|  |  |  |  |
| --- | --- | --- | --- |
| Лабораторная работа №2 | Б10 | 2022 | |
| Моделирование схем в Verilog | Барковская Мария Александровна | |
|  | |

**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.

**Описание:** в качестве варианта нам были даны некоторые параметры системы и задача, которую надо сначала решить аналитически на основании полученных параметров системы, а затем промоделировать ее на языке Verilog и сравнить полученные результаты (а именно: сколько тактов займет выполнение предоставленной нам программы).

Вычисление недостающих параметров системы.

* **CACHE\_SIZE:** весь размер кэша = количество кэш-линий на размер кэш-линий (то есть `CACHE\_LINE\_SIZE \* `CACHE\_LINE\_COUNT) = 16 \* 64 = 1024
* **CACHE\_SETS\_COUNT:** ассоциативность равна двум, то есть все кэш линии мы храним в наборах по 2, тогда количество наборов кэщ линий равно `CACHE\_LINE\_COUNT / `CACHE\_WAY = 64 / 2 = 32
* **CACHE\_SET\_SIZE:** логарифм от количества наборов, то есть 5
* **CACHE\_OFFSET\_SIZE:** offset задает смещение внутри кэш-линии, поэтому его размер – логарифм от размера кэш-линии, то есть 4
* **ADDR1\_BUS\_SIZE**, **ADDR2\_BUS\_SIZE:** максимальный размер передаваемой в один момент (в нашем случае – это первый такт из двух, так как во второй передается offset, занимающий всего 4 бита) времени информации равен `SET\_SIZE + `CACHE\_TAG\_SIZE = 15 бит
* **CTR1\_BUS\_SIZE**, **CTR2\_BUS\_SIZE:** размер шин для передачи команд равен количеству всех возможных для передачи по этой шине команд (записываем в бинарной записи, поэтому нужно будет взять двоичный логарифм от десятичного числа команд, округленный вверх). Получаем размер шин, равный 3 и 2 соответственно

Аналитическое решение задачи

Поставленная задача была решена аналитически в виде кода на языке программирования kotlin (код представлен ниже). Функция main симулирует задачу, а класс Cache – схематичную работу кэша (те его действия, которые помогают посчитать количество тактов).

В кэше хранятся наборы по две кэш-линии (каждая кэш-линия представляет из себя сами данные и набор информации о них и их состоянии (valid, dirty, tag, lru (lru = true, если в своем наборе кэш-линия не использовалась дольше))).

При получении запроса к памяти (то есть соответстующей команды) кэш проверяет, есть ли у него нужные данные (то есть, находятся ли они в какой-то из кэш-линий (для этого нужно пройтись по двум кэш-линиям из набора с номером set и если у какой-то из них совпал tag с запрошенным, то в кэше есть нужные данные, иначе – нужно обращаться к памяти)). В каждом случае (нашлись данные или нет) дополнительно учитывается время ответа кэша, а в случае, если кэш не нашел у себя этих данных, и при этом кэш-линия, которую мы «вытесняем», “dirty”, то учитываем, что обращения к памяти необходимо два: для сохранения текущей информации этой кэш-линии и для получения в нее из памяти новой.

tickCounter подсчитывает количество тактов, затраченных на выполнение поставленной нам задачи (учитывая не только работу кэша и памяти, но и сложение, умножение и так далее).

Моделирование заданной системы на Verilog + Воспроизведение задачи на Verilog

Так как мы моделировали систему “процессор-кэш-память”, то в моей реализации есть три основных блока, эту самую систему реализующие: cpu, cache и memory.

Права на чтение/запись данных для модулей в системе такие: при clk == 1 есть право записывать данные на провода, при clk == 0 – cчитывать с них (эта разница в такт позволяет удобно пользоваться inout шинами, так как мы не можем записать данные, пока с другой стороны не прочитали предыдущие).

Во время реализации возникла определенная проблема ожидания внутри always блоков: неопределенное значение clk после выхода из задержки и возникновение гонок. Для решения этой проблемы мои одногруппники (Артем Пешков и Тимофей Малов) придумали макрос ‘delay’ (реализацию можно найти в листинге кода).

Модуль cache

Реализованный в Verilog-е, кэш работает очень похоже на свой аналитический вариант, с той лишь поправкой, что теперь работа с памятью прописана напрямую. К Кэшу подключены шины, отвечающие за взаимодействие с процессором и памятью, а также в кэше хранятся сами данные кэша (те самые наборы кэш-линий, о которых говорилось в описании аналитического решения), и массив, соответствующий lru, описанному ранее. Всего мы можем получить от процессора три типа команд (не считая C1\_NOP):

* Read: позволяет прочитать 8, 16 или 32 бита, в зависимости от команды (указано в ее названии). Происходит чтение адреса по шине a1 за два такта (сначала tag и set, потом offset), проверка на наличие данных в кэше, обработка в случае промаха (если не нашли, то обращаемся к памяти, получая в кэш-линию нужные данные) и возвращение процессору запрошенных данных
* Write: позволяет записать 8, 16 или 32 бита, в зависимости от команды (указано в ее названии). Происходит чтение адреса по шине a1 за два такта (сначала tag и set, потом offset), проверка на наличие данных в кэше, обработка в случае промаха (если не нашли, то обращаемся к памяти, записывая туда данные, передаваемые нам процессором (не забываем про обработку случая 'dirty’)).
* InvalidLine: ставим линии бит валидности 0, при этом если линия была ‘dirty’, записываем ее данные перед этим в память

Общение с модулем памяти происходит следующим образом: ставим на С2 нужную нам команду (чтение/запись), на А2 – необходимый нам адрес, и ставим значение 1 для соответствующего флага, означающего для кэша нашу работу с памятью (reading/writing) и переходим в режим ожидания, пока значение этой переменной не станет обратно равно 0 (что будет значить, что мы закончили работать с памятью). В это время активизируется другой always-блок, в нем мы ждем соответствующей нам конфигурации некоторых из переменных: reading/writing == 1, clk == 0 или 1, в зависимости от того, хотим ли мы считать данные из памяти или записать в нее, а также c2 == `C2\_RESPONSE для чтения (что будет нам говорить, что память передает нужные нам данные). Потратив восемь тактов на необходимые нам чтение/запись, возвращаем значение переменной reading/writing в ноль, выходя из этого always-блока.

Модуль memory

К памяти подключены шины, отвечающие за взаимодействие с кэшем. Сама память хранит данные (заполняется изначально в initial-блоке так, как указано в задании). При получении команды, она считывает адрес с шины a2 за два такта, а дальше за 8 тактов в зависимости от полученной команды либо считывает в себя данные с шины d2, либо записывает свои данные в шину d2. При этом учитываем, что, согласно условию, от первого такта команды до первого такта ответа должно пройти 100 тактов.

Модуль cpu

Основная задача просимулирована именно здесь.

К процессору подключены шины, отвечающие за взаимодействие с кэшем. Реализованы таски read\_data\_8, read\_data\_16, write\_data\_32, необходимые для симуляции задачи, а также task\_simulation, являющаяся самой симуляцией. Запускается все через initial-блок. Из него же получаем ответ

Сравнение полученных результатов

Из аналитического решения получили:

Total ticks: 5090975

Total accesses: 249600

Cache hits: 228080

Cache misses: 21520

Part of hits: 0.9137820512820513

Из решения на Verilog:

Total ticks: 5042495

Total memory accesses: 249600

Cache hits: 228080

Cache misses: 21520

Part of hits: 0.913782

Листинг кода

Аналитическое решение:

|  |
| --- |
| var tickCounter = 0  val cache\_line\_count = 64  val cache\_way = 2  val cache\_offset\_bits = 4 //log\_2(cache\_line\_size)  val cache\_set\_size = 32 //cache\_line\_count/cache\_way  val cache\_set\_bits = 5 //log\_2(cache\_set\_size)  val tag\_bits = 10  data class Cache\_Line(  var lru: Boolean,  var valid: Boolean,  var dirty: Boolean,  var tag: UInt  )  class Cache {  var cacheMissCounter = 0  var cacheHitCounter = 0    private val cache\_lines = List(cache\_line\_count) { Cache\_Line(lru = true, valid = false, dirty = false, tag = 0u) }  fun read\_data(address: Int, bytes: Int) {  val address = address.toUInt()  val offset = address.shl(32 - cache\_offset\_bits).shr(32 - cache\_offset\_bits) //last cache\_offset\_bits bits  val set = address.shr(cache\_offset\_bits).shl(32 - cache\_set\_bits).shr(32 - cache\_set\_bits) //last cache\_set\_bits bits  val tag = address.shr(cache\_offset\_bits + cache\_set\_bits).shl(32 - tag\_bits).shr(32 - tag\_bits) //last tag\_bits bits  val left = set.toInt() \* cache\_way  val right = (set.toInt()+1) \* cache\_way - 1  for (i in left..right) {  if (cache\_lines[i].tag == tag) {  cache\_lines[i].lru = false  cache\_lines[i xor 1].lru = true  if (cache\_lines[i].valid) {  // нашли в кэше  tickCounter += 6 //время, через которое в результате кэш попадания, кэш начинает отвечать  tickCounter += 1 //отправка данных по шине d1  cacheHitCounter += 1  return  }  }  }  cache\_miss(set, tag)  tickCounter += 1 //отправка данных по шине d1  }  fun write\_data(address: Int, bytes: Int) {  val address = address.toUInt()  val offset = address.shl(32 - cache\_offset\_bits).shr(32 - cache\_offset\_bits) //last cache\_offset\_bits bits  val set = address.shr(cache\_offset\_bits).shl(32 - cache\_set\_bits).shr(32 - cache\_set\_bits) //last cache\_set\_bits bits  val tag = address.shr(cache\_offset\_bits + cache\_set\_bits).shl(32 - tag\_bits).shr(32 - tag\_bits) //last tag\_bits bits  val left = set.toInt() \* cache\_way  val right = (set.toInt()+1) \* cache\_way - 1  for (i in left..right) {  if (cache\_lines[i].tag == tag) {  if (cache\_lines[i].valid) {  // нашли в кэше  tickCounter += 6 // время, через которое в результате кэш попадания, кэш начинает отвечать  cacheHitCounter += 1  cache\_lines[i].lru = false  cache\_lines[i xor 1].lru = true  cache\_lines[i].dirty = true  return  }  }  }  for (i in left..right) {  if (cache\_lines[i].lru) {  cache\_lines[i].dirty = true  }  }  cache\_miss(set, tag)  }  private fun cache\_miss(set: UInt, tag: UInt) {  tickCounter += 4 // время, через которое в результате кэш промаха, кэш посылает запрос к памяти.  tickCounter += 100 // MemCTR обработка  cacheMissCounter += 1  val left = set.toInt() \* cache\_way  val right = (set.toInt()+1) \* cache\_way - 1  for (i in left..right) {  if (cache\_lines[i].lru) {  cache\_lines[i].valid = true  if (cache\_lines[i].dirty) {  tickCounter += 100  }  cache\_lines[i].dirty = false  cache\_lines[i].tag = tag  cache\_lines[i].lru = false  cache\_lines[i xor 1].lru = true  return  }  }  }  }  fun main() {  val cache = Cache()  //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.  val M = 64  val N = 60  val K = 32  var pa = 0 //указатель на массив a  tickCounter += 1 //инициализация  val b = M \* K //адрес начала массива b  var pc = b + K \* N \* 2 //указатель на массив с  tickCounter += 1 //инициализация  repeat(M) {  repeat(N) { x ->  var pb = b //указатель на массив b  tickCounter += 1 //инициализация  tickCounter += 1 //инициализация переменной s  repeat(K) { k ->  cache.read\_data(pa + k, 8/8)  cache.read\_data(pb + x \* 2, 16/8)  tickCounter += 5 //умножение  tickCounter += 1 //сложение  pb += N \* 2  tickCounter += 1 //сложение  tickCounter += 1 //итерация цикла  }  cache.write\_data(pc + x \* 4, 32/8)  tickCounter += 1 //итерация цикла  }  pa += K  tickCounter += 1 //сложение  pc += N \* 4  tickCounter += 1 //сложение  tickCounter += 1 //итерация цикла  }  tickCounter += 1 //выход из функции  println("Total ticks: $tickCounter")  println("Total accesses: ${cache.cacheMissCounter + cache.cacheHitCounter}")  println("Cache hits: ${cache.cacheHitCounter}")  println("Cache misses: ${cache.cacheMissCounter}")  println("Part of hits: ${cache.cacheHitCounter.toDouble() / (cache.cacheMissCounter + cache.cacheHitCounter)}")  } |

Реализация на языке Verilog:

|  |
| --- |
| //delay фиксит возможную гонку при чтении значения clk, придуман Артемом Пешковым и Тимофеем Маловым  `define delay(TIME, CLOCK) \  for (int i = 0; i < TIME; i++) begin \  wait(clk == (i + !CLOCK) % 2); \  end  `define BYTE 8  `define C1\_NOP 0  `define C1\_READ8 1  `define C1\_READ16 2  `define C1\_READ32 3  `define C1\_INVALIDATE\_LINE 4  `define C1\_WRITE8 5  `define C1\_WRITE16 6  `define C1\_WRITE32\_OR\_RESPONSE 7  `define C2\_NOP 0  `define C2\_RESPONSE 1  `define C2\_READ\_LINE 2  `define C2\_WRITE\_LINE 3  `define VALID 1  `define DIRTY 1  `define CACHE\_WAY 2  `define CACHE\_TAG\_SIZE 10  `define SET\_SIZE 5  `define OFFSET\_SIZE 4  `define MEM\_SIZE 524288 //2^19  `define CACHE\_LINE\_SIZE 16  `define CACHE\_LINE\_COUNT 64  `define CACHE\_SETS\_COUNT (`CACHE\_LINE\_COUNT / `CACHE\_WAY)  `define CACHE\_LINE\_SIZE\_IN\_BITS (`CACHE\_LINE\_SIZE \* `BYTE)  `define WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS (`VALID + `DIRTY + `CACHE\_TAG\_SIZE + `CACHE\_LINE\_SIZE\_IN\_BITS)  `define MEMCTR\_RESPONSE\_TIME 100  `define CACHE\_HIT\_RESPONSE\_TIME 6  `define CACHE\_MIS\_RESPONSE\_TIME 4  `define MAX\_POSSIBLE\_SIZE\_OF\_REQUESTED\_DATA 32  `define SEND\_FROM\_MEM (`CACHE\_LINE\_SIZE/`CACHE\_WAY)  `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  module test;  reg clk = 0;  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;  reg c\_dump = 0;  reg m\_dump = 0;  reg reset = 0;  integer mdump\_file = 0;  integer cdump\_file = 0;  integer log\_file = 0;  cpu test\_cpu(clk, a1, d1, c1);  cache test\_cache(clk, a1, d1, c1, a2, d2, c2, c\_dump, reset);  memory test\_memory(clk, a2, d2, c2, m\_dump, reset);  initial begin  for (int i = 0; i < 14000000; i++) begin  #1;  clk = 1-clk;  end  end  endmodule  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 [`ADDR1\_BUS\_SIZE -1:0] inner\_a1 = 'z;  reg [`DATA1\_BUS\_SIZE - 1 : 0] inner\_d1 = 'z;  reg [`CTR1\_BUS\_SIZE - 1 : 0] inner\_c1 = 'z;  assign a1 = inner\_a1;  assign d1 = inner\_d1;  assign c1 = inner\_c1;  reg [7:0] result\_for\_reading\_8 = 'z;  bit reading\_8 = 0;  reg [15:0] result\_for\_reading\_16 = 'z;  bit reading\_16 = 0;  reg [31:0] result\_for\_writing\_32 = 'z;  bit writing\_32 = 0;  //начало симуляции задачи  int M = 64;  int N = 60;  int K = 32;  int pa = 0;  int b = M \* K;  int pb = 0;  int pc = b + K \* N \* 2;  int s = 0;  int additional\_tick\_counter = 0;  initial begin  test.log\_file = $fopen("log.txt", "w");  task\_simulation();  $display("Total ticks: %0t", $time/2 + additional\_tick\_counter);  $display("Total memory accesses: %0d", test.test\_cache.cacheMissCounter + test.test\_cache.cacheHitCounter);  $display("Cache hits: %0d", test.test\_cache.cacheHitCounter);  $display("Cache misses: %0d", test.test\_cache.cacheMissCounter);  $display("Part of hits: %0f", test.test\_cache.cacheHitCounter\*1.0 / (test.test\_cache.cacheMissCounter + test.test\_cache.cacheHitCounter));  test.reset = 1;  `delay(2,1);  $fclose(test.mdump\_file);  $fclose(test.cdump\_file);  $fclose(test.log\_file);  end  always @(negedge clk) begin  if (c1 == `C1\_WRITE32\_OR\_RESPONSE) begin  if (reading\_8 == 1) begin  $fdisplay(test.log\_file, "END OF READING 8, time = %0d", $time/2);  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "");  result\_for\_reading\_8 = d1[7:0];  reading\_8 = 0;  end else if (reading\_16 == 1) begin  $fdisplay(test.log\_file, "END OF READING 16, time = %0d", $time/2);  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "");  result\_for\_reading\_8 = d1[15:0];  reading\_16 = 0;  end else if (writing\_32 == 1) begin  $fdisplay(test.log\_file, "END OF WRITING 32, time = %0d", $time/2);  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "");  inner\_d1 = 'z;  writing\_32 = 0;  end  end  end  task read\_data\_8(reg[`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : 0] from);  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "START OF READING 8, time = %0d", $time/2);  wait(clk == 1);  inner\_c1 = `C1\_READ8;  inner\_a1 = from[`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : `OFFSET\_SIZE];  `delay(2,1);  inner\_c1 = 'z;  inner\_d1 = 'z;  inner\_a1 = from[`OFFSET\_SIZE-1:0];  reading\_8 = 1;  wait(reading\_8 == 0);  endtask  task read\_data\_16(reg [`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : 0] from);  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "START OF READING 16, time = %0d", $time/2);  wait(clk == 1);  inner\_a1 = from[`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : `OFFSET\_SIZE];  inner\_c1 = `C1\_READ16;    `delay(2,1);  inner\_a1 = from[`OFFSET\_SIZE-1:0];  inner\_c1 = 'z;  inner\_d1 = 'z;  reading\_16 = 1;  wait(reading\_16 == 0);  endtask  task write\_data\_32(reg [`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : 0] to, reg[31:0] data);  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "------------------------------------------");  $fdisplay(test.log\_file, "START OF WRITING 32, time = %0d", $time/2);  wait(clk == 1);  inner\_a1 = to[`ADDR1\_BUS\_SIZE + `OFFSET\_SIZE -1 : `OFFSET\_SIZE];  inner\_c1 = `C1\_WRITE32\_OR\_RESPONSE;  inner\_d1 = data[`DATA1\_BUS\_SIZE - 1:0];  `delay(2,1);  inner\_a1 = to[`OFFSET\_SIZE-1:0];  inner\_c1 = 'z;  inner\_d1 = data[`DATA1\_BUS\_SIZE\*2-1:`DATA1\_BUS\_SIZE];    writing\_32 = 1;  wait(writing\_32 == 0);  endtask  task task\_simulation;  $fdisplay(test.log\_file, "START SIMULATION, time = %0d", $time/2);  additional\_tick\_counter += 2; //инициализация pa, pc  for (int y = 0; y < M; y++) begin  for (int x = 0; x < N; x++) begin  pb = b;  s = 0;  additional\_tick\_counter += 2; //инициализация b, s  for (int k = 0; k < K; k++) begin  read\_data\_8(pa + k);  read\_data\_16(pb + 2\*x);  s += result\_for\_reading\_8 \* result\_for\_reading\_16;  additional\_tick\_counter += 5; //умножение  additional\_tick\_counter += 1; //сложение  pb += N \* 2;  additional\_tick\_counter += 1; //сложение  additional\_tick\_counter += 1; //итерация цикла  end  write\_data\_32(pc + x \* 4, s);    additional\_tick\_counter += 1; //итерация цикла  end  pa += K;  additional\_tick\_counter += 1; //сложение  pc += N \* 4;  additional\_tick\_counter += 1; //сложение  additional\_tick\_counter += 1; //итерация цикла  end  additional\_tick\_counter += 1; //выход из функции  $fdisplay(test.log\_file, "END SIMULATION, time = %0d", $time/2);  endtask  endmodule  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 c\_dump, input reset);  reg [`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - 1 : 0] cache\_data [`CACHE\_SETS\_COUNT - 1 : 0][`CACHE\_WAY-1:0];  bit lru [`CACHE\_SETS\_COUNT - 1: 0];  bit[`MAX\_POSSIBLE\_SIZE\_OF\_REQUESTED\_DATA-1:0] inner\_data;  integer hit = -1;  integer command = 0;  integer writing = 0;  integer reading = 0;  bit [`CACHE\_TAG\_SIZE -1:0] tag;  bit [`SET\_SIZE -1:0] set;  bit [`OFFSET\_SIZE-1:0] offset;  reg [`CTR2\_BUS\_SIZE - 1 : 0] inner\_c2 = 'z;  reg [`DATA2\_BUS\_SIZE - 1 : 0] inner\_d2 = 'z;  reg [`ADDR2\_BUS\_SIZE -1:0] inner\_a2 = 'z;  reg [`DATA1\_BUS\_SIZE - 1 : 0] inner\_d1 = 'z;  reg [`CTR1\_BUS\_SIZE - 1 : 0] inner\_c1 = 'z;  assign c2 = inner\_c2;  assign d2 = inner\_d2;  assign a2 = inner\_a2;  assign d1 = inner\_d1;  assign c1 = inner\_c1;  integer cacheHitCounter = 0;  integer cacheMissCounter = 0;  int i = 0;  initial begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "----------------------------------");  $fdisplay(test.log\_file, "INIT OF CACHE, t = %0d", $time/2);  for (int outer = 0; outer < `CACHE\_SETS\_COUNT; outer++) begin  for (int inner = 0; inner < `CACHE\_WAY; inner++) begin  cache\_data[outer][inner] = 0;  end  lru[outer] = 0;  end  $fdisplay(test.log\_file, "----------------------------------");  end  always @(clk) begin  if (clk == 0 && reading == 1 && c2==`C2\_RESPONSE) begin  for (int i = 0; i < `SEND\_FROM\_MEM; i++) begin  cache\_data[set][hit][`DATA2\_BUS\_SIZE \* i +: `DATA2\_BUS\_SIZE] = d2;  `delay(2,0);  end  reading = 0;  end else if (writing == 1 && clk == 1) begin  for (int i = 0; i < `SEND\_FROM\_MEM; i++) begin  inner\_d2 = cache\_data[set][hit][`DATA2\_BUS\_SIZE \* i +: `DATA2\_BUS\_SIZE];  `delay(2,1);    inner\_c2 = 'z;  inner\_a2 = 'z;  end  writing = 2;  end  end  always @(negedge clk) begin  if (writing == 2 && c2==`C2\_RESPONSE) begin  writing = 0;  end  end  always @(posedge c\_dump) begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "C\_DUMP, time = %0d", $time/2);  for (int i = 0; i < `CACHE\_SETS\_COUNT; i++) begin  $fdisplay(test.cdump\_file, "line %d: %d %d %d %d", 2\*i, cache\_data[i][0][127:96], cache\_data[i][0][95:64], cache\_data[i][0][63:32], cache\_data[i][0][31:0]);  $fdisplay(test.cdump\_file, "line %d: %d %d %d %d", 2\*i + 1, cache\_data[i][1][127:96], cache\_data[i][1][95:64], cache\_data[i][1][63:32], cache\_data[i][1][31:0]);  end  end  always @(posedge reset) begin  $fdisplay(test.log\_file, "RESET, time = %0d", $time/2);  for (int outer = 0; outer < `CACHE\_SETS\_COUNT; outer++) begin  for (int inner = 0; inner < `CACHE\_WAY; inner++) begin  cache\_data[outer][inner] = 0;  end  lru[outer] = 0;  end  end  always @(negedge clk) begin  case (c1)  `C1\_READ8, `C1\_READ16, `C1\_READ32: begin  command = c1;  set[`SET\_SIZE-1:0] = a1[`SET\_SIZE-1:0];  tag[`CACHE\_TAG\_SIZE-1:0] = a1[`ADDR1\_BUS\_SIZE-1:`SET\_SIZE];  `delay(2,0);  offset[`OFFSET\_SIZE-1:0] = a1[`OFFSET\_SIZE-1:0];  `delay(1,0);  hit = -1;  inner\_c1 = `C1\_NOP;  i = 0;  while (hit == -1 && i < `CACHE\_WAY) begin  if (cache\_data[set][i][`CACHE\_TAG\_SIZE + `CACHE\_LINE\_SIZE\_IN\_BITS -1:`CACHE\_LINE\_SIZE\_IN\_BITS] == tag) begin  if (cache\_data[set][i][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID]) begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "HIT CACHE, time = %0d", $time/2);  lru[set] = i;  cacheHitCounter++;  `delay((`CACHE\_HIT\_RESPONSE\_TIME - 2)\* 2, 1);  hit = i;  end  end  i++;  end  //miss  if (hit == -1) begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "CACHE MISS, time = %0d", $time/2);  cacheMissCounter++;  hit = 1 - lru[set];  `delay((`CACHE\_MIS\_RESPONSE\_TIME - 2)\* 2, 0);  if (cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID] && cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID - `DIRTY]) begin  inner\_a2[`SET\_SIZE - 1: 0] = set;  inner\_a2[`ADDR2\_BUS\_SIZE-1:`SET\_SIZE] = cache\_data[set][hit][`CACHE\_LINE\_SIZE\_IN\_BITS + `CACHE\_TAG\_SIZE - 1 : `CACHE\_LINE\_SIZE\_IN\_BITS];  inner\_c2 = `C2\_WRITE\_LINE;  writing = 1;  wait(writing == 0);  `delay(1,0);  end  inner\_a2[`SET\_SIZE - 1: 0] = set;  inner\_a2[`ADDR2\_BUS\_SIZE-1:`SET\_SIZE] = tag;  inner\_c2 = `C2\_READ\_LINE;    `delay(2,1);  inner\_c2 = 'z;  inner\_a2 = 'z;  inner\_d2 = 'z;  reading = 1;  wait(reading == 0);  cache\_data[set][hit][`CACHE\_LINE\_SIZE\_IN\_BITS + `CACHE\_TAG\_SIZE] = 0;  cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS-`VALID] = 1;  cache\_data[set][hit][`CACHE\_TAG\_SIZE+`CACHE\_LINE\_SIZE\_IN\_BITS-1:`CACHE\_LINE\_SIZE\_IN\_BITS] = tag;  lru[set] = hit;  `delay(1,0);  end  inner\_c1 = `C1\_WRITE32\_OR\_RESPONSE;  case (command)  `C1\_READ8: begin  inner\_d1[7:0] = cache\_data[set][hit][offset \* `BYTE +: 8];  end  `C1\_READ16: begin  inner\_d1[15:0] = cache\_data[set][hit][offset \* `BYTE +: 16];  end  `C1\_READ32: begin  inner\_d1[15:0] = cache\_data[set][hit][offset \* `BYTE +: 16];  `delay(2,1);  inner\_d1[15:0] = cache\_data[set][hit][16 + offset +: 16];  end  endcase  `delay(2,1);  inner\_c1 = 'z;  inner\_d1 = 'z;  end  `C1\_WRITE8, `C1\_WRITE16, `C1\_WRITE32\_OR\_RESPONSE: begin  command = c1;  set[`SET\_SIZE-1:0] = a1[`SET\_SIZE-1:0];  tag[`CACHE\_TAG\_SIZE-1:0] = a1[`ADDR1\_BUS\_SIZE-1:`SET\_SIZE];    inner\_data[15:0] = d1;  `delay(2,0);  offset[`OFFSET\_SIZE-1:0] = a1[`OFFSET\_SIZE-1:0];  if (command == `C1\_WRITE32\_OR\_RESPONSE) begin  inner\_data[31:16] = d1;  end    `delay(1,0);  hit = -1;  inner\_c1 = `C1\_NOP;  i = 0;  while (hit == -1 && i < `CACHE\_WAY) begin  if (cache\_data[set][i][`CACHE\_TAG\_SIZE + `CACHE\_LINE\_SIZE\_IN\_BITS -1:`CACHE\_LINE\_SIZE\_IN\_BITS] == tag) begin  if (cache\_data[set][i][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID]) begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "HIT CACHE, time = %0d", $time/2);  lru[set] = i;  cacheHitCounter++;  `delay((`CACHE\_HIT\_RESPONSE\_TIME-2) \* 2, 0);  hit = i;  end  end  i++;  end  //miss  if (hit == -1) begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "CACHE MISS, time = %0d", $time/2);  cacheMissCounter++;  hit = 1 - lru[set];  `delay((`CACHE\_MIS\_RESPONSE\_TIME - 2) \* 2, 0);  if (cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID] && cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID - `DIRTY]) begin  inner\_a2[`SET\_SIZE - 1: 0] = set;  inner\_a2[`ADDR2\_BUS\_SIZE-1:`SET\_SIZE] = cache\_data[set][hit][`CACHE\_LINE\_SIZE\_IN\_BITS + `CACHE\_TAG\_SIZE - 1 : `CACHE\_LINE\_SIZE\_IN\_BITS];  inner\_c2 = `C2\_WRITE\_LINE;    writing = 1;  wait(writing == 0);  `delay(1,0);  end  inner\_a2[`SET\_SIZE - 1: 0] = set;  inner\_a2[`ADDR2\_BUS\_SIZE-1:`SET\_SIZE] = tag;  inner\_c2 = `C2\_READ\_LINE;  `delay(2,1);    inner\_a2 = 'z;  inner\_c2 = 'z;  inner\_d2 = 'z;  reading = 1;  wait(reading == 0);  cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS-`VALID] = 1;  cache\_data[set][hit][`CACHE\_TAG\_SIZE+`CACHE\_LINE\_SIZE\_IN\_BITS-1:`CACHE\_LINE\_SIZE\_IN\_BITS] = tag;  lru[set] = hit;  `delay(1,0);  end  cache\_data[set][hit][`CACHE\_TAG\_SIZE + `CACHE\_LINE\_SIZE\_IN\_BITS] = 0;  if (command == `C1\_WRITE8) begin  cache\_data[set][hit][offset \* 8 +: 8] = inner\_data[7:0];  end else if (command == `C1\_WRITE16) begin  cache\_data[set][hit][offset \* 8 +: 16] = inner\_data[15:0];  end else if (command == `C1\_WRITE32\_OR\_RESPONSE) begin  cache\_data[set][hit][offset \* 8 +: 32] = inner\_data[31:0];  end  inner\_c1 = `C1\_WRITE32\_OR\_RESPONSE;  `delay(2,1);  inner\_c1 = 'z;  inner\_d1 = 'z;  end  `C1\_INVALIDATE\_LINE: begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "INVELIDATE LINE, time = %0d", $time/2);    command = inner\_c1;  set[`SET\_SIZE-1:0] = a1[`SET\_SIZE-1:0];  tag[`CACHE\_TAG\_SIZE-1:0] = a1[`ADDR1\_BUS\_SIZE-1:`SET\_SIZE];    `delay(2,0);    offset[`OFFSET\_SIZE-1:0] = a1[`OFFSET\_SIZE-1:0];  `delay(1,0);  hit = -1;  inner\_c1 = `C1\_NOP;  for (int i = 0; i < `CACHE\_WAY && (hit==-1); i++) begin  if (cache\_data[set][i][`CACHE\_TAG\_SIZE + `CACHE\_LINE\_SIZE\_IN\_BITS -1:`CACHE\_LINE\_SIZE\_IN\_BITS] == tag) begin  if (cache\_data[set][i][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID]) begin  lru[set] = i;  cacheHitCounter++;  `delay(`CACHE\_HIT\_RESPONSE\_TIME \* 2 - 4, 0);  hit = i;  if (cache\_data[set][hit][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID - `DIRTY]) begin  inner\_a2[`SET\_SIZE - 1: 0] = set;  inner\_a2[`ADDR2\_BUS\_SIZE-1:`SET\_SIZE] = tag;  inner\_c2 = `C2\_WRITE\_LINE;  writing = 1;  wait(writing == 0);  `delay(1,0);  end  cache\_data[set][hit][`CACHE\_LINE\_SIZE\_IN\_BITS + `CACHE\_TAG\_SIZE] = 0;  cache\_data[set][i][`WHOLE\_CACHE\_LINE\_SIZE\_IN\_BITS - `VALID] = 0;  end  end  end  end  endcase  end  endmodule    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 m\_dump, input reset);  integer SEED = \_SEED;  logic [`BYTE -1:0] memory\_data[`MEM\_SIZE -1:0];  reg [`DATA2\_BUS\_SIZE - 1 : 0] inner\_d2 = 'z;  reg [`CTR2\_BUS\_SIZE - 1 : 0] inner\_c2 = 'z;  assign c2 = inner\_c2;  assign d2 = inner\_d2;  bit [`ADDR2\_BUS\_SIZE-1:0] inner\_a = 'z;  initial begin  $fdisplay(test.log\_file, "----------------------------------");  $fdisplay(test.log\_file, "INIT MEMORY, t = %0d", $time/2);  for (int i = 0; i < (1 << `MEM\_SIZE); i++) begin  memory\_data[i] = $random(SEED)>>16;  end  end  int i;  //прописать еще reset, dump  always @(posedge m\_dump) begin  $fdisplay(test.log\_file, "MDUMP, time = %0d", $time/2);  for (i = 0; i < `MEM\_SIZE; i++) begin  $fdisplay(test.mdump\_file, "line %d: %d", i, memory\_data[i]);  end  end  always @(posedge reset) begin  $fdisplay(test.log\_file, "RESET MEMORY, time = %0d", $time/2);  SEED = \_SEED;  for (i = 0; i < `MEM\_SIZE; i++) begin  memory\_data[i] = $random(SEED)>>16;  end  end  always @(negedge clk) begin  case (c2)  `C2\_READ\_LINE: begin  $fdisplay(test.log\_file, "", $time/2);  $fdisplay(test.log\_file, "READ FROM MEM, time = %0d", $time/2);  inner\_a = a2;  `delay(1,0);    inner\_c2 = `C2\_NOP;  `delay(`MEMCTR\_RESPONSE\_TIME\*2, 1);  inner\_c2 = `C2\_RESPONSE;  for (int i = 0; i < `SEND\_FROM\_MEM; i++) begin  inner\_d2[`BYTE-1:0] = memory\_data[inner\_a \* (1<<`OFFSET\_SIZE) + 2 \* i];  inner\_d2[`DATA2\_BUS\_SIZE-1:`BYTE] = memory\_data[inner\_a \* (1<<`OFFSET\_SIZE) + 2 \* i + 1];  `delay(2,1);  end  inner\_c2 = 'z;  inner\_d2 = 'z;  inner\_a = 'z;  end  `C2\_WRITE\_LINE: begin  $fdisplay(test.log\_file, "");  $fdisplay(test.log\_file, "WRITE TO MEM, time = %0d", $time/2);  inner\_a = a2;  for (int i = 0; i < `SEND\_FROM\_MEM; i++) begin  memory\_data[inner\_a \* (1<<`OFFSET\_SIZE) + 2 \* i] = inner\_d2[`BYTE-1:0];  memory\_data[inner\_a \* (1<<`OFFSET\_SIZE) + 2 \* i + 1] = inner\_d2[`DATA2\_BUS\_SIZE-1:`BYTE];  `delay(2,0);  end  `delay(`MEMCTR\_RESPONSE\_TIME\*2 - `SEND\_FROM\_MEM\*2,1);  inner\_c2 = `C2\_RESPONSE;  `delay(2,1);  inner\_c2 = 'z;  inner\_a = 'z;  end  endcase  end  endmodule |