Лабораторная работа №2	Б09	2022
Моделирование схем в Verilog	Васильев Дмитрий Сергеевич	

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

Вся работа велась на Icarus Verilog 11.

Описание: Необходимо аналитически решить поставленную задачу - по известным параметрам системы вычислить, сколько тактов займет выполнение данной в условии функции, а также определить количество и процент кэш попаданий при ее выполнении. После чего, промоделировать работу системы с заданными параметрами на языке verilog. Сравнить полученные результаты.

Вычисление недостающий параметров системы:

1) **`ADDR1_BUS_SIZE, ADDR2_BUS_SIZE`** - в силу того, что по шинам адреса нужно за один такт передать tag и set, то размер этих шин составляет 10 (tag_size) + 5 (set_size) = 15 бит.

(Очевидно, что этого хватит, чтобы по первой шине передать ещё 4 бита оффсета на следующем такте).

2) **CTR1_BUS_SIZE, CTR2_BUS_SIZE -** понятно, что размер шин команд равняется логарифму количества соответствующих команд, округлённому вверх. Ясно, что тогда **CTR1_BUS_SIZE = 3, CTR2_BUS_SIZE = 2.**
3) **CACHE_SIZE -** понятно, что полезных данных в кеше - количество кэш линий умножить на количество полезных данных в кэш линии - то есть 64 * 16B = 8kB.
4) **CACHE_SETS_COUNT -** количество наборов кэш линий - то есть количество линий, делить на ассоциативность: 64 / 2 = 32.
5) **CACHE_SET_SIZE -** логарифм количества наборов: 5.
6) **CACHE_OFFSET_SIZE -** смещение происходит внутри одной кэш линии, поэтому оно варьируется от 0 до 15 байт, поэтому на запись оффеста нужно log(16) = 4 бита.

Аналитическое решение задачи: я решил поставленную задачу на языке C++. Для этого я написал класс Cache, эмулирующий поведение кеша. Для подсчета количества кэш попаданий и промахов, а также для вычисления количества тактов - я создал ещё один класс - counter.

Логика поведения Cache довольно простая - в нём хранится полезная информация о кэш линиях (tag, valid, dirty, lastUsed (true стоит у той кэш линии из двух в ‘наборе’, которая использовалась в последний раз)) (сами данные не хранятся). При получении очередной команды из main, cache определяет, если ли у него нужная линия: для этого он проверяет все свои

линии из группы с данным сетом, и если находит ту, у которой tag совпадает с tag - ом запроса, значит случится cache hit, иначе - cache miss. В каждом из случаев необходимо добавить нужное количество таков (не забыв в случае miss посчитать время общения с памятью, а также проверить линию на ‘dirty’ - если да, то обращение к памяти будет 2: первое - для сохранения туда информации из ‘грязной’ линии, а второе - для получения в нее новой порции данных) и обновить состояния кэш линий - например, после команды write, независимо от того, попали ли мы в кэш - кэш линия станет ‘грязной’ - нужно не забыть выставить ей соответствующее состояния.

Счётчик тактов разбит на два слагаемых: clock и extraTicks. Первый считает время на обработку команд, связанных с памятью (read / write), а второй суммирует время на команды внутри процессора (арифметика, выход из функции, итерации циклов и тп).

Результаты аналитического решения таковы:

```
Total ticks: 4223640 + 998660 = 5222300 (clock + extraTicks)
Total memory accesses: 249600
Cache hits: 228080
Part of hits: 0.913782
```

Моделирование заданной системы на Verilog: в выданной модели 3 основных блока: cache, cpu и mem (каждый их них - отдельный модуль). Есть ещё несколько вспомогательных модулей, но о них разумнее рассказать в следующем пункте.

Во всей системе поддерживается следующий инвариант: при clk = 0 все модули могут только читать из входящих в них inout проводов, а при clk = 1,

они могут записывать туда данные. Мне этот подход показался удобным: в силу того, что при аккуратном написании кода, всегда понятно, кто сейчас владеет шиной, - не нужно задумываться, может ли владелец в неё что - то написать: если синхронизация 1 - пишет спокойно.

Для реализации ожидания внутри always блоков, мне помог макрос, придуманный моими одногруппниками:

```
`define delay(TIME, CLOCK) \
   for (int i = 0; i < TIME; i++) begin \
       wait(clk == (i + !CLOCK) % 2); \
   end
```

Подобное ухищрения понадобилось из - за того, что при написании ‘#n’ внутри always блоков, программа вела себя непредсказуемым образом, возникали гонки. Поэтому, подобный маркрос упрощает жизнь - ты понимаешь, что произойдет, после его использования, в отличие от ‘#n’.

Остановимся чуть подробнее на реализованных модулях:

Cache - самый крупный модуль так как ему нужно уметь общаться и с памятью и с процессором. Как и в других модулях, у кэша есть регистры, подключенные к его inout и out проводам, несколько флагов, и самое главное - кэш линии. Также, дополнительно к кэш линиям хранится массив из 32 (количество кэш линий / ассоциативность) битов, i - тый бит в нем отвечает за то, 0 или 1 элемент ‘группы’ с сетом i использовался последним.

Обработка команд от процессора происходит в always блоке. Всего существует 3 вида ‘полезных’ (то есть не равных NOP) команд, которые может получить кэш от процессора: readN, writeN и invalidate.

Команда read устроена так: мы за два такта читаем адрес, потом так же, как в C++ проверям, есть ли у нас указанные данные в кэше (смотрим на наши линии с сетом из адреса и проверяем их валидность, а также совпадение тегов). В случае попадания - запоминаем номер кэш линии, в которой хранятся данные. В случае промаха - сгружаем dirty line, если нужно, и получаем данные из памяти. Теперь, независимо от того, промахнулись мы или нет - у нас есть кэш линия, в которой лежат нужные процессору данные. Поэтому просто отдадим нужное количество бит обратно процессору.

Стоит сказать, как происходит чтение / запись кэшем из памяти / в память. Рассмотрим чтение из памяти: в моей реализации, кэш ставит на шину C2 команду, означающую чтение, на шину адреса ставит нужный адрес, назначает вспомогательную переменную `wantsToReadFromMemory = 1,` и ждет, пока она не станет 0. (просто пишем `wait(wantsToReadFromMemory == 0)).` В это же время в другом always блоке на clk, мы проверяем, что `wantsToReadFromMemory == 1, clk == 0` (то есть можно читать) и `C2 == C2_RESPONSE.` Если это так, то память готова отдать нам запрошенные данные - нужно просто их прочитать за 8 тактов и вернуть `wantsToReadFromMemory = 0.`

Аналогичным образом устроены запись в память и write из процессора в кэш.

Инвалидация устроена еще проще: если ячейка лежит в одной из кэш линий, то мы делаем эту линию не валидной, а если она была грязной, то выгружаем в память. Иначе - ничего делать не надо.

Теперь должно быть понятно, как устроена память: в initial блоке, она заполняет себя так, как сказано в условии. При получении команды, например на запись, она считывает адрес, 8 порций по 16 бит и ждёт, пока от первой полученной команды не пройдет 100 тактов. После этого, записав полученные данные в себя, память на 1 такт выставляет респонс, давая кэшу понять, что закончила работу.

Дампы и ресеты реализованы через отдельные always блоки. Очевидно, как записать каждый из них в коде.

Работу процессора проще объяснить в следующем блоке.

Воспроизведение задачи на Verilog: для того чтобы проверить кэш и память, нам понадобилось написать подобие процессора. В силу того, что в поставленной задаче будут использоваться только функции read8, read16, write32 - ровно они и будут реализованы в процессоре. Их реализация крайне проста: мы отправляем кэшу команду, адрес и, при необходимости - данные. А дальше идея такая же, как была с `wantsToReadFromMemory` в кэше: Сеттим флаг, в другом always блоке он проверяется, ожидая, когда кэш ответит респонзом. После этого, при необходимости, читаем данные из ответа кэша, возвращая флаг в исходное состояние.

Все приготовления проделаны - теперь можно моделировать задачу. Для этого в initial блоке cpu запишем те самые вложенные циклы из условия (функцию mmul). В случаях, когда нам нужно получить ячейку из памяти - будем применять команду read (read8 для a и read16 для b). Когда же мы записываем переменную s в память - процессор будет применять команду write32.

Для подсчета статистики, мне понадобился модуль counter, с полями hits, misses. Также, для моделирования системы пригодился модуль тест, в котором и располагаются cpu, mem и cache, соединенные проводами, а также в нём написан большой цикл, меняющий синхронизацию.

После столь долгих приготовлений можно, наконец, сравнить результаты аналитического решения и моделирования, и, о чудо, они совпали во всем с точностью до такта!

```
Total ticks: 4223640 + 998660 = 5222300
Total memory accesses: 249600
Cache hits: 228080
Part of hits: 0.913782
```

Лог выполнения программы можно найти в репозитории (log.txt).
Дамп файлы находятся там же (называются cDump.txt и mDump.txt).
Также в папке /analytics находится проект на C++, эмулирующий эту задачу.

Листинг кода приведен ниже.

Листинг кода:

testbench.sv

```
`include "constants.sv"
`include "mem.sv"
`include "cpu.sv"
`include "cache.sv"

module test;
  bit clk = 0;
  wire [`ADDR1_BUS_SIZE - 1 : 0] a1;
  wire [`DATA1_BUS_SIZE - 1 : 0] d1;
  wire [`CTR1_BUS_SIZE - 1 : 0] c1;
  wire [`DATA2_BUS_SIZE - 1 : 0] d2;
  wire [`CTR2_BUS_SIZE - 1 : 0] c2;
  wire [`ADDR2_BUS_SIZE - 1 : 0] a2;

  bit C_DUMP;
  bit M_DUMP;
  bit RESET;

  cache myChace(clk, C_DUMP, RESET, a1, d1, c1, d2, c2, a2);
  mem myMem(clk, M_DUMP, RESET, d2, c2, a2);
  cpu myCpu(clk, a1, d1, c1);

 initial begin
     for(int i = 0; i < 15000000; i++) begin
         #1;
         clk = 1 - clk;
     end
 end

endmodule

module counter;
```

```
 integer hits = 0;
 integer misses = 0;
 integer mDump = 0;
 integer cDump = 0;
endmodule
```

cpu.sv

```
`include "constants.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 [`DATA1_BUS_SIZE - 1 : 0] cpu_d1 = 'z;
 reg [`ADDR1_BUS_SIZE - 1 : 0] cpu_a1 = 'z;
 reg [`CTR1_BUS_SIZE - 1 : 0]  cpu_c1 = 'z;
 assign a1 = cpu_a1;
 assign d1 = cpu_d1;
 assign c1 = cpu_c1;

 reg [ 7 : 0] resultRead8   = 'z;
 reg [15 : 0] resultRead16  = 'z;
 reg [31 : 0] resultWrite32 = 'z;

 bit wantsToRead8 = 0;
 bit wantsToRead16 = 0;
 bit wantsToWrite32 = 0;

 integer M = 64;
 integer N = 60;
 integer K = 32;
 integer a = 0;
 integer b = M * K;
 integer c = b + K * N * 2;
 integer s = 0;

 integer extraTicks = 0;

 int pa = 0, pb = 0, pc = 0;

 initial begin
   counter.mDump = $fopen("mDump.txt", "w");
   counter.cDump = $fopen("cDump.txt", "w");
```

```
    counter.log = $fopen("log.txt", "w");
    mmul();

      $display("Total ticks: %0t + %0t = %0t", $time / 2, extraTicks, $time / 2 +
extraTicks);
    $display("Total memory accesses: %0d", counter.hits + counter.misses);
    $display("Cache hits: %0d", counter.hits);
        $display("Part of hits: %0f", counter.hits * 1.0 / (counter.hits +
counter.misses));
    test.M_DUMP = 1;
    test.C_DUMP = 1;
    test.RESET = 1;
    wait(clk == 0);
    wait(clk == 1);
    $fclose(counter.log);
    $fclose(counter.mDump);
    $fclose(counter.cDump);
 end

 task mmul;
     $fdisplay(counter.log, "Stardted working in mull(), time =  %0d", $time / 2);
     pa = a;
     pc = c;
     extraTicks += 3; // pa, pc, y init
     for(int y = 0; y < M; y++) begin
         extraTicks += 1; // x init
         for(int x = 0; x < N; x++) begin
             pb = b;
             s = 0;
             extraTicks += 3; // b, s, k init
             for(int k = 0; k < K; k++) begin
                 read8 (pa + k); // resultRead8
                 read16 (pb + x * 2); //resultRead16
                 s += resultRead8 * resultRead16;
                 pb += 2 * N;
                 extraTicks += 5 + 1 + 1; // mul, add, ad
                 extraTicks += 1; // loop
             end
               $fdisplay(counter.log, "Stardted writing 's', x = %0d, y = %0d, time =
%0d", x, y, $time / 2);
             write32 (pc + x * 4, s);
```

```
                $fdisplay(counter.log, "Finished writing 's', x = %0d, y = %0d, time =
%0d", x, y, $time / 2);
              extraTicks += 1; // loop
            end
            extraTicks += 1; // add
            extraTicks += 1; // add
            pa += K;
            pc += 4 * N;
            extraTicks += 1; // loop
          end
          extraTicks += 1; // func exit
          $fdisplay(counter.log, "Ended working in mull(), time =  %0d", $time / 2);
 endtask
  task read8 (reg [`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : 0] address);
   wait(clk == 1);
   cpu_c1 = `C1_READ8;
   cpu_a1 = address[`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : `CACHE_OFFSET_SIZE];
   `delay(2, 1)
   cpu_c1 = 'z;
   cpu_d1 = 'z;
   cpu_a1 = address[`CACHE_OFFSET_SIZE - 1 : 0];
   wantsToRead8 = 1;
   wait(wantsToRead8 == 0);
 endtask

 task read16 (reg [`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : 0] address);
   wait(clk == 1);
   cpu_c1 = `C1_READ16;
   cpu_a1 = address[`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : `CACHE_OFFSET_SIZE];
   `delay(2, 1)
   cpu_c1 = 'z;
   cpu_d1 = 'z;
   cpu_a1 = address[`CACHE_OFFSET_SIZE - 1 : 0];
   wantsToRead16 = 1;
   wait(wantsToRead16 == 0);
 endtask

 task write32 (reg [`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : 0] address, reg[31 : 0]
data);
   wait(clk == 1);
   cpu_c1 = `C1_WRITE32;
   cpu_a1 = address[`ADDR1_BUS_SIZE + `CACHE_OFFSET_SIZE - 1 : `CACHE_OFFSET_SIZE];
```

```
    cpu_d1 = data[`DATA1_BUS_SIZE - 1 : 0];
    `delay(2, 1)
    cpu_c1 = 'z;
    cpu_a1 = address[`CACHE_OFFSET_SIZE - 1 : 0];
    cpu_d1 = data[`DATA1_BUS_SIZE * 2 - 1 : `DATA1_BUS_SIZE];
    wantsToWrite32 = 1;
    wait(wantsToWrite32 == 0);
  endtask

  always @(negedge clk) begin
      if(wantsToRead8 == 1 && c1 == `C1_RESPONSE) begin
         $fdisplay(counter.log, "Cpu got response in read8, time = %0d", $time / 2);
         resultRead8 = d1[7 : 0];
         wantsToRead8 = 0;
      end else if (wantsToRead16 == 1 && c1 == `C1_RESPONSE) begin
         $fdisplay(counter.log, "Cpu got response in read16, time = %0d", $time / 2);
         resultRead16 = d1[15 : 0];
         wantsToRead16 = 0;
      end else if (wantsToWrite32 == 1 && c1 == `C1_RESPONSE) begin
          $fdisplay(counter.log, "Cpu got response in write32, time = %0d", $time / 2);
          cpu_d1 = 'z;
          wantsToWrite32 = 0;
      end
  end
endmodule
```

mem.sv

```
`include "constants.sv"

module mem #(parameter _SEED = 225526) (input clk, input M_DUMP, input RESET, inout
wire [`DATA2_BUS_SIZE - 1 : 0] d2, inout wire [`CTR2_BUS_SIZE - 1 : 0] c2, input wire
[`ADDR2_BUS_SIZE - 1 : 0] a2);
 integer SEED = _SEED;
 logic [`BITS_IN_BYTE - 1 : 0] memLines[`MEM_SIZE - 1 : 0];

 reg [`DATA2_BUS_SIZE - 1 : 0] mem_d2 = 'z;
 reg [`CTR2_BUS_SIZE - 1 : 0]  mem_c2 = 'z;
 assign d2 = mem_d2;
 assign c2 = mem_c2;

 bit [`ADDR2_BUS_SIZE - 1 : 0] readedSetTag;

 initial begin
     for (int i = 0; i < `MEM_SIZE; i += 1) begin
         memLines[i] = $random(SEED)>>16;
     end
 end

 integer i = 0;

 always @(negedge clk) begin
     if(c2 == `C2_READ_LINE) begin
        $fdisplay(counter.log, "Mem got C2_READ_LINE request, time = %0d", $time / 2);
       readedSetTag = a2;
       `delay(1, 0)

       mem_c2 = `C2_NOP;
       `delay(`MEM_CTR_WAIT * 2 - 2, 1)
```

```
        mem_c2 = `C2_RESPONSE;
        for(i = 0; i < `SEND_FROM_MEM; i++) begin
            mem_d2[`BITS_IN_BYTE - 1 : 0] =                memLines[readedSetTag * (1 <<
`CACHE_OFFSET_SIZE) + 2 * i];
            mem_d2[`DATA2_BUS_SIZE - 1 : `BITS_IN_BYTE] = memLines[readedSetTag * (1 <<
`CACHE_OFFSET_SIZE) + 2 * i + 1];
           `delay(2, 1)
        end
        mem_c2 = 'z;
        mem_d2 = 'z;
      end else if(c2 == `C2_WRITE_LINE) begin
        $fdisplay(counter.log, "Mem got C2_WRITE_LINE request, time = %0d", $time / 2);
        readedSetTag = a2;
        for(i = 0; i < `SEND_FROM_MEM; i++) begin
                    memLines[readedSetTag * (1 << `CACHE_OFFSET_SIZE) + 2 * i] =
d2[`BITS_IN_BYTE - 1 : 0];
                memLines[readedSetTag * (1 << `CACHE_OFFSET_SIZE) + 2 * i + 1] =
d2[`DATA2_BUS_SIZE - 1 : `BITS_IN_BYTE];
           `delay(2, 0)
        end
       `delay(`MEM_CTR_WAIT * 2 - `SEND_FROM_MEM * 2 - 3, 0)
        mem_c2 = `C2_RESPONSE;
          $fdisplay(counter.log, "Mem send C2_RESPONSE respoce after reading, time =
%0d", $time / 2);
        `delay(2, 1)
        mem_c2 = 'z;
      end
  end

  always @(posedge M_DUMP) begin
      $fdisplay(counter.log, "Mem DUMP, time = %0d", $time / 2);
      for (i = 0; i < `MEM_SIZE; i++ ) begin
          $fdisplay(counter.mDump, "%d : %d", i, memLines[i]);
      end
  end

  always @(posedge RESET) begin
     $fdisplay(counter.log, "Mem RESET, time = %0d", $time / 2);
      SEED = _SEED;
       for (int i = 0; i < `MEM_SIZE; i += 1) begin
          memLines[i] = $random(SEED)>>16;
       end
```

```
 end
endmodule
```

cache.sv

```
`include "constants.sv"

module cache(input clk, input C_DUMP, input RESET, input wire [`ADDR1_BUS_SIZE - 1 :
0] a1, inout wire [`DATA1_BUS_SIZE - 1 : 0] d1, inout wire [`CTR1_BUS_SIZE - 1 : 0]
c1, inout wire [`DATA2_BUS_SIZE - 1 : 0] d2, inout wire [`CTR2_BUS_SIZE - 1 : 0] c2,
output wire [`ADDR2_BUS_SIZE - 1 : 0] a2);
 reg [`CACHE_LINE_WHOLE_SIZE_BITS - 1 : 0] lines[`CACHE_LINE_COUNT / `CACHE_WAY - 1 :
0][`CACHE_WAY - 1 : 0];
 bit lastUsed [`CACHE_LINE_COUNT / `CACHE_WAY - 1 : 0];

 reg [`DATA1_BUS_SIZE - 1 : 0] cache_d1 = 'z;
 reg [`CTR1_BUS_SIZE - 1 : 0] cache_c1 = 'z;
 reg [`DATA2_BUS_SIZE - 1 : 0] cache_d2 = 'z;
 reg [`CTR2_BUS_SIZE - 1 : 0] cache_c2 = 'z;
 reg [`ADDR2_BUS_SIZE - 1 : 0] cache_a2 = 'z;
 assign d1 = cache_d1;
 assign c1 = cache_c1;
 assign d2 = cache_d2;
 assign c2 = cache_c2;
 assign a2 = cache_a2;

 initial begin
     for(int i = 0; i < `CACHE_LINE_COUNT / `CACHE_WAY; i++) begin
         for(int j = 0; j < `CACHE_WAY; j++) begin
           lines[i][j] = 0;
         end
         lastUsed[i] = 0;
     end
 end

 bit [`CACHE_TAG_SIZE - 1 : 0] readedTag;
```

```
 bit [`CACHE_SET_SIZE - 1 : 0] readedSet;
 bit [`CACHE_OFFSET_SIZE - 1 : 0] readedOffset;

 bit [`MAX_CPU_ASK - 1 : 0] tmpBuffer;

 integer target = -1; // it helps to write back to cpu only once
 integer whatToDoWithCpu = 0; // just last command
 integer i = 0;

 integer wantsToWriteToMemory = 0;
 integer wantsToReadFromMemory = 0;

 always @(negedge clk) begin
   // READ
   if (c1 == `C1_READ8 || c1 == `C1_READ16 ||c1 == `C1_READ32) begin
       whatToDoWithCpu = c1;
       readedTag[`CACHE_TAG_SIZE - 1 : 0] = a1[`ADDR1_BUS_SIZE - 1 : `CACHE_SET_SIZE];
       readedSet[`CACHE_SET_SIZE - 1 : 0] = a1[`CACHE_SET_SIZE - 1 : 0];
       `delay(2, 0)
       readedOffset[`CACHE_OFFSET_SIZE - 1 : 0] = a1[`CACHE_OFFSET_SIZE - 1 : 0];
       `delay(1, 0)
       target = -1;
       cache_c1 = `C1_NOP;
       for(i = 0; i < `CACHE_WAY && (target == -1); i++) begin
                if(lines[readedSet][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BITS - 1 :
`CACHE_LINE_SIZE_BITS] == readedTag) begin
           // hit
           if(lines[readedSet][i][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] == 1) begin
               $fdisplay(counter.log, "Cache hit in read, time = %0d", $time / 2);
               lastUsed[readedSet] = i;
               counter.hits++;
                  `delay(`CACHE_HIT_WAIT * 2 - 3 - 1, 1) // - 3 cause we have already
wait 3 half ticks. - 1 cause we read info after 0.5 ticks
               target = i;
           end
           end
       end

       // miss
       if(target == -1) begin
           $fdisplay(counter.log, "Cache mis in read, time = %0d", $time / 2);
           counter.misses++;
```

```
            target = 1 - lastUsed[readedSet];
            `delay(`CACHE_MIS_WAIT * 2 - 4, 1)

            // DIRTY
             if(lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID - `DIRTY]
== 1 && lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] == 1) begin
                $fdisplay(counter.log, "Dirty line in read, time = %0d", $time / 2);
                cache_c2 = `C2_WRITE_LINE;
                cache_a2[`CACHE_SET_SIZE - 1 : 0] = readedSet;
                                cache_a2[`ADDR2_BUS_SIZE  -  1  :  `CACHE_SET_SIZE]  =
lines[readedSet][target][`CACHE_LINE_SIZE_BITS    +    `CACHE_TAG_SIZE    -    1    :
`CACHE_LINE_SIZE_BITS];
                wantsToWriteToMemory = 1;
                wait(wantsToWriteToMemory == 0);
                `delay(1, 0)
            end
            cache_c2[`CTR2_BUS_SIZE - 1 : 0] = `C2_READ_LINE;
            cache_a2[`CACHE_SET_SIZE - 1 : 0] = readedSet;
            cache_a2[`ADDR2_BUS_SIZE - 1 : `CACHE_SET_SIZE] = readedTag;
            `delay(2, 1)
            cache_c2[`CTR2_BUS_SIZE - 1 : 0] = 'z;
            cache_a2[`ADDR2_BUS_SIZE - 1 : 0] = 'z;
            cache_d2[`DATA2_BUS_SIZE - 1 : 0] = 'z;
            wantsToReadFromMemory = 1;
            wait(wantsToReadFromMemory == 0);

              lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID - `DIRTY] =
0;
            lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] = 1;
                lines[readedSet][target][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BITS - 1 :
`CACHE_LINE_SIZE_BITS] = readedTag;
            lastUsed[readedSet] = target;
            `delay(1, 0)
        end

        cache_c1 = `C1_RESPONSE;
        case (whatToDoWithCpu)
        `C1_READ8 : begin
        cache_d1[7 : 0] =  lines[readedSet][target][readedOffset * 8 +: 8];
        end
        `C1_READ16 : begin
        cache_d1[15 : 0] =  lines[readedSet][target][readedOffset * 8 +: 16];
```

```
        end
        `C1_READ32 : begin
            cache_d1[15 : 0] =  lines[readedSet][target][readedOffset * 8 +: 16];
            `delay(2, 1)
            cache_d1[15 : 0] =  lines[readedSet][target][16 + readedOffset +: 16];
        end
        endcase

        `delay(2, 1)
        cache_c1 = 'z;
        cache_d1 = 'z;
    end
    // WRITE
    else if (c1 == `C1_WRITE8 || c1 == `C1_WRITE16 || c1 == `C1_WRITE32) begin
        whatToDoWithCpu = c1;
        readedTag[`CACHE_TAG_SIZE - 1 : 0] = a1[`ADDR1_BUS_SIZE - 1 : `CACHE_SET_SIZE];
        readedSet[`CACHE_SET_SIZE - 1 : 0] = a1[`CACHE_SET_SIZE - 1 : 0];
        tmpBuffer[15 : 0] = d1;
        `delay(2, 0)
        readedOffset[`CACHE_OFFSET_SIZE - 1 : 0] = a1[`CACHE_OFFSET_SIZE - 1 : 0];
        tmpBuffer[31 : 16] = d1; // better not to do like that when command != write32;
        `delay(1, 0)
        target = -1;
        cache_c1 = `C1_NOP;
        for(i = 0; i < `CACHE_WAY && (target == -1); i++) begin
                if(lines[readedSet][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BITS - 1 :
`CACHE_LINE_SIZE_BITS] == readedTag) begin
            // hit
            if(lines[readedSet][i][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID]) begin
                $fdisplay(counter.log, "Cache hit in write, time =  %0d", $time / 2);
                lastUsed[readedSet] = i;
                counter.hits++;
                   `delay(`CACHE_HIT_WAIT * 2 - 3 - 1, 1) // - 3 cause we have already
wait 3 half ticks. - 1 cause we read info after 0.5 ticks
                target = i;
            end
            end
        end

        // miss
        if(target == -1) begin
            $fdisplay(counter.log, "Cache mis in write, time = %0d", $time / 2);
```

```
            counter.misses++;
            target = 1 - lastUsed[readedSet];
            `delay(`CACHE_MIS_WAIT * 2 - 4, 1)

            // DIRTY
             if(lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID - `DIRTY]
== 1 && lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] == 1) begin
                $fdisplay(counter.log, "Dirty line in write, time = %0d", $time / 2);
                cache_c2 = `C2_WRITE_LINE;
                cache_a2[`CACHE_SET_SIZE - 1 : 0] = readedSet;
                                cache_a2[`ADDR2_BUS_SIZE  -  1  :  `CACHE_SET_SIZE]  =
lines[readedSet][target][`CACHE_LINE_SIZE_BITS    +    `CACHE_TAG_SIZE    -    1    :
`CACHE_LINE_SIZE_BITS];
                wantsToWriteToMemory = 1;
                wait(wantsToWriteToMemory == 0);
                `delay(1, 0)
            end

            cache_c2[`CTR2_BUS_SIZE - 1 : 0] = `C2_READ_LINE;
            cache_a2[`CACHE_SET_SIZE - 1 : 0] = readedSet;
            cache_a2[`ADDR2_BUS_SIZE - 1 : `CACHE_SET_SIZE] = readedTag;
            `delay(2, 1)

            cache_c2[`CTR2_BUS_SIZE - 1 : 0] = 'z;
            cache_a2[`ADDR2_BUS_SIZE - 1 : 0] = 'z;
            cache_d2[`DATA2_BUS_SIZE - 1 : 0] = 'z;
            wantsToReadFromMemory = 1;
            wait(wantsToReadFromMemory == 0);

            lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] = 1;
                lines[readedSet][target][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BITS - 1 :
`CACHE_LINE_SIZE_BITS] = readedTag;
            lastUsed[readedSet] = target;
            `delay(1, 0)
        end

        // save data:
        lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID - `DIRTY] = 1;

        case (whatToDoWithCpu)
        `C1_WRITE8 : begin
            lines[readedSet][target][readedOffset * 8 +: 8] = tmpBuffer[7 : 0];
```

```
        end
        `C1_WRITE16 : begin
            lines[readedSet][target][readedOffset * 8 +: 16] = tmpBuffer[15 : 0];
        end
        `C1_WRITE32 : begin
            lines[readedSet][target][readedOffset * 8 +: 32] = tmpBuffer[31 : 0];
        end
        endcase


        cache_c1 = `C1_RESPONSE;
        `delay(2, 1)
        cache_c1 = 'z;
        cache_d1 = 'z;
    end
    // INVALIDATE
    else if (c1 == `C1_INVALIDATE_LINE) begin
        $fdisplay(counter.log, "Line invalidation, time = %0d", $time / 2);
        whatToDoWithCpu = cache_c1;
        readedTag[`CACHE_TAG_SIZE - 1 : 0] = a1[`ADDR1_BUS_SIZE - 1 : `CACHE_SET_SIZE];
        readedSet[`CACHE_SET_SIZE - 1 : 0] = a1[`CACHE_SET_SIZE - 1 : 0];
        `delay(2, 0)
        readedOffset[`CACHE_OFFSET_SIZE - 1 : 0] = a1[`CACHE_OFFSET_SIZE - 1 : 0];
        `delay(1, 0)
        target = -1;
        cache_c1 = `C1_NOP;
        for(i = 0; i < `CACHE_WAY && (target == -1); i++) begin
                 if(lines[readedSet][i][`CACHE_TAG_SIZE + `CACHE_LINE_SIZE_BITS - 1 :
`CACHE_LINE_SIZE_BITS] == readedTag) begin
            // hit
            if(lines[readedSet][i][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID]) begin
                lastUsed[readedSet] = i;
                counter.hits++; // ?
                   `delay(`CACHE_HIT_WAIT * 2 - 3 - 1, 1) // - 3 cause we have already
wait 3 half ticks. - 1 cause we read info after 0.5 ticks
                target = i;

                //  invalidation
                     if(lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID -
`DIRTY] == 1) begin
                    cache_c2 = `C2_WRITE_LINE;
                    cache_a2[`CACHE_SET_SIZE - 1 : 0] = readedSet;
```

```
                    cache_a2[`ADDR2_BUS_SIZE - 1 : `CACHE_SET_SIZE] = readedTag;
                    wantsToWriteToMemory = 1;
                    wait(wantsToWriteToMemory == 0);
                    `delay(1, 0)
                    end
                        lines[readedSet][target][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID -
`DIRTY] = 0;
                    lines[readedSet][i][`CACHE_LINE_WHOLE_SIZE_BITS - `VALID] = 0;
                end
            end
        end
    end
 end

 always @(clk) begin
     if(wantsToWriteToMemory == 1 && clk == 1) begin
         for(i = 0; i < `SEND_FROM_MEM; i++) begin
                      cache_d2  =  lines[readedSet][target][`DATA2_BUS_SIZE  *  i  +:
`DATA2_BUS_SIZE];
            `delay(2, 1)
             cache_c2[`CTR2_BUS_SIZE - 1 : 0] = 'z;
             cache_a2[`ADDR2_BUS_SIZE - 1 : 0] = 'z;
         end
         wantsToWriteToMemory = 2;
     end else if (clk == 0 && wantsToReadFromMemory == 1 && c2 == `C2_RESPONSE) begin
           for(i = 0; i < `SEND_FROM_MEM; i++) begin
               lines[readedSet][target][`DATA2_BUS_SIZE * i +: `DATA2_BUS_SIZE] = d2;
               `delay(2, 0)
           end
           wantsToReadFromMemory = 0;
     end
 end

 always @(negedge clk) begin
     if (wantsToWriteToMemory == 2 && c2 == `C2_RESPONSE) wantsToWriteToMemory = 0;
 end

 always @(posedge C_DUMP) begin
   $fdisplay(counter.log, "Cache DUMP, time = %0d", $time / 2);
   for (i = 0; i < `CACHE_LINE_COUNT / `CACHE_WAY; i++ ) begin
          $fdisplay(counter.cDump, "%d:  %d  %d  %d  %d", 2 * i, lines[i][0][127  :  96],
lines[i][0][95 : 64], lines[i][0][63 : 31], lines[i][0][31 : 0]);
```

```
        $fdisplay(counter.cDump, "%d: %d %d %d %d", 2 * i + 1, lines[i][1][127 : 96],
lines[i][1][95 : 64], lines[i][1][63 : 31], lines[i][1][31 : 0]);
    end
  end

  always @(posedge RESET) begin
    $fdisplay(counter.log, "Cache RESET, time = %0d", $time / 2);
    for(int i = 0; i < `CACHE_LINE_COUNT / `CACHE_WAY; i++) begin
        for(int j = 0; j < `CACHE_WAY; j++) begin
            lines[i][j] = 0;
        end
            lastUsed[i] = 0;
    end
  end
endmodule
```

constans.sv

```
// Created by my unimates
`define delay(TIME, CLOCK) \
    for (int i = 0; i < TIME; i++) begin \
        wait(clk == (i + !CLOCK) % 2); \
    end

`define BITS_IN_BYTE 8

`define CACHE_LINE_COUNT 64
`define CACHE_WAY 2
`define MEM_SIZE 524288 // 2 ^ 19

`define SEND_FROM_MEM (`CACHE_LINE_SIZE  / `DATA_BUS_SIZE)

// time
`define CACHE_HIT_WAIT 6
`define CACHE_MIS_WAIT 4
`define MEM_CTR_WAIT 100

// in bytes
`define CACHE_LINE_SIZE 16
```

```
`define DATA_BUS_SIZE (16 / `BITS_IN_BYTE)

// in bits
`define CACHE_OFFSET_SIZE 4 // log2(CACHE_LINE_SIZE);
`define CACHE_SET_SIZE 5  //  log2(CACHE_LINE_COUNT / CACHE_WAY);
`define CACHE_TAG_SIZE 10
`define CACHE_LINE_SIZE_BITS (`CACHE_LINE_SIZE * `BITS_IN_BYTE)
`define   CACHE_LINE_WHOLE_SIZE_BITS   (`VALID   +   `DIRTY   +   `CACHE_TAG_SIZE   +
`CACHE_LINE_SIZE_BITS)
`define VALID 1
`define DIRTY 1
`define MAX_CPU_ASK 32 // read32 / write32

`define ADDR1_BUS_SIZE (`CACHE_SET_SIZE + `CACHE_TAG_SIZE)
`define ADDR2_BUS_SIZE (`CACHE_SET_SIZE + `CACHE_TAG_SIZE)

`define DATA1_BUS_SIZE 16
`define DATA2_BUS_SIZE 16

`define CTR1_BUS_SIZE 3
`define CTR2_BUS_SIZE 2

// comands
`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
```