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:

$$\text{CACHE\_SET\_SIZE} = log_2(\text{CACHE\_SETS\_COUNT}) = log_2(32) = 5 \text{ бит}$$

- Размер смещения:

$$\text{CACHE\_OFFSET\_SIZE} = log_2(\text{CACHE\_LINE\_SIZE}) = log_2(16 \text{ байт}) = 4 \text{ бит}$$

- Размер адреса:

$$\text{CACHE\_ADDR\_SIZE} = log_2(\text{MEM\_SIZE}) = log_2(512 \cdot 1024 \text{ байт}) = 19 \text{ бит}$$

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:

- $\text{ADDR1\_BUS\_SIZE} = max(\text{CACHE\_TAG\_SIZE} + \text{CACHE\_SET\_SIZE}, \text{CACHE\_OFFSET\_SIZE}) = max(10 + 5, 7) = 15 \text{ бит}$
- **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 $log_2(7 + 1) = 3$ и $log_2(3 + 1) = 2$ соответственно.

Все параметры системы указаны в файле src/parameters.sv

Аналитическое решение задачи

Аналитическое решение было выполнено с помощью кода на языке Python (находится в файле “`Analytical solution/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[→].

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` и `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 сéте 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`[→], 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
))
```