Лабораторная работа №2	Б10	2022
Моделирование схем в Verilog	Пешков Артём Вадимович	

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

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 12 (iverilog-v12-20220611-x64).

Описание

Дана задача и некоторые параметры системы. Необходимо сначала аналитически решить поставленную задачу: на основе параметров системы вычислить время в тактах, затраченное на исполнение данной в условии функции, определить общее количество обращений в память, а также количество и процент кэш-попаданий. Также необходимо промоделировать систему с заданными параметрами на языке verilog, промоделировать задачу в этой системе и сравнить результаты с результатами аналитического решения.

Вариант

В качестве варианта вам даны некоторые параметры системы и задача, которую надо сначала решить аналитически на основании параметров системы, а затем промоделировать и сравнить полученные результаты.

Вычисление недостающих параметров системы.

Вычисления общего размера полезных данных кэша.

$CACHE\_SIZE = CACHE\_LINE\_SIZE * CACHE\_LINE\_COUNT = 16 * 64 = 1024$ байта

Вычисление параметров интерпретации адреса кэшем.

Размер адреса памяти равен 19 битам ($log(MEM\_SIZE$ в байтах)), так как общий размер памяти 512 Кб, что равно 2^{19} байт.
Значит в интерпретации адреса кэшем
$CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE + CACHE\_OFFSET\_SIZE = 19.$
$CACHE\_OFFSET\_SIZE$ - это смещение внутри одной кэш-линии, поскольку размер кэш-линии равен 16 байт, $CACHE\_OFFSET\_SIZE = 4$ бита.
$CACHE\_SET\_SIZE$ можно вычислить двумя способами:
$CACHE\_SET\_SIZE = log(CACHE\_LINE\_COUNT / CACHE\_WAY) = 5$ бит
или
$CACHE\_SET\_SIZE = 19 - CACHE\_TAG\_SIZE - CACHE\_OFFSET\_SIZE = 5$ бит

Вычисление размеров шин.

По шинам адреса **ADDR1_BUS_SIZE** и **ADDR2_BUS_SIZE** мы хотим передавать адрес между процессором и кэшем (**A1**), передавая tag + set за первый такт и offset за второй, и адрес между кэшем и памятью(**A2**), передавая tag+set за один такт (offset не нужен, так как кэш и память общаются кэш-линиями). Максимум мы передаём tag + set, а значит 15 бит для шин **A1** и **A2** будет достаточно (**ADDR1_BUS_SIZE** = 15 бит, **ADDR2_BUS_SIZE** = 15 бит).

По шине **C1** передаются команды, кодируемые числами от 0 до 7, а значит достаточно 3 бита, чтобы закодировать команды, идущие по этой шине. По шине **C2** передаются команды, кодируемые числами от 0 до 3, а значит достаточно 2 бита, чтобы закодировать команды, идущие по этой шине. **CTR1_BUS_SIZE** = 3 бита, **CTR2_BUS_SIZE** = 2 бита.

Аналитическое решение задачи.

Реализация.

Для решения я выбрал язык Kotlin. Для решения аналитической задачи было достаточно прописать кэш и логику его работы с адресами, а также переписать код задачи. Далее реализация кэша.

Кэш хранит в себе список из $2^{CACHE\_SET\_SIZE}$ кэш линий, в каждой линии есть бит валидности, бит dirty и тэг (для аналитического решения полезные данные не нужны). Кэш-линии реализованы отдельным data классом.

Адрес, который мы передаём - это тоже отдельный data класс, который хранит в себе tag и set (offset в аналитическом решении опять же не нужен)

Как только кэшу поступает запрос на чтение/запись по какому-то адресу, он проверяет, есть ли в его внутреннем массиве на индексе соответствующем переданному set-у кэш-линия с tag-ом, совпадающим с переданным tag-ом, если такая линия есть, мы запоминаем её индекс и увеличиваем счётчик кэш-хитов.

Если кэш не нашёл линию внутри себя, мы обрабатываем кэш-miss. Увеличиваем счётчик кэш-миссов. Для всех set-ов мы храним последнюю использованную в нём кэш-линию, тогда для переданного set-а выберем ту из двух линий, которую мы дольше не использовали и считаем данные из памяти в неё, но перед этим в случае если линия валидна (valid = true) и является dirty (dirt = true), нам необходимо выгрузить данные этой линии в память и только потом считать в неё данные (в аналитическом решении это влияет только на то, сколько раз и на сколько мы увеличиваем общее число тиков).

После обработки кэш-hit/кэш-miss нам остаётся либо записать данные, пометив линию dirty = true, либо считать данные и отправить их процессору.

Осталось только проставить задержки в основной функции и в кэше согласно условию и параметрам системы и вывести необходимые данные на экран.

Результат аналитического решения.

Total ticks: 5307782

Total memory accesses: 249600

Cache hits: 228080

Success hits (%): 91.37820512820512

Моделирование заданной системы на Verilog.

Общие факты.

Наша система состоит из 3 элементов: процессор, кэш и память, между которыми находятся несколько шин по которым происходит общение между элементами и всё это работает по общей синхронизации. Для удобства я завёл такой инвариант: если clk = 1, устройства могут записывать данные на шины, если clk = 0, устройства могут только читать с шин. Записывает всегда устройство владеющее шиной, а читает то, которое в данный момент шиной не владеет. Такой подход позволяет избежать гонок, ведь если устройство пишет на шину, а другое пытается прочитать с шины, может произойти dirty read.

Все устройства действуют по-очереди, процессор делает запрос, кэш его обрабатывает, делает запрос к памяти, если это необходимо, память обрабатывает запрос, отдаёт результат кэшу, а кэш возвращает результат процессору, потом владение шинами возвращается в первоначальное состояние и на шинах команд устанавливаются NOP. Далее подробнее поговорим о каждом модуле и что в нём происходит. Говоря об отдельных модулях в начале будут перечислены основные переменные и к ним будет краткое пояснение. Про остальные переменные либо будет сказано при описании частей в которых они используются, либо они были введены для удобства/упрощения кода.

При инициализации модулей процессора, кэша и памяти мы (там где необходимо) передаём им провода отвечающие за синхронизацию, dump-ы, reset и шины (a1, c1, d1, a2, c2, d2). Шины a - обычные input/output

(т. к. являются однонаправленными), шины c и d - inout (т. к. они двунаправленные).

Реализация процессора (модуль cpu, файл cpu.sv)

Основные переменные:

command - отвечает за команды, которые cpu посылает кэшу
data_to_send, addr_to_send - данные, которые передаются по шинам
finished_query - меняется с 0 на 1, когда на последний отправленный запрос был получен ответ
is_owning - говорит владеет ли процессор шинами в данный момент

Реализация:

Основная часть логики находится в always блоке, который реагирует на изменение clk, проверяя внутри себя владеет ли cpu шинами d1, c1. Если clk = 1 и is_owning = 1, то процессор может отправлять команду и данные по шинам, если clk и is_owning = 0 - читать информация с шин.

В начале всех команд мы разбиваем адрес на 2 части (tag + set и offset), после чего, если command = числу от 0 до 7, отправляем по шинам нужную команду, адрес и данные (если необходимо).

Чтобы принимающая сторона успевала обработать отправленные данные, после отправки данных выставляется задержка, которая блокирует always блок на нужное число тактов, не давая ему менять значения на шинах раньше времени (подробнее про задержку отдельно). Когда отправка очередной команды окончена, процессор ставит is_owning = 0, command = -1, а на шины ставится высокоимпидансное состояние, чтобы владеющее устройство могло изменять данные на шинах.

Если процессор не владеет шинами, он ждёт, пока кэш поставит на шину c1 команду C1_RESPONSE, что будет означать, что кэш начал выдавать ответ на последний запрос. Если данные передаются за несколько тактов, в нужных местах выставляются задержки, чтобы кэш успевал записывать данные на шину, а процессор - читать. После считывания всех данных процессор возвращает себе владение шиной устанавливая command = 0 и is_owning = 1.

Команды с шины c1 не снимаются до момента завершения отправки данных.

Происходящее в initial блоке будет описано, когда мы будем говорить о симуляции функции из задачи.

Реализация кэша (модуль cache, файл cache.sv)

Немного про кэш:

Основная задача кэша - ускорение доступа к памяти. Обращение к ram - дорогая операция, т.к. ram находится далеко от процессора, поэтому рядом с процессором расположен кэш, который гораздо меньше памяти по объёму хранимых данных, зато обращение к нему работает гораздо быстрее. В нашей модели реализован look through (при каждом обращении в память процессор сначала ищет в кэше), write back(данные записываются сначала в кэш и только потом, по мере необходимости - в память) кэш. В реализации процессор общается только с кэшем, поэтому все действия проходят сначала через него и только в случае необходимости кэш отправляет запросы в память (честный look through и write back). Наш кэш реализует политику вытеснения LRU. Как только мы поработали с какой-то кэш линией во время выполнения программы, мы помечаем её, как последнюю использованную, а поскольку для одного set-а у нас предусмотрено 2 кэш-линии (ассоциативность = 2), при вытеснении мы будем выкидывать ту линию, которая не помечена, как последняя использованная. Разные линии в памяти записываются в разные места в кэше. Память бьётся на 1024 блока (tag - номер такого большого блока), каждый из них бьётся на 32 кэш-линии (set - номер кэш-линии внутри такого блока из кэш-линий) линии с одинаковым set-ом претендуют на одну и ту же пару позиций в кэше.

Основные переменные для общения с процессором:

last_operation_from_cpu - хранит последнюю команду, полученную от cpu
is_owning_cpu - говорит владеет ли кэш шинами c1, d1 в данный момент
addr_tagset, addr_offset, data_to_write - данные, которые были переданы по шинам a1 и d1 от процессора

Основные переменные для общения с памятью:

command_for_mem - отвечает за команды, которые cache посылает памяти
finished_query_in_mem - меняется с 0 на 1, когда на последний отправленный в память запрос был получен ответ
is_owning_memory - говорит владеет ли кэш шинами c2, d2 в данный момент

Основные переменные кэша:

last_used - для каждого set-а хранит последнюю использованную в нём линию
inner_cache_data - хранит в себе кэш-линии в виде двумерного массива (первый индекс - номер сета, второй индекс - номер линии в сете)

Реализация:

В основе работы снова лежит always блок, реагирующий на изменение синхронизации и проверяющий владеем ли мы сейчас шиной, но для удобства таких блоков 2, чтобы разделить общение с процессором и общение с памятью. Рассмотрим каждый из блоков.

Изначально кэш не владеет шинами c1, d1, поэтому он будет ждать, когда процессор отправит какую-то команду. Как только это происходит с помощью нескольких задержек происходит считывание данных с шин и перехват владения шинами.

Как только кэш завладел шинами, он ставит на c1 NOP и начинает обработку полученных данных (далее опишем поведение для команд чтения/записи). Первым делом кэш по переданному сету проверяет, есть ли у него внутри валидная линия с тэгом равным переданному, если такая линия нашлась, произошёл кэш-хит (в этот момент увеличился глобальный счётчик кэш-хитов), а значит дальше мы будем работать с этой линией (работа с линией будет описана дальше), если же такой линии не нашлось, мы фиксируем кэш-мисс, запоминаем последнюю неиспользованную линию и готовимся поработать с даннми, но перед этим мы проверяем, не надо ли нам выполнить вытеснение.

Если линия, которую мы давно не используем является валидной и при этом у неё dirty = 1, значит перед тем как считать нужную линию из памяти, нам нужно выгрузить текущую в память, после чего можно спокойно считать данные по нужному адресу и продолжить выполнение.

Считывание данных из памяти в кэш и их запись в память будут описаны дальше, когда мы будем говорить о втором always блоке.

Когда кэш обработал хит/мисс, нам нужно записать данные в запомненную кэш-линию или считать данные из неё и выдать их процессору (какая именно часть линии нам нужна определяется offset - ом, который мы сохранили в addr_offset).

Если в кэш пришла команда инвалидации линии, он пытается найти внутри линию по нужному сету с совпадающим тэгом и если такая линия есть, надо инвалидировать её, если она не dirty или выгрузить в память перед инвалидацией, если она dirty.

При возвращении данных кэш ставит на c1 C1_RESPONSE.

Теперь поговорим о втором always блоке. В этом блоке реализована логика общения кэша с памятью. Когда кэшу нужно получить данные из памяти или записать данные в память, он устанавливает значение command_for_mem на 1 или 2 соответственно, устанавливает задержку до момента, пока не будет получен ответ от памяти, тем самым блокируя первый always блок, а после отправки команд и данных отдаёт владение шинами c2, d2 памяти. В случае записи по шине a2 мы передаём tag+set линии, которую вытесняем из кэша, а по d2 данные в этой линии, в случае чтения мы передаём адрес, который передал нам процессор, ведь мы обращаемся к памяти только в случае кэш-мисса.

В конце обработки внутри памяти меняем переменную finished_query_in_mem, чтобы сказать первому блоку, что запрос был успешно обработан и он может продолжить общение с процессором.

Дамп кэша выводит все данные в файл cache.txt, reset устанавливает данные во всех линиях на 0, сбрасывает биты валидности и dirty, ставит 0 индекс в каждом сете, как индекс последней использованной кэш-линии.

Реализация памяти (модуль memory, файл memory.sv)

Основные переменные памяти:

addr_to_work_with - адрес, который кэш передал для чтения данных
last_operation_from_cache - запоминает последнюю команду, полученную от кэша
mem_data - хранит в себе все данные памяти
is_owning - говорит владеет ли память шинами в данный момент

Реализация:

Основная идея такая же, как и в предыдущих модулях, всё реализовано через always блок, если мы пытаемся считать данные, мы достаём из памяти нужную кэш линию по переданным tag и set, а если пытаемся записать данные, пишем переданную кэш линию по переданному адресу.

Дамп памяти выводит все данные в файл memory.txt, reset устанавливает данные во всех байтах согласно алгоритму из условия.

При работе с памятью реализована задержка в 100 тактов, чтобы просимулировать долгий ответ памяти.

Константы (файл consts.sv)

В этом файле перечислены полезные константы и макрос, реализующий задержку. С задержками вида #x возникла проблема гонки. Когда в блоке, где clk = 1 устанавливалась чётная задержка, на выходе clk не обязательно был равен 1, чтобы поправить это достаточно было использовать wait, который не продолжает выполнение, пока не выполнено внутреннее условие, таким образом удалось получить валидное значение clk после всех задержек.

Воспроизведение задачи на Verilog

Инстансы нужных модулей, шины для них и clk создавались в testbench.sv. Код задачи я воспроизводил в initial блоке модуля cpu, отправляя запросы на чтение и запись с помощью логики отправки команд процессором, описанной ранее. Также там где надо я выставил задержки на выполнение различных операций и посчитал общее число попаданий и промах с помощью глобальных счётчиков.

Код достаточно подробно залогирован, если возникнет необходимость в выводе лога, можно раскомментировать все строки, которые начинаются с //$fdisplay(glob.fl,...

Чтобы это сделать достаточно заменить все вхождения //$fdisplay(glob.fl, во всех файлах на $fdisplay(glob.fl,

Результат:

Total ticks: 5307782

Total memory accesses: 249600

Cache hits: 228080

Success hits (%): 91.378205

Сравнение результатов

Результат аналитического решения совпал с результатом симуляции в Verilog, но для этого пришлось в одном месте аналитического решения добавить 1 такт, в силу того, что в моей модели 2 последовательные команды отправляются с задержкой в 1 такт, а в этом месте кэш сначала пишет в память, потом читает из неё, т.е. отправляет 2 последовательные команды.

Листинг кода.

Verilog

cpu.sv

```
`include "consts.sv"


module cpu(input clk, output wire [`ADDR1_BUS_SIZE - 1 : 0] a1, inout wire [`DATA1_BUS_SIZE - 1 : 0] d1, inout wire [`CTR1_BUS_SIZE - 1 : 0] c1);
    reg [31:0] inner_data [7:0];
    reg finished_query = 0;
    integer reg_number_to_write;
    bit is_owning = 1;

    integer command = 0;
    integer data_to_send = 'z;
    integer addr_to_send = 0; // отправляется в байтах

    reg [14 : 0] addr_tagset = 'z;
    reg [3 : 0] addr_offset = 'z;

    logic [`ADDR1_BUS_SIZE - 1 : 0] inner_cpu_a1 = 'z;
    assign a1 = inner_cpu_a1;
    logic [`CTR1_BUS_SIZE - 1 : 0] last_operation = `C2_NOP;
    logic [`CTR1_BUS_SIZE - 1 : 0] inner_cpu_c1 = `C2_NOP;
    assign c1 = inner_cpu_c1;
    logic [`DATA1_BUS_SIZE - 1 : 0] inner_cpu_d1 = 'z;
    assign d1 = inner_cpu_d1;

    integer i = 0;
```

```
//simulation
int M = 64;
int N = 60;
int K = 32;
int a = 0;
int b = a + M * K * 1;
int c = b + K * N * 2;
int pa = 0;
int pb = 0;
int pc = 0;
integer ticks_in_sample = 0;
integer s;
initial begin
    for (i = 0; i < 8; i++) begin
        inner_data[i] = '0;
    end
    glob.fl = $fopen("logger.txt", "w");
    glob.fdm = $fopen("memory.txt", "w");
    glob.fdc = $fopen("cache.txt", "w");
    pa = a;
    `delay(2,1); //init
    pc = c;
    `delay(2,1);//init
    `delay(2,1);//init y
    for (int y = 0; y < M; y++) begin
        `delay(2,1);//init x
        for (int x = 0; x < N; x++) begin
            pb = b;
            `delay(2,1); //init
```

```
s = 0;
`delay(2,1);//init
`delay(2,1);//init k
for (int k = 0; k < K; k++) begin
    finished_query = 0;
    addr_to_send = pa + k * 1;
    command = 1;
    reg_number_to_write = 0;
    wait(finished_query == 1 && clk == 1);
    `delay(10,1); // mul
    finished_query = 0;
    addr_to_send = pb + x * 2;
    command = 2;
    reg_number_to_write = 1;
    wait(finished_query == 1 && clk == 1);
    finished_query = 0;
    s += inner_data[0] * inner_data[1];
    `delay(2,1); // add
    pb += N * 2;
    `delay(2,1); // add

    `delay(2,1); // k++
    `delay(2,1); // loop iteratrion
end
//$fdisplay(glob.fl,"y = %0d, x = %0d, t = %0t", y, x, $time);
data_to_send = s;
addr_to_send = pc + x * 4;
command = 7;
wait(finished_query == 1 && clk == 1);
finished_query = 0;
```

```
            `delay(2,1); // x++
            `delay(2,1); // loop iteratrion
        end
        pa += K * 1;
        `delay(2,1); // add
        pc = pc + N * 4;
        `delay(2,1); // add

        `delay(2,1); // y++
        `delay(2,1); // loop iteratrion
    end
    `delay(2,1); // exit function
    //$fdisplay(glob.fl, "Total ticks: %0d", ($time)/2);
    //$fdisplay(glob.fl, "Total accesses: %0d", glob.hits + glob.miss);
    //$fdisplay(glob.fl, "Cache hits: %0d", glob.hits);
    //$fdisplay(glob.fl, "Part of hits: %0f", glob.hits / (glob.hits + glob.miss));

    $display("Total ticks: %0d", ($time)/2);
    $display("Total accesses: %0d", glob.hits + glob.miss);
    $display("Cache hits: %0d", glob.hits);
    $display("Part of hits: %0f", glob.hits / (glob.hits + glob.miss));

    `delay(2, 1);
    $fclose(glob.fl);
    $fclose(glob.fdm);
    $fclose(glob.fdc);
end

task drop_leading;
    inner_cpu_a1 = 'z;
```

```
            inner_cpu_d1 = 'z;
            inner_cpu_c1 = 'z;
            is_owning = 0;
            command = -1;
        endtask

        task split_addr;
            addr_tagset = addr_to_send[`TAG_SIZE + `SET_SIZE + `OFFSET_SIZE - 1 : `OFFSET_SIZE];
            addr_offset = addr_to_send[`OFFSET_SIZE - 1 : 0];
            //$fdisplay(glob.fl,"INIT t=%0t, tagset = %b, offset = %b, sent addr = %0d", $time, addr_tagset, addr_offset, addr_to_send);
        endtask

        task take_leading;
            command = 0;
            is_owning = 1;
        endtask

        always @(clk) begin
            if (clk == 1 && is_owning == 1) begin
                case (command)
                    0: begin
                        //$fdisplay(glob.fl,"Cpu in deal. t=%0t", $time);
                        inner_cpu_a1 = 'z;
                        last_operation = `C1_NOP;
                        inner_cpu_c1 = `C1_NOP;
                        inner_cpu_d1 = 'z;
                        is_owning = 1;
                        command = -1;
                        finished_query = 1;
```

```
end
1: begin
   //$fdisplay(glob.fl,"Cpu wants to read 8. addr part 1t=%0t", $time);
   split_addr();
   last_operation = `C1_READ8;
   inner_cpu_a1 = addr_tagset;
   inner_cpu_d1 = 'z;
   inner_cpu_c1 = `C1_READ8;
   `delay(2,1)
   //$fdisplay(glob.fl,"Cpu wants to read 8. addr part 2 t=%0t", $time);
   inner_cpu_a1 = addr_offset;
   `delay(2,1)
   //$fdisplay(glob.fl,"Cpu gave up after asking to read 8. t=%0t", $time);
   drop_leading();

end
2: begin
   //$fdisplay(glob.fl,"Cpu wants to read 16. addr part 2 t=%0t", $time);
   split_addr();
   last_operation = `C1_READ16;
   inner_cpu_a1 = addr_tagset;
   inner_cpu_d1 = 'z;
   inner_cpu_c1 = `C1_READ16;
   `delay(2,1)
   //$fdisplay(glob.fl,"Cpu wants to read 16. addr part 2 t=%0t", $time);
   inner_cpu_a1 = addr_offset;
   `delay(2,1)
   //$fdisplay(glob.fl,"Cpu gave up after asking to read 16. t=%0t", $time);
   drop_leading();
end
```

```
                3: begin
                   //$fdisplay(glob.fl,"Cpu wants to read 32. addr part 1 t=%0t", $time);
                   split_addr();
                   last_operation = `C1_READ32;
                   inner_cpu_a1 = addr_tagset;
                   inner_cpu_d1 = 'z;
                   inner_cpu_c1 = `C1_READ32;
                   `delay(2,1)
                   //$fdisplay(glob.fl,"Cpu wants to read 32. addr part 2 t=%0t", $time);
                   inner_cpu_a1 = addr_offset;
                   `delay(2,1)
                   //$fdisplay(glob.fl,"Cpu gave up after asking to read 32. t=%0t", $time);
                   drop_leading();

                end
                4: begin
                   //$fdisplay(glob.fl,"Cpu wants to invalidate some line. t=%0t", $time);
                   split_addr();
                   last_operation = `C1_INVALIDATE_LINE;
                   inner_cpu_a1 = addr_tagset;
                   inner_cpu_d1 = 'z;
                   inner_cpu_c1 = `C1_INVALIDATE_LINE;
                   `delay(2,1)
                       //$fdisplay(glob.fl,"Cpu gave up after asking to invalidate some line.
t=%0t", $time);
                   drop_leading();

                end
                5: begin
                   //$fdisplay(glob.fl,"Cpu wants to write 8. addr part 1 t=%0t", $time);
                   split_addr();
```

```
    last_operation = `C1_WRITE8;
    inner_cpu_a1 = addr_tagset;
    inner_cpu_d1 = data_to_send & 255;
    inner_cpu_c1 = `C1_WRITE8;
    `delay(2,1)
    //$fdisplay(glob.fl,"Cpu wants to write 8. addr part 2 t=%0t", $time);
    inner_cpu_a1 = addr_offset;
    `delay(2,1)
    //$fdisplay(glob.fl,"Cpu gave up after asking to write 8. t=%0t", $time);
    drop_leading();
end
6: begin
    //$fdisplay(glob.fl,"Cpu wants to write 16. addr part 1 t=%0t", $time);
    split_addr();
    last_operation = `C1_WRITE16;
    inner_cpu_a1 = addr_tagset;
    inner_cpu_d1 = data_to_send & 65535;
    inner_cpu_c1 = `C1_WRITE16;
    `delay(2,1)
    //$fdisplay(glob.fl,"Cpu wants to write 16. addr part 2 t=%0t", $time);
    inner_cpu_a1 = addr_offset;
    `delay(2,1)
    //$fdisplay(glob.fl,"Cpu gave up after asking to write 16. t=%0t", $time);
    drop_leading();

end
7: begin
    split_addr();
    last_operation = `C1_WRITE32;
    inner_cpu_a1 = addr_tagset;
```

```
                    inner_cpu_d1 = data_to_send & 65535;
                    inner_cpu_c1 = `C1_WRITE32;
                        //$fdisplay(glob.fl,"Cpu wants to write 32. addr part 1 data = %b t=%0t",inner_cpu_d1, $time);
                    `delay(2,1)
                    inner_cpu_a1 = addr_offset;
                    inner_cpu_d1 = (data_to_send >> 16) & 65535;
                     //$fdisplay(glob.fl,"Cpu wants to write 32 addr part 2 data = %b. t=%0t", inner_cpu_d1, $time);
                    `delay(2,1)
                    //$fdisplay(glob.fl,"Cpu gave up after asking to write 32. t=%0t", $time);
                    drop_leading();
                  end
                  default: begin
                  end
                endcase
              end
              else if (is_owning == 0 && clk == 0) begin
                case (c1)
                  `C1_RESPONSE: begin
                    //$fdisplay(glob.fl,"Cpu got the result. t=%0t, clk = %0b", $time, clk);
                    case (last_operation)
                      `C1_READ8: begin
                        inner_data[reg_number_to_write] = d1 & 255;
                            //$fdisplay(glob.fl,"Cpu saved 8 bit number to read. t=%0t num=%0b", $time,  d1);
                      end
                      `C1_READ16: begin
                        inner_data[reg_number_to_write] = d1;
                            //$fdisplay(glob.fl,"Cpu saved 16 bit number to read. t=%0t num=%0b", $time,  d1);
                      end
```

```
                    `C1_READ32: begin
                        inner_data[reg_number_to_write][15:0] = d1;
                                //$fdisplay(glob.fl,"Cpu saved 32 bit number to read. t=%0t num=%0b", $time,  d1);
                        `delay(2,0)
                        inner_data[reg_number_to_write][31:16] = d1;
                            //$fdisplay(glob.fl,"Cpu saved 32 bit number to read part 2. t=%0t num=%0b", $time, d1);
                    end
                    default: begin
                    end
                endcase
            take_leading();
            end
            default: begin
            end
        endcase
    end
end

endmodule
```

cache.sv

```
`include "consts.sv"

module cache(input clk, 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, input dump_cache, input reset);
// CPU side
    bit is_owning_cpu = 0;
    bit last_used [`CACHE_LINE_COUNT/`CACHE_WAY - 1 : 0];
```

```
        //V + D + TAG_SIZE + CACHE_LINE_SIZE = 1 + 1 + 10 + 16*8 = 1 + 1 + 10 +
128

        reg [(`V + `D + `TAG_SIZE + `CACHE_LINE_SIZE) - 1 : 0] inner_cache_data
[`CACHE_LINE_COUNT/`CACHE_WAY - 1 : 0][`CACHE_WAY - 1 : 0];

        integer i = 0;

        reg[`CACHE_LINE_SIZE/4 - 1 : 0] data1;

        reg[`CACHE_LINE_SIZE/4 - 1 : 0] data2;

        reg[`CACHE_LINE_SIZE/4 - 1 : 0] data3;

        reg[`CACHE_LINE_SIZE/4 - 1 : 0] data4;


        logic [`CTR1_BUS_SIZE - 1 : 0] last_operation_from_cpu = 'z;

        logic [`CTR1_BUS_SIZE - 1 : 0] inner_cache_c1 = 'z;

        assign c1 = inner_cache_c1;

        logic [`DATA1_BUS_SIZE - 1 : 0] inner_cache_d1 = 'z;

        assign d1 = inner_cache_d1;


    // Memory side

        bit is_owning_memory = 1;

        integer data_from_memory = 'z;

        integer command_for_mem = 0; // команду посылает кэш

        integer finished_query_in_mem = 0;


        logic [`ADDR2_BUS_SIZE - 1 : 0] inner_cache_a2 = 'z;

        assign a2 = inner_cache_a2;

        logic [`CTR2_BUS_SIZE - 1 : 0] last_operation_for_memory = 'z;

        logic [`CTR2_BUS_SIZE - 1 : 0] inner_cache_c2 = 'z;

        assign c2 = inner_cache_c2;

        logic [`DATA2_BUS_SIZE - 1 : 0] inner_cache_d2 = 'z;

        assign d2 = inner_cache_d2;


        integer sutable_line = -1;
```

```
// Inner data

    reg [`TAG_SIZE - 1 : 0] resived_tag = 'z;
    reg [`SET_SIZE - 1 : 0] resived_set = 'z;
    reg [`TAG_SIZE - 1 : 0] inner_tag = 'z;
    reg inner_v = 'z;
    reg inner_d = 'z;

    reg [`TAG_SIZE + `SET_SIZE - 1 : 0] addr_tagset;
    reg [`OFFSET_SIZE - 1 : 0] addr_offset;
    reg [`CACHE_LINE_SIZE : 0] data_to_read;
    reg [2 * `DATA1_BUS_SIZE : 0] data_to_write;

    task init_reset_cache;
      //$fdisplay(glob.fl,"Cache init/reset. t=%0t", $time);
      for (int l = 0; l < `CACHE_LINE_COUNT/`CACHE_WAY; l++) begin
        for (int j = 0; j < `CACHE_WAY; j++)
          inner_cache_data[l][j] = 0;
        last_used[i] = 0;
      end
      //$fdisplay(glob.fl,"-----------------------------------------------------");
    endtask

    task dump_cache_task;
      $fdisplay(glob.fdc,"Cache dump. t = %0t", $time);
      for (int l = 0; l < `CACHE_LINE_COUNT/`CACHE_WAY; l++) begin
        for (int j = 0; j < `CACHE_WAY; j++)begin
          inner_tag = inner_cache_data[l][j][(`TAG_SIZE + `CACHE_LINE_SIZE) -
1 : `CACHE_LINE_SIZE];
                inner_v = inner_cache_data[l][j][`D + `TAG_SIZE +
`CACHE_LINE_SIZE];
```

```
                inner_d = inner_cache_data[l][j][`TAG_SIZE + `CACHE_LINE_SIZE];
                data1 = inner_cache_data[l][j][`CACHE_LINE_SIZE/4 - 1 : 0];
                        data2 = inner_cache_data[l][j][`CACHE_LINE_SIZE/2 - 1: `CACHE_LINE_SIZE/4];
                    data3 = inner_cache_data[l][j][`CACHE_LINE_SIZE * 3 / 4 - 1 : `CACHE_LINE_SIZE/2];
                        data4 = inner_cache_data[l][j][`CACHE_LINE_SIZE - 1 : `CACHE_LINE_SIZE * 3 / 4 ];

                $fdisplay(glob.fdc,"[set = %0d][way = %0d]  v = %0d, d = %0d, tag = %0d, data = %0d %0d %0d %0d", l, j, inner_v, inner_d, inner_tag, data4, data3, data2, data1);
            end
            $fdisplay(glob.fdc,"for set: %0d,last used = %0d", l, last_used[l]);
        end
        $fdisplay(glob.fdc, "-----------------------------------------------------");
    endtask

    initial begin
        init_reset_cache();
    end

    task drop_leading;
        //$fdisplay(glob.fl,"Cache gave up. t=%0t", $time);
        inner_cache_d1 = 'z;
        inner_cache_c1 = 'z;
        is_owning_cpu = 0;
    endtask

    always@(posedge dump_cache) begin
        dump_cache_task();
    end
```

```
always@(posedge reset) begin
   init_reset_cache();
end

always @(clk) begin
   if (clk == 1 && is_owning_cpu == 1) begin
      inner_cache_c1 = `C1_NOP;
                        //$fdisplay(glob.fl,"Cache in deal last_op = %b. t=%0t", last_operation_from_cpu, $time);
                  if (last_operation_from_cpu == `C1_READ8 || last_operation_from_cpu == `C1_READ16 || last_operation_from_cpu == `C1_READ32 || last_operation_from_cpu == `C1_WRITE8 || last_operation_from_cpu == `C1_WRITE16 || last_operation_from_cpu == `C1_WRITE32) begin
                  sutable_line = -1;
                  resived_tag = addr_tagset[(`TAG_SIZE + `SET_SIZE) - 1 : `SET_SIZE];
                  resived_set = addr_tagset[`SET_SIZE - 1 : 0];
                  for (int j = 0; j < `CACHE_WAY; j++) begin
                        inner_tag = inner_cache_data[resived_set][j][(`TAG_SIZE + `CACHE_LINE_SIZE) - 1 : `CACHE_LINE_SIZE];
                     inner_v = inner_cache_data[resived_set][j][`D + `TAG_SIZE + `CACHE_LINE_SIZE];
                     if (inner_tag == resived_tag && inner_v == 1) begin
                        `delay(8,1)
                        //$fdisplay(glob.fl,"Cache hit j = %b. t=%0t", j,$time);
                        glob.hits += 1;
                        sutable_line = j;
                        last_used[resived_set] = j;
                     end
                  end
                  if (sutable_line == -1) begin
                     glob.miss += 1;
                        sutable_line = (last_used[resived_set] + 1) % 2; // отрицание last_used для integer
```

```
                    inner_v = inner_cache_data[resived_set][sutable_line][`D + `TAG_SIZE
+ `CACHE_LINE_SIZE];
                     inner_d = inner_cache_data[resived_set][sutable_line][`TAG_SIZE +
`CACHE_LINE_SIZE];
                      //$fdisplay(glob.fl,"Cache miss, valid = %0b, dirty = %0b, set = %b,
sutable_line = %b, . t=%0t\n", inner_v,inner_d, resived_set, sutable_line,$time);
                  `delay(4,1)
                  if (inner_v == 1 && inner_d == 1) begin
                     finished_query_in_mem = 0;
                     command_for_mem = 2;
                     wait(clk == 1 && finished_query_in_mem == 1);
                  end
                  finished_query_in_mem = 0;
                  command_for_mem = 1;
                  wait(clk == 1 && finished_query_in_mem == 1);
                  //$fdisplay(glob.fl,"Cache miss finished. t = %0t", $time);
                              inner_cache_data[resived_set][sutable_line][`TAG_SIZE +
`CACHE_LINE_SIZE - 1 : `CACHE_LINE_SIZE] = resived_tag;
                              inner_cache_data[resived_set][sutable_line][`TAG_SIZE +
`CACHE_LINE_SIZE] = 0; //dirty
                               inner_cache_data[resived_set][sutable_line][`D + `TAG_SIZE +
`CACHE_LINE_SIZE] = 1; //valid
                  finished_query_in_mem = 0;
                  last_used[resived_set] = ~last_used[resived_set];


                end
                 if (last_operation_from_cpu == `C1_READ8) begin
                  //$fdisplay(glob.fl,"Cache sent 8 bit data. t=%0t", $time);
                                                                inner_cache_d1     =
inner_cache_data[resived_set][sutable_line][addr_offset*`BYTE +: `BYTE];
                  inner_cache_c1 = `C1_RESPONSE;
                end else if (last_operation_from_cpu == `C1_READ16) begin
                  //$fdisplay(glob.fl,"Cache sent 16 bit data. t=%0t", $time);
```

```
                                                    inner_cache_d1    = inner_cache_data[resived_set][sutable_line][addr_offset * `BYTE +: `BYTE * 2];
                    inner_cache_c1 = `C1_RESPONSE;
                end else if (last_operation_from_cpu == `C1_READ32) begin
                        //$fdisplay(glob.fl,"Cache sent 1 pack data = %0b. t=%0t, clk = %0b",inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 16], $time, clk);
                                                    inner_cache_d1    = inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 16];
                    inner_cache_c1 = `C1_RESPONSE;
                    `delay(2,1)
                                        //$fdisplay(glob.fl,"Cache sent 2 pack data = %0b. t=%0t",inner_cache_data[resived_set][sutable_line][addr_offset*8 + 16 +: 16], $time);
                                                    inner_cache_d1    = inner_cache_data[resived_set][sutable_line][addr_offset*8 + 16 +: 16];
                end else if (last_operation_from_cpu == `C1_WRITE8) begin
                    //$fdisplay(glob.fl,"Cache wrote 8 bit data. t=%0t", $time);
                        inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 8] = data_to_write;
                                inner_cache_data[resived_set][sutable_line][`TAG_SIZE + `CACHE_LINE_SIZE] = 1;
                    inner_cache_c1 = `C1_RESPONSE;
                end else if (last_operation_from_cpu == `C1_WRITE16) begin
                    //$fdisplay(glob.fl,"Cache wrote 16 bit data. t=%0t", $time);
                        inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 16] = data_to_write;
                                inner_cache_data[resived_set][sutable_line][`TAG_SIZE + `CACHE_LINE_SIZE] = 1;
                    inner_cache_c1 = `C1_RESPONSE;
                end else if (last_operation_from_cpu == `C1_WRITE32) begin
                        inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 16] = data_to_write[15 : 0];
                                        //$fdisplay(glob.fl,"Cache wrote 1 pack data = %0b. t=%0t",inner_cache_data[resived_set][sutable_line][addr_offset*8 +: 16], $time);
                        inner_cache_data[resived_set][sutable_line][addr_offset*8 + 16 +: 16] = data_to_write[31 : 16];
```

```
                                    //$fdisplay(glob.fl,"Cache wrote 2 pack data = %0b. t=%0t",inner_cache_data[resived_set][sutable_line][addr_offset*8 + 16 +: 16], $time);
                                    inner_cache_data[resived_set][sutable_line][`TAG_SIZE + `CACHE_LINE_SIZE] = 1;
                        inner_cache_c1 = `C1_RESPONSE;
                    end
                end else if (last_operation_from_cpu == `C1_INVALIDATE_LINE) begin
                    resived_tag = addr_tagset[(`TAG_SIZE + `SET_SIZE) - 1 : `SET_SIZE];
                    resived_set = addr_tagset[`SET_SIZE - 1 : 0];
                    for (int j = 0; j < `CACHE_WAY; j++) begin
                                    inner_tag = inner_cache_data[resived_set][j][(`TAG_SIZE + `CACHE_LINE_SIZE) - 1 : `CACHE_LINE_SIZE];
                                inner_v = inner_cache_data[resived_set][j][`D + `TAG_SIZE + `CACHE_LINE_SIZE];
                                    inner_d = inner_cache_data[resived_set][j][`TAG_SIZE + `CACHE_LINE_SIZE];
                        if (inner_tag == resived_tag && inner_v == 1) begin
                            //$fdisplay(glob.fl,"Cache found what to invalidate j = %b. t=%0t", j,$time);
                            sutable_line = j;
                            if (inner_d == 1) begin
                                //$fdisplay(glob.fl,"The line was dirty. t=%0t", j,$time);
                                finished_query_in_mem = 0;
                                command_for_mem = 2;
                                wait(clk == 1 && finished_query_in_mem == 1);
                            end
                                inner_cache_data[resived_set][sutable_line][`D + `TAG_SIZE + `CACHE_LINE_SIZE] = 0;
                                                inner_cache_data[resived_set][j][`TAG_SIZE + `CACHE_LINE_SIZE] = 0;
                            last_used[resived_set] = (sutable_line + 1) % 2;
                        end
                    end
```

```
                inner_cache_c1 = `C1_RESPONSE;
            end
            `delay(2,1)
            drop_leading();
        end
        else if (is_owning_cpu == 0 && clk == 0) begin
            case (c1)
                `C1_READ8: begin
                    //$fdisplay(glob.fl,"Cache prepared to give out 8 bit data. addr part 1 t=%0t", $time);
                    addr_tagset = a1;
                    last_operation_from_cpu = c1;
                    `delay(2,0)
                    //$fdisplay(glob.fl,"Cache prepared to give out 8 bit data. addr part 2 t=%0t", $time);
                    addr_offset = a1;
                    is_owning_cpu = 1;
                end
                `C1_READ16: begin
                    //$fdisplay(glob.fl,"Cache prepared to give out 16 bit data. addr part 1 t=%0t", $time);
                    addr_tagset = a1;
                    last_operation_from_cpu = c1;
                    `delay(2,0)
                    //$fdisplay(glob.fl,"Cache prepared to give out 16 bit data. addr part 2 t=%0t", $time);
                    addr_offset = a1;
                    is_owning_cpu = 1;
                end
                `C1_READ32: begin
                    //$fdisplay(glob.fl,"Cache prepared to give out 32 bit data. addr part 1 t=%0t", $time);
```

```
                addr_tagset = a1;
                last_operation_from_cpu = c1;
                `delay(2,0)
                    //$fdisplay(glob.fl,"Cache prepared to give out 32 bit data. addr part 2 t=%0t", $time);
                addr_offset = a1;
                is_owning_cpu = 1;
            end
            `C1_INVALIDATE_LINE: begin
                //$fdisplay(glob.fl,"Cache prepared to invalidate line.t=%0t", $time);
                addr_tagset = a1;
                last_operation_from_cpu = c1;
                is_owning_cpu = 1;
            end
            `C1_WRITE8: begin
                data_to_write = d1;
                last_operation_from_cpu = c1;
                addr_tagset = a1;
                    //$fdisplay(glob.fl,"Cache got the 8 bit data. addr part 2 t=%0t, tagset = %b, offset = %b", $time, addr_tagset, addr_offset);
                `delay(2,0)
                addr_offset = a1;
                is_owning_cpu = 1;
                    //$fdisplay(glob.fl,"Cache got the 8 bit data. addr part 2 t=%0t, tagset = %b, offset = %b", $time, addr_tagset, addr_offset);
            end
            `C1_WRITE16: begin
                //$fdisplay(glob.fl,"Cache got the 16 bit data. addr part 1 t=%0t", $time);
                data_to_write = d1;
                last_operation_from_cpu = c1;
                addr_tagset = a1;
```

```
            `delay(2,0)
            //$fdisplay(glob.fl,"Cache got the 16 bit data. addr part 2 t=%0t", $time);
            addr_offset = a1;
            is_owning_cpu = 1;
        end
        `C1_WRITE32: begin
            //$fdisplay(glob.fl,"Cache got the 32 bit data. t=%0t", $time);
            last_operation_from_cpu = c1;
            addr_tagset = a1;
            data_to_write[`DATA1_BUS_SIZE - 1 : 0] = d1;
            `delay(2,0)
            //$fdisplay(glob.fl,"Cache got the 32 bit data part 2. t=%0t", $time);
            addr_offset = a1;
            data_to_write[2*`DATA1_BUS_SIZE - 1 : `DATA1_BUS_SIZE] = d1;
            is_owning_cpu = 1;
        end
        default: begin
        end
      endcase
   end
end

always @(clk) begin
   if (clk == 1 && is_owning_memory == 1) begin
      case (command_for_mem)
         0: begin
            //$fdisplay(glob.fl,"Cache in deal(memory part). t=%0t", $time);
            inner_cache_a2 = 'z;
            last_operation_for_memory = `C2_NOP;
            inner_cache_c2 = `C2_NOP;
```

```
                inner_cache_d2 = 'z;
                is_owning_memory = 1;
                command_for_mem = -1;
                finished_query_in_mem = 1;
            end
            1: begin
                    //$fdisplay(glob.fl,"Cache wants to read line.(memory part) t=%0t",
$time);
                last_operation_for_memory = `C2_READ_LINE;
                inner_cache_a2 = addr_tagset;
                inner_cache_d2 = 'z;
                inner_cache_c2 = `C2_READ_LINE;
                `delay(2,1)
                //$fdisplay(glob.fl,"Cache gave up after asking to read line.(memory part)
t=%0t", $time);
                inner_cache_a2 = 'z;
                inner_cache_d2 = 'z;
                inner_cache_c2 = 'z;
                is_owning_memory = 0;
            end
            2: begin
                    //$fdisplay(glob.fl,"Cache wants to write line, last_op = %0d. t=%0t",
last_operation_from_cpu,$time);
                last_operation_for_memory = `C2_WRITE_LINE;
                                                    inner_cache_a2 =
inner_cache_data[resived_set][sutable_line][`CACHE_LINE_SIZE +: `TAG_SIZE] * (1 <<
`SET_SIZE) + resived_set;
                inner_cache_c2 = `C2_WRITE_LINE;
                for ( i = 0; i < `MEM_LINE_SIZE/`DATA2_BUS_SIZE; i++) begin
                    //$fdisplay(glob.fl,"Cache writing 16 byte data, part %0b. t=%0t", i +
1,$time);
                                                    inner_cache_d2[0 +: `BYTE] =
inner_cache_data[resived_set][sutable_line][2*i*`BYTE +: `BYTE];
```

```
                                                    inner_cache_d2[`BYTE +: `BYTE] = inner_cache_data[resived_set][sutable_line][(2*i + 1) * `BYTE +: `BYTE];
                        `delay(2,1)
                    end
                        //$fdisplay(glob.fl,"Cpu gave up after asking to write line. t=%0t", $time);
                    inner_cache_a2 = 'z;
                    inner_cache_d2 = 'z;
                    inner_cache_c2 = 'z;
                    is_owning_memory = 0;

                end
                default: begin
                end
                endcase
            end
            else if (is_owning_memory == 0 && clk == 0) begin
                case (c2)
                `C2_RESPONSE: begin
                    //$fdisplay(glob.fl,"Cache getting the result. t=%0t", $time);
                    case (last_operation_for_memory)
                        `C2_READ_LINE: begin
                        for ( i = 0; i < `MEM_LINE_SIZE/`DATA2_BUS_SIZE; i++) begin
                            inner_cache_data[resived_set][sutable_line][2*i*`BYTE +: `BYTE] = d2[0 +: `BYTE];
                                inner_cache_data[resived_set][sutable_line][(2*i + 1) * `BYTE +: `BYTE] = d2[`BYTE +: `BYTE];
                            //$fdisplay(glob.fl,"Cache getting 16 byte data, part %0d, data = %b. t=%0t", i + 1,d2[0 +: `BYTE],$time);
                            if (i != `MEM_LINE_SIZE/`DATA2_BUS_SIZE - 1)
                                `delay(2,0)
                        end
```

```
                end
                default: begin
                end
              endcase
              // finished_query_in_mem = 1;
               command_for_mem = 0;
               is_owning_memory = 1;
            end
            default: begin
            end
          endcase
        end
      end
    endmodule
```

memory.sv

```
`include "consts.sv"



module memory #(parameter _SEED = 225526) (input clk, input wire [`ADDR2_BUS_SIZE - 1 : 0] a2, inout wire [`DATA2_BUS_SIZE - 1 : 0] d2, inout wire [`CTR2_BUS_SIZE - 1 : 0] c2, input dump_memory, input reset);
    integer SEED = _SEED;
    bit is_owning = 0;
    logic [`BYTE - 1 : 0] mem_data [(1 << `MEM_BYTE_SIZE) - 1 : 0];


    integer addr_to_work_with;


    logic [`CTR2_BUS_SIZE - 1 : 0] last_operation_from_cache = 'z;
    logic [`CTR2_BUS_SIZE - 1 : 0] inner_mem_c2 = 'z;
    assign c2 = inner_mem_c2;
    logic [`DATA1_BUS_SIZE - 1 : 0] inner_mem_d2 = 'z;
```

```
    assign d2 = inner_mem_d2;

    task init_reset_memory;
      //$fdisplay(glob.fl,"Memory reset/init. t=%0t", $time);
      for (int i = 0; i < (1 << `MEM_BYTE_SIZE); i++)
        mem_data[i] = $random(SEED) >> 16;
      for (int i = 0; i < (1 << `MEM_BYTE_SIZE); i++)
          //$fdisplay(glob.fl,"[%d] %d", i, mem_data[i]);
      //$fdisplay(glob.fl,"------------------------------------------------------");
    endtask

    initial begin
      init_reset_memory();
    end

    always@(posedge dump_memory) begin
      $fdisplay(glob.fdm,"Memory dump. t=%0t", $time);
      for (i = 0; i < (1 << `MEM_BYTE_SIZE); i++ ) begin
                $fdisplay(glob.fdm, "[tag = %0d, set = %0d, offset = %0d] %0d",
i[`SET_SIZE + `OFFSET_SIZE +: `TAG_SIZE], i[`OFFSET_SIZE +: `SET_SIZE] ,i[0 +:
`OFFSET_SIZE], mem_data[i]);
      end
      $fdisplay(glob.fdm, "------------------------------------------------------");
    end

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

    integer i = 0;
```

```
always @(clk) begin
  if (clk == 1 && is_owning == 1) begin
    //$fdisplay(glob.fl,"Memory in deal. t=%0t", $time);
    inner_mem_c2 = `C2_NOP;
    if (last_operation_from_cache == `C2_READ_LINE)
      `delay(196,1)
    else if (last_operation_from_cache == `C2_WRITE_LINE)
      `delay(180,1)
    inner_mem_c2 = `C2_RESPONSE;
    if (last_operation_from_cache == `C2_READ_LINE) begin
      for ( i = 0; i < `MEM_LINE_SIZE/`DATA2_BUS_SIZE; i++) begin
        inner_mem_d2[`BYTE - 1 : 0] = mem_data[addr_to_work_with * (`CACHE_LINE_SIZE / `BYTE) + i*2];
        inner_mem_d2[2 * `BYTE - 1 : `BYTE] = mem_data[addr_to_work_with * (`CACHE_LINE_SIZE / `BYTE) + i*2 + 1];
        last_operation_from_cache = c2;
        //$fdisplay(glob.fl,"Memory giving 16 byte data, part %0d, data = %b. t=%0t", i + 1, mem_data[addr_to_work_with * `CACHE_LINE_SIZE + i*2],$time);
        if (i != `MEM_LINE_SIZE/`DATA2_BUS_SIZE - 1)
          `delay(2,1)
      end
    end
    `delay(2,1)
    //$fdisplay(glob.fl,"Memory gave up. t=%0t", $time);
    inner_mem_d2 = 'z;
    inner_mem_c2 = 'z;
    is_owning = 0;
  end
  else if (is_owning == 0 && clk == 0) begin
    case (c2)
      `C2_READ_LINE: begin
```

```
                    //$fdisplay(glob.fl,"Memory prepared to give out data. t=%0t", $time);
                    addr_to_work_with = a2;
                    last_operation_from_cache = c2;
                    is_owning = 1;
                end
                `C2_WRITE_LINE: begin
                    for (int i = 0; i <  `MEM_LINE_SIZE/`DATA2_BUS_SIZE; i++) begin
                        //$fdisplay(glob.fl,"Memory writing the 16 byte data, part %0b. t=%0t, d2 = %0b", i + 1,$time, d2);
                        mem_data[a2 * (`CACHE_LINE_SIZE / `BYTE) + i*2] = d2[`BYTE - 1 : 0];
                          mem_data[a2 * (`CACHE_LINE_SIZE / `BYTE) + i*2 + 1] = d2[2 * `BYTE - 1 : `BYTE];
                        last_operation_from_cache = c2;
                        `delay(2,0)
                    end
                    is_owning = 1;
                end
            endcase
        end
    end

endmodule
```

consts.sv

```
`ifndef CONSTANTS
`define CONSTANTS

// Разработано совместно с Тимофеем Маловым
`define delay(TIME, CLOCK) \
    for (int i = 0; i < TIME; i++) \
```

```
        wait(clk == (i + !CLOCK) % 2);

`define C1_NOP 3'b000
`define C1_READ8 3'b001
`define C1_READ16 3'b010
`define C1_READ32 3'b011
`define C1_INVALIDATE_LINE 3'b100
`define C1_WRITE8 3'b101
`define C1_WRITE16 3'b110
`define C1_WRITE32 3'b111
`define C1_RESPONSE 3'b111

`define C2_NOP 2'b00
`define C2_READ_LINE 2'b10
`define C2_WRITE_LINE 2'b11
`define C2_RESPONSE 2'b01

`define MEM_LINE_SIZE (1 << 7)
`define MEM_LINE_COUNT (1 << 15)

`define CACHE_LINE_SIZE (1 << 7)
`define CACHE_LINE_COUNT 64
`define CACHE_WAY 2

`define TAG_SIZE 10
`define SET_SIZE 5
`define OFFSET_SIZE 4
`define V 1
`define D 1
```

```
`define ADDR1_BUS_SIZE 15
`define ADDR2_BUS_SIZE 15

`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16

`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2

`define BYTE 8
`define MEM_BYTE_SIZE 19



module glob;
    real  hits = 0;
    integer miss = 0;
    integer fl, fdm, fdc;
    reg reset;
    reg dump_m;
    reg dump_c;
endmodule

`endif
```

testbench.sv

```
`include "consts.sv"
`include "cpu.sv"
`include "cache.sv"
`include "memory.sv"
```

```
module testbench;

    reg clk = 1;
    wire [`ADDR1_BUS_SIZE - 1 : 0] a1;
    wire [`DATA1_BUS_SIZE - 1 : 0] d1;
    wire [`CTR1_BUS_SIZE - 1 : 0] c1;
    wire [`ADDR2_BUS_SIZE - 1 : 0] a2;
    wire [`DATA2_BUS_SIZE - 1 : 0] d2;
    wire [`CTR2_BUS_SIZE - 1 : 0] c2;
    integer i = 0;
    cpu my_cpu(clk, a1, d1, c1);
    cache my_cache(clk, a1, d1, c1, a2, d2, c2, glob.dump_c, glob.reset);
    memory my_memory(clk, a2, d2, c2, glob.dump_m, glob.reset);

    initial begin
        for (i = 1; i < 20000000; i++) begin
            #1;
            clk = ~clk;
        end
    end

endmodule
```

Kotlin

Main.kt

```
var ticks = 0
val cache = MyCache()

var totalAsksCnt = 0

fun cacheRead8(address: Address) {
    ticks += 2
    cache.processData(address, 1, false)
    totalAsksCnt += 1
}

fun cacheRead16(address: Address) {
    ticks += 2
    cache.processData(address, 2, false)
    totalAsksCnt += 1
}

fun cacheWrite32(address: Address) {
    ticks += 2
    cache.processData(address, 4, true)
    totalAsksCnt += 1
}

fun multi() {
    ticks += 5
}
```

```
fun add() {
    ticks += 1
}

fun init() {
    ticks += 1
}

fun iteration() {
    ticks += 1 // итерация цикла
    ticks += 1 // (x++ / y++ / k++)
}

fun ext() {
    ticks += 1
}

fun endBits(num: Int, number: UInt) = number.shl(32 - num).shr(32 - num)

fun intToAddress(address: UInt): Address {
    val addressSet = endBits(cacheSetBits, address.shr(offsetBits)).toInt()
    val addressTag = endBits(tagBits,address.shr(offsetBits + cacheSetBits)).toInt()

    return Address(addressTag, addressSet)
}


const val M = 64
const val N = 60
const val K = 32
```

```
const val aBegPtr = 0
const val bBegPtr = M * K * 1 //массив элементов по 2 байта, поэтому в след. строке * 2
const val cBegPtr = M * K * 1 + K * N * 2 //массив элементов по 4 байта, учитываем это в (*)

fun main() {

    var pa = aBegPtr
    init() // pa
    var pc = cBegPtr
    init() // pc
    init() // y
    for (y in 0 until M) {
        init() // x
        for (x in 0 until N) {
            var pb = bBegPtr
            init() // pb
            init() // s
            init() // k
            for (k in 0 until K) {
                cacheRead8(intToAddress((pa + k * 1).toUInt()))
                multi()
                cacheRead16(intToAddress((pb + x * 2).toUInt()))
                add()
                pb += N * 2
                add()
                iteration()
            }
            cacheWrite32(intToAddress((pc + x * 4).toUInt()))
            iteration()
```

```
        }
        pa += K
        add()
        pc += N * 4 // (*)
        add()
        iteration()
    }
    ext() // exit function
    println("Total ticks: $ticks")
    println("Total memory accesses: $totalAsksCnt")
    println("Cache hits: ${cache.cacheHitCnt}")
    println("Success hits (%): ${cache.cacheHitCnt.toDouble() * 100 / totalAsksCnt}")
}
```

Cache.kt

```
const val cacheWay = 2
const val cacheLineCount = 64
// const val cacheLineSize = 16

const val offsetBits = 4 // log(2, cacheLineSize)
const val cacheSetBits = 5 // log(2, cacheLineCount / cacheWay)
const val tagBits = 10



data class CacheLine(
    var valid: Boolean,
    var dirty: Boolean,
    var tag: Int
)

data class Address(
```

```
        var tag: Int,
        var set: Int
    )

    class MyCache {
        var cacheHitCnt = 0
        var cacheMissCnt = 0
         private val innerCacheData = List(cacheLineCount / cacheWay) { List (cacheWay) {CacheLine(false,  false, 0) }}
        private val lastUsed = MutableList(cacheLineCount / cacheWay) {0}
        fun processData(address : Address, sentDataSizeByte : Int, writing : Boolean) {
            var matched = -1;
            innerCacheData[address.set].forEachIndexed { index, cacheLine ->
                if (cacheLine.tag == address.tag && cacheLine.valid){
                    ticks += 4 // 4 т.к. за 2 такта мы передаём данные от cpu
                    matched = index
                    cacheHitCnt += 1
                    lastUsed[address.set] = index;
                }
            }
            if (matched == -1) {
                cacheMissCnt += 1
                matched = (lastUsed[address.set] + 1) % 2
                val interestingLine = innerCacheData[address.set][matched]
                    ticks += 2 // кэш посылает запрос через 4 такта после miss, возможно нужно учитывать что команда началась 2 такта назад (тогда тут 2) иначе 4
                if (interestingLine.valid && interestingLine.dirty) {
                    ticks += 100
                    ticks += 1 // Эти 4 такта появляются во время
                }
                ticks += 100
```

```
            ticks += 7 //данные идут от памяти к кэшу
            innerCacheData[address.set][matched].dirty = false
            innerCacheData[address.set][matched].valid = true
            innerCacheData[address.set][matched].tag = address.tag
            lastUsed[address.set] = (lastUsed[address.set] + 1) % 2
        }
        ticks += 1//отправка данных обратно к процессору
        if (!writing && sentDataSizeByte == 4)
            ticks += 1//если 32 бита, то нужно 2 такта, а не 1
        if (writing)
            innerCacheData[address.set][matched].dirty = true
    }
}
```