| Лабораторная работа №2 | B10 | 2022 |
| --- | --- | --- |
| Моделирование схем в Verilog | Малов Тимофей Иванович | |
|

**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](https://docs.google.com/document/u/0/d/179Da-G5IvQp12NpAxpFa0MLwzdFsBTVp/edit)). 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**

Имеется следующее определение глобальных переменных и функций:

| #define M 64  #define N 60  #define K 32  int8 a[M][K];  int16 b[K][N];  int32 c[M][N];  void mmul()  {  int8 \*pa = a;  int32 \*pc = c;  for (int y = 0; y < M; y++)  {  for (int x = 0; x < N; x++)  {  int16 \*pb = b;  int32 s = 0;  for (int k = 0; k < K; k++)  {  s += pa[k] \* pb[x];  pb += N;  }  pc[x] = s;  }  pa += K;  pc += N;  }  } |
| --- |

Сложение, инициализация переменных и переход на новую итерацию цикла, выход из функции занимают 1 такт. Умножение – 5 тактов. Обращение к памяти вида pc[x] считается за одну команду.

Массивы последовательно хранятся в памяти, и первый из них начинается с 0.

Все локальные переменные лежат в регистрах процессора.

По моделируемой шине происходит только обмен данными (не командами).

Определите процент попаданий (число попаданий к общему числу обращений) для кэша и общее время (в тактах), затраченное на выполнение этой функции.

# Идея устройства памяти и вычисление недостающих параметров системы

Нам требуется реализовать look-through two-way-associative кэш - это значит, что все кэш линии объединены в блоки ассоциативности размером в 2 кэш линии. Введём удобные обозначения: все биты оперативной памяти последовательно объединены в байты по 8 бит. Байты последовательно объединены в *линии памяти* - блоки, по размеру равные размеру кэш-линии. Линии памяти последовательно объединены в *сеты* - блоки, по размеру равные числу кэш линий делённому на ассоциативность - идея этих блоков в том, что внутри блока каждую очередную линию памяти мы ассоциируем с очередным блоком ассоциативности в кэше. Линия памяти может потенциально хранится в кэше только в тех двух кэш линиях, которые принадлежат ассоциированному ей блоку ассоциативности. Причём, как можно заметить, любые две соответствующие линии памяти из двух разных сетов ассоциированы с одним и тем же блоком ассоциативности. Значит можно ввести такой способ адресации: мы хотим обратится к байту с индексом j в памяти, тогда пусть **tag** = j / (размер сета \* размер линии) **=** j / (число кэш линий / ассоциативность \* размер линии) - номер сета, в котором он лежит, **set =** (j / размер линии) % размер сета - номер линии внутри сета, в которой он лежит, **offset** = j % 16 - номер интересующего нас байта внутри линии. Посчитаем из этих наблюдений и из данных в условии значения всех констант:

* **MEM\_SIZE** = 2^19 байт
* **CACHE\_LINE\_SIZE** = 16 байт
* **CACHE\_LINE\_COUNT** = 64 (В коде называется CACHE\_LINE\_NUMBER)
* **CACHE\_SIZE** = **CACHE\_LINE\_COUNT \* CACHE\_LINE\_SIZE** = 1024 байта = 1 килобайт
* **CACHE\_WAY** = 2
* **CACHE\_SETS\_COUNT** = **CACHE\_LINE\_COUNT / CACHE\_WAY** = 32 (В коде называется CACHE\_WAY\_BLOCKS\_NUMBER)
* **CACHE\_TAG\_SIZE** = log2(**MEM\_SIZE / (CACHE\_SETS\_COUNT \* CACHE\_LINE\_SIZE** ) = 10 бит
* **CACHE\_SET\_SIZE** = log2(**CACHE\_SETS\_COUNT** ) = 5 бит
* **CACHE\_OFFSET\_SIZE** = log2(**CACHE\_LINE\_SIZE**) = 4 бит
* **ADDR1\_BUS\_SIZE** = max(**CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE, CACHE\_OFFSET\_SIZE** **)** = 15 бит
* **ADDR2\_BUS\_SIZE = CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE** = 15 бит
* **CTR1\_BUS\_SIZE** = 3
* **CTR2\_BUS\_SIZE =** 2

# Аналитическое решение

Для практической реализации мне понадобится поддерживать инвариант на синхронизацию: менять данные на выходах из схем я могу только на выключенной синхронизации, а читать данные на входах - только на включенной. Такое соглашение позволяет избежать гонок и чтения “грязных” данных. Также важно, что два такта моего практического, а следственно и теоретического, решения равны одному такту из условия, так как задержка полного цикла синхронизации(0 -> 1 -> 0) равна 2 такта. Я пишу это здесь, так как я учитывал это в написании теоретического решения.

Я написал код на языке Kotlin, который моделирует взаимодействие между процессором, кэшем и памятью на самом базовом уровне - для кэша он помнит только для каждой линии тэг set’а, из которого в нём прямо сейчас лежит линия памяти (lineTag), изменена ли она по сравнению с памятью (modified), а также для каждого блока ассоциативности последнюю использованную линию (lastUsed). В main реализована задача, в нужные моменты моделируя запросы в кэш (processOperationOnAddress), считая при этом число запросов и число *моих* тактов от запуска программы. При каждом запросе в начале моделируется поведение процессора: он ожидает отрицательную синхронизацию, после чего отправляет запрос в кэш. Кэш проверяет, хранится ли линия памяти с заданным тегом сейчас у него, или нет. Если да, то это считается *попаданием* и кэш возвращает результат через 12 *моих* тактов(или через 6 тактов условия). Иначе это считается *промахом*, и он ждёт 8 *моих* тактов, после чего проверяет, верно ли, что в кэш линии, которую необходимо заменить, лежат изменённые данные. Если да, то кэш отправляет запрос на запись в память, ответ на который приходит ровно через 200 тактов(далее слово *такты* будет означать *мои такты*, в конце вычислений их достаточно будет поделить на 2), кэш же видит его на один такт позже, так как ждёт нужную синхронизацию, а также ждёт ещё один такт, чтобы забрать владение шиной. После этого, независимо от того, нужна ли была запись, кэш запрашивает из памяти интересующую его линию, память начинает ответ через 200 тактов, а также сам ответ длится 16 тактов, так как размер линии в 8 раз больше пропускной способности шины, после чего кэш дослушивает ответ, обновляет данные о последней использованной линии и её изменённости относительно памяти, и ждёт один такт, чтобы перенять шину c2 и начать отвечать. Процессор слушает ответ кэша всего за 1 такт, так как он отправляет только запросы на чтение 8 и 16 бит и на запись 32, ответ на которые влезает в шину c1, после чего ждёт ещё 1, чтобы перенять владение c1 - одно полное взаимодействие с памятью завершено. Итого, алгоритм выдаёт такие результаты:

Cache requests: 249600

Cache hits: 228080. Percent of hits: 91

Total ticks: 5346917

# Моделирование заданной системы на Verilog

В моделировании на верилоге я придерживался двух принципов, описанных в теоретической части, реализуя смену синхронизации отдельным модулем, который просто меняет синхронизацию каждый такт. Также я старался как можно сильнее избегать параллельного исполнения внутри одного модуля, дабы избежать трудностей. Опишу работу каждого модуля по-отдельности.

### Процессор:

Он написан почти идентично теоретическому решению - всё выполняется “высокоуровнево”, за исключением task’ов read8, read16, read32, write32 - в них реализована низкоуровневая логика общения с кэшем. Так как они практически эквивалентны, то опишу работу только одной из них, например, read8: для начала процессор ожидает нулевую синхронизацию, чтобы писать на шины, на которой он на шину c1 выставляет код запроса (в случае read8 это 1), а на шину адреса объединённые tag и set, а на следующей синхронизации offset. Ещё через 2 такта он отдаёт владение шинами кэшу и ждёт ответ. Когда ответ пришёл, процессор его обрабатывает и ждёт такт, чтобы перенять владение.

### Память:

Она представлена массивом ram, в котором содержится 2^19 байт. В обычном состоянии память ничего не делает и ожидает запроса от кэша. На синхронизации 1 она слушает шину c2, и, как только на неё приходит запрос, она, в зависимости от его типа, считывает необходимые входные данные (для чтения это просто адрес за одну итерацию “прослушивания” шины, а для записи это данные линии за восемь итераций по два такта). Так как наша реализация призвана лишь моделировать, но не в точности реализовывать память, то запрос исполняется в момент считывания, его тип запоминается в специальную переменную requested\_operation, но результат на него выдаётся с требуемой задержкой. На 0 синхронизации же она проверяет направлен ли к ней в данный момент какой-то запрос, если это так, то она ждёт столько тактов, чтобы начать отвечать ровно через 200 тактов после начала запроса (ждать нужно по-разному, так как время передачи запросов тоже разное), после чего отвечает, при необходимости отправляя запрошенные на чтение данные за 16 тактов. После этого, с задержкой в 2 такта, чтобы кэш увидел ответ, она отдаёт владение шиной c2, запоминая, что теперь нет никаких операций, на которые нужно дать ответ.

### Кэш:

Внутреннее устройство кэша представлено в виде двумерного массива cache\_memory, в котором хранятся байты, объединённые в кэш линии, которые объединены в блоки ассоциативности. Также в отдельных массивах хранятся биты invalid и modified (dirty) для каждой линии. И для каждого блока ассоциативности хранится индекс последней использованной кэш линии - last\_used.

Кэш, так же как и память в состоянии покоя не делает ничего более, чем слушает на синхронизации 1 шину c1, ожидая команду процессора. Когда она приходит, то он читает и запоминает все входные параметры (требуемую операцию - requested\_operation, адрес - requested\_tag, requested\_set, requested\_offset, и, если нужно, данные для записи - data\_to\_write). На синхронизации 0 он начинает выполнять запрос: для начала проверяя хранится ли в нём сейчас валидная (то есть та, invalid для которой 0) линия данных с требуемым tag’ом. Если нет, то он должен заменить самую давно использованную линию в соответствующем блоке ассоциативности, но если она не invalid и modified относительно памяти, то нужно вначале её записать в память, после чего, независимо от записи, запросить на её место необходимую сейчас линию памяти. За запросы к памяти отвечают task’и write\_to\_memory() и read\_from\_memory(), о них отдельно позже. Выполнив такие запросы мы точно знаем с какой линией уже внутри кэша нужно выполнить запрос - ind\_in\_way\_block. Выполняем запрос: для чтения кладём на шину данных требуемые байты, для записи меняем требуемые байты на данные, после чего, через 2 такта, отдаём владение шинами процессору. task’и, о которых упоминалось выше, выполняются очень похоже на соответствующие task’и общения с кэшем в процессоре: выставляем на шины c2 и d2 необходимые данные, после чего ожидаем ответа памяти “вешая” на это время работу кэша, как только запрос приходит, обрабатываем и через 2 такта перенимаем владение шинами.

Команда “invalidate” в кэше обрабатывается отдельно, отлично от всех остальных: мы запоминаем её при “прослушивании” c1, как и раньше, а на выключенной синхронизации выставляем invalid требуемой линии на 1, записывая её в память, если она была modified относительно памяти.

Команда “dump” в кэше и памяти реализован одинаково: он происходит в момент, когда на этом соответствующем входе появляется сигнал, независимо от синхронизации - это просто полный вывод памяти в соответствующий файл.

Команда “reset” в кэше инвалидирует все кэш линии, в то время как в памяти она обновляет все значения во внутреннем массиве по данному для этого в условии алгоритму.

Мой код во время исполнения пишет много полезной отладочной информации в файл “log.txt”, при запросах на “dump” кэша или памяти пишет их в соответствующие файлы “\*\_dump.txt”, результат же данного в задаче кода (то есть подсчитанная матрица C) выводится в файл “result.txt”.

Для запуска всех схемы используется модуль “test\_simulaton”, который просто подключает все шины всех устройств между собой, открывает на запись нужные файлы и запускает clock.

# Сравнение полученных результатов

В результате исполнения, симуляция на Verilog выдала такие же результаты, что и теоретическое решение, то есть:  
 Cache requests: 249600

Cache hits: 228080

Total ticks: 5346917

# Листинг кода

### Теоретическое решение на языке Kotlin.

Файл “main.kt”:

| const val *M* = 64  const val *N* = 60  const val *K* = 32  var *tickCounter* = 0  var *cacheMisses* = 0  var *cacheHits* = 0  object MemorySimulator {  private val lineTag = *List*(32) **{** *mutableListOf*(-1, -1) **}**  private val modified = *List*(32) **{** *mutableListOf*(false, false) **}**  private val lastUsed = *MutableList*(32) **{** 0 **}**  fun processOperationOnAddress(tag: Int, set: Int, offset: Int, write: Boolean) {  if (*tickCounter* % 2 == 1) *//cpu waits for 0 clock and sends request to the cache*  *tickCounter*++  *//cache work:*  if (tag in lineTag[set]) {  *cacheHits* += 1 *//cache hit*  *tickCounter* += 12 *//time until cache answers*  } else {  *cacheMisses* += 1 *//cache miss*  *tickCounter* += 8 *//cache sends request to memory*  if (modified[set][1 - lastUsed[set]]) {  *//cache sends to memory write request*  *//memory work:*  *tickCounter* += 200 *//time until memory answers*  *//memory answered*  *tickCounter* += 1 *//cache waits for 1 clock and listens to answer*  *tickCounter* += 1 *//cache waits for 0 clock, takes the bus, then continues working*  }  *//memory work:*  *tickCounter* += 200 *//time unlit memory answers*  *tickCounter* += 16 *//memory answering*  *//memory answered*  *tickCounter* += 1 *//cache listens to answer*  *tickCounter* += 1 *//wait until clock is 0 to cache answer*  modified[set][1 - lastUsed[set]] = false  lineTag[set][1 - lastUsed[set]] = tag  }  if (write)  modified[set][lineTag[set].indexOf(tag)] = true  lastUsed[set] = lineTag[set].indexOf(tag)  *//cache answered*  *tickCounter* += 1 *//cpu wait for 1 clock and listens the answer*  *tickCounter* += 1 *//cpu wait for 0 clock, takes the bus, then continues working*  }  }  fun readA(row: Int, col: Int) {  val j = row \* *K* + col  MemorySimulator.processOperationOnAddress(j / (16 \* 32), (j / 16) % 32, j % 16, false)  }  fun readB(row: Int, col: Int) {  val j = *M* \* *K* + (row \* *N* + col) \* 2  MemorySimulator.processOperationOnAddress(j / (16 \* 32), (j / 16) % 32, j % 16, false)  }  fun writeC(row: Int, col: Int) {  val j = *M* \* *K* + *K* \* *N* \* 2 + (row \* *N* + col) \* 4  MemorySimulator.processOperationOnAddress(j / (16 \* 32), (j / 16) % 32, j % 16, true)  }  fun main() {  *tickCounter* += 2 *//pa init*  *tickCounter* += 2 *//pc init*  for (y in 0 *until M*) {  *tickCounter* += 2 *//y init and ++ later*  for (x in 0 *until N*) {  *tickCounter* += 2 *//x init and ++ later*  *tickCounter* += 2 *//pb init*  *tickCounter* += 2 *//s init*  for (k in 0 *until K*) {  *tickCounter* += 2 *//k init and ++ later*  *readA*(y, k)  *readB*(k, x)  *tickCounter* += 12 *// += and \**  *tickCounter* += 2 *//pb +=*  *tickCounter* += 2 *//k new iter*  }  *writeC*(y, x)  *tickCounter* += 2 *//x new iter*  }  *tickCounter* += 2 *//pa +=*  *tickCounter* += 2 *//pc +=*  *tickCounter* += 2 *//y new iter*  }  *tickCounter* += 2 *//fun exit*  *println*("Cache requests: ${*cacheHits* + *cacheMisses*}")  *println*("Cache hits: $*cacheHits*. Percent of hits: ${*cacheHits* \* 100 / (*cacheHits* + *cacheMisses*)}")  *print*("Total ticks: ${*tickCounter* / 2}")  } |
| --- |

### Моделирование на SystemVerilog версии “Icarus Verilog version 12.0 (devel) (s20150603-1539-g2693dd32b)”.

Файл “testbench.sv”:

| `include "constants.sv"  `include "memCTR.sv"  `include "cache.sv"  `include "cpu.sv"  module test\_simulation(  output wire clk, reset,  output reg c\_dump, m\_dump,  output wire[`ADDR1\_BUS\_SIZE - 1 : 0] a1,  output wire[`DATA1\_BUS\_SIZE - 1 : 0] d1,  output wire[`CTR1\_BUS\_SIZE - 1 : 0] c1,  output wire[`ADDR2\_BUS\_SIZE - 1 : 0] a2,  output wire[`DATA2\_BUS\_BIT\_SIZE - 1 : 0] d2,  output wire[`CTR2\_BUS\_SIZE - 1 : 0] c2  );  int log\_stream = $fopen("output\_files/log.txt", "w");  int mem\_stream = $fopen("output\_files/memory\_dump.txt", "w");  int cache\_stream = $fopen("output\_files/cache\_dump.txt", "w");  clock c(clk);  cpu cpu\_inst(log\_stream, mem\_stream, cache\_stream, clk, a1, d1, c1);  cache cache\_inst(log\_stream, cache\_stream, clk, reset, c\_dump, a1, d1, c1, a2, d2, c2);  memCTR mem\_inst(log\_stream, mem\_stream, clk, reset, m\_dump, a2, d2, c2);  endmodule;  module clock(output reg clk);  always begin  clk = 0;  #1;  clk = 1;  #1;  end  endmodule |
| --- |

Файл “memCTR.sv”:

| `include "constants.sv"  module memCTR #(parameter \_SEED = 225526) (  input int log\_stream,  input int mem\_stream,  input wire clk, reset, m\_dump,  input wire[`ADDR2\_BUS\_SIZE - 1 : 0] a2\_in,  inout wire[`DATA2\_BUS\_BIT\_SIZE - 1 : 0] d2\_in,  inout wire[`CTR2\_BUS\_SIZE - 1 : 0] c2\_in  );  integer SEED = \_SEED;  reg[`DATA2\_BUS\_BIT\_SIZE - 1 : 0] d2\_reg = 'z;  reg[`CTR2\_BUS\_SIZE - 1 : 0] c2\_reg = 'z;  assign d2\_in = d2\_reg;  assign c2\_in = c2\_reg;  reg[`BYTE - 1 : 0] ram[`MEM\_BYTE\_SIZE - 1 : 0];  task reset\_ram;  for (int i = 0; i < `MEM\_BYTE\_SIZE; i++)  ram[i] = $random(SEED) >> 16;  $fdisplay(log\_stream, "Memory reset");  endtask  task dump\_memory;  $fdisplay(mem\_stream,"Memory dump. t: %0t", $time);  for (int i = 0; i < 50000; i++)  $fdisplay(mem\_stream, "[tag = %0d, set = %0d, offset = %0d] %0d", i / (16 \* 32), (i / 16) % 32, i % 16, ram[i]);  $fdisplay(mem\_stream, "-------------------------------------------------------");  // for (int i = 0; i < `MEM\_BYTE\_SIZE; i++)  // $fdisplay(mem\_stream, "tag: %0d, set: %0d, offset: %0d, data: %0d", i / (16 \* 32), (i / 16) % 32, i % 16, ram[i]);  endtask  reg[`CTR2\_BUS\_SIZE - 1 : 0] requested\_operation = NONE;  reg[`ADDR2\_BUS\_SIZE - 1 : 0] data\_from = 0;  initial begin  reset\_ram();  end  always @(posedge reset)  reset\_ram();  always @(posedge m\_dump)  $dumpvars(0, memCTR);  always @(clk) begin  if (m\_dump)  $dumpvars(0, memCTR);  if (reset)  reset\_ram();  if (clk == 1 && c2\_in != NONE) begin  requested\_operation = c2\_in;  case (c2\_in)  READ: begin  $fdisplay(log\_stream, "Time: %0t, memory read at adress: %b", $time, a2\_in);  data\_from = a2\_in;  end  WRITE: begin  $fdisplay(log\_stream, "Time: %0t, memory write at adress: %b", $time, a2\_in);  for (int i = 0; i < `MEMORY\_INTERACTION\_TICK\_NUMBER; i++) begin  if (i > 0)  `delay(`CLOCK\_DELAY, 1)  for (int j = 0; j < `DATA2\_BUS\_BYTE\_SIZE; j++)  ram[a2\_in \* `LINE\_BYTE\_SIZE + i \* `DATA2\_BUS\_BYTE\_SIZE + j] = (d2\_in >> (j \* `BYTE)) & ((1 << `BYTE) - 1);  end  end  endcase  end    if (clk == 0 && requested\_operation != NONE) begin  c2\_reg = NONE;  $fdisplay(log\_stream, "Time: %0t, memory has taken the bus and started processing request", $time);  if (requested\_operation == READ)  `delay(`MEMORY\_RESPONSE\_TIME - `CLOCK\_DELAY, 0)  else  `delay(`MEMORY\_RESPONSE\_TIME - `MEMORY\_INTERACTION\_TICK\_NUMBER \* `CLOCK\_DELAY, 0)  c2\_reg = RESPONSE;  $fdisplay(log\_stream, "Time: %0t, memory start answering", $time);  if (requested\_operation == READ) begin  for (int i = 0; i < `MEMORY\_INTERACTION\_TICK\_NUMBER; i++) begin  if (i > 0)  `delay(`CLOCK\_DELAY, 0)  d2\_reg = 0;  for (int j = 0; j < `DATA2\_BUS\_BYTE\_SIZE; j++)  d2\_reg[j \* `BYTE +: `BYTE] = ram[data\_from \* `LINE\_BYTE\_SIZE + i \* `DATA2\_BUS\_BYTE\_SIZE + j];  end  end  `delay(`CLOCK\_DELAY, 0)  c2\_reg = 'z;  d2\_reg = 'z;  $fdisplay(log\_stream, "Time: %0t, memory has given the bus", $time);  requested\_operation = NONE;  end  end  endmodule |
| --- |

Файл “cpu.sv”:

| `include "constants.sv"  module cpu #(parameter M = 64, parameter N = 60, parameter K = 32) (  input int log\_stream,  input int mem\_stream,  input int cache\_stream,  input wire clk,  output wire[`ADDR1\_BUS\_SIZE - 1 : 0] a1\_bus,  inout wire[`DATA1\_BUS\_SIZE - 1 : 0] d1\_in,  inout wire[`CTR1\_BUS\_SIZE - 1 : 0] c1\_in  );  reg[`ADDR1\_BUS\_SIZE -1 : 0] a1\_reg = 'z;  reg[`DATA1\_BUS\_SIZE - 1 : 0] d1\_reg = 'z;  reg[`CTR1\_BUS\_SIZE - 1 : 0] c1\_reg = 'z;  assign a1\_bus = a1\_reg;  assign d1\_in = d1\_reg;  assign c1\_in = c1\_reg;    int cache\_hits;  int cache\_misses;  task read8(  input [`TAG\_SIZE - 1 : 0] tag,  input [`SET\_SIZE - 1 : 0] set,  input [`OFFSET\_SIZE - 1: 0] offset,  output reg[7 : 0] res  );  wait(clk == 0);  //clock = 0  c1\_reg = READ8;  a1\_reg[`SET\_SIZE +: `TAG\_SIZE] = tag;  a1\_reg[0 +: `SET\_SIZE] = set;  `delay(`CLOCK\_DELAY, 0)  a1\_reg = offset;  `delay(`CLOCK\_DELAY, 0)  c1\_reg = 'z;  d1\_reg = 'z;  a1\_reg = 'z;  $fdisplay(log\_stream, "Time: %0t, cpu has given c1: %b", $time, c1\_in);  wait(c1\_in == WRITE32\_OR\_RESPONSE && clk == 1);  //Clock = 1  res = d1\_in;  $fdisplay(log\_stream, "Time: %0t, cpu listening d1: %b, c1: %b", $time, d1\_in, c1\_in);  $fdisplay(log\_stream, "Time: %0t, cash answer on read c1: %b, res: %0d", $time, c1\_in, res);  wait(clk == 0);  c1\_reg = NONE\_C1;  endtask  task read16(  input [`TAG\_SIZE - 1 : 0] tag,  input [`SET\_SIZE - 1 : 0] set,  input [`OFFSET\_SIZE - 1: 0] offset,  output reg[15 : 0] res  );  wait(clk == 0);  //clock = 0  c1\_reg = READ16;  a1\_reg[`SET\_SIZE +: `TAG\_SIZE] = tag;  a1\_reg[0 +: `SET\_SIZE] = set;  `delay(`CLOCK\_DELAY, 0)  a1\_reg = offset;  `delay(`CLOCK\_DELAY, 0)  c1\_reg = 'z;  d1\_reg = 'z;  a1\_reg = 'z;  $fdisplay(log\_stream, "Time: %0t, cpu has given c1: %b", $time, c1\_in);  wait(c1\_in == WRITE32\_OR\_RESPONSE && clk == 1);  //Clock = 1  res = d1\_in;  $fdisplay(log\_stream, "Time: %0t, cpu listening d1: %b, c1: %b", $time, d1\_in, c1\_in);  $fdisplay(log\_stream, "Time: %0t, cash answer on read c1: %b, res: %0d", $time, c1\_in, res);  wait(clk == 0);  c1\_reg = NONE\_C1;  endtask  task read32(  input [`TAG\_SIZE - 1 : 0] tag,  input [`SET\_SIZE - 1 : 0] set,  input [`OFFSET\_SIZE - 1: 0] offset,  output reg[31 : 0] res  );  wait(clk == 0);  //clock = 0  c1\_reg = READ32;  a1\_reg[`SET\_SIZE +: `TAG\_SIZE] = tag;  a1\_reg[0 +: `SET\_SIZE] = set;  `delay(`CLOCK\_DELAY, 0)  a1\_reg = offset;  `delay(`CLOCK\_DELAY, 0)  c1\_reg = 'z;  d1\_reg = 'z;  a1\_reg = 'z;  $fdisplay(log\_stream, "Time: %0t, cpu has given c1: %b", $time, c1\_in);  wait(c1\_in == WRITE32\_OR\_RESPONSE && clk == 1);  //Clock = 1  res[15 : 0] = d1\_in;  $fdisplay(log\_stream, "Time: %0t, cpu listening d1: %b, c1: %b", $time, d1\_in, c1\_in);  `delay(`CLOCK\_DELAY, 1)  res[31 : 16] = d1\_in;  $fdisplay(log\_stream, "Time: %0t, cpu listening d1: %b, c1: %b", $time, d1\_in, c1\_in);  $fdisplay(log\_stream, "Time: %0t, cash answer on read c1: %b, res: %0d", $time, c1\_in, res);  wait(clk == 0);  c1\_reg = NONE\_C1;  endtask  task write32(  input [`TAG\_SIZE - 1 : 0] tag,  input [`SET\_SIZE - 1 : 0] set,  input [`OFFSET\_SIZE - 1: 0] offset,  input reg[31 : 0] info  );  wait(clk == 0);  //Clock = 0  c1\_reg = WRITE32\_OR\_RESPONSE;  a1\_reg[`SET\_SIZE +: `TAG\_SIZE] = tag;  a1\_reg[0 +: `SET\_SIZE] = set;  d1\_reg = info & ((1 << `DATA1\_BUS\_SIZE) - 1);  `delay(`CLOCK\_DELAY, 0)  a1\_reg = offset;  d1\_reg = (info >> `DATA1\_BUS\_SIZE) & ((1 << `DATA1\_BUS\_SIZE) - 1);  `delay(`CLOCK\_DELAY, 0)    //Clock = 0  c1\_reg = 'z;  d1\_reg = 'z;  wait(c1\_in == WRITE32\_OR\_RESPONSE && clk == 1);  //Clock = 1  $fdisplay(log\_stream, "Time: %0t, cash answer on write c1:%b", $time, c1\_in);  wait(clk == 0);  //Clock = 0  c1\_reg = NONE\_C1;  endtask  int j;    reg[7 : 0] a;  task readA(input int row, input int col);  j = row \* K + col;  read8(j / ((1 << `OFFSET\_SIZE) \* (1 << `SET\_SIZE)), (j / (1 << `OFFSET\_SIZE)) % (1 << `SET\_SIZE), j % (1 << `OFFSET\_SIZE), a);  endtask  reg[15 : 0] b;  task readB(input int row, input int col);  j = M \* K + (row \* N + col) \* 2;  read16(j / ((1 << `OFFSET\_SIZE) \* (1 << `SET\_SIZE)), (j / (1 << `OFFSET\_SIZE)) % (1 << `SET\_SIZE), j % (1 << `OFFSET\_SIZE), b);  endtask  reg[31 : 0] c;  task writeC(input int row, input int col);  j = M \* K + K \* N \* 2 + (row \* N + col) \* 4;  write32(j / ((1 << `OFFSET\_SIZE) \* (1 << `SET\_SIZE)), (j / (1 << `OFFSET\_SIZE)) % (1 << `SET\_SIZE), j % (1 << `OFFSET\_SIZE), c);  endtask  task readC(input int row, input int col);  j = M \* K + K \* N \* 2 + (row \* N + col) \* 4;  read32(j / ((1 << `OFFSET\_SIZE) \* (1 << `SET\_SIZE)), (j / (1 << `OFFSET\_SIZE)) % (1 << `SET\_SIZE), j % (1 << `OFFSET\_SIZE), c);  endtask  int s;  int result\_stream;  initial begin  `delay(`CLOCK\_DELAY, 0) //pa init  `delay(`CLOCK\_DELAY, 0) //pc init  for (int y = 0; y < M; y++) begin  `delay(`CLOCK\_DELAY, 0) //y init and ++ later  for (int x = 0; x < N; x++) begin  `delay(`CLOCK\_DELAY, 0) //x init and ++ later  `delay(`CLOCK\_DELAY, 0) //pb init  c = 0;  `delay(`CLOCK\_DELAY, 0) //s init  for (int k = 0; k < K; k++) begin  `delay(`CLOCK\_DELAY, 0) //k init and ++ later  readA(y, k);  readB(k, x);  c += a \* b;  `delay(`CLOCK\_DELAY \* (5 + 1), 0) // += and \*  `delay(`CLOCK\_DELAY, 0) //pb +=    `delay(`CLOCK\_DELAY, 0) //k new iter  end  writeC(y, x);    `delay(`CLOCK\_DELAY, 0) //x new iter  end  `delay(`CLOCK\_DELAY, 0) //pa +=  `delay(`CLOCK\_DELAY, 0) //pc +=  `delay(`CLOCK\_DELAY, 0) //y new iter  end  `delay(`CLOCK\_DELAY, 0) //fun exit  $display("Cache requests: %0d", cache\_misses + cache\_hits);  $display("Cache hits: %0d", cache\_hits);  $display("Total ticks: %0t", $time / `CLOCK\_DELAY);    result\_stream = $fopen("output\_files/result.txt", "w");  for (int y = 0; y < M; y++) begin  for (int x = 0; x < N; x++) begin  readC(y, x);  $fdisplay(result\_stream, "%0d", c);  end  end  `delay(`CLOCK\_DELAY, 0)  $fclose(result\_stream);  $fclose(log\_stream);  $fclose(mem\_stream);  $fclose(cache\_stream);  $finish;  end  endmodule; |
| --- |

Файл “cache.sv”:

| `include "constants.sv"  module cache(  input int log\_stream,  input int cache\_stream,  input wire clk, reset, c\_dump,  input wire[`ADDR1\_BUS\_SIZE - 1 : 0] a1\_in,  inout wire[`DATA1\_BUS\_SIZE - 1 : 0] d1\_in,  inout wire[`CTR1\_BUS\_SIZE - 1 : 0] c1\_in,  output wire[`ADDR2\_BUS\_SIZE - 1 : 0] a2\_out,  inout wire[`DATA2\_BUS\_BIT\_SIZE - 1 : 0] d2\_in,  inout wire[`CTR2\_BUS\_SIZE - 1 : 0] c2\_in  );  reg[`DATA1\_BUS\_SIZE - 1 : 0] d1\_reg = 'z;  reg[`CTR1\_BUS\_SIZE - 1 : 0] c1\_reg = 'z;  reg[`ADDR2\_BUS\_SIZE - 1 : 0] a2\_reg = 'z;  reg[`DATA2\_BUS\_BIT\_SIZE - 1 : 0] d2\_reg = 'z;  reg[`CTR2\_BUS\_SIZE - 1 : 0] c2\_reg = NONE;  assign d1\_in = d1\_reg;  assign c1\_in = c1\_reg;  assign d2\_in = d2\_reg;  assign a2\_out = a2\_reg;  assign c2\_in = c2\_reg;  reg[`LINE\_BIT\_SIZE - 1 : 0] cache\_memory[`CACHE\_WAY\_BLOCKS\_NUMBER - 1 : 0][`CACHE\_WAY - 1 : 0];  reg[`TAG\_SIZE - 1 : 0] line\_tag[`CACHE\_WAY\_BLOCKS\_NUMBER - 1 : 0][`CACHE\_WAY - 1 : 0];  reg modified[`CACHE\_WAY\_BLOCKS\_NUMBER - 1 : 0][`CACHE\_WAY - 1 : 0];  reg invalid[`CACHE\_WAY\_BLOCKS\_NUMBER - 1 : 0][`CACHE\_WAY - 1 : 0];  reg last\_used[`CACHE\_WAY\_BLOCKS\_NUMBER - 1 : 0];  reg[`TAG\_SIZE - 1 : 0] requested\_tag;  reg[`SET\_SIZE - 1 : 0] requested\_set;  reg[`OFFSET\_SIZE - 1 : 0] requested\_offset;  reg[`CTR1\_BUS\_SIZE - 1 : 0] requested\_operation = NONE;  reg[`MAX\_WRITE\_REQUEST\_SIZE - 1 : 0] data\_to\_write;  task write\_to\_memory(  input reg[`SET\_SIZE - 1 : 0] set,  input int ind\_in\_way\_block  );  c2\_reg = WRITE;  a2\_reg[`SET\_SIZE +: `TAG\_SIZE] = line\_tag[set][ind\_in\_way\_block];  a2\_reg[0 +: `SET\_SIZE] = set;  $fdisplay(log\_stream, "Time: %0t, cache send write request to memory on a2: %b", $time, a2\_out);  for (int i = 0; i < `MEMORY\_INTERACTION\_TICK\_NUMBER; i++) begin  d2\_reg = (cache\_memory[set][ind\_in\_way\_block] >> i \* `DATA2\_BUS\_BIT\_SIZE) & ((1 << `DATA2\_BUS\_BIT\_SIZE) - 1);  `delay(`CLOCK\_DELAY, 0)  end  modified[set][ind\_in\_way\_block] = 0;  c2\_reg = 'z;  wait(clk == 1 && c2\_in == RESPONSE);  //Clock = 1  $fdisplay(log\_stream, "Time: %0t, memory answered to cache on write c2: %b", $time, c2\_in);  wait(clk == 0);  //Clock = 0    c2\_reg = NONE;  $fdisplay(log\_stream, "Time: %0t, cache has taken the c2: %b", $time, c2\_in);  endtask;  task read\_from\_memory(  input reg[`TAG\_SIZE - 1 : 0] tag,  input reg[`SET\_SIZE - 1 : 0] set,  input int ind\_in\_way\_block  );  c2\_reg = READ;  a2\_reg[`SET\_SIZE +: `TAG\_SIZE] = tag;  a2\_reg[0 +: `SET\_SIZE] = set;  $fdisplay(log\_stream, "Time: %0t, cache send read request to memory, a2: %b", $time, a2\_out);  `delay(`CLOCK\_DELAY, 0)  c2\_reg = 'z;  d2\_reg = 'z;  wait(clk == 1 && c2\_in == RESPONSE);  //Clock = 1  cache\_memory[set][ind\_in\_way\_block] = 0;  for (int i = 0; i < `MEMORY\_INTERACTION\_TICK\_NUMBER; i++) begin  cache\_memory[set][ind\_in\_way\_block][i \* `DATA2\_BUS\_BIT\_SIZE +: `DATA2\_BUS\_BIT\_SIZE] = d2\_in;  `delay(`CLOCK\_DELAY, 1)  end;  invalid[set][ind\_in\_way\_block] = 0;  modified[set][ind\_in\_way\_block] = 0;  line\_tag[set][ind\_in\_way\_block] = tag;  $fdisplay(log\_stream, "Time: %0t, memory answered to cache on read with data: %b", $time, cache\_memory[set][ind\_in\_way\_block]);  wait(clk == 0);  //Clock = 0    c2\_reg = NONE;  $fdisplay(log\_stream, "Time: %0t, cache has taken the c2: %b", $time, c2\_in);  endtask;  task reset\_cache;  $fdisplay(log\_stream, "Cache reset");  for (int i = 0; i < `CACHE\_WAY\_BLOCKS\_NUMBER; i++) begin  for (int j = 0; j < `CACHE\_WAY; j++) begin  cache\_memory[i][j] = 0;  invalid[i][j] = 1;  modified[i][j] = 0;  end;  last\_used[i] = 0;  end  endtask  task dump\_cache;  for (int i = 0; i < `CACHE\_WAY\_BLOCKS\_NUMBER; i++) begin  for (int j = 0; j < `CACHE\_WAY; j++) begin  $fdisplay(cache\_stream, "tag: %b, set: %b, ind\_in\_way\_block: %b, cache\_line: %b", line\_tag[i][j], i, j, cache\_memory[i][j]);  end  end  endtask  initial reset\_cache();  int ind\_in\_way\_block = -1;  int requested\_bit\_offset;  always @(posedge reset)  reset\_cache();  always @(posedge c\_dump)  dump\_cache();  always @(clk) begin  if (clk == 1 && c1\_in != NONE\_C1) begin  requested\_operation = c1\_in;  requested\_set = a1\_in & ((1 << `SET\_SIZE) - 1);  requested\_tag = (a1\_in >> `SET\_SIZE);  if (c1\_in == WRITE8)  data\_to\_write = d1\_in & ((1 << 8) - 1);  else if (c1\_in == WRITE16 || c1\_in == WRITE32\_OR\_RESPONSE)  data\_to\_write = d1\_in;  if (c1\_in != INVALIDATE) begin  `delay(`CLOCK\_DELAY, 1)  requested\_offset = a1\_in & ((1 << `OFFSET\_SIZE) - 1);  if (c1\_in == WRITE32\_OR\_RESPONSE)  data\_to\_write[`DATA1\_BUS\_SIZE +: `DATA1\_BUS\_SIZE] = d1\_in;  end    $fdisplay(log\_stream, "Time: %0t, cache got the request: %0d, tag: %b, set: %b, offset: %b", $time, requested\_operation, requested\_tag, requested\_set, requested\_offset);  if (c1\_in == WRITE8 || c1\_in == WRITE16 || c1\_in == WRITE32\_OR\_RESPONSE)  $fdisplay(log\_stream, "with data to write: %b", data\_to\_write);  end    if (clk == 0 && requested\_operation != NONE) begin  c1\_reg = NONE\_C1;  $fdisplay(log\_stream, "Time: %0t, cache has taken the c1 bus and started processing request", $time);  if (requested\_operation == INVALIDATE) begin  for (int j = 0; j < `CACHE\_WAY; j++) begin  if (invalid[requested\_set][j] == 0 && line\_tag[requested\_set][j] == requested\_tag) begin  last\_used[requested\_set] = !j;  if (modified[requested\_set][j])  write\_to\_memory(requested\_set, j);  invalid[requested\_set][j] = 1;  end  end  c1\_reg = WRITE32\_OR\_RESPONSE;  end else begin  ind\_in\_way\_block = -1;  for (int j = 0; j < `CACHE\_WAY; j++)  if (invalid[requested\_set][j] == 0 && line\_tag[requested\_set][j] == requested\_tag)  ind\_in\_way\_block = j;    if (ind\_in\_way\_block == -1) begin  $fdisplay(log\_stream, "Time: %0t, cache miss at set: %b at tag: %b", $time, requested\_set, requested\_tag);  test\_simulation.cpu\_inst.cache\_misses++;  `delay(`CACHE\_MISS\_RESPONSE\_TIME - `CACHE\_INTERACTION\_TICK\_NUMBER \* `CLOCK\_DELAY, 0)  ind\_in\_way\_block = !last\_used[requested\_set];    if (modified[requested\_set][ind\_in\_way\_block] == 1 && invalid[requested\_set][ind\_in\_way\_block] == 0)  write\_to\_memory(requested\_set, ind\_in\_way\_block);  read\_from\_memory(requested\_tag, requested\_set, ind\_in\_way\_block);  end else begin  $fdisplay(log\_stream, "Time: %0t, cache hit at set: %b at tag: %b", $time, requested\_set, requested\_tag);  test\_simulation.cpu\_inst.cache\_hits++;  `delay(`CACHE\_HIT\_RESPONSE\_TIME - `CACHE\_INTERACTION\_TICK\_NUMBER \* `CLOCK\_DELAY, 0)  end  c1\_reg = WRITE32\_OR\_RESPONSE;  requested\_bit\_offset = requested\_offset \* 8; //bytes to bits    case (requested\_operation)  WRITE8: begin  cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 8] = data\_to\_write;  modified[requested\_set][ind\_in\_way\_block] = 1;  end  WRITE16: begin  cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 16] = data\_to\_write;  modified[requested\_set][ind\_in\_way\_block] = 1;  end  WRITE32\_OR\_RESPONSE: begin  cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 32] = data\_to\_write;  modified[requested\_set][ind\_in\_way\_block] = 1;  end  READ8: d1\_reg = cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 8];  READ16: d1\_reg = cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 16];  READ32: begin  $fdisplay(log\_stream, "Time: %0t, cache\_memory: %b", $time, cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 16]);  d1\_reg = cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset +: 16];  `delay(`CLOCK\_DELAY, 0);  $fdisplay(log\_stream, "Time: %0t, cache\_memory: %b", $time, cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset + 16 +: 16]);  d1\_reg = cache\_memory[requested\_set][ind\_in\_way\_block][requested\_bit\_offset + 16 +: 16];  end  endcase  last\_used[requested\_set] = ind\_in\_way\_block;  end  `delay(`CLOCK\_DELAY, 0)  $fdisplay(log\_stream, "Time: %0t, cache has given the c1 bus", $time);  c1\_reg = 'z;  d1\_reg = 'z;  requested\_operation = NONE;  end  end;  endmodule |
| --- |

Файл “constants.sv”:

| `ifndef GUARD  `define GUARD  `define delay(TIME, CLOCK) \  for (int i = 0; i < TIME; i++) begin \  wait(clk == (i + !CLOCK) % 2); \  end  `define BYTE 8  `define LINE\_BYTE\_SIZE 16  `define LINE\_BIT\_SIZE 16 \* `BYTE  `define MEM\_BYTE\_SIZE (1 << 19)  `define CACHE\_LINE\_NUMBER 64  `define CACHE\_WAY 2  `define CACHE\_WAY\_BLOCKS\_NUMBER (`CACHE\_LINE\_NUMBER / `CACHE\_WAY)  `define CTR1\_BUS\_SIZE 3  `define CTR2\_BUS\_SIZE 2  typedef enum int { NONE\_C1 = 0, READ8 = 1, READ16 = 2, READ32 = 3, INVALIDATE = 4, WRITE8 = 5, WRITE16 = 6, WRITE32\_OR\_RESPONSE = 7 } C1\_state;  typedef enum int { NONE = 0, RESPONSE = 1, READ = 2, WRITE = 3 } C2\_state;  `define DATA1\_BUS\_SIZE 16  `define DATA2\_BUS\_BIT\_SIZE 16  `define DATA2\_BUS\_BYTE\_SIZE (`DATA2\_BUS\_BIT\_SIZE / `BYTE)  `define ADDR1\_BUS\_SIZE 15  `define ADDR2\_BUS\_SIZE 15  `define TAG\_SIZE 10  `define SET\_SIZE 5  `define OFFSET\_SIZE 4  `define MAX\_WRITE\_REQUEST\_SIZE 32  `define CLOCK\_DELAY 2  `define MEMORY\_RESPONSE\_TIME (`CLOCK\_DELAY \* 100)  `define CACHE\_HIT\_RESPONSE\_TIME (`CLOCK\_DELAY \* 6)  `define CACHE\_MISS\_RESPONSE\_TIME (`CLOCK\_DELAY \* 4)  `define MEMORY\_INTERACTION\_TICK\_NUMBER (`LINE\_BYTE\_SIZE / `DATA2\_BUS\_BYTE\_SIZE)  `define CACHE\_INTERACTION\_TICK\_NUMBER 2  `endif //GUARD |
| --- |