE;
    int set_num = addr % (1 << CACHE_SET_SIZE);
    int tag = addr >> CACHE_SET_SIZE;

    if (check_entry(set_num, tag, write)) {
        tacts_counter += 7 + (bytes + 1) / 2;
        cache_hit++;
    } else {
        cache_miss++;
        cache_replacement(set_num, tag, write);
        tacts_counter += 106 + CACHE_LINE_SIZE /
DATA2_BUS_SIZE + (bytes + 1) / 2;
    }
}
```

Листинг кода 3 – Вытеснение из кэша

Функция ищет линию, которую нужно вытеснить и ставит на ее место текущую. Если write = true, то помечает линию как измененную.

К количеству тактов прибавляем время задержки + время передачи

```
void memory_request(int addr, int bytes, bool write) {
    addr = addr >> CACHE_OFFSET_SIZE;
    int set_num = addr % (1 << CACHE_SET_SIZE);
    int tag = addr >> CACHE_SET_SIZE;

    if (check_entry(set_num, tag, write)) {
        tacts_counter += 6 + bytes / 2;
        cache_hit++;
    } else {
        cache_miss++;
        cache_replacement(set_num, tag, write);
        tacts_counter += 105 + CACHE_LINE_SIZE /
DATA2_BUS_SIZE + bytes / 2;
    }
}
```

Листинг кода 4 – Моделирование обращения к памяти

Функция моделирует обращение к памяти. Если write = true, то мы пытаемся изменить линию.

Основной код аналитики

```
void process() {
    for (int i = 0; i < CACHE_SETS_COUNT; ++i) {
        for (int j = 0; j < CACHE_WAY; ++j) {
            cache_tags[i][j] = -1;
        }
    }
//    int8 *pa = a;
    int pa = A;
    tacts_counter++;
//    int32 *pc = c;
    int pc = C;
    tacts_counter++;

    tacts_counter++; // int y = 0
    for (int y = 0; y < M; y++)
    {
        tacts_counter++; // int x = 0
        for (int x = 0; x < N; x++)
        {
//            int16 *pb = b;
            int pb = B;
            tacts_counter++;
//            int32 s = 0;
            tacts_counter++;

            tacts_counter++; // int k = 0;
            for (int k = 0; k < K; k++)
            {
```

```
//                  s += pa[k] * pb[x];
                    tacts_counter++; // +=

                    tacts_counter += 5; // *

                    tacts_counter++; // pa + k
                    memory_request(pa + A_STEP * k, A_STEP, 0);

                    tacts_counter++; // pb + x;
                    memory_request(pb + B_STEP * x, B_STEP, 0);

                    pb += N * B_STEP;
                    tacts_counter++;

                    tacts_counter++; // k++
                }
//                pc[x] = s;
                tacts_counter++; // pc + x
                memory_request(pc + C_STEP * x, C_STEP, 1);

                tacts_counter++; // x++
            }
            pa += K * A_STEP;
            tacts_counter++;

            pc += N * C_STEP;
            tacts_counter++;

            tacts_counter++; // y++
        }
        tacts_counter++; //ret
}
```

Итоговые значения аналитики

Количество тактов – 7737520

Количество кэш-промахов – 35839

Количество кэш-попаданий – 213761

Процент попаданий – 85.6414%

Моделирование заданной системы на Verilog

Составные элементы

test_bench – Создание шины, элементов. Объединение всех элементов. Генерация тактов.

CPU – Моделирование кода из задачи. Создание обращений к кэшу.

Cache – Модель LRU кэша. Реализует обработку запросов процессора и создание запросов к памяти.

MemCTR – модель контроллера памяти. Генерирует начальные значения. Отвечает на запросы кэша.

Подробное описание каждого элемента

test_bench

```
wire[ADDR1_BUS_SIZE - 1:0] A1;
wire[ADDR2_BUS_SIZE - 1:0] A2;
wire[DATA1_BUS_SIZE - 1:0] D1;
wire[DATA2_BUS_SIZE - 1:0] D2;
wire[CTR1_BUS_SIZE - 1:0] C1;
wire[CTR2_BUS_SIZE - 1:0] C2;

reg C_DUMP;
reg M_DUMP;

reg RESET = 0;

reg CLK = 0;
```

Листинг кода 5 – Создание шины (файл test_bench.sv)

Создается 6 пучков проводов, каждый из которых подключается в соответствующие модули и подсоединяется к регистрам каждого модуля.

```
reg [0:CTR1_BUS_SIZE - 1] C1_out = 'hz;
reg [0:CTR2_BUS_SIZE - 1] C2_out = 'hz;
reg [0:DATA1_BUS_SIZE - 1] D1_out = 'hz;
reg [0:DATA2_BUS_SIZE - 1] D2_out = 'hz;
reg [0:ADDR2_BUS_SIZE - 1] A2_out = 'hz;

assign C1 = C1_out;
assign C2 = C2_out;
assign D1 = D1_out;
assign D2 = D2_out;
assign A2 = A2_out;
```

Листинг кода 6 – Пример подключения модуля к шине (файл CPU.sv)

По умолчанию все регистры передают сигнал z. Если с 2х концов провода подается не z, то на проводе будет x, поэтому чтобы этого избежать необходимо поддерживать хотя бы на 1 из концов провода сигнал z.

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

Листинг кода 7 – Генерация тактов (файл test_bench.sv)

Значение в CLK меняется на противоположное с задержкой 1.

Cache (описание работы, вспомогательные функции)

В задание необходимо использовать политику замещения LRU, то есть вытесняется та линия, обращений к которой не было дольше всего.

```
parameter BIT_CACHE_LINE_SIZE = CACHE_LINE_SIZE * 8;
parameter LINE_IN_CACHE_SIZE = 2 + CACHE_TAG_SIZE +
BIT_CACHE_LINE_SIZE;
```

Листинг кода 8 – создание дополнительных констант (файл Cache.sv)

Константа BIT_CACHE_LINE_SIZE – размер данных в кэш линии в битах

Константа LINE_IN_CACHE_SIZE – полный размер кэш линии (Включая valid dirty tag data)

```
reg [CACHE_ADDR_SIZE - 1:0] addr;
reg [CACHE_TAG_SIZE - 1:0] tag;
reg [CACHE_SET_SIZE - 1:0] set;
reg [CACHE_OFFSET_SIZE - 1:0] offset;
reg [BIT_CACHE_LINE_SIZE - 1:0] data;
reg [LINE_IN_CACHE_SIZE - 1:0] cache[0:CACHE_LINE_COUNT - 1];
reg [CACHE_LINE_COUNT - 1:0] times;
reg [CTR1_BUS_SIZE - 1:0] request;
```

Листинг кода 9 – Создание локальных переменных кэша (файл Cache.sv)

В cache хранятся все кэш линии.

В для каждой линии в time хранится 0/1. Если хранится 0, то данная линия не использовалась дольше всего или не была использована вообще, и ее можно вытеснить.

Все остальные переменные хранят информацию о запросе, соответствующую своему названию.

```
task dump();
    $display("hit = %0d, miss = %0d", cache_hit, cache_miss);
    fd = $fopen("CacheDump.ext", "w");
    for (i = 0; i < CACHE_LINE_COUNT; i++) begin
        $fdisplay(fd, "[%d] %b", i, cache[i]);
    end
    $fclose(fd);
endtask
```

Листинг кода 10 – dump кэша (файл Cache.sv)

Вывод статистики кэш-попаданий/промахов в консоль. Вывод кэша в CacheDump.ext.

```
task reset();
    C2_out = 0;
    for (i = 0; i < CACHE_LINE_COUNT; ++i) begin
        cache[i] = 0;
        times[i] = 0;
    end
endtask
```

Листинг кода 11 – reset кэша (файл Cache.sv)

Выставление значений по умолчанию, подача на C2 NOP.

```
always @(negedge CLK) begin
    if (C_DUMP) dump();
    if (RESET) reset();
end
```

Листинг кода 12 – обработка сигналов C_DUMP и RESET (файл Cache.sv)

Если на каком-то такте C_DUMP или RESET будут равны 1, то выполняется соответствующая функция.

```
task automatic upd_time();
    for (int i = set * CACHE_WAY; i < (set + 1) * CACHE_WAY;
++i) begin
        if (get_tag(i) != tag) times[i] = 0;
        else times[i] = 1;
    end
endtask
```

Листинг кода 12 – Обновление time

Помечает линию из запроса как последнюю использованную.

```
function reg[LINE_IN_CACHE_SIZE - 1:0] get_line(int pos);
    return cache[pos];
endfunction

function automatic reg[CACHE_TAG_SIZE - 1:0] get_tag(int
pos);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    return (line >> BIT_CACHE_LINE_SIZE) % (1 <<
CACHE_TAG_SIZE);
endfunction

function reg[CACHE_TAG_SIZE - 1:0] get_set(int pos);
    return pos / CACHE_WAY;
endfunction

function automatic reg[BIT_CACHE_LINE_SIZE - 1:0]
get_data(int pos);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    return line % (1 << BIT_CACHE_LINE_SIZE);
endfunction

function automatic reg get_dirty(int pos);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    return (line >> (BIT_CACHE_LINE_SIZE + CACHE_TAG_SIZE)) %
2;
endfunction

function automatic reg get_valid(int pos);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    return line >> (BIT_CACHE_LINE_SIZE + CACHE_TAG_SIZE +
1);
endfunction

task set_line(int pos, reg[LINE_IN_CACHE_SIZE - 1:0] line);
    cache[pos] = line;
endtask

task automatic set_valid(int pos, reg valid);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    line[LINE_IN_CACHE_SIZE - 1] = valid;
    set_line(pos, line);
endtask

task automatic set_dirty(int pos, reg dirty);
    reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
    line[LINE_IN_CACHE_SIZE - 2] = dirty;
    set_line(pos, line);
endtask
```

Листинг кода 13 – getter-ы и setter-ы (файл Cache.sv)

get_*(pos) – получает соответствующие данные о линии с номером pos.

set_*(pos, data) – выставляет для линии pos соответствующие данные.

```
function int get_line_number();
    for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1); i++)
begin
        if (get_valid(i) != 0 && get_tag(i) == tag) return i;
    end
    return -1;
endfunction
```

Листинг кода 14 – Получение линии запроса (файл Cache.sv)

Получает номер линии из запроса процессора. Если она отсутствует возвращает -1.

```
function automatic int get_replacement();
    for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1); i++)
begin
        if (get_valid(i) == 0) return i;
    end

    for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1); i++)
begin
        if (times[i] == 0) return i;
    end
endfunction
```

Листинг кода 15 – Получение номера вытесняемой линии (файл Cache.sv)

Получает номер вытесняемой линии из текущего блока.

```
function reg[LINE_IN_CACHE_SIZE - 1:0] build_new_line();
   return data + (tag << BIT_CACHE_LINE_SIZE) +
   (1 << (BIT_CACHE_LINE_SIZE + CACHE_TAG_SIZE + 1));
endfunction
```

Листинг кода 16 – Создание новой линии (файл Cache.sv)

Создает новую линию с тегом из запроса процессора.

```
function reg[7:0] get8();
        return (data >> offset * 8) % (1 << 8);
    endfunction
    function reg[15:0] get16();
        return (data >> offset * 8) % (1 << 16);
    endfunction
    function reg[31:0] get32();
    return (data >> offset * 8) % (1 << 32);
endfunction
```

Листинг кода 17 – Получение байтов числа запроса (файл Cache.sv)

Возвращает байты запроса процессора.

```
task CLKwaiting(int cnt);
    for (i = 0; i < cnt; ++i) begin
        @(negedge CLK);
    end
endtask
```

Листинг кода 18 – Задержка на cnt тактов (файл Cache.sv)

Взаимодействие Cache и MemCTR

```
always @(negedge CLK) begin
        if (C2 === 2 || C2 === 3) begin
```

Листинг кода 19 – Обработка такта в памяти (файл MemCTR)

Память ждет, когда по C2 придет 2 или 3.

Обработка чтения из MemCTR

```
task automatic read_mem(int tag, int set);
    C2_out = 2;
    A2_out = (tag << CACHE_SET_SIZE) + set;
    CLKwaiting(1);
    A2_out = 'hz;
    C2_out = 'hz;
    while (!(C2 === 1)) CLKwaiting(1);
    data = 0;
    shift = 0;
    for (int x = 0; x < 16; ++x) begin
        data += (D2 << shift);
        shift += DATA2_BUS_SIZE;
        CLKwaiting(1);
    end
    C1_out = 0;
Endtask
```

Листинг кода 20 – Чтение данных из RAM (файл Cache.sv)

```
addr = A2 << CACHE_OFFSET_SIZE;
    if (C2 == 2) begin
        CLKwaiting(1);
        C2_out = 0;
        CLKwaiting(99);
        C2_out = 1;
        for (i = addr; i + 1 < addr +
CACHE_LINE_SIZE; i += 2) begin
            D2_out = ram[i] + (ram[i + 1] << 8);
            CLKwaiting(1);
        end
        D2_out = 'hz;
        C2_out = 'hz;
```

Листинг кода 21 – Передача данных памятью (файл MemCTR.sv)

1. Кэш отправляет 2 по C2 и set + tag по A2.
2. Через такт кэш ставит z на C2 и A2 и ждет пока память не отправит 1 на C2, а память отправляет 0 по C2.
3. После задержки память начинает передавать данные и ставит на C2 1.
4. После окончания передачи на D2 ставится z, а на C2 ставится 0 со стороны кэша.
5. Считанные данные хранятся в глобальной переменной data.

Обработка записи в память

```
delay = 100;
i = addr;
for (int j = 0; j < 16; ++j) begin
    ram[i] = D2 % (1 << 8);
    ram[i + 1] = D2 >> 8;
    CLKwaiting(1);
    C2_out = 0;
    delay--;
    i += 2;
end
C2_out = 0;
CLKwaiting(delay);
C2_out = 1;
CLKwaiting(1);
C2_out = 'hz;
```

Листинг кода 22 – Чтение данных памятью (файл MemCTR.sv)

```
task automatic write_mem(int tag, int set,
reg[BIT_CACHE_LINE_SIZE - 1:0] wr);
    C2_out = 3;
    A2_out = (tag << CACHE_SET_SIZE) + set;
    for (int i = 0; i < BIT_CACHE_LINE_SIZE; i +=
DATA2_BUS_SIZE) begin
        D2_out = (wr >> i) % (1 << 8) + (((wr >> (i + 8)) %
(1 << 8)) << 8);
        CLKwaiting(1);
        A2_out = 'hz;
        C2_out = 'hz;
    end
    D2_out = 'hz;

    while (!(C2 === 1)) CLKwaiting(1);
    CLKwaiting(1);
    C2_out = 0;
endtask
```

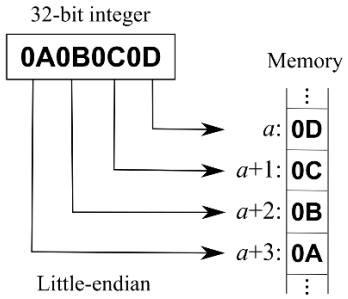
Листинг кода 23 – Передача данных от кэша к памяти (файл Cache.sv)

1. Кэш передает 3 по C2 и set + tag по A2
2. Через такт C2 в кэше становится z, а в памяти 0. Параллельно начинают передаваться данные.
3. После передачи линии кэш ждет пока память не ответит ему 1 (сигнал означает, что запись произошла)
4. Память делает задержку на оставшееся из 100 тактов время и отправляет 1 по C2

Взаимодействие CPU и Cache

Запросы на чтение из кэша

Все данные хранятся в little-endian



То есть, если шина данных может передавать по 2 байта за такт, то для передачи 32х битного числа необходимо считать первые 2 байта, потом следующие 2 прибавить со битовым сдвигом на 16.

```
task read8(reg [CACHE_ADDR_SIZE - 1:0] addr);
    C1_out = 1;
    A1_out = addr >> CACHE_OFFSET_SIZE;
    CLKwaiting(1);
    A1_out = addr % (1 << CACHE_OFFSET_SIZE);
    CLKwaiting(1);
    A1_out = 'hz;
    C1_out = 'hz;
    while (!(C1 === 7)) CLKwaiting(1);
    data = D1;
    CLKwaiting(1);
    C1_out = 0;
endtask
```

Листинг кода 24 – Чтение 1 байта (файл CPU.sv)

```
task read16(reg [CACHE_ADDR_SIZE - 1:0] addr);
   C1_out = 2;
   A1_out = addr >> CACHE_OFFSET_SIZE;
   CLKwaiting(1);
   A1_out = addr % (1 << CACHE_OFFSET_SIZE);
   CLKwaiting(1);
   C1_out = 'hz;
   A1_out = 'hz;
   while (!(C1 === 7)) CLKwaiting(1);
   data = D1;
   CLKwaiting(1);
   C1_out = 0;
endtask
```

Листинг кода 25 – Чтение 2 байтов (файл CPU.sv)

```
task read32(reg [CACHE_ADDR_SIZE - 1:0] addr);
    C1_out = 3;
    A1_out = addr >> CACHE_OFFSET_SIZE;
    CLKwaiting(1);
    A1_out = addr % (1 << CACHE_OFFSET_SIZE);
    CLKwaiting(1);
    A1_out = 'hz;
    C1_out = 'hz;
    while (!(C1 === 7)) CLKwaiting(1);
    data = D1;
    CLKwaiting(1);
    data += D1 << 16;
    CLKwaiting(1);
    C1_out = 0;
endtask
```

Листинг кода 26 – Чтение 4 байтов (файл CPU.sv)

1. Процессор передает соответствующий сигнал на C1 и set + tag на A1.
2. На следующем такте процессор передает offset.
3. На следующем такте процессор ставит z на C1 и A1 и ждет ответ от кэша.
4. После того как пришли данные по D1
    a. Если запрашивался 1 или 2 байта, то data = D1
    b. Если запрашивалось 4 байта, то считываем D1, на следующем такте прибавляем D1 << 16
5. Считанные данные хранятся в глобальной переменной data.

Запросы на запись в кэш

```
task write8(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [7:0] wr);
    C1_out = 5;
    A1_out = addr >> (CACHE_OFFSET_SIZE);
    D1_out = wr;
    CLKwaiting(1);
    D1_out = 'hz;
    A1_out = addr % (1 << CACHE_OFFSET_SIZE);
    CLKwaiting(1);
    D1_out = 'hz;
    A1_out = 'hz;
    C1_out = 'hz;
    while (!(C1 === 7)) CLKwaiting(1);
    CLKwaiting(1);
    C1_out = 0;
endtask
```

Листинг кода 27 – Запись 1 байта в кэш (файл CPU.sv)

```
task write16(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [15:0]
wr);
    C1_out = 6;
    A1_out = addr >> (CACHE_OFFSET_SIZE);
    D1_out = wr;
    CLKwaiting(1);
    D1_out = 'hz;
    A1_out = addr % (1 << CACHE_OFFSET_SIZE);
    CLKwaiting(1);
    D1_out = 'hz;
    C1_out = 'hz;
    A1_out = 'hz;
    while (!(C1 === 7)) CLKwaiting(1);
    CLKwaiting(1);
    C1_out = 0;
endtask
```

Листинг кода 28 – Запись 2 байтов в кэш (файл CPU.sv)

```
task write32(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [31:0]
wr);
    C1_out = 7;
    A1_out = addr >> (CACHE_OFFSET_SIZE);
    D1_out = wr % (1 << 16);
    CLKwaiting(1);
    D1_out = wr >> 16;
    A1_out = addr % (1 << CACHE_OFFSET_SIZE);
    CLKwaiting(1);
    D1_out = 'hz;
    A1_out = 'hz;
    C1_out = 'hz;
    while (!(C1 === 7)) CLKwaiting(1);
    CLKwaiting(1);
    C1_out = 0;
Endtask
```

Листинг кода 29 – Запись 4 байтов в кэш (файл CPU.sv)

1. Процессор передает соответствующий сигнал на C1, set + tag на A1 и данные на D1 (в случае write32 передается 1я половина данных).
2. На следующем такте процессор передает offset (в случае write32 передается 2я половина данных.
3. На следующем такте процессор ставит z на C1, A1 и D1 и ждет ответ от кэша.
4. После того как пришел ответ от кэша ожидаем 1 такт и ставим на C1 0.

Обработка запросов кэшем

```
task invalidate(int pos);
    if (pos != -1 && get_valid(pos) != 0) begin
        set_valid(pos, 0);
        if (get_dirty(pos) == 1) begin
            write_mem(get_tag(pos), get_set(pos),
get_data(pos));
        end
    end
endtask
```

Листинг кода 30 – Инвалидация (файл Cache.sv)

Функция делает инвалидацию линии с номером pos. Если значение dirty этой линии было равно 1, то перезаписываем линию в память.

```
task read8_ans(reg[7:0] wr);
    C1_out = 7;
    D1_out = wr;
    CLKwaiting(1);
    C1_out = 'hz;
    D1_out = 'hz;
endtask

task read16_ans(reg[15:0] wr);
    C1_out = 7;
    D1_out = wr;
    CLKwaiting(1);
    C1_out = 'hz;
    D1_out = 'hz;
endtask

task read32_ans(reg[31:0] wr);
    C1_out = 7;
    D1_out = wr % (1 << DATA1_BUS_SIZE);
    CLKwaiting(1);
    D1_out = wr >> DATA1_BUS_SIZE;
    CLKwaiting(1);
    C1_out = 'hz;
    D1_out = 'hz;
endtask
```

Листинг кода 31 – Передача ответа на запрос чтения из кэша (файл Cache.sv)

В функции передается значение, которое необходимо передать процессору. Если чисто 32х битное, то значение передается за 2 такта.

```
task automatic write8_ans(reg [DATA1_BUS_SIZE - 1:0] data1);
    reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
    for (int i = 0; i < 8; ++i) begin
        line[i + offset * 8] = data1[i];
    end
    set_line(line_number, line);

    C1_out = 7;
    CLKwaiting(1);
    C1_out = 'hz;
endtask

task automatic write16_ans(reg [DATA1_BUS_SIZE - 1:0] data1);
    reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
    for (int i = 0; i < 16; ++i) begin
        line[i + offset * 8] = data1[i];
    end
    set_line(line_number, line);

    C1_out = 7;
    CLKwaiting(1);
    C1_out = 'hz;
endtask

task automatic write32_ans(reg [DATA1_BUS_SIZE - 1:0] data1,
reg [DATA1_BUS_SIZE - 1:0] data2);
    reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
    for (int i = 0; i < 32; ++i) begin
        if (i < 16) line[i + offset * 8] = data1[i];
        else line[i + offset * 8] = data2[i - 16];
    end
    set_line(line_number, line);
    C1_out = 7;
    CLKwaiting(1);
    C1_out = 'hz;
endtask
```

Листинг кода 32 – Передача ответа на запрос записи в кэш (файл Cache.sv)

В функцию передаются данные, полученные от процессора.

Находим линию (Листинг кода 13) и меняем биты на полученные.

Присваиваем измененную линию на текущее место в кэше и отправляем 7 по C1.

```
always @(negedge CLK) begin
    if (C1 === 1 || C1 === 2 || C1 === 3 || C1 === 4 ||
    C1 === 5 || C1 === 6 || C1 === 7) begin
        reg [DATA1_BUS_SIZE - 1:0] data1;
        reg [DATA1_BUS_SIZE - 1:0] data2;
        addr = A1;
        data1 = D1;
        tag = addr >> CACHE_SET_SIZE;
        set = addr % (1 << CACHE_SET_SIZE);
        request = C1;
        CLKwaiting(1);
        if (request == 7) data2 = D1;
        offset = A1;

        CLKwaiting(1);
        C1_out = 0;
        line_number = get_line_number();
```

Листинг кода 33 – Обработка запроса (файл Cache.sv)

На каждом такте считываем и парсим данные, переданные процессором.

В конце на C1 подаем NOP и ищем линию из запроса (Листинг кода 14).

```
if (request == 4) begin
    CLKwaiting(4);
    invalidate(line_number);
end else if (line_number == -1) begin
    CLKwaiting(2);
    cache_miss++;
    read_mem(tag, set);
    line_number = get_replacement();
    invalidate(line_number);
    set_line(line_number, build_new_line());
end else begin
    CLKwaiting(4);
    cache_hit++;
end
```

Листинг кода 34 – Проверка на кэш попадание (файл Cache.sv)

Если приходит запрос на инвалидацию, то после 4х тактов ожидания начинаем выполнять функцию инвалидации (Листинг кода 30)

Если произошел кэш промах, то ждем оставшиеся 2 такта и читаем из памяти по текущему адресу (Листинг кода 20). Находим номер линии, которую надо вытеснить (Листинг кода 15) и делаем инвалидацию (листинг кода 30). Заменяем предыдущую линию на текущую (Листинг кода 13).

Если произошло кэш попадание, то делаем задержку на оставшиеся 4 такта.

```
data = get_data(line_number);

if (request == 1) begin
    read8_ans(get8());
end
if (request == 2) begin
    read16_ans(get16());
end
if (request == 3) begin
    read32_ans(get32());
end
if (request == 5) begin
    write8_ans(data1);
    set_dirty(line_number, 1);
end
if (request == 6) begin
    write16_ans(data1);
    set_dirty(line_number, 1);
end
if (request == 7) begin
    write32_ans(data1, data2);
    set_dirty(line_number, 1);
end
upd_time();
```

Листинг кода 35 – Обработка каждого запроса (файл Cache.sv)

Для ответа на запросы чтения используем функции из листинга кода 17 и 31.

Для ответа на запросы записи используем функции из листинга кода 32 и 13.

В конце помечаем текущую линию как последнюю использованную (Листинг кода 12).

Воспроизведение задачи на Verilog

```
pa = A_BEGIN;
CLKwaiting(1);
pc = C_BEGIN;
CLKwaiting(1);
CLKwaiting(1);
for (int y = 0; y < M; y++) begin
    CLKwaiting(1);
    for (int x = 0; x < N; x++) begin
        pb = B_BEGIN;
        CLKwaiting(1);
        s = 0;
        CLKwaiting(1);

        CLKwaiting(1);
        for (int k = 0; k < K; k++) begin
            CLKwaiting(1);
            read8(pa + k * A_STEP);
            a_el = data;
            CLKwaiting(1);
            read16(pb + x * B_STEP);
            b_el = data;
            CLKwaiting(6); //math
            s += a_el * b_el;
            // s += pa[k] * pb[x];
            CLKwaiting(1);
            pb += N * B_STEP;
            CLKwaiting(1); //iteration
        end
        // pc[x] = s;
        write32(pc + x * C_STEP, s);
        CLKwaiting(2);
    end
    pa += K * A_STEP;
    pc += N * C_STEP;
    CLKwaiting(3); //iteration
end
CLKwaiting(1); //return
```

Листинг кода 36 – Модель задачи на языке SystemVerilog (файл CPU.sv)

После подсчета тактов, кэш промахов и кэш попаданий получаются следующие значения:

- Clocks – 7737315
- Cache hit – 213761
- Cache miss – 35839
- Cache hit chance – 85.6414%

После обработки всей функции можно положить все посчитанные значения из кэша в память, но т.к. это не указано в тз, то я посчитал это излишним.

Сравнение полученных результатов

Количество тактов отличается незначительно (на 205).

Количество кэш промахов и кэш попаданий совпадает полностью.

Листинг кода

test_bench.sv

```
// CLS && iverilog -g2012 -o a.out test_bench.sv && vvp a.out
`include "Cache.sv"
`include "CPU.sv"
`include "MemCTR.sv"


module Test_bench;
    parameter MEM_SIZE = 1048576;
    parameter CACHE_SIZE = 1024;
    parameter CACHE_LINE_SIZE = 32;
    parameter CACHE_LINE_COUNT = 32;
    parameter CACHE_WAY = 2;
    parameter CACHE_SETS_COUNT = 16;
    parameter CACHE_TAG_SIZE = 11;
    parameter CACHE_SET_SIZE = 4;
    parameter CACHE_OFFSET_SIZE = 5;
    parameter CACHE_ADDR_SIZE = 20;
    parameter ADDR1_BUS_SIZE = 15;
    parameter ADDR2_BUS_SIZE = 15;
    parameter DATA1_BUS_SIZE = 16;
    parameter DATA2_BUS_SIZE = 16;
    parameter CTR1_BUS_SIZE = 3;
    parameter CTR2_BUS_SIZE = 2;

    wire[ADDR1_BUS_SIZE - 1:0] A1;
    wire[ADDR2_BUS_SIZE - 1:0] A2;
    wire[DATA1_BUS_SIZE - 1:0] D1;
    wire[DATA2_BUS_SIZE - 1:0] D2;
    wire[CTR1_BUS_SIZE - 1:0] C1;
    wire[CTR2_BUS_SIZE - 1:0] C2;
```

```
    reg C_DUMP;
    reg M_DUMP;

    reg RESET = 0;

    reg CLK = 0;

    CPU cpu(
        .A1(A1),
        .C1(C1),
        .D1(D1),
        .M_DUMP(M_DUMP),
        .C_DUMP(C_DUMP),

        .CLK(CLK)
    );

    Cache cache(
        .C1(C1),
        .C2(C2),
        .D1(D1),
        .D2(D2),
        .A2(A2),

        .CLK(CLK),
        .RESET(RESET),
        .C_DUMP(C_DUMP),
        .A1(A1)
    );

    MemCTR mem(
        .D2(D2),
        .C2(C2),

        .M_DUMP(M_DUMP),
        .RESET(RESET),
        .CLK(CLK),
        .A2(A2)
    );

    task automatic CLKwaiting(int cnt);
        for (int i = 0; i < cnt; ++i) begin
            @(negedge CLK);
        end
    endtask

    initial begin
    end

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

```
    end

endmodule
```

CPU.sv

```
// CLS && iverilog -g2012 -o a.out CPU.sv && vvp a.out

module CPU #(
    parameter MEM_SIZE = 1048576,
    parameter CACHE_SIZE = 1024,
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
)

(
    output wire [ADDR1_BUS_SIZE - 1:0] A1,
    inout wire [CTR1_BUS_SIZE - 1:0] C1,
    inout wire [DATA1_BUS_SIZE - 1:0] D1,

    output reg M_DUMP,
    output reg C_DUMP,

    input reg CLK
);

    reg [ADDR1_BUS_SIZE - 1:0] A1_out = 'hz;
    reg [CTR1_BUS_SIZE - 1:0] C1_out = 'hz;
    reg [DATA1_BUS_SIZE - 1:0] D1_out = 'hz;

    assign A1 = A1_out;
    assign C1 = C1_out;
    assign D1 = D1_out;

    parameter M = 64;
    parameter N = 60;
    parameter K = 32;
    parameter A_BEGIN = 0;
    parameter B_BEGIN = M * K * A_STEP;
```

```
    parameter C_BEGIN = B_BEGIN + K * N * B_STEP;
    parameter A_STEP = 1;
    parameter B_STEP = 2;
    parameter C_STEP = 4;

    int data;
    int a_el;
    int b_el;
    int pa, pb, pc;
    int i;
    int s;
    int clk_counter = 0;

    task CLKwaiting(int cnt);
        for (i = 0; i < cnt; ++i) begin
            @(negedge CLK);
        end
    endtask

    task read8(reg [CACHE_ADDR_SIZE - 1:0] addr);
        C1_out = 1;
        A1_out = addr >> CACHE_OFFSET_SIZE;
        CLKwaiting(1);
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        A1_out = 'hz;
        C1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        data = D1;
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task read16(reg [CACHE_ADDR_SIZE - 1:0] addr);
        C1_out = 2;
        A1_out = addr >> CACHE_OFFSET_SIZE;
        CLKwaiting(1);
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        C1_out = 'hz;
        A1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        data = D1;
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task read32(reg [CACHE_ADDR_SIZE - 1:0] addr);
        C1_out = 3;
        A1_out = addr >> CACHE_OFFSET_SIZE;
        CLKwaiting(1);
```

```
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        A1_out = 'hz;
        C1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        data = D1;
        CLKwaiting(1);
        data += D1 << 16;
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task invalidate(reg [CACHE_ADDR_SIZE - 1:0] addr);
        C1_out = 4;
        A1_out = addr >> (CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        C1_out = 'hz;
        A1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task write8(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [7:0]
wr);
        C1_out = 5;
        A1_out = addr >> (CACHE_OFFSET_SIZE);
        D1_out = wr;
        CLKwaiting(1);
        D1_out = 'hz;
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        D1_out = 'hz;
        A1_out = 'hz;
        C1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task write16(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [15:0]
wr);
        C1_out = 6;
        A1_out = addr >> (CACHE_OFFSET_SIZE);
        D1_out = wr;
        CLKwaiting(1);
        D1_out = 'hz;
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
```

```
        D1_out = 'hz;
        C1_out = 'hz;
        A1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task write32(reg [CACHE_ADDR_SIZE - 1:0] addr, reg [31:0]
wr);
        C1_out = 7;
        A1_out = addr >> (CACHE_OFFSET_SIZE);
        D1_out = wr % (1 << 16);
        CLKwaiting(1);
        D1_out = wr >> 16;
        A1_out = addr % (1 << CACHE_OFFSET_SIZE);
        CLKwaiting(1);
        D1_out = 'hz;
        A1_out = 'hz;
        C1_out = 'hz;
        while (!(C1 === 7)) CLKwaiting(1);
        CLKwaiting(1);
        C1_out = 0;
    endtask

    task dump_cache();
        C_DUMP = 1;
        CLKwaiting(1);
        C_DUMP = 0;
        CLKwaiting(1);
    endtask
    task dump_mem();
        M_DUMP = 1;
        CLKwaiting(1);
        M_DUMP = 0;
        CLKwaiting(1);
    endtask

    initial begin
        $display("CPU");

        C_DUMP = 0;
        M_DUMP = 0;
        C1_out = 0;

        pa = A_BEGIN;
        CLKwaiting(1);
        pc = C_BEGIN;
        CLKwaiting(1);
        CLKwaiting(1);
        for (int y = 0; y < M; y++) begin
```

```
            CLKwaiting(1);
            for (int x = 0; x < N; x++) begin
                pb = B_BEGIN;
                CLKwaiting(1);
                s = 0;
                CLKwaiting(1);

                CLKwaiting(1);
                for (int k = 0; k < K; k++) begin
                    CLKwaiting(1);
                    read8(pa + k * A_STEP);
                    a_el = data;
                    CLKwaiting(1);
                    read16(pb + x * B_STEP);
                    b_el = data;

                    // $display("addr a = %0d, addr b = %0d",
pa + k * A_STEP, pb + x * B_STEP);
                    // $display("a = %0d, b = %0d", a_el,
b_el);

                    CLKwaiting(6); //math
                    s += a_el * b_el;
                    // s += pa[k] * pb[x];
                    CLKwaiting(1);
                    pb += N * B_STEP;
                    CLKwaiting(1); //iteration
                end
                // pc[x] = s;
                // $display("c = %0d, addr = %0b", s, pc + x
* C_STEP);
                write32(pc + x * C_STEP, s);
                CLKwaiting(2);
            end
            pa += K * A_STEP;
            pc += N * C_STEP;
            CLKwaiting(3); //iteration
        end

        // for (int i = 0; i < M; ++i) begin
        //     for (int j = 0; j < N; ++j) begin
        //         read32(C_BEGIN + (i * N + j) * C_STEP);
        //         $display("c = %0d, addr = %0b", data,
C_BEGIN + (i * N + j) * C_STEP);
        //     end
        //     $finish();
        // end

        $display("Clocks = %0d", clk_counter);
        dump_cache();
        $display("FINISH time = %0t", $time());
```

```
        $finish();
    end

    always @(negedge CLK) begin
        clk_counter++;
    end

endmodule
```

MemCTR.sv

```
// CLS && iverilog -g2012 -o a.out MemCTR.sv && vvp a.out

module MemCTR #(
    parameter MEM_SIZE = 1048576,
    parameter CACHE_SIZE = 1024,
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
)

(
    inout wire [DATA2_BUS_SIZE - 1:0] D2,
    input wire [CTR2_BUS_SIZE - 1:0] C2,

    input reg M_DUMP,
    input reg RESET,
    input reg CLK,
    input wire [ADDR2_BUS_SIZE - 1:0] A2
);

    reg [DATA2_BUS_SIZE - 1:0] D2_out = 'hz;
    reg [CTR2_BUS_SIZE - 1:0] C2_out = 'hz;

    assign D2 = D2_out;
    assign C2 = C2_out;

    parameter _SEED = 225526;

    integer SEED = _SEED;
```

```
    reg[7:0] ram[0:MEM_SIZE - 1];
    integer i = 0;
    integer fd;
    reg[CACHE_ADDR_SIZE:0] addr;

    logic write = 0;
    int curWrite;
    int endWrite;
    logic read = 0;
    int curRead;
    int endRead;

    int delay;

    initial begin
        $display("RAM");
        reset();
        // dump();

        // for (i = 0; i < MEM_SIZE; i += 1) begin
        //     $display("[%d] %d", i, ram[i]);
        // end
    end

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

    task automatic dump();
        fd = $fopen("RAMdump.ext", "w");
        for (i = 0; i < MEM_SIZE; i++) begin
            $fdisplay(fd, "[%d] %b", i, ram[i]);
        end
        $fclose(fd);
    endtask

    task automatic CLKwaiting(int cnt);
        for (int i = 0; i < cnt; ++i) begin
            @(negedge CLK);
        end
    endtask

    always @(negedge CLK) begin
        if (RESET == 1) reset();
        if (M_DUMP == 1) dump();

    end

    always @(negedge CLK) begin
```

```
        if (C2 === 2 || C2 === 3) begin
            addr = A2 << CACHE_OFFSET_SIZE;
            if (C2 == 2) begin
                CLKwaiting(1);
                C2_out = 0;
                CLKwaiting(99);
                C2_out = 1;
                for (i = addr; i + 1 < addr +
CACHE_LINE_SIZE; i += 2) begin
                    D2_out = ram[i] + (ram[i + 1] << 8);
                    CLKwaiting(1);
                end
                D2_out = 'hz;
                C2_out = 'hz;
            end else begin
                delay = 100;
                i = addr;
                for (int j = 0; j < 16; ++j) begin
                    ram[i] = D2 % (1 << 8);
                    ram[i + 1] = D2 >> 8;
                    CLKwaiting(1);
                    C2_out = 0;
                    delay--;
                    i += 2;
                end
                C2_out = 0;
                CLKwaiting(delay);
                C2_out = 1;
                CLKwaiting(1);
                C2_out = 'hz;
            end
        end
    end

endmodule
```

Cache.sv

```
// CLS && iverilog -g2012 -o a.out Cache.sv && vvp a.out

module Cache #(
    parameter MEM_SIZE = 1048576,
    parameter CACHE_SIZE = 1024,
    parameter CACHE_LINE_SIZE = 32,
    parameter CACHE_LINE_COUNT = 32,
    parameter CACHE_WAY = 2,
    parameter CACHE_SETS_COUNT = 16,
    parameter CACHE_TAG_SIZE = 11,
    parameter CACHE_SET_SIZE = 4,
    parameter CACHE_OFFSET_SIZE = 5,
    parameter CACHE_ADDR_SIZE = 20,
```

```
    parameter ADDR1_BUS_SIZE = 15,
    parameter ADDR2_BUS_SIZE = 15,
    parameter DATA1_BUS_SIZE = 16,
    parameter DATA2_BUS_SIZE = 16,
    parameter CTR1_BUS_SIZE = 3,
    parameter CTR2_BUS_SIZE = 2
)

(
    inout wire [0:CTR1_BUS_SIZE - 1] C1,
    inout wire [0:CTR2_BUS_SIZE - 1] C2,
    inout wire [0:DATA1_BUS_SIZE - 1] D1,
    inout wire [0:DATA2_BUS_SIZE - 1] D2,
    output wire [0:ADDR2_BUS_SIZE - 1] A2,

    input reg CLK,
    input reg RESET,
    input reg C_DUMP,
    input wire [ADDR1_BUS_SIZE - 1:0] A1
);


    reg [0:CTR1_BUS_SIZE - 1] C1_out = 'hz;
    reg [0:CTR2_BUS_SIZE - 1] C2_out = 'hz;
    reg [0:DATA1_BUS_SIZE - 1] D1_out = 'hz;
    reg [0:DATA2_BUS_SIZE - 1] D2_out = 'hz;
    reg [0:ADDR2_BUS_SIZE - 1] A2_out = 'hz;

    assign C1 = C1_out;
    assign C2 = C2_out;
    assign D1 = D1_out;
    assign D2 = D2_out;
    assign A2 = A2_out;

    parameter BIT_CACHE_LINE_SIZE = CACHE_LINE_SIZE * 8;
    parameter LINE_IN_CACHE_SIZE = 2 + CACHE_TAG_SIZE +
BIT_CACHE_LINE_SIZE;

    reg [CACHE_ADDR_SIZE - 1:0] addr;
    reg [CACHE_TAG_SIZE - 1:0] tag;
    reg [CACHE_SET_SIZE - 1:0] set;
    reg [CACHE_OFFSET_SIZE - 1:0] offset;
    reg [BIT_CACHE_LINE_SIZE - 1:0] data;
    reg [LINE_IN_CACHE_SIZE - 1:0] cache[0:CACHE_LINE_COUNT -
1];
    reg [CACHE_LINE_COUNT - 1:0] times;
    reg [CTR1_BUS_SIZE - 1:0] request;

    int line_number;
    int shift;
    integer fd;
```

```
    int i;
    int cache_hit = 0;
    int cache_miss = 0;

    task dump();
        $display("hit = %0d, miss = %0d", cache_hit,
cache_miss);
        fd = $fopen("CacheDump.ext", "w");
        for (i = 0; i < CACHE_LINE_COUNT; i++) begin
            $fdisplay(fd, "[%d] %b", i, cache[i]);
        end
        $fclose(fd);
    endtask

    task reset();
        C2_out = 0;
        for (i = 0; i < CACHE_LINE_COUNT; ++i) begin
            cache[i] = 0;
            times[i] = 0;
        end
    endtask

    initial begin
        $display("Cache");
        reset();
    end

    always @(negedge CLK) begin
        if (C_DUMP) dump();
        if (RESET) reset();
    end

    task automatic upd_time();
        for (int i = set * CACHE_WAY; i < (set + 1) *
CACHE_WAY; ++i) begin
            if (get_tag(i) != tag) times[i] = 0;
            else times[i] = 1;
        end
    endtask

    function reg[LINE_IN_CACHE_SIZE - 1:0] get_line(int pos);
        return cache[pos];
    endfunction

    function automatic reg[CACHE_TAG_SIZE - 1:0] get_tag(int
pos);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        return (line >> BIT_CACHE_LINE_SIZE) % (1 <<
CACHE_TAG_SIZE);
    endfunction
```

```
    function reg[CACHE_TAG_SIZE - 1:0] get_set(int pos);
        return pos / CACHE_WAY;
    endfunction

    function automatic reg[BIT_CACHE_LINE_SIZE - 1:0]
get_data(int pos);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        return line % (1 << BIT_CACHE_LINE_SIZE);
    endfunction

    function automatic reg get_dirty(int pos);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        return (line >> (BIT_CACHE_LINE_SIZE +
CACHE_TAG_SIZE)) % 2;
    endfunction

    function automatic reg get_valid(int pos);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        return line >> (BIT_CACHE_LINE_SIZE + CACHE_TAG_SIZE
+ 1);
    endfunction

    task set_line(int pos, reg[LINE_IN_CACHE_SIZE - 1:0]
line);
        cache[pos] = line;
    endtask

    task automatic set_valid(int pos, reg valid);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        line[LINE_IN_CACHE_SIZE - 1] = valid;
        set_line(pos, line);
    endtask

    task automatic set_dirty(int pos, reg dirty);
        reg[LINE_IN_CACHE_SIZE - 1:0] line = get_line(pos);
        line[LINE_IN_CACHE_SIZE - 2] = dirty;
        set_line(pos, line);
    endtask

    function int get_line_number();
        for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1);
i++) begin
            if (get_valid(i) != 0 && get_tag(i) == tag)
return i;
        end
        return -1;
    endfunction

    function automatic int get_replacement();
        for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1);
i++) begin
```

```
            if (get_valid(i) == 0) return i;
        end

        for (i = CACHE_WAY * set; i < CACHE_WAY * (set + 1);
i++) begin
            if (times[i] == 0) return i;
        end
    endfunction

    function reg[LINE_IN_CACHE_SIZE - 1:0] build_new_line();
        return data + (tag << BIT_CACHE_LINE_SIZE) + (1 <<
(BIT_CACHE_LINE_SIZE + CACHE_TAG_SIZE + 1));
    endfunction

    function reg[7:0] get8();
        return (data >> offset * 8) % (1 << 8);
    endfunction
    function reg[15:0] get16();
        return (data >> offset * 8) % (1 << 16);
    endfunction
    function reg[31:0] get32();
        return (data >> offset * 8) % (1 << 32);
    endfunction

    task CLKwaiting(int cnt);
        for (i = 0; i < cnt; ++i) begin
            @(negedge CLK);
        end
    endtask

    task invalidate(int pos);
        if (pos != -1 && get_valid(pos) != 0) begin
            set_valid(pos, 0);
            if (get_dirty(pos) == 1) begin
                write_mem(get_tag(pos), get_set(pos),
get_data(pos));
            end
        end
    endtask

    task automatic write_mem(int tag, int set,
reg[BIT_CACHE_LINE_SIZE - 1:0] wr);
        C2_out = 3;
        A2_out = (tag << CACHE_SET_SIZE) + set;
        for (int i = 0; i < BIT_CACHE_LINE_SIZE; i +=
DATA2_BUS_SIZE) begin
            D2_out = (wr >> i) % (1 << 8) + (((wr >> (i +
8)) % (1 << 8)) << 8);
            CLKwaiting(1);
            A2_out = 'hz;
            C2_out = 'hz;
```

```
        end
        D2_out = 'hz;

        while (!(C2 === 1)) CLKwaiting(1);
        CLKwaiting(1);
        C2_out = 0;
    endtask

    task automatic read_mem(int tag, int set);
        C2_out = 2;
        A2_out = (tag << CACHE_SET_SIZE) + set;
        CLKwaiting(1);
        A2_out = 'hz;
        C2_out = 'hz;
        while (!(C2 === 1)) CLKwaiting(1);
        data = 0;
        shift = 0;
        for (int x = 0; x < 16; ++x) begin
            data += (D2 << shift);
            shift += DATA2_BUS_SIZE;
            CLKwaiting(1);
        end
        C1_out = 0;
    endtask

    task read8_ans(reg[7:0] wr);
        C1_out = 7;
        D1_out = wr;
        CLKwaiting(1);
        C1_out = 'hz;
        D1_out = 'hz;
    endtask

    task read16_ans(reg[15:0] wr);
        C1_out = 7;
        D1_out = wr;
        CLKwaiting(1);
        C1_out = 'hz;
        D1_out = 'hz;
    endtask

    task read32_ans(reg[31:0] wr);
        C1_out = 7;
        D1_out = wr % (1 << DATA1_BUS_SIZE);
        CLKwaiting(1);
        D1_out = wr >> DATA1_BUS_SIZE;
        CLKwaiting(1);
        C1_out = 'hz;
        D1_out = 'hz;
    endtask
```

```
    task automatic write8_ans(reg [DATA1_BUS_SIZE - 1:0]
data1);
        reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
        for (int i = 0; i < 8; ++i) begin
            line[i + offset * 8] = data1[i];
        end
        set_line(line_number, line);

        C1_out = 7;
        CLKwaiting(1);
        C1_out = 'hz;
    endtask

    task automatic write16_ans(reg [DATA1_BUS_SIZE - 1:0]
data1);
        reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
        for (int i = 0; i < 16; ++i) begin
            line[i + offset * 8] = data1[i];
        end
        set_line(line_number, line);

        C1_out = 7;
        CLKwaiting(1);
        C1_out = 'hz;
    endtask

    task automatic write32_ans(reg [DATA1_BUS_SIZE - 1:0]
data1, reg [DATA1_BUS_SIZE - 1:0] data2);
        reg[LINE_IN_CACHE_SIZE - 1:0] line =
get_line(line_number);
        for (int i = 0; i < 32; ++i) begin
            if (i < 16) line[i + offset * 8] = data1[i];
            else line[i + offset * 8] = data2[i - 16];
        end
        set_line(line_number, line);
        C1_out = 7;
        CLKwaiting(1);
        C1_out = 'hz;
    endtask

    always @(negedge CLK) begin
        if (C1 === 1 || C1 === 2 || C1 === 3 || C1 === 4 ||
        C1 === 5 || C1 === 6 || C1 === 7) begin
            reg [DATA1_BUS_SIZE - 1:0] data1;
            reg [DATA1_BUS_SIZE - 1:0] data2;
            addr = A1;
            data1 = D1;
            tag = addr >> CACHE_SET_SIZE;
            set = addr % (1 << CACHE_SET_SIZE);
```

```
            request = C1;
            CLKwaiting(1);
            if (request == 7) data2 = D1;
            offset = A1;

            CLKwaiting(1);
            C1_out = 0;
            line_number = get_line_number();


            if (request == 4) begin
                CLKwaiting(4);
                invalidate(line_number);
            end else if (line_number == -1) begin
                CLKwaiting(2);
                cache_miss++;
                read_mem(tag, set);
                line_number = get_replacement();
                invalidate(line_number);
                set_line(line_number, build_new_line());
            end else begin
                CLKwaiting(4);
                cache_hit++;
            end

            data = get_data(line_number);

            if (request == 1) begin
                read8_ans(get8());
            end
            if (request == 2) begin
                read16_ans(get16());
            end
            if (request == 3) begin
                read32_ans(get32());
            end
            if (request == 5) begin
                write8_ans(data1);
                set_dirty(line_number, 1);
            end
            if (request == 6) begin
                write16_ans(data1);
                set_dirty(line_number, 1);
            end
            if (request == 7) begin
                write32_ans(data1, data2);
                set_dirty(line_number, 1);
            end
            upd_time();
        end
    end
```

```
endmodule
```

cache_simulation.cpp

```
#include <bits/stdc++.h>

using namespace std;

#define _ << " " <<

const int CACHE_LINE_SIZE = 1 << 5;
const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 1 << 4;
const int CACHE_SET_SIZE = 4;
const int CACHE_OFFSET_SIZE = 5;

const int DATA1_BUS_SIZE = 2;
const int DATA2_BUS_SIZE = 2;

//int8 a[M][K];
//int16 b[K][N];
//int32 c[M][N];

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

const int A_STEP = 1;
const int B_STEP = 2;
const int C_STEP = 4;

const int A = 0;
const int B = M * K * A_STEP;
const int C = B + K * N * B_STEP;

int timer = 1;

int cache_tags[CACHE_SETS_COUNT][CACHE_WAY];
int cache_times[CACHE_SETS_COUNT][CACHE_WAY];
int cache_valid[CACHE_SETS_COUNT][CACHE_WAY];
int cache_dirty[CACHE_SETS_COUNT][CACHE_WAY];

int tacts_counter = 0;

int cache_miss = 0;
int cache_hit = 0;

bool check_entry(int set_num, int tag, bool write) {
    for (int i = 0; i < CACHE_WAY; ++i) {
        if (cache_tags[set_num][i] == tag) {
```

```
            cache_times[set_num][i] = timer++;
            if (write) cache_dirty[set_num][i] = 1;
            return true;
        }
    }
    return false;
}

void cache_replacement(int set_num, int tag, bool write) {
    int imn = 0;
    for (int i = 0; i < CACHE_WAY; ++i) {
        if (cache_valid[set_num][i] == 0 ||
cache_times[set_num][i] < cache_times[set_num][imn]) {
            imn = i;
        }
    }
    cache_times[set_num][imn] = timer++;
    if (cache_dirty[set_num][imn] == 1) {
        tacts_counter += 101;
    }
    cache_valid[set_num][imn] = 1;
    cache_dirty[set_num][imn] = 0;
    if (write) cache_dirty[set_num][imn] = 1;
    cache_tags[set_num][imn] = tag;
}

void memory_request(int addr, int bytes, bool write) {
    addr = addr >> CACHE_OFFSET_SIZE;
    int set_num = addr % (1 << CACHE_SET_SIZE);
    int tag = addr >> CACHE_SET_SIZE;

    if (check_entry(set_num, tag, write)) {
        tacts_counter += 7 + (bytes + 1) / 2;
        cache_hit++;
    } else {
        cache_miss++;
        cache_replacement(set_num, tag, write);
        tacts_counter += 106 + CACHE_LINE_SIZE /
DATA2_BUS_SIZE + (bytes + 1) / 2;
    }
}

void process() {
    for (int i = 0; i < CACHE_SETS_COUNT; ++i) {
        for (int j = 0; j < CACHE_WAY; ++j) {
            cache_tags[i][j] = -1;
        }
    }
//    int8 *pa = a;
    int pa = A;
    tacts_counter++;
```

```
//    int32 *pc = c;
    int pc = C;
    tacts_counter++;

    tacts_counter++; // int y = 0
    for (int y = 0; y < M; y++)
    {
        tacts_counter++; // int x = 0
        for (int x = 0; x < N; x++)
        {
//            int16 *pb = b;
            int pb = B;
            tacts_counter++;
//            int32 s = 0;
            tacts_counter++;

            tacts_counter++; // int k = 0;
            for (int k = 0; k < K; k++)
            {
//                s += pa[k] * pb[x];
                tacts_counter++; // +=

                tacts_counter += 5; // *

                tacts_counter++; // pa + k
                memory_request(pa + A_STEP * k, A_STEP, 0);

                tacts_counter++; // pb + x;
                memory_request(pb + B_STEP * x, B_STEP, 0);

                pb += N * B_STEP;
                tacts_counter++;

                tacts_counter++; // k++
            }
//            pc[x] = s;
            tacts_counter++; // pc + x
            memory_request(pc + C_STEP * x, C_STEP, 1);

            tacts_counter++; // x++
        }
        pa += K * A_STEP;
        tacts_counter++;

        pc += N * C_STEP;
        tacts_counter++;

        tacts_counter++; // y++
    }
}
```

```
signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    process();

    cout << "Memory request count:" _ cache_miss + cache_hit
<< endl;
    cout << "Cache miss:" _ cache_miss << endl;
    cout << "Cache hit:" _ cache_hit << endl;
    cout << fixed << setprecision(4) << "Statistic:" _
    (double)cache_hit / (double)(cache_hit + cache_miss) *
100 << "%" << endl;
    cout << "Tacts count:" _ tacts_counter << endl;
}
```