|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2 | M3136 | 2022 |
| Моделирование схем в Verilog | Панюхин Никита Константинович | |
|

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

**Инструментарий и требования к работе:**используемый язык: SystemVerilog, компиляция и симуляция: Icarus Verilog 12.

**Описание:** аналитически решить данную в условии задачу (найти процент кэш-попадания и длительность выполнения программы в тактах); построить и отладить модуль кэша на языке Verilog, после чего запустить на нём поставленную задачу и сравнить полученные результаты с аналитическим решением.

**Вариант**: 1

# Вычисление недостающих параметров системы

Параметры из условия для варианта №1:

* Ассоциативность: **CACHE\_WAY = 2**
* Размер тэга адреса: **CACHE\_TAG\_SIZE = 10 бит**
* Размер кэш-линии (полезных данных): **CACHE\_LINE\_SIZE = 16 байт**
* Кол-во кэш-линий: **CACHE\_LINE\_COUNT = 64**
* Размер памяти: **MEM\_SIZE = 512 Кбайт**

Вычислим остальные параметры:

* Размер кэша:

**CACHE\_SIZE = CACHE\_LINE\_SIZE × CACHE\_LINE\_COUNT = 64 × 16 байт = 1024 байт**

* Кол-во наборов кэш-линий:

**CACHE\_SETS\_COUNT = CACHE\_LINE\_COUNT / CACHE\_WAY = 64 / 2 = 32**

* Размер индекса в наборе кэш-линий:

**CACHE\_SET\_SIZE = = = 5 бит**

* Размер смещения**:**

**CACHE\_OFFSET\_SIZE = = = 4 бит**

* Размер адреса**:**

**CACHE\_ADDR\_SIZE = = = 19 бит**

Размер частей адреса кэша:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **tag** | **set** | **offset** |
| Размер | **CACHE\_TAG\_SIZE** | **CACHE\_SET\_SIZE** | **CACHE\_OFFSET\_SIZE** |
| **10 бит** | **5 бит** | **4 бита** |
| **19 бит** | | |

Таблица №1 – Интерпретация адреса кэшем

Отсюда получим размеры шин A1 и A2:

* **ADDR1\_BUS\_SIZE = =  
  = = 15 бит**
* **ADDR2\_BUS\_SIZE = CACHE\_TAG\_SIZE + CACHE\_SET\_SIZE = 15 бит**

Размеры шин C1 и C2, очевидно, равны логарифму от максимального значения сигнала на них, то есть и соответственно.

Все параметры системы указаны в файле [src/parameters.sv](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/parameters.sv)

# Аналитическое решение задачи

Аналитическое решение было выполнено с помощью кода на языке Python (находится в файле [“Analytical solution/solution.py”](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/Analytical%20solution/solution.py)).

Поскольку в задаче требуется лишь посчитать количество кэш-попаданий и суммарное число тактов работы программы, можно смоделировать лишь часть процессов, достаточную для получения ответа. Так, например, можно опустить все операции с памятью и значениями переменных, поскольку, записанные в память (Mem) не влияют на оба требуемых ответа. Также, поскольку в задаче нет вызовов операции read32, упрощается операция read\_bytes – байты данных передаются одновременно в такт с командой C1\_RESPONSE, и считать дополнительный такт не надо[[➜]](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/Analytical%20solution/solution.py#L92).

В остальном, программа на питоне полностью повторяет программу на Verilog, смысл большинства задержек подписан в коде. В некоторых местах требуется подождать изменения значения синхронизации (например, команда отправляется по спаду CLK и принимается по фронту CLK, см. соотв. соглашение), для этого используется функция Cache.wait\_clk

В результате получаем следующие ответы:

|  |
| --- |
| Total time: 4785493.0 tacts  Cache hits: 228080/249600 = 91.38% |

# Моделирование заданной системы на Verilog

*Поскольку условие лабораторной допускает множество вариаций и отклонений при решении, были использованы некоторые соглашения. Далее они будут помечаться значком « ».*

**Идеология кода без числовых констант:**

Изначально планировалось написать код так, чтобы он работал при любом варианте условия, то есть с использованием синтаксиса констант, нежели напрямую вставкой числовых значений в код. Однако позже пришлось отказаться от некоторых из констант, из-за чего итоговый код напрямую зависит от ассоциативности (CACHE\_WAY = 2) и размера шин D1 и D2 (DATA\_BUS\_SIZE = 16 бит). Все остальные параметры, теоретически, изменяемы без нарушения работоспособности программы.

**Составные части (модули):**

Было решено совместить CPU с testbench. Кэш и модуль памяти (MemCTR вместе с хранящимися данными Mem) хранятся в отдельных файлах [src/cache.sv](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/cache.sv) и [src/mem.sv](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/mem.sv) соответственно. Остальные файлы содержат константы и общие функции, которые используются во всей программе.

**Общение модулей по шине:**

Для удобства обе шины были разделены на 3 части (команда, адрес и данные), каждая из которых состоит из проводов и имеет заданную ширину. Все провода шин подключаются к модулям на вход и выход (inout) для простоты. Владение шиной устроено следующий образом: модуль, который владеет шиной, устанавливает в регистр (reg), соединённый с проводом (assign), значение. На противоположном конце другой модуль читает данные с провода. При этом модуль, который в данный момент не владеет шиной, устанавливает на ней высокоимпедансное состояние ('z). Пример для A1:

|  |
| --- |
| **testbench.sv**  wire[ADDR1\_BUS\_SIZE-1:0] A1\_WIRE;  reg[ADDR1\_BUS\_SIZE-1:0] A1 = 'z; assign A1\_WIRE = A1;  A1 = 100; // Отправка значения 100 по шине A1  **cache.sv**  module Cache(  inout[ADDR1\_BUS\_SIZE-1:0] A1\_WIRE  );  reg[ADDR1\_BUS\_SIZE-1:0] A1 = 'z; assign A1\_WIRE = A1;  req\_tag = A1\_WIRE >> CACHE\_SET\_SIZE; // Чтение данных с шины |

**Тайминги и синхронизация:**

* Данные на шине меняются при CLK = 0 (по спаду, negedge), а принимаются на другой стороне при CLK = 1 (по фронту, posedge)

Следуя этому соглашению, модуль ставит значение в регистр шины при CLK = 0 и читает данные с провода при CLK = 1. Для ожидания ответа используются конструкции wait(CLK == 1 && C2\_WIRE == C2\_RESPONSE), которые ждут не только изменения команды на шине, но и правильного такта синхронизации, после чего продолжают работу.

Для каждой задачи (task) в коде подписано время входа и выхода (если её время работы ненулевое), таком образом в каждый момент времени легко понять, какое сейчас значение CLK.

В результате, написанная система, при передаче сигнала RESPONSE от MemCTR к кэшу, а затем сразу к CPU (например, при операции INVALIDATE\_LINE или WRITE), тратит один лишний такт, который можно избежать, усложнив логику владения проводами и “перенаправляя” сигнал C2\_REPONSE сразу в C1\_RESPONSE на том же такте, на котором он был послан. В данный момент в коде присутствует задержка в 1 такт между этими командами.

**Передача данных по шинам D1 и D2 – кодирование в little-endian:**

* Данные в памяти и кэш линиях хранятся в формате big-endian

Это соглашение было принято из-за удобства понимания человеком хранящихся значений при дебаге. По условию, данные должны отправляться в little-endian, поэтому их нужно кодировать. Как было сказано ранее, размер шин данных принимается константой равной 16 бит или 2 байта. Кодирование двух байт в формате little-endian происходит следующим образом: байты просто меняются местами. Первые 8 бит шины данных содержат значение второго байта, а вторые 8 бит – первого. Здесь также полагается, что “начало” (лево) у регистра a размера N находится в a[N-1], а “конец” (право) в a[0], что соответствует способу побитовой печати регистра. В коде это выглядит следующим образом:

|  |
| --- |
| task send\_bytes\_D1(input [7:0] byte1, input [7:0] byte2);  D1[15:8] = byte2; D1[7:0] = byte1;  endtask  task receive\_bytes\_D1(output [7:0] byte1, output [7:0] byte2);  byte2 = D1\_WIRE[15:8]; byte1 = D1\_WIRE[7:0];  endtask |

**Устройство кэша – политика замещения (LRU):**

Следуя политике Least Recently Used, для вытеснения выбирается линия, которая дольше всего не использовалась. Поскольку мы принимаем параметр ассоциативности константой, равной двум, достаточно хранить лишь 1 бит (LRU\_bit) для каждой линии – в се́те всего две линии и та, у которой этот бит равен 0, не использовалась дольше другой.

При чтении и записи по заданному адресу определяется линия, над которой происходит операция. В конце операции этой линии устанавливается LRU\_bit равный 1, а соседней линии в сете – бит 0.

**Отладка:**

Модель была отлажена на четырёх тестовых случаях (INVALIDATE\_LINE при найденной линии и не найденной, READ32 в обоих и WRITE32 также в обоих случаях). Логи тестов можно посмотреть в папке tests.

Для отладки используется глобальная переменная DEBUG\_MODE доступная в файле [src/common.sv](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/common.sv)[[➜]](https://github.com/skkv-itmo/itmo-comp-arch22-lab2-npanuhin/blob/main/src/common.sv#L22), а также связка конструкции `log и функции $display. Для показа текущего такта для человека при дебаге используется равенство CLK = $time % 2 из-за особенностей запуска функции always на параметрах с приставкой posedge/negedge – если выводить просто CLK, то значение будет немного отличаться.

*Помимо описанного в отчёте, в коде присутсвуют комментарии, которые поясняют технические аспекты реализации*

# Воспроизведение задачи на Verilog

Для упрощения кода задержка в 1 такт использовалась после каждой итерации цикла, то есть на 1 раз больше, чем если бы она использовалась между итерациями (как задано в условии). Это допущение использовалось также в аналитическом решении и не влияет на сравнение ответов.

В целом моделирование задачи не представляло никакой сложности, так как использовало уже отлаженные вызовы команд READ и WRITE. В результате были получены следующий ответы:

|  |
| --- |
| Total time: 4785493 tacts  Cache hits: 228080/249600 = 91.38% |

# Сравнение полученных результатов

Ответ аналитического решения на Python и решения с помощью модели кэша на Verilog сошёлся. Учитывая, что моделируемая задача состоит в перемножении двух матриц, высокий процент кэш-попаданий вполне реалистичен.

Стоит заметить, что аналитическое решение было написано после основного, вследствие чего уже были известны точные значения задержек, которые нужно выставить в коде. Аналитическое решение проще в понимании, меньше в объёме, но всё ещё показывает, как изнутри устроена система.

# Листинг кода

**testbench.sv**

`include "src/common.sv"

`include "src/parameters.sv"

`include "src/commands.sv"

`include "src/statistics.sv"

`include "src/cache.sv"

`include "src/mem.sv"

// `define assert(signal, value) \

// if (signal !== value) begin \

// $display("ASSERTION FAILED in %m: signal != value"); \

// $finish; \

// end

module cache\_test;

reg CLK = 0,

RESET = 0,

C\_DUMP = 0,

M\_DUMP = 0;

always #1 CLK = ~CLK;

wire[ADDR1\_BUS\_SIZE-1:0] A1\_WIRE;

wire[ADDR2\_BUS\_SIZE-1:0] A2\_WIRE;

wire[DATA\_BUS\_SIZE-1:0] D1\_WIRE;

wire[DATA\_BUS\_SIZE-1:0] D2\_WIRE;

wire[CTR1\_BUS\_SIZE-1:0] C1\_WIRE;

wire[CTR2\_BUS\_SIZE-1:0] C2\_WIRE;

`map\_bus1; // Initialize wires

Cache Cache\_instance(CLK, A1\_WIRE, D1\_WIRE, C1\_WIRE, A2\_WIRE, D2\_WIRE, C2\_WIRE, RESET, C\_DUMP);

MemCTR Mem\_instance(CLK, A2\_WIRE, D2\_WIRE, C2\_WIRE, RESET, M\_DUMP);

task wait\_response;

#2 `close\_bus1;

wait(CLK == 1 && C1\_WIRE == C1\_RESPONSE);

endtask

// For testing

reg[CACHE\_TAG\_SIZE-1:0] tag;

reg[CACHE\_SET\_SIZE-1:0] set;

reg[CACHE\_OFFSET\_SIZE-1:0] offset;

reg[CACHE\_ADDR\_SIZE-1:0] address;

task send\_bytes\_D1(input [7:0] byte1, input [7:0] byte2);

`log $display("CPU: Sending byte: %d = %b", byte1, byte1);

`log $display("CPU: Sending byte: %d = %b", byte2, byte2);

D1[15:8] = byte2; D1[7:0] = byte1;

endtask

task receive\_bytes\_D1;

`log $display("CPU: Received byte: %d = %b", D1\_WIRE[7:0], D1\_WIRE[7:0]);

`log $display("CPU: Received byte: %d = %b", D1\_WIRE[15:8], D1\_WIRE[15:8]);

endtask

// initial begin

// $dumpfile("dump.vcd"); $dumpvars;

// -------------------------------------------- Test C1\_INVALIDATE\_LINE --------------------------------------------

// tag = 1; // 0 — found, 1 — not found

// set = 2;

// offset = 3;

// address = tag;

// address = (((address << CACHE\_SET\_SIZE) + set) << CACHE\_OFFSET\_SIZE) + offset;

// $display("Testbench: sending C1\_INVALIDATE\_LINE, A1 = %b|%b|%b\n", tag, set, offset);

// // Передача команды и первой части адреса

// `log $display("<Sending C1 and first half of A1>");

// C1 = C1\_INVALIDATE\_LINE;

// A1 = `discard\_last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// #2;

// // Передача второй части адреса

// `log $display("<Sending second half of A1>");

// A1 = `last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// // Завершение взаимодействия

// wait\_response();

// `log $display("CPU received C1\_RESPONSE");

// ---------------------------------------------- Test C1\_READ8/16/32 ----------------------------------------------

// tag = 1;

// set = 2;

// offset = 3;

// address = tag;

// address = (((address << CACHE\_SET\_SIZE) + set) << CACHE\_OFFSET\_SIZE) + offset;

// $display("Testbench: sending C1\_READ32, A1 = %b|%b|%b\n", tag, set, offset);

// // Прочитаем один и те же данные два раза - во второй раз не должно быть похода в память

// for (int iteration = 0; iteration < 2; ++iteration) begin

// // Передача команды и первой части адреса

// `log $display("<Sending C1 and first half of A1>");

// C1 = C1\_READ32;

// A1 = `discard\_last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// #2

// // Передача второй части адреса

// `log $display("<Sending second half of A1>");

// A1 = `last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// // Завершение взаимодействия

// wait\_response();

// `log $display("CPU received C1\_RESPONSE");

// for (int bbytes\_start = 0; bbytes\_start < 32 / 8; bbytes\_start += 2) begin

// receive\_bytes\_D1();

// if (bbytes\_start + 2 < CACHE\_LINE\_SIZE) #2; // Ждать надо везде, кроме последней передачи данных

// end

// $display("\n---------- Iteration %0d finished ----------\n", iteration);

// #3;

// end

// ---------------------------------------------- Test C1\_WRITE8/16/32 ----------------------------------------------

// tag = 1;

// set = 2;

// offset = 3;

// address = tag;

// address = (((address << CACHE\_SET\_SIZE) + set) << CACHE\_OFFSET\_SIZE) + offset;

// $display("Testbench: sending C1\_WRITE32, A1 = %b|%b|%b\n", tag, set, offset);

// // Прочитаем один и те же данные два раза - во второй раз не должно быть похода в память

// for (int iteration = 0; iteration < 2; ++iteration) begin

// // Передача команды, первой части адреса и первой части данных

// `log $display("<Sending C1 and first half of A1>");

// C1 = C1\_WRITE32;

// A1 = `discard\_last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// D1[15:8] = 200; D1[7:0] = 124;

// #2

// // Передача второй части адреса и второй части данных

// `log $display("<Sending second half of A1>");

// A1 = `last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

// D1[15:8] = 37; D1[7:0] = 5;

// // Завершение взаимодействия

// wait\_response();

// `log $display("CPU received C1\_RESPONSE");

// $display("\n---------- Iteration %0d finished ----------\n", iteration);

// #3;

// end

// -----------------------------------------------------------------------------------------------------------------

// DUMP everything and finish

// #3;

// C\_DUMP = 1;

// M\_DUMP = 1;

// #3 $finish;

// end

// --------------------------------------------------- Actual task ---------------------------------------------------

// ---------- READ8/16/32 ----------

task common\_read(input int address, input int command);

// `log $display("CPU sending READ command");

C1 = command;

A1 = `discard\_last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

#2 A1 = `last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

wait\_response();

endtask

task read8(input int address, output [7:0] result);

common\_read(address, C1\_READ8);

result = D1; // byte 1

#1; // Wait for CLK -> 0

endtask

task read16(input int address, output [15:0] result);

common\_read(address, C1\_READ16);

result[15:8] = D1[7:0]; // byte 1

result[7:0] = D1[15:8]; // byte 2

#1; // Wait for CLK -> 0

endtask

task read32(input int address, output [31:0] result);

common\_read(address, C1\_READ32);

result[31:24] = D1[7:0]; // byte 1

result[23:16] = D1[15:8]; // byte 2

#2;

result[15:8] = D1[7:0]; // byte 3

result[7:0] = D1[15:8]; // byte 4

#1; // Wait for CLK -> 0

endtask

// ---------- WRITE8/16/32 ----------

task common\_write(input int address, input int command);

// `log $display("CPU sending WRITE command");

C1 = command;

A1 = `discard\_last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

#2 A1 = `last\_n\_bits(address, CACHE\_OFFSET\_SIZE);

wait\_response();

#1; // Wait for CLK -> 0

endtask

task write8(input int address, input [7:0] data);

fork

common\_write(address, C1\_WRITE8);

D1 = data; // byte 1

join

endtask

task write16(input int address, input [15:0] data);

fork

common\_write(address, C1\_WRITE16);

begin

D1[7:0] = data[15:8]; // byte 1

D1[15:8] = data[7:0]; // byte 2

end

join

endtask

task write32(input int address, input [31:0] data);

fork

common\_write(address, C1\_WRITE32);

begin

D1[7:0] = data[31:24]; // byte 1

D1[15:8] = data[23:16]; // byte 2

#2;

D1[7:0] = data[15:8]; // byte 3

D1[15:8] = data[7:0]; // byte 4

end

join

endtask

localparam M = 64; // #define M 64

localparam N = 60; // #define N 60

localparam K = 32; // #define K 32

// reg[7:0] a[M][K]; // int8 a[M][K]; — 1 byte

// reg[15:0] b[K][N]; // int16 b[K][N]; — 2 bytes

// reg[31:0] c[M][N]; // int32 b[K][N]; — 4 bytes

int pa, pb, pc, s, tmp\_mul, tmp\_pa\_k, tmp\_pb\_x;

int a = 0,

b = M \* K,

c = b + 2 \* K \* N;

initial begin

#2 pa = a; // int8 \*pa = a;

#2 pc = c; // int32 \*pc = c;

for (int y = 0; y < M; ++y) begin // for (int y = 0; y < M; y++) {

for (int x = 0; x < N; ++x) begin // for (int x = 0; x < N; x++) {

#2 pb = b; // int16 \*pb = b;

#2 s = 0; // int32 s = 0;

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

read8(pa + k, tmp\_pa\_k);

read16(pb + 2 \* x, tmp\_pb\_x);

#12 s += tmp\_pa\_k \* tmp\_pb\_x; // s += pa[k] \* pb[x];

#2 pb += 2 \* N; // pb += N;

#2; end // }

write32(pc + 4 \* x, s); // pc[x] = s;

#2; end // }

#2 pa += K; // pa += K;

#2 pc += 4 \* N; // pc += N;

#2; end // }

#2; // Выход из функции

$display("Total time: %0d tacts", $time / 2);

$display("Cache hits: %0d/%0d = %0.2t%%", cache\_hits, cache\_hits + cache\_misses, real'(cache\_hits) \* 100 / (cache\_hits + cache\_misses));

$finish;

// $display();

// #10 C\_DUMP = 1;

// #10 M\_DUMP = 1;

// #10 $finish;

end

// initial #10000 $finish;

always @(CLK) begin

`log $display("C1\_WIRE = %d, C2\_WIRE = %d", C1\_WIRE, C2\_WIRE);

end

endmodule

**src/cache.sv**

module Cache (

input CLK,

inout[ADDR1\_BUS\_SIZE-1:0] A1\_WIRE,

inout[DATA\_BUS\_SIZE-1 :0] D1\_WIRE,

inout[CTR1\_BUS\_SIZE-1 :0] C1\_WIRE,

inout[ADDR2\_BUS\_SIZE-1:0] A2\_WIRE,

inout[DATA\_BUS\_SIZE-1 :0] D2\_WIRE,

inout[CTR2\_BUS\_SIZE-1 :0] C2\_WIRE,

input RESET,

input C\_DUMP

);

`map\_bus1; `map\_bus2; // Initialize wires

// Cache system

reg[7:0] data [CACHE\_SETS\_COUNT] [CACHE\_WAY] [CACHE\_LINE\_SIZE];

reg[7:0] tags [CACHE\_SETS\_COUNT] [CACHE\_WAY];

bit LRU\_bit [CACHE\_SETS\_COUNT] [CACHE\_WAY],

valid [CACHE\_SETS\_COUNT] [CACHE\_WAY],

dirty [CACHE\_SETS\_COUNT] [CACHE\_WAY];

// For storing A1 parts

reg[CACHE\_TAG\_SIZE-1:0] req\_tag;

reg[CACHE\_SET\_SIZE-1:0] req\_set;

reg[CACHE\_OFFSET\_SIZE-1:0] req\_offset;

// Internal variables

bit listening\_bus1 = 1;

reg[7:0] write\_buffer [4]; // Max is WRITE32 = 4 bytes

int found\_line;

// Initialization & RESET

task reset\_line(int set, int line);

LRU\_bit[set][line] = 0;

valid[set][line] = DEBUG\_MODE ? 1 : 0;

dirty[set][line] = DEBUG\_MODE ? 1 : 0;

tags[set][line] = DEBUG\_MODE ? 0 :'x;

for (int bbyte = 0; bbyte < CACHE\_LINE\_SIZE; ++bbyte) // Optional

data[set][line][bbyte] = DEBUG\_MODE ? ($random(SEED) >> 16) : 'x;

endtask

task reset;

for (int set = 0; set < CACHE\_SETS\_COUNT; ++set)

for (int line = 0; line < CACHE\_WAY; ++line)

reset\_line(set, line);

endtask

initial reset();

always @(posedge RESET) reset();

// Dumping

always @(posedge C\_DUMP)

for (int set = 0; set < CACHE\_SETS\_COUNT; ++set) begin

$display("Set #%0d", set);

for (int line = 0; line < CACHE\_WAY; ++line) begin

$write("Line #%0d (%0d): ", line, set \* CACHE\_WAY + line);

for (int bbyte = 0; bbyte < CACHE\_LINE\_SIZE; ++bbyte) $write("%b ", data[set][line][bbyte]);

$display("| TAG:%b | V:%b | D:%b | LRU:%b", tags[set][line], valid[set][line], dirty[set][line], LRU\_bit[set][line]);

end

$display();

end

// --------------------------------------------------- Main logic ----------------------------------------------------

// Передаём и получаем данные в little-endian, то есть вначале (слева) идёт второй байт ([15:8]), потом (справа) первый ([7:0])

// Тогда D = (второй байт, первый байт) -> второй байт = D2[15:8], первый байт = D2[7:0]

task send\_bytes\_D1(input [7:0] byte1, input [7:0] byte2);

`log $display("Cache: Sending byte: %d = %b", byte1, byte1);

`log $display("Cache: Sending byte: %d = %b", byte2, byte2);

D1[15:8] = byte2; D1[7:0] = byte1;

endtask

task send\_bytes\_D2(input [7:0] byte1, input [7:0] byte2);

`log $display("Cache: Sending byte: %d = %b", byte1, byte1);

`log $display("Cache: Sending byte: %d = %b", byte2, byte2);

D2[15:8] = byte2; D2[7:0] = byte1;

endtask

task receive\_bytes\_D1(output [7:0] byte1, output [7:0] byte2);

byte2 = D1\_WIRE[15:8]; byte1 = D1\_WIRE[7:0];

endtask

task receive\_bytes\_D2(output [7:0] byte1, output [7:0] byte2);

byte2 = D2\_WIRE[15:8]; byte1 = D2\_WIRE[7:0];

endtask

// Parses A1 bus to A1 parts + finds valid line corresponding to these parts

task parse\_A1; // Called on CLK = 1, return: CLK = 1

req\_tag = `discard\_last\_n\_bits(A1\_WIRE, CACHE\_SET\_SIZE);

req\_set = `last\_n\_bits(A1\_WIRE, CACHE\_SET\_SIZE);

#2 req\_offset = A1\_WIRE;

`log $display("tag = %b, set = %b, offset = %b", req\_tag, req\_set, req\_offset);

found\_line = -1;

for (int test\_line = 0; test\_line < CACHE\_WAY; ++test\_line)

if (valid[req\_set][test\_line] == 1 && tags[req\_set][test\_line] == req\_tag) found\_line = test\_line;

endtask

task read\_line\_from\_MEM(input [CACHE\_TAG\_SIZE-1:0] tag, input [CACHE\_SET\_SIZE-1:0] set, input int line); // Called on CLK = 0, return: CLK = 1

`log $display("Reading line from MemCTR");

tags[req\_set][found\_line] = tag;

C2 = C2\_READ\_LINE;

A2[CACHE\_TAG\_SIZE+CACHE\_SET\_SIZE-1:CACHE\_SET\_SIZE] = tag;

A2[CACHE\_SET\_SIZE-1:0] = set;

#2 `close\_bus2;

wait(CLK == 1 && C2\_WIRE == C2\_RESPONSE);

`log $display("Cache received C2\_RESPONSE");

for (int bytes\_start = 0; bytes\_start < CACHE\_LINE\_SIZE; bytes\_start += 2) begin

receive\_bytes\_D2(data[set][line][bytes\_start], data[set][line][bytes\_start + 1]);

`log $display("Cache: Wrote byte %d = %b to data[%0d][%0d][%0d]", data[set][line][bytes\_start], data[set][line][bytes\_start], set, line, bytes\_start);

`log $display("Cache: Wrote byte %d = %b to data[%0d][%0d][%0d]", data[set][line][bytes\_start + 1], data[set][line][bytes\_start + 1], set, line, bytes\_start + 1);

if (bytes\_start + 2 < CACHE\_LINE\_SIZE) #2; // Ждать надо везде, кроме последней передачи данных

end

valid[set][line] = 1;

dirty[set][line] = 0;

endtask

task write\_line\_to\_MEM(input [CACHE\_SET\_SIZE-1:0] set, input int line); // Called on CLK = 0, return: CLK = 1

C2 = C2\_WRITE\_LINE;

A2[CACHE\_TAG\_SIZE+CACHE\_SET\_SIZE-1:CACHE\_SET\_SIZE] = tags[set][line];

A2[CACHE\_SET\_SIZE-1:0] = set;

for (int bytes\_start = 0; bytes\_start < CACHE\_LINE\_SIZE; bytes\_start += 2) begin

send\_bytes\_D2(data[set][line][bytes\_start], data[set][line][bytes\_start + 1]);

if (bytes\_start + 2 < CACHE\_LINE\_SIZE) #2; // Ждать надо везде, кроме последней передачи данных

end

#1 `close\_bus2;

wait(CLK == 1 && C2\_WIRE == C2\_RESPONSE);

`log $display("Cache received C2\_RESPONSE");

endtask

task invalidate\_line(input [CACHE\_SET\_SIZE-1:0] set, input int line); // Called on CLK = 0, return: CLK = 0

`log $display("Invalidating line: set = %b, line = %0d | D: %0d", set, line, dirty[set][line]);

// Если линия Dirty, то нужно сдампить её содержимое в Mem

if (dirty[set][line]) write\_line\_to\_MEM(set, line);

// reset\_line(set, line); // Правильнее будет сделать valid[set][line] = 0, но так проще тестировать

valid[set][line] = 0;

#1; // Wait for CLK -> 0

endtask

task find\_spare\_line; // Called on CLK = 0, return: CLK = 0

// Сначала ищем пустую линию

for (int test\_line = 0; test\_line < CACHE\_WAY; ++test\_line)

if (valid[req\_set][test\_line] == 0) found\_line = test\_line;

// Если таковой не нашлось, то по LRU берём самую давнюю занятую (LRU\_bit = 0) и инвалидируем

if (found\_line == -1) begin

for (int test\_line = 0; test\_line < CACHE\_WAY; ++test\_line)

if (LRU\_bit[req\_set][test\_line] == 0) found\_line = test\_line;

invalidate\_line(req\_set, found\_line);

end

endtask

task handle\_c1\_read(int read\_bits); // Called on CLK = 1

`log $display("Cache: C1\_READ%0d, A1 = %b", read\_bits, A1\_WIRE);

listening\_bus1 = 0; parse\_A1();

#1 C1 = C1\_NOP;

if (found\_line == -1) begin

`log $display("Line not found, finding spare one");

++cache\_misses;

#(CACHE\_MISS\_DELAY - 4);

find\_spare\_line();

read\_line\_from\_MEM(req\_tag, req\_set, found\_line);

end else begin

`log $display("Found line #%0d", found\_line);

++cache\_hits;

#(CACHE\_HIT\_DELAY - 5);

end

LRU\_bit[req\_set][found\_line] = 1;

LRU\_bit[req\_set][!found\_line] = 0;

#1 C1 = C1\_RESPONSE;

case (read\_bits)

8: send\_bytes\_D1(data[req\_set][found\_line][req\_offset], 0);

16: send\_bytes\_D1(data[req\_set][found\_line][req\_offset], data[req\_set][found\_line][req\_offset + 1]);

32: begin

send\_bytes\_D1(data[req\_set][found\_line][req\_offset], data[req\_set][found\_line][req\_offset + 1]);

#2 send\_bytes\_D1(data[req\_set][found\_line][req\_offset + 2], data[req\_set][found\_line][req\_offset + 3]);

end

endcase

#2 `close\_bus1; listening\_bus1 = 1;

endtask

task handle\_c1\_write(int write\_bits); // Called on CLK = 1

`log $display("Cache: C1\_WRITE%0d, A1 = %b", write\_bits, A1\_WIRE);

listening\_bus1 = 0;

fork // duration: 2 tacks

parse\_A1();

case (write\_bits)

8: receive\_bytes\_D1(write\_buffer[0], write\_buffer[1]); // Second byte is just a placeholder

16: receive\_bytes\_D1(write\_buffer[0], write\_buffer[1]);

32: begin

receive\_bytes\_D1(write\_buffer[0], write\_buffer[1]);

#2 receive\_bytes\_D1(write\_buffer[2], write\_buffer[3]);

end

endcase

join

#1 C1 = C1\_NOP;

if (found\_line == -1) begin

`log $display("Line not found, finding spare one");

++cache\_misses;

#(CACHE\_MISS\_DELAY - 4);

find\_spare\_line();

read\_line\_from\_MEM(req\_tag, req\_set, found\_line);

end else begin

`log $display("Found line #%0d", found\_line);

++cache\_hits;

#(CACHE\_HIT\_DELAY - 5);

end

dirty[req\_set][found\_line] = 1;

LRU\_bit[req\_set][found\_line] = 1;

LRU\_bit[req\_set][!found\_line] = 0;

for (int i = 0; i < write\_bits / 8; i += 1) begin

data[req\_set][found\_line][req\_offset + i] = write\_buffer[i];

`log $display("Cache: Wrote byte %d = %b to data[%0d][%0d][%0d]", write\_buffer[i], write\_buffer[i], req\_set, found\_line, req\_offset + i);

end

#1 C1 = C1\_RESPONSE;

#2 `close\_bus1; listening\_bus1 = 1;

endtask

always @(posedge CLK) begin

if (listening\_bus1) case (C1\_WIRE)

C1\_NOP: begin `log $display("Cache: C1\_NOP"); end

C1\_READ8: handle\_c1\_read(8);

C1\_READ16: handle\_c1\_read(16);

C1\_READ32: handle\_c1\_read(32);

C1\_WRITE8: handle\_c1\_write(8);

C1\_WRITE16: handle\_c1\_write(16);

C1\_WRITE32: handle\_c1\_write(32);

C1\_INVALIDATE\_LINE: begin

`log $display("Cache: C1\_INVALIDATE\_LINE, A1 = %b", A1\_WIRE);

listening\_bus1 = 0; parse\_A1();

#1 C1 = C1\_NOP;

if (found\_line == -1) begin

$display("Line not found");

#(CACHE\_HIT\_DELAY - 4); // Для реалистичности поставим задежку между C1\_INVALIDATE\_LINE и отправкой данных/C1\_RESPONSE равную CACHE\_HIT\_DELAY тактов

end else begin

$display("Found line #%0d", found\_line);

invalidate\_line(req\_set, found\_line);

end

C1 = C1\_RESPONSE;

`log $display("Cache: Sending C1\_RESPONSE");

#2 `close\_bus1; listening\_bus1 = 1;

end

endcase

end

endmodule

**src/commands.sv**

typedef enum {

C1\_NOP,

C1\_READ8,

C1\_READ16,

C1\_READ32,

C1\_INVALIDATE\_LINE,

C1\_WRITE8,

C1\_WRITE16,

C1\_WRITE32

} C1\_COMMANDS; // CPU <-> Cache (BUS 1)

localparam C1\_RESPONSE = 7;

typedef enum {

C2\_NOP,

C2\_RESPONSE,

C2\_READ\_LINE,

C2\_WRITE\_LINE

} C2\_COMMANDS; // Cache <-> Mem (BUS 2)

**src/common.sv**

// Tools

`define discard\_last\_n\_bits(register, n) (register >> n)

`define first\_n\_bits(register, n) `discard\_last\_n\_bits(register, $size(register) - n)

`define last\_n\_bits(register, n) (register & ((1 << n) - 1))

`define log \

if (DEBUG\_MODE == 1) $write("[%3t | CLK=%0d] ", $time, $time % 2); \ // CLK = $time % 2 representation works much better, more suitable for debugging

if (DEBUG\_MODE == 1) // $display(...)

// BUSes

`define map\_bus1 \

reg[ADDR1\_BUS\_SIZE-1:0] A1 = 'z; assign A1\_WIRE = A1; \

reg[DATA\_BUS\_SIZE-1 :0] D1 = 'z; assign D1\_WIRE = D1; \

reg[CTR1\_BUS\_SIZE-1 :0] C1 = 'z; assign C1\_WIRE = C1;

`define map\_bus2 \

reg[ADDR2\_BUS\_SIZE-1:0] A2 = 'z; assign A2\_WIRE = A2; \

reg[DATA\_BUS\_SIZE-1 :0] D2 = 'z; assign D2\_WIRE = D2; \

reg[CTR2\_BUS\_SIZE-1 :0] C2 = 'z; assign C2\_WIRE = C2;

`define close\_bus1 C1 = 'z; A1 = 'z; D1 = 'z;

`define close\_bus2 C2 = 'z; A2 = 'z; D2 = 'z;

localparam DEBUG\_MODE = 0;

**src/mem.sv**

module MemCTR (

input CLK,

inout[ADDR2\_BUS\_SIZE-1:0] A2\_WIRE,

inout[DATA\_BUS\_SIZE-1 :0] D2\_WIRE,

inout[CTR2\_BUS\_SIZE-1 :0] C2\_WIRE,

input RESET,

input M\_DUMP

);

`map\_bus2; // Initialize wires

reg[7:0] ram [MEM\_SIZE];

reg[CACHE\_ADDR\_SIZE-1:0] address;

bit listening\_bus2 = 1;

// Initialization & RESET

task intialize\_ram;

for (int i = 0; i < MEM\_SIZE; ++i) ram[i] = $random(SEED) >> 16;

endtask

always @(RESET) intialize\_ram();

initial begin

intialize\_ram();

// $display("RAM:");

// for (memory\_pointer = 0; memory\_pointer < 100; memory\_pointer += 1)

// $display("[%2d] %d", memory\_pointer, ram[memory\_pointer]);

// $display();

end

// Dumping

always @(posedge M\_DUMP)

for (int i = 0; i < 100; ++i) // 100 for testing, should be MEM\_SIZE (warning: MEM\_SIZE ~= 500'000, you don't want to print this)

$display("Byte %2d: %d = %b", i, ram[i], ram[i]);

// --------------------------------------------------- Main logic ----------------------------------------------------

task send\_bytes\_D2(input [7:0] byte1, input [7:0] byte2);

D2[15:8] = byte2; D2[7:0] = byte1;

endtask

task receive\_bytes\_D2(output [7:0] byte1, output [7:0] byte2);

byte2 = D2\_WIRE[15:8]; byte1 = D2\_WIRE[7:0];

endtask

task parse\_A2;

address = A2\_WIRE; address <<= CACHE\_OFFSET\_SIZE;

endtask

always @(posedge CLK) begin

if (listening\_bus2) case (C2\_WIRE)

C2\_NOP: begin `log $display("MemCTR: C2\_NOP"); end

C2\_READ\_LINE: begin

`log $display("MemCTR: C2\_READ\_LINE, A2 = %b", A2\_WIRE);

listening\_bus2 = 0; parse\_A2();

#1 C2 = C2\_NOP;

#(MEM\_CTR\_DELAY - 3);

#1 C2 = C2\_RESPONSE;

`log $display("MemCTR: Sending C2\_RESPONSE");

for (int bytes\_start = 0; bytes\_start < CACHE\_LINE\_SIZE; bytes\_start += 2) begin

send\_bytes\_D2(ram[address], ram[address + 1]);

`log $display("MemCTR: Sent byte %d = %b from ram[%b]", ram[address], ram[address], address);

++address;

`log $display("MemCTR: Sent byte %d = %b from ram[%b]", ram[address], ram[address], address);

++address;

if (bytes\_start + 2 < CACHE\_LINE\_SIZE) #2; // Ждать надо везде, кроме последней передачи данных

end

#2 `close\_bus2; listening\_bus2 = 1;

end

C2\_WRITE\_LINE: begin

`log $display("MemCTR: C2\_WRITE\_LINE, A2 = %b", A2\_WIRE);

listening\_bus2 = 0; parse\_A2();

fork

#(MEM\_CTR\_DELAY - 2); // С одной стороны ждём MEM\_CTR\_DELAY тактов до отправки C2\_RESPONSE, а с другой параллельно читаем и пишем данные

begin

for (int bytes\_start = 0; bytes\_start < CACHE\_LINE\_SIZE; bytes\_start += 2) begin

receive\_bytes\_D2(ram[address], ram[address + 1]);

`log $display("MemCTR: Wrote byte %d = %b to ram[%b]", ram[address], ram[address], address);

++address;

`log $display("MemCTR: Wrote byte %d = %b to ram[%b]", ram[address], ram[address], address);

++address;

if (bytes\_start + 2 < CACHE\_LINE\_SIZE) #2; // Ждать надо везде, кроме последней передачи данных

end

C2 = C2\_NOP;

end

join

#1 C2 = C2\_RESPONSE;

`log $display("MemCTR: Sending C2\_RESPONSE");

#2 `close\_bus2; listening\_bus2 = 1;

end

endcase

end

endmodule

**src/parameters.sv**

// Given parameters

localparam CACHE\_WAY = 2;

localparam CACHE\_TAG\_SIZE = 10; // [бит]

localparam CACHE\_LINE\_SIZE = 16; // [байт] 16 байт

localparam CACHE\_LINE\_COUNT = 64;

localparam MEM\_SIZE = 512 \* 1024; // [байт] 512 Кбайт

// Calculated parameters

localparam CACHE\_SIZE = 1024; // [байт] CACHE\_LINE\_SIZE × CACHE\_LINE\_COUNT

localparam CACHE\_SETS\_COUNT = 32; // CACHE\_LINE\_COUNT / CACHE\_WAY

localparam CACHE\_SET\_SIZE = 5; // [бит] log(CACHE\_SETS\_COUNT)

localparam CACHE\_OFFSET\_SIZE = 4; // [бит] log(CACHE\_LINE\_SIZE)

localparam CACHE\_ADDR\_SIZE = 19; // [бит] log(MEM\_SIZE)

// BUS sizes

localparam ADDR1\_BUS\_SIZE = 15; // [бит]

localparam ADDR2\_BUS\_SIZE = 15; // [бит]

localparam DATA\_BUS\_SIZE = 16; // [бит] по условию

localparam CTR1\_BUS\_SIZE = 3; // [бит], так как команды 0..7

localparam CTR2\_BUS\_SIZE = 2; // [бит], так как команды 0..3

// Memory initialization seed

int SEED = 225526;

// Delays (\*2 because CLK changes every #1, so 1 -> 0 -> 1 equals #2)

localparam CACHE\_HIT\_DELAY = 4 \* 2;

localparam CACHE\_MISS\_DELAY = 6 \* 2;

localparam MEM\_CTR\_DELAY = 100 \* 2;

**src/statistics.sv**

int cache\_hits = 0;

int cache\_misses = 0;

**Analytical solution\parameters.py**

# Copy of src/parameters.sv

# Given parameters

CACHE\_WAY = 2

CACHE\_TAG\_SIZE = 10

CACHE\_LINE\_SIZE = 16

CACHE\_LINE\_COUNT = 64

MEM\_SIZE = 512 \* 1024

# Calculated parameters

CACHE\_SIZE = 1024

CACHE\_SETS\_COUNT = 32

CACHE\_SET\_SIZE = 5

CACHE\_OFFSET\_SIZE = 4

CACHE\_ADDR\_SIZE = 19

# BUS sizes

ADDR1\_BUS\_SIZE = 15

ADDR2\_BUS\_SIZE = 15

DATA\_BUS\_SIZE = 16

CTR1\_BUS\_SIZE = 3

CTR2\_BUS\_SIZE = 2

# Memory initialization seed

SEED = 225526

# Delays

CACHE\_HIT\_DELAY = 4

CACHE\_MISS\_DELAY = 6

MEM\_CTR\_DELAY = 100

**Analytical solution\solution.py**

from parameters import \*

TIME = 0

class CacheLine:

tag = None

LRU\_bit = 0

valid = dirty = False

class Cache:

def \_\_init\_\_(self):

self.lines = [[CacheLine() for \_ in range(CACHE\_WAY)] for \_ in range(CACHE\_SETS\_COUNT)]

self.hits = self.misses = 0

self.read8 = self.read16 = lambda addr: self.access(addr)

self.write32 = lambda addr: self.access(addr, True)

def wait\_clk(self, clk\_value):

global TIME

if (TIME % 1) \* 2 != clk\_value:

TIME += 0.5

def read\_line\_from\_MEM(self, tag, sset, line):

global TIME

self.lines[sset][line].tag = tag

self.wait\_clk(1)

TIME += MEM\_CTR\_DELAY

for bbytes\_start in range(0, CACHE\_LINE\_SIZE, 2):

TIME += 1

TIME -= 1

self.lines[sset][line].valid = True

self.lines[sset][line].dirty = False

def write\_line\_to\_MEM(self):

global TIME

self.wait\_clk(1)

TIME += MEM\_CTR\_DELAY # MemCTR

def find\_valid\_line(self, tag, sset):

for line in range(CACHE\_WAY):

if self.lines[sset][line].valid and self.lines[sset][line].tag == tag:

return line

def invalidate\_line(self, sset, line):

global TIME

if self.lines[sset][line].dirty:

self.write\_line\_to\_MEM()

self.lines[sset][line].valid = False

TIME += 0.5

def find\_spare\_line(self, sset):

global TIME

for line in range(CACHE\_WAY):

if not self.lines[sset][line].valid:

return line

for line in range(CACHE\_WAY):

if self.lines[sset][line].LRU\_bit == 0:

self.invalidate\_line(sset, line)

return line

def access(self, addr, is\_write=False):

global TIME

self.wait\_clk(0) # To send command

self.wait\_clk(1) # To receive command

# req\_offset = addr % (2 \*\* CACHE\_OFFSET\_SIZE)

addr = addr >> CACHE\_OFFSET\_SIZE

req\_tag = addr >> CACHE\_SET\_SIZE

req\_set = addr % (2 \*\* CACHE\_SET\_SIZE)

found\_line = self.find\_valid\_line(req\_tag, req\_set)

TIME += 1 # parse\_A1

TIME += 0.5 # C1\_NOP

if found\_line is None:

self.misses += 1

TIME += CACHE\_MISS\_DELAY - 2

found\_line = self.find\_spare\_line(req\_set)

self.read\_line\_from\_MEM(req\_tag, req\_set, found\_line)

else:

self.hits += 1

TIME += CACHE\_HIT\_DELAY - 2.5

if is\_write:

self.lines[req\_set][found\_line].dirty = True

self.lines[req\_set][found\_line].LRU\_bit = 1

self.lines[req\_set][not found\_line].LRU\_bit = 0

TIME += 0.5 # C1\_RESPONSE

# В READ send\_bytes не нужен, так как посылаем либо 8, либо 16 бит, одновременно с C1\_RESPONSE

self.wait\_clk(1)

TIME += 0.5 # // Wait for CLK -> 0

cache = Cache()

# ---------------------------------------------------- Actual task -----------------------------------------------------

def assign(value):

global TIME

TIME += 1

return value

def add(target, value):

global TIME

TIME += 1

return target + value

M = 64 # #define M 64

N = 60 # #define N 60

K = 32 # #define K 32

a = 0 # int8 a[M][K];

b = M \* K # int16 b[K][N];

c = b + 2 \* K \* N # int32 c[M][N];

pa = assign(a)

pc = assign(c)

for y in range(M):

for x in range(N):

pb = assign(b)

s = assign(0)

for k in range(K):

cache.read8(pa + k)

cache.read16(pb + 2 \* x)

TIME += 5 + 1 # 1 умножение и 1 сложение

pb = add(pb, 2 \* N)

TIME += 1 # end of "for"

cache.write32(pc + 4 \* x)

TIME += 1 # end of "for"

pa = add(pa, K)

pc = add(pc, 4 \* N)

TIME += 1 # end of "for"

TIME += 1 # end of function

print(f"Total time: {TIME} tacts")

print("Cache hits: {}/{} = {}%".format(

cache.hits, cache.hits + cache.misses,

round(cache.hits \* 100 / (cache.hits + cache.misses), 2) if cache.hits + cache.misses else 0

))