Лабораторная работа №2	М3137	2022
Моделирование схем в Verilog	Нартов Дмитрий Николаевич	

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

Инструментарий и требования к работе: Icarus Verilog 11, GNU C++ 17.

Описание

1. Описание работы системы.
2. Вычисление недостающих параметров системы.
3. Аналитическое решение задачи.
4. Моделирование заданной системы на Verilog.
5. Воспроизведение задачи на Verilog.
6. Сравнение полученных результатов.
7. Листинг кода

Вариант

Вариант №1.

Описание работы системы.

Коротко опишем что такое кэш и какие принципы он руководствуется. Кэш - это память, которая находиться между процессором и оперативной памятью, и значительно ускоряет доступ к некоторым данным. Существует несколько подходов, как строить кэш, но мы будем рассматривать наборно-ассоциативный. В таком варианте кэш представляет собой следующее:

Доступ к данным происходит следующим образом: к сету мы можем обратиться сразу (на картинке сет - это кэш-линии обведенные в прямоугольники со скругленными краями), а вот внутри каждого сета придется искать подходящую линию проходясь по всем и сравнивая на равенство тегов, которые служат “ключом”. Рассмотрим устройство кэш линии:

valid	dirty	tag	data
1	1	CACHE_TAG_SIZE	CACHE_LINE_SIZE

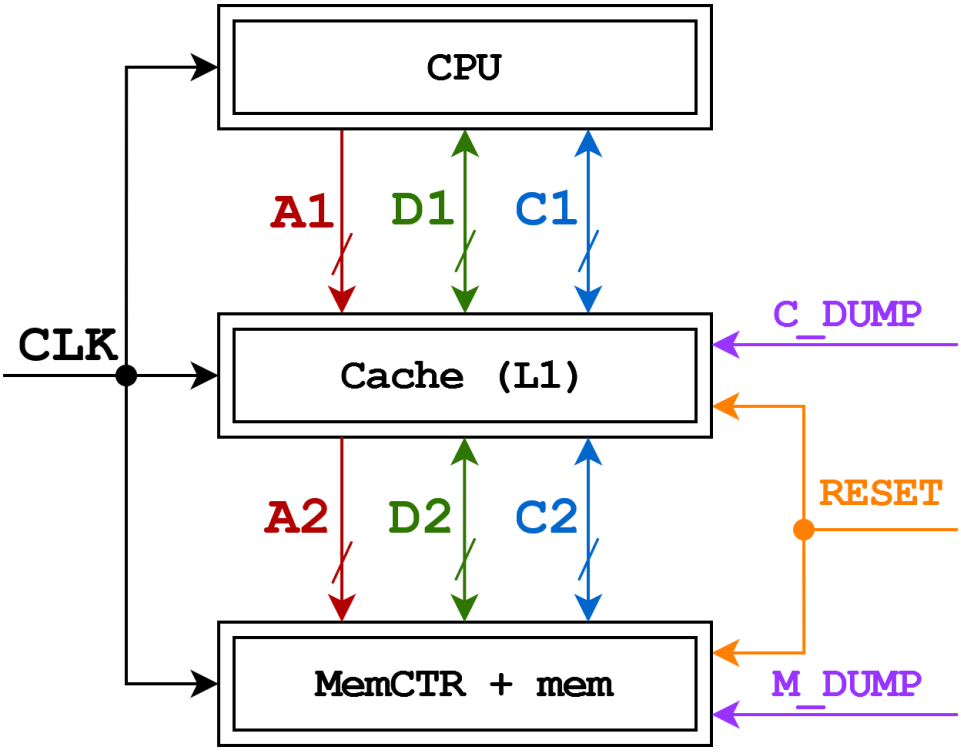
Как мы видим кэш-линия, кроме полезных данных также содержит некоторое количество служебной информации. Valid это один бит, который отвечает за корректность кэш-линии, он равен 0, когда кэш линия свободна (не имеем никаких полезных данных)б иначе 1. Dirty - это тоже один бит, но который отвечает за то, какие данные хранит линия (0 - если полезные данные являются точной копией данных из оперативной памяти, 1 иначе). Про тэг было написано выше, а data -это полезные данные, которые хранит наш кэш. Стоит заметить, что кэш хранит не один байт данных в качестве полезных, а какую-то непрерывную последовательность битов из оперативной памяти, так как это существенно позволяет увеличить производительность. Также немаловажной частью кэш является, то как

адрес оперативной памяти интерпретирует кэш:

tag	set	offset
CACHE_TAG_SIZE	CACHE_SET_SIZE	CACHE_OFFSET_SIZE

Тэг здесь является частью, которую мы будем сравнивать с тегом в кэш-линии, чтобы понять подходит ли она нам. Set - это номер сета в котором должна находиться искомая кэш-линия. И, наконец, offset - это номер первого байта последовательность, которая нам нужна (это важно, так как мы должны уметь отдавать последовательность из некоторого количества подряд идущих байтиков).

Теперь обсудим принципы, которые используются в нашей схеме. Первый из них - это look-through. Так как не находиться в вакууме а тесно взаимосвязан с памятью и процессором, нужно описать поведение в случае запроса данных от процессора. Здесь поможет следующий рисунок:



Кэш находиться между памятью и процессором, и когда процессор посылает запрос к памяти сначала этот запрос приходит к кэшу и он пытается найти у себя подходящие данные, если такие есть, то он отдает их, иначе идет к памяти и просит у нее. Именно такой подход называется look-through. Также существует такое понятие, как политика замещения,

которая описывает поведение, когда нам нужно добавить новую кэш-линию, а свободного места нет. Мы будем иметь дело с политикой, называющейся lru (Least recently used). Здесь появляется еще один служебный бит в кэш-линии, который отвечает за возраст кэш-линии. Возраст - это сколько запросов назад обращались к данной кэш-линии. Важно заметить, что возраст не единый для всех линии в кэше, а отдельный для каждого сета. В случае, когда свободного места нет мы должны найти линию, которая дольше всего не использовалась, то есть, которая имеет максимальный возраст. Но что же с ней делать? Тут в бой идет принцип называющийся write-back. Мы утверждает, что если линия, на место которой мы хотим поместить новые данные, должна быть проверена на следующее: если данная кэш-линия не содержит модифицированные данные, то мы можем заменить ее на свою, но если нет, то мы должны сначала её записать в память, а только потом можно её заменить. Более подробно работа этих принципов вместе будет разобрана, на примере аналитического решения.

Вычисление недостающих параметров системы.

```
MEM_SIZE                                                                                  = 512 Kbyte
CACHE_SIZE          = CACHE_LINE_SIZE * CACHE_LINE_COUNT = 16 * 64                        = 1024 Kbyte
CACHE_LINE_SIZE                                                                           = 16 byte
CACHE_LINE_COUNT                                                                          = 64
CACHE_WAY                                                                                 = 2
CACHE_SETS_COUNT    = CACHE_LINE_COUNT / CACHE_WAY = 2^6 / 2 = 2^5                        = 32
CACHE_TAG_SIZE                                                                            = 10 bit
CACHE_SETS_SIZE     = log2(CACHE_SETS_COUNT) = log2(32)                                   = 5 bit
CACHE_OFFSET_SIZE   = log2(CACHE_LINE_SIZE) = log2(16)                                    = 4 bit
CACHE_ADDR_SIZE     = CACHE_TAG_SIZE + CACHE_SETS_SIZE + CACHE_OFFSET_SIZE = 10 + 4 + 5   = 19 bit
VALID                                                                                     = 1 bit
DIRTY                                                                                     = 1 bit
AGE                 = log2(CACHE_WAY)                                                     = 1 bit
```

На данной картинке приведены расчеты параметров, формулы вытекают из устройства кэша, а также были разобраны на занятиях, поэтому объяснения формул не приводятся. Также прикладываю картинку с расчетами размера шин:

```
ADDR1_BUS_SIZE = max(CACHE_TAG_SIZE + CACHE_SETS_SIZE, CACHE_OFFSET_SIZE) = max(10 + 5, 4)  = 15 bit
ADDR2_BUS_SIZE = CACHE_TAG_SIZE + CACHE_SETS_SIZE = 10 + 5                                  = 15 bit
DATA1_BUS_SIZE                                                                              = 16 bit
DATA2_BUS_SIZE                                                                              = 16 bit
CTR1_BUS_SIZE = max(log2(8), log2(2)) = max(3, 1)                                           = 3 bit
CTR2_BUS_SIZE = max(log2(3), log2(2)) = max(2, 1)                                           = 2 bit
```

Файл с расчету прикреплю с остальным файлам.

Аналитическое решение задачи.

Аналитическое решение считает кол-во тактов и отношение кэш попаданий к общему числу обращений в память.

```
#include <iostream>
#include <map>

using namespace std;

#define M 64
#define N 60
#define K 32

const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 32;

const int CACHE_HIT = 6;
const int CACHE_MISS = 4;
const int MEM_CTR = 100;
const int DATA_TRANSPORT = 1;
const int D1_BUS_DATA = 16;
const int D2_BUS_DATA = 16;

int my_time = 0;
int cache_hits = 0, cache_queries = 0;


//read

//write

struct cache_line {
	int valid, dirty, age;
	int tag, data;

	cache_line() {
		valid = 0;
		tag = -1;
	}

	cache_line(int valid, int dirty, int age, int tag, int data) {
		this -> valid = valid;
		this -> dirty = dirty;
		this -> age = age;
		this -> tag = tag;
		this -> data = data;
	}
};

cache_line cache[32][2];

void read(int addr) {
	cache_queries++;
	int tag = (addr >> 9);
```

```
51.         int set = ((addr >> 4) & ((1 << 5) - 1));
52.         int offset = (addr & ((1 << 4) - 1));
53.
54.         if (cache[set][0].tag == tag) {
55.               cache_hits++;
56.               my_time += CACHE_HIT;
57.               cache[set][0].age = 0;
58.               cache[set][1].age = 1;
59.               return;
60.         }
61.
62.         if (cache[set][1].tag == tag) {
63.               cache_hits++;
64.               my_time += CACHE_HIT;
65.               cache[set][1].age = 0;
66.               cache[set][0].age = 1;
67.               return;
68.         }
69.
70.         my_time += CACHE_MISS;
71.         int num = -1;
72.
73.         if (cache[set][0].valid == 0) {
74.               num = 0;
75.         } else if (cache[set][1].valid == 0) {
76.               num = 1;
77.         }
78.
79.         if (num == -1) {
80.               if (cache[set][0].age == 1) {
81.                     num = 0;
82.               } else {
83.                     num = 1;
84.               }
85.         }
86.
87.         if (cache[set][num].valid != 0 && cache[set][num].dirty == 1) {
88.               my_time += MEM_CTR;
89.         }
90.         my_time += MEM_CTR;
91.         cache[set][num] = cache_line(1, 0, 0, tag, 1010);
92.         cache[set][1 - num].age = 1;
93. }
94.
95. void write(int addr) {
96.         cache_queries++;
97.         int tag = (addr >> 9);
98.         int set = ((addr >> 4) & ((1 << 5) - 1));
99.         int offset = (addr & ((1 << 4) - 1));
100.
101.        if (cache[set][0].tag == tag) {
102.              cache_hits++;
103.              my_time += CACHE_HIT;
104.              cache[set][0].dirty = 1;
105.              cache[set][0].age = 0;
106.              cache[set][1].age = 1;
107.              return;
108.        }
```

```
109.
110.        if (cache[set][1].tag == tag) {
111.              cache_hits++;
112.              my_time += CACHE_HIT;
113.              cache[set][1].dirty = 1;
114.              cache[set][1].age = 0;
115.              cache[set][0].age = 1;
116.              return;
117.        }
118.
119.        my_time += CACHE_MISS;
120.        int num = -1;
121.
122.        if (cache[set][0].valid == 0) {
123.              num = 0;
124.        } else if (cache[set][1].valid == 0) {
125.              num = 1;
126.        }
127.
128.        if (num == -1) {
129.              if (cache[set][0].age == 1) {
130.                    num = 0;
131.              } else {
132.                    num = 1;
133.              }
134.        }
135.
136.        if (cache[set][num].valid != 0 && cache[set][num].dirty == 1) {
137.              my_time += MEM_CTR;
138.        }
139.        my_time += MEM_CTR;
140.        cache[set][num] = cache_line(1, 1, 0, tag, 1010);
141.        cache[set][1 - num].age = 1;
142.
143.    }
144.
145.    int main() {
146.        for (int i = 0; i < 32; i++) {
147.              for (int j = 0; j < 2; j++) {
148.                    cache[i][j] = cache_line();
149.              }
150.        }
151.
152.        int pa, pb, pc;
153.        pa = 0;
154.        my_time++;
155.        pc = 5888;
156.        my_time++;
157.        int addr = 0;
158.        my_time++;
159.        for (int y = 0; y < M; y++) {
160.              my_time++;
161.              my_time++;
162.              for (int x = 0; x < N; x++) {
163.                    my_time++;
164.                    my_time++;
165.                    pb = 2048;
166.                    my_time += 2;
```

```
167.                    for (int k = 0; k < K; k++) {
168.                          my_time++;
169.                          addr = (pa + k);
170.                          read(addr);
171.                          my_time += 2;
172.                          addr = (pb + 2 * x);
173.                          my_time++;
174.                          read(addr);
175.                          my_time += 2;
176.                          pb += (2 * N);
177.                          my_time++;
178.                          my_time += 6;
179.                    }
180.                    addr = (pc + 4 * x);
181.                    my_time++;
182.                    write(addr);
183.                    my_time += 32 / 16;
184.              }
185.              pc += 4 * N;
186.              pa += K;
187.              my_time += 2;
188.        }
189.
190.        cout << "cache_hits: " << cache_hits << ", cache_queries: " <<
    cache_queries << '\n';
191.        cout << "tacts: " << my_time + 1 << '\n';
192.        return 0;
193.    }
```

Разберем этот код. На строке 28 объявляется структура, которая представляет собой кэш-линию. На 46 строке объявляется двумерный массив кэш-линий, который и представляет собой кэш. Первое измерение - это кэш-сет, второе кэш-линия в этом сете. Дальше есть две функции read и write, которые начинаются соответственно на 48 и 95 строке соответственно. И main, в котором симулируется работа алгоритма из условия. Так как нам нужно только посчитать количество кэш попаданий и такты, то пересылкой полезных данных и не используемыми функциями можно пренебречь. Коротко пробежимся по работе функции read и write.

В функцию read поступает какой-то адрес, нам нужно найти данные по этому адресу. Сперва проверяем есть ли нужные нам данные в кэше, если да, то радуемся и засчитываем кэш-попадание и не забываем

пересчитать возрасты. По коду видно что сначала проверяется первая кэш-линия, а затем вторая. Если попадания в кэш не произошло, то не отчаиваемся и идем искать место под кэш линию, так как мы не можем просто переслать нужные данные из оперативной памяти в процессор игнорируя кэш. Место ищется просто: если в сете есть пустое место, то давайте запишем новую кэш-линию, иначе найдем самую старую линию и заместим её, попутно не забывая проверить не лежат в старой линии модифицированные данные, которые нужно отправить в оперативку.

Write работает почти также, только нужно не забыть поставить линии в которую записали данные dirty в 1. Если писать некуда то проделываем точно такую же последовательность с поиском места, записываем на найденное место новую линию (её прислали из оперативки) и счастливо пишем в неё уже то, что уже нужно, опять же, не забывая поставить dirty в 1.

Конечно же в процессе работы не забываем считать время, если попали в кэш, то это занимает 6 перед тем как начинаем отвечать полезные данные, если нет, то через 4 такта идем к памяти и просим у нее данные. Ну а данные от оперативки будем ждать 100 тактов. Безусловно, нужно также учитывать время на пересылку данных по шине, работу циклов, инициализацию переменных и остальные команды, цена которых описана в тз. В большинстве случаев время работы операции суммируется с остальным после операции, для которой задано такое время.НомерРаботы

Чуть не забыл, одним из ключевых фактов, которым мы руководствуемся во время подсчета ответа - это то, что все локальные переменные лежат в регистрах процессора. Остаётся, что все обращения к памяти - это получение элемента из массива.

Ответ на задачу получился следующий: процент кэш-попаданий = 0.9137 такты = 5404940.

Моделирование задачи на Verilog.

Ну логика работы кода, ровно такая же как и в аналитическом решение, только это реализовано конструкциями языка Verilog. Главное отличие - это задержки, задержки, задержки… А еще это то, что здесь мы отправляем данные и реализуем все функции (по крайней мере, пытаемся). Не забываем про то, что данные отправляются по принципу little-endian (это когда данные передаются от младшего бита к старшему). Сам код модели не сильно отличается от кода аналитического решения. Есть пара интересных нюансов реализации. Чтобы данные доходили вовремя куда надо, мы реализуем следующее: синхронизируем запись на шину по спаду, а прием по фронту, но даже так возникает проблема с состоянием гонки, когда на одном конце ставят z, а на другом команду. Решить эту проблему можно подгонкой задержек.

Воспроизведение задачи на верилог.

Тут есть небольшая проблемка. Я не смог до конца отдебагать код. Но в случае доведения этой схемы до ума она будет работать, так как логика обработки команд правильная и проверена в аналитическом решении.

Сравнение результатов решений.

При сравнении результатов стоит сказать, что такты посчитанные в Verilog, могут не сходиться с аналитическим решением так как мы подкручиваем задержки, чтобы все нормально работало. Ну а кэш-попадания должны сойтись.

Листинг.

Аналитическое решение

```
#include <iostream>
#include <map>

using namespace std;

#define M 64
#define N 60
#define K 32

const int CACHE_WAY = 2;
const int CACHE_SETS_COUNT = 32;

const int CACHE_HIT = 6;
const int CACHE_MISS = 4;
const int MEM_CTR = 100;
const int DATA_TRANSPORT = 1;
const int D1_BUS_DATA = 16;
const int D2_BUS_DATA = 16;

int my_time = 0;
int cache_hits = 0, cache_queries = 0;


//read

//write

struct cache_line {
	int valid, dirty, age;
	int tag, data;

	cache_line() {
		valid = 0;
		tag = -1;
	}

	cache_line(int valid, int dirty, int age, int tag, int data) {
		this -> valid = valid;
		this -> dirty = dirty;
		this -> age = age;
		this -> tag = tag;
		this -> data = data;
	}
};

cache_line cache[32][2];

void read(int addr) {
	cache_queries++;
	int tag = (addr >> 9);
	int set = ((addr >> 4) & ((1 << 5) - 1));
	int offset = (addr & ((1 << 4) - 1));

	if (cache[set][0].tag == tag) {
		cache_hits++;
		my_time += CACHE_HIT;
```

```
57.             cache[set][0].age = 0;
58.             cache[set][1].age = 1;
59.             return;
60.       }
61.
62.       if (cache[set][1].tag == tag) {
63.             cache_hits++;
64.             my_time += CACHE_HIT;
65.             cache[set][1].age = 0;
66.             cache[set][0].age = 1;
67.             return;
68.       }
69.
70.       my_time += CACHE_MISS;
71.       int num = -1;
72.
73.       if (cache[set][0].valid == 0) {
74.             num = 0;
75.       } else if (cache[set][1].valid == 0) {
76.             num = 1;
77.       }
78.
79.       if (num == -1) {
80.             if (cache[set][0].age == 1) {
81.                   num = 0;
82.             } else {
83.                   num = 1;
84.             }
85.       }
86.
87.       if (cache[set][num].valid != 0 && cache[set][num].dirty == 1) {
88.             my_time += MEM_CTR;
89.       }
90.       my_time += MEM_CTR;
91.       cache[set][num] = cache_line(1, 0, 0, tag, 1010);
92.       cache[set][1 - num].age = 1;
93.}
94.
95.void write(int addr) {
96.       cache_queries++;
97.       int tag = (addr >> 9);
98.       int set = ((addr >> 4) & ((1 << 5) - 1));
99.       int offset = (addr & ((1 << 4) - 1));
100.
101.      if (cache[set][0].tag == tag) {
102.            cache_hits++;
103.            my_time += CACHE_HIT;
104.            cache[set][0].dirty = 1;
105.            cache[set][0].age = 0;
106.            cache[set][1].age = 1;
107.            return;
108.      }
109.
110.      if (cache[set][1].tag == tag) {
111.            cache_hits++;
112.            my_time += CACHE_HIT;
113.            cache[set][1].dirty = 1;
114.            cache[set][1].age = 0;
115.            cache[set][0].age = 1;
116.            return;
117.      }
118.
119.      my_time += CACHE_MISS;
```

```
120.      int num = -1;
121.
122.      if (cache[set][0].valid == 0) {
123.            num = 0;
124.      } else if (cache[set][1].valid == 0) {
125.            num = 1;
126.      }
127.
128.      if (num == -1) {
129.            if (cache[set][0].age == 1) {
130.                  num = 0;
131.            } else {
132.                  num = 1;
133.            }
134.      }
135.
136.      if (cache[set][num].valid != 0 && cache[set][num].dirty == 1) {
137.            my_time += MEM_CTR;
138.      }
139.      my_time += MEM_CTR;
140.      cache[set][num] = cache_line(1, 1, 0, tag, 1010);
141.      cache[set][1 - num].age = 1;
142.
143.   }
144.
145.   int main() {
146.      for (int i = 0; i < 32; i++) {
147.            for (int j = 0; j < 2; j++) {
148.                  cache[i][j] = cache_line();
149.            }
150.      }
151.
152.      int pa, pb, pc;
153.      pa = 0;
154.      my_time++;
155.      pc = 5888;
156.      my_time++;
157.      int addr = 0;
158.      my_time++;
159.      for (int y = 0; y < M; y++) {
160.            my_time++;
161.            my_time++;
162.            for (int x = 0; x < N; x++) {
163.                  my_time++;
164.                  my_time++;
165.                  pb = 2048;
166.                  my_time += 2;
167.                  for (int k = 0; k < K; k++) {
168.                        my_time++;
169.                        addr = (pa + k);
170.                        read(addr);
171.                        my_time += 2;
172.                        addr = (pb + 2 * x);
173.                        my_time++;
174.                        read(addr);
175.                        my_time += 2;
176.                        pb += (2 * N);
177.                        my_time++;
178.                        my_time += 6;
179.                  }
180.                  addr = (pc + 4 * x);
181.                  my_time++;
182.                  write(addr);
```

```
                    my_time += 32 / 16;
                }
                pc += 4 * N;
                pa += K;
                my_time += 2;
        }

        cout << "cache_hits: " << cache_hits << ", cache_queries: " << cache_queries << '\n';
        cout << "tacts: " << my_time + 1 << '\n';
        return 0;
    }
```

Verilog

testbench.sv

```
`include "cache.sv"

module mem_tb;

`define C1_READ8 3'b001
`define C1_READ16 3'b010
`define C1_READ32 3'b011
`define C1_WRITE8 3'b101
`define C1_WRITE16 3'b110
`define C1_WRITE32 3'b111
`define C1_NOP 3'b000
`define C1_INVALIDATE_LINE 3'b100
`define C1_RESPONSE 3'b111

`define C2_NOP 2'b00
`define C2_RESPONSE 2'b01
`define C2_READ_LINE 2'b10
`define C2_WRITE_LINE 2'b11

reg [14: 0] addr1_bus;
reg [15: 0] data1_bus;
reg [2: 0] command1_bus;
reg [14: 0] addr2_bus;
reg [15: 0] data2_bus;
reg [1: 0] command2_bus;

reg[1 : 0] CLK = 0;
reg [1 : 0] RESET = 0;
reg [1 : 0] C_DUMP = 0;

wire c1_bus;
wire d1_bus;
wire a1_bus;
wire c2_bus;
wire d2_bus;
wire a2_bus;

wire clk;
wire reset;
wire c_dump;
wire [31 : 0] hit;
```

```
42.wire [31 : 0] miss;
43.
44.assign a1_bus = addr1_bus;
45.assign d1_bus = data1_bus;
46.assign c1_bus = command1_bus;
47.
48.assign a2_bus = addr2_bus;
49.assign d2_bus = data2_bus;
50.assign c2_bus = command2_bus;
51.
52.assign clk = CLK;
53.assign reset = RESET;
54.assign c_dump = C_DUMP;
55.
56.reg [31 : 0] pa;
57.reg [31 : 0] pb;
58.reg [31 : 0] pc;
59.
60.reg [7 : 0] a;
61.reg [15 : 0] b;
62.reg [31 : 0] c;
63.
64.integer i = 0;
65.integer j = 0;
66.integer k = 0;
67.
68.cache _cache(
69.   clk,
70.   c_dump,
71.   reset,
72.
73.   a1_bus,
74.   d1_bus,
75.   c1_bus,
76.
77.   a2_bus,
78.   d2_bus,
79.   c2_bus,
80.
81.   hit,
82.   miss
83.);
84.
85.initial begin
86.   addr1_bus [14 : 0] = 14'bz;
87.   addr2_bus [14 : 0] =  14'bz;
88.   command1_bus [2 : 0] = `C1_NOP;
89.   command2_bus [1 : 0] = `C2_NOP;
90.   data1_bus [15 : 0] = 15'bz;
91.   data2_bus [15 : 0] = 15'bz;
92.   pa = 32'd0;
93.   pb = 32'd2048;
94.   pc = 32'd3969;
95.
96.   for (i = 0; i < 64; i++) begin
97.       for (j = 0; j < 60; j++) begin
98.           for (k = 0; k < 32; k++) begin
99.               #1;
100.                command1_bus = `C1_READ8;
101.                addr1_bus = (pa + k);
102.                #1;
103.                command1_bus = 3'bz;
104.                wait(command1_bus == `C1_RESPONSE);
```

```
                a = data1_bus;
                #1;

                #1;
                command1_bus = `C1_READ16;
                addr1_bus = (pb + 2 * j);
                #1;
                command1_bus = 3'bz;
                wait(command1_bus == `C1_RESPONSE);
                b [7 : 0] = data1_bus;
                #2;
                b [15 : 8] = data1_bus;
                #1;

                c = a * b;

            end
            #1;
            command1_bus = `C1_WRITE32;
            addr1_bus = (pc + 4 * j);
            #1;
            command1_bus = 3'bz;
            wait(command1_bus == `C1_RESPONSE);
            #1;
        end
    end

    $display("cache_hits = %d, cache_mises = %d", hit, miss);
    $display("tacts = %d", $time / 2);
    $finish;
end

always #1 begin
    CLK = ~CLK;
end

endmodule
```

cache.sv

```
1. `include "mem.sv"
2.
3. module cache(
4.    input wire CLK,
5.    input wire C_DUMP,
6.    input wire RESET,
7.
8.    input wire [14: 0] a1_bus,
9.    inout wire [15: 0] d1_bus,
10.   inout wire [2: 0] c1_bus,
11.
12.   input wire [14: 0] a2_bus,
13.   inout wire [15: 0] d2_bus,
14.   inout wire [1: 0] c2_bus,
15.
16.   output wire [31 : 0] cache_miss,
17.   output wire [31 : 0] cache_hit
18.);
```

```
19.
20.`define CACHE_WAY 2
21.
22.`define ADDR1_BUS_SIZE 15
23.`define DATA1_BUS_SIZE 16
24.`define CTR1_BUS_SIZE 3
25.
26.`define ADDR2_BUS_SIZE 15
27.`define DATA2_BUS_SIZE 16
28.`define CTR2_BUS_SIZE 2
29.
30.`define CACHE_LINE_SIZE 128
31.`define CACHE_FULL_LINE_SIZE 141
32.`define CACHE_SETS_COUNT 32
33.`define CACHE_LINE_COUNT 64
34.`define CACHE_ADDR_SIZE 19
35.`define CACHE_OFFSET_SIZE 4
36.`define CACHE_VALID 1
37.`define CACHE_DIRTY 1
38.`define CACHE_AGE 1
39.`define CACHE_TAG_SIZE 10
40.`define CACHE_LINE_SIZE 128
41.`define CACHE_SETS_SIZE 5
42.`define CACHE_WAY 2
43.
44.`define C1_READ8 3'b001
45.`define C1_READ16 3'b010
46.`define C1_READ32 3'b011
47.`define C1_WRITE8 3'b101
48.`define C1_WRITE16 3'b110
49.`define C1_WRITE32 3'b111
50.`define C1_NOP 3'b000
51.`define C1_INVALIDATE_LINE 3'b100
52.`define C1_RESPONSE 3'b111
53.
54.`define C2_NOP 2'b00
55.`define C2_RESPONSE 2'b01
56.`define C2_READ_LINE 2'b10
57.`define C2_WRITE_LINE 2'b11
58.
59.`define CACHE_MISS 4
60.`define CACHE_HIT 6
61.
62.
63.reg [`ADDR1_BUS_SIZE - 1: 0] b1_addr;
64.reg [`DATA1_BUS_SIZE - 1: 0] b1_data;
65.reg [`CTR1_BUS_SIZE - 1: 0] b1_command;
66.
67.reg [`ADDR2_BUS_SIZE - 1: 0] b2_addr;
68.reg [`DATA2_BUS_SIZE - 1: 0] b2_data;
69.reg [`CTR2_BUS_SIZE - 1: 0] b2_command;
70.
71.reg [32 : 0] miss;
72.reg [32 : 0] hit;
73.
74.integer i = 0;
75.integer j = 0;
76.integer cnt = 0;
77.integer cache_hit = 0;
78.integer shift = 0;
79.integer vnt = 0;
80.integer num_of_line = 0;
81.
```

```
82.
83.assign a1_bus = b1_addr;
84.assign d1_bus = b1_data;
85.assign c1_bus = b1_command;
86.
87.assign a2_bus = b2_addr;
88.assign d2_bus = b2_data;
89.assign c2_bus = b2_command;
90.
91.assign cache_hit = hit;
92.assign cache_miss = miss;
93.
94.reg [`CACHE_FULL_LINE_SIZE - 1 : 0] cache[0 : `CACHE_SETS_COUNT - 1][0
   : `CACHE_WAY - 1];
95.
96.reg [`DATA1_BUS_SIZE - 1 : 0] tmp_data1;
97.reg [`DATA2_BUS_SIZE - 1 : 0] tmp_data2;
98.reg [`CACHE_ADDR_SIZE - 1 : 0] tmp_addr;
99.reg [`CACHE_TAG_SIZE + `CACHE_SETS_SIZE - 1 : 0] tmp_addr2;
100.  reg [`CACHE_LINE_SIZE - 1 : 0] tmp_buffer;
101.  reg [`CACHE_SETS_SIZE : 0] tmp_set;
102.
103.  initial begin
104.     b1_addr [`ADDR1_BUS_SIZE - 1: 0] = 15'bz;
105.     b1_data [`DATA1_BUS_SIZE - 1: 0] = 16'bz;
106.     b1_command [`CTR1_BUS_SIZE - 1: 0] = 3'bz;
107.
108.     b2_addr [`ADDR2_BUS_SIZE - 1 : 0] = 15'bz;
109.     b2_data [`DATA2_BUS_SIZE - 1 : 0] = 16'bz;
110.     b2_command [`CTR2_BUS_SIZE - 1 : 0] = `C2_NOP;
111.
112.     for (i = 0; i < `CACHE_LINE_COUNT; i += 1) begin
113.         cache[i][`CACHE_FULL_LINE_SIZE - 1] = 0;
114.     end
115.  end
116.
117.  mem _mem(
118.     CLK,
119.     M_DUMP,
120.     RESET,
121.     a2_bus,
122.     d2_bus,
123.     c2_bus
124.  );
125.
126.  always @(C_DUMP) begin
127.     $dumpfile("output.txt");
128.  end
129.
130.  always @(RESET) begin
131.     b1_addr [`ADDR2_BUS_SIZE - 1: 0] = 15'bz;
132.     b1_data [`DATA2_BUS_SIZE - 1: 0] = 16'bz;
133.     b1_command [`CTR2_BUS_SIZE - 1: 0] = 3'bz;
134.
135.     b2_addr [`ADDR2_BUS_SIZE - 1 : 0] = 15'bz;
136.     b2_data [`DATA2_BUS_SIZE - 1 : 0] = 16'bz;
137.     b2_command [`CTR2_BUS_SIZE - 1 : 0] = `C2_NOP;
138.
139.     for (i = 0; i < `CACHE_LINE_COUNT; i += 1) begin
140.         cache[i][`CACHE_FULL_LINE_SIZE - 1] = 0;
141.     end
142.  end
143.
```

```
144.  always @(posedge CLK) begin
145.     case(b1_command)
146.         (`C1_READ8) : begin
147.             if (cnt == 0) begin
148.                 tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
   = a1_bus;
149.                 cnt += 1;
150.             end if (cnt == 1) begin
151.                 tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
   bug
152.                 cnt += 1;
153.             end else begin
154.                 // find cacheLine
155.                 if (cnt == 2) begin
156.                     tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
   `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
157.                     if (
158.                         cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
159.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
160.                     ) begin
161.                         cache_hit = 1;
162.                     end
163.
164.                     if (
165.                         cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
166.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
167.                     ) begin
168.                         cache_hit = 2;
169.                     end
170.                     cnt += 1;
171.                 end
172.                 // cache_hit
173.                 if (cache_hit != 0) begin
174.                     hit++;
175.                     // cache_hit += 1;
176.                     #(`CACHE_HIT - 1);
177.                     b1_command = `C1_RESPONSE;
178.                     // send answer
179.                     vnt = 7;
180.                     for (i = 0; i < 8; i++) begin
181.                         tmp_data1[vnt] = cache[tmp_set][cache_hit -
   1][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 + i];
182.                     end
183.                     b1_data = tmp_data1;
184.                     #1;
185.                     b1_command = 3'bz;
186.                 end else begin
187.                     #(`CACHE_MISS);
188.                     miss++;
189.                     // go to mem
190.                     if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1]
   == 1'b0) begin
191.                         #1;
192.                         b2_command = `C2_READ_LINE;
193.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
194.                         #1;
195.                         b2_command = 2'bz;
196.                         b2_addr = 2'bz;
```

```
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][0][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                        end else if
   (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][1][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 1;
                        end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
                            // write back if needed
                            if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
                                #1;
                                b2_command = `C2_WRITE_LINE;
                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
                                b2_addr = tmp_addr2;

                                if (cnt < 3 + 8) begin
                                    vnt = 15;
                                    for (i = 0; i < 16; i += 1) begin
                                        tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
                                        vnt -= 1;
                                        #2;
                                    end
                                    shift += 16;
                                end
                                b2_command = 2'bz;
                                #1;
                                wait(b2_command == `C2_RESPONSE);
                            end
```

```
                            shift = 0;
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][0][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 0;
                        end else begin
                            // write back if needed
                            // write back if needed
                            if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
                                #1;
                                b2_command = `C2_WRITE_LINE;
                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
                                b2_addr = tmp_addr2;

                                if (cnt < 3 + 8) begin
                                    vnt = 15;
                                    for (i = 0; i < 16; i += 1) begin
                                        tmp_data2[vnt] =
   cache[tmp_set][1][i + shift];
                                        vnt -= 1;
                                        #2;
                                    end
                                    shift += 16;
                                end
                                b2_command = 2'bz;
                                #1;
                                wait(b2_command == `C2_RESPONSE);
                            end
                            shift = 0;
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
```

```
                                    cache[tmp_set][1][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 1;
                        end
                    end
                    #1;
                    b1_command = `C1_RESPONSE;
                    vnt = 7;
                    for (i = 0; i < 8; i++) begin
                        tmp_data1[vnt] =
   cache[tmp_set][num_of_line][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 +
   i];
                    end
                    b1_data = tmp_data1;
                    #1;
                    b1_command = 3'bz;
                    // send cpu
                end
                cnt = 0;
            end

            (`C1_READ16) : begin //must-have
                // find cache line
                // if yes print
                // if no go to mem
                // if have empty space cache
                // else delete old
                // if old modified write back
                // set new one line
                if (cnt == 0) begin
                    tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
   = a1_bus;
                    cnt += 1;
                end if (cnt == 1) begin
                    tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
   bug
                    cnt += 1;
                end else begin
                    // find cacheLine
                    if (cnt == 2) begin
                        tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
   `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
                        if (
                            cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
                            tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
                        ) begin
                            cache_hit = 1;
                        end

                        if (
                            cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
                            tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
                        ) begin
                            cache_hit = 2;
```

```
                        end
                        cnt += 1;
                    end
                    // cache_hit
                    if (cache_hit != 0) begin
                        // cache_hit += 1;
                        #(`CACHE_HIT - 1);
                        hit++;
                        b1_command = `C1_RESPONSE;
                        // send answer
                        vnt = 15;
                        for (i = 0; i < 16; i++) begin
                            tmp_data1[vnt] = cache[tmp_set][cache_hit - 1][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 + i];
                        end
                        b1_data = tmp_data1;
                        #1;
                        b1_command = 3'bz;
                    end else begin
                        #(`CACHE_MISS);
                        miss++;
                        // go to mem
                        if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][0][shift + i] = tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                        end else if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][1][shift + i] = tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
```

```
412.                            end
413.                            num_of_line = 1;
414.                        end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
415.                            // write back if needed
416.                            if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
417.                                #1;
418.                                b2_command = `C2_WRITE_LINE;
419.                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
420.                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
421.                                b2_addr = tmp_addr2;
422.
423.                                if (cnt < 3 + 8) begin
424.                                    vnt = 15;
425.                                    for (i = 0; i < 16; i += 1) begin
426.                                        tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
427.                                        vnt -= 1;
428.                                        #2;
429.                                    end
430.                                    shift += 16;
431.                                end
432.                                b2_command = 2'bz;
433.                                #1;
434.                                wait(b2_command == `C2_RESPONSE);
435.                            end
436.                            shift = 0;
437.                            #1;
438.                            b2_command = `C2_READ_LINE;
439.                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
440.                            #1;
441.                            b2_command = 2'bz;
442.                            b2_addr = 2'bz;
443.                            wait(b2_command == `C2_RESPONSE);
444.                            if (cnt < 3 + 8 + 8) begin
445.                                tmp_data2 = b2_data;
446.                                j = 15;
447.                                for (i = 0; i < 16; i++) begin
448.                                    cache[tmp_set][0][shift + i] =
   tmp_data2[j];
449.                                    j -= 1;
450.                                end
451.                                shift += 16;
452.                                cnt += 1;
453.                            end
454.                            num_of_line = 0;
455.                        end else begin
456.                            // write back if needed
457.                            // write back if needed
458.                            if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
459.                                #1;
460.                                b2_command = `C2_WRITE_LINE;
461.                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
```

```
462.                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
463.                                b2_addr = tmp_addr2;
464.
465.                                if (cnt < 3 + 8) begin
466.                                    vnt = 15;
467.                                    for (i = 0; i < 16; i += 1) begin
468.                                        tmp_data2[vnt] =
   cache[tmp_set][1][i + shift];
469.                                        vnt -= 1;
470.                                        #2;
471.                                    end
472.                                    shift += 16;
473.                                end
474.                                b2_command = 2'bz;
475.                                #1;
476.                                wait(b2_command == `C2_RESPONSE);
477.                            end
478.                            shift = 0;
479.                            #1;
480.                            b2_command = `C2_READ_LINE;
481.                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
482.                            #1;
483.                            b2_command = 2'bz;
484.                            b2_addr = 2'bz;
485.                            wait(b2_command == `C2_RESPONSE);
486.                            if (cnt < 3 + 8 + 8) begin
487.                                tmp_data2 = b2_data;
488.                                j = 15;
489.                                for (i = 0; i < 16; i++) begin
490.                                    cache[tmp_set][1][shift + i] =
   tmp_data2[j];
491.                                    j -= 1;
492.                                end
493.                                shift += 16;
494.                                cnt += 1;
495.                            end
496.                            num_of_line = 1;
497.                        end
498.                    end
499.                    #1;
500.                    b1_command = `C1_RESPONSE;
501.                    vnt = 15;
502.                    for (i = 0; i < 16; i++) begin
503.                        tmp_data1[vnt] =
   cache[tmp_set][num_of_line][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 +
   i];
504.                    end
505.                    b1_data = tmp_data1;
506.                    #1;
507.                    b1_command = 3'bz;
508.                    // send cpu
509.                end
510.                cnt = 0;
511.            end
512.
513.            (`C1_READ32) : begin
514.                if (cnt == 0) begin
515.                    tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
   = a1_bus;
516.                    cnt += 1;
517.                end if (cnt == 1) begin
```

```
518.                 tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
   bug
519.                 cnt += 1;
520.             end else begin
521.                 // find cacheLine
522.                 if (cnt == 2) begin
523.                     tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
   `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
524.                     if (
525.                         cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
526.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
527.                     ) begin
528.                         cache_hit = 1;
529.                     end
530.
531.                     if (
532.                         cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
533.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
534.                     ) begin
535.                         cache_hit = 2;
536.                     end
537.                     cnt += 1;
538.                 end
539.                 // cache_hit
540.                 if (cache_hit != 0) begin
541.                     // cache_hit += 1;
542.                     #(`CACHE_HIT - 1);
543.                     hit++;
544.                     b1_command = `C1_RESPONSE;
545.                     // send answer
546.                     vnt = 15;
547.                     for (i = 0; i < 16; i++) begin
548.                         tmp_data1[vnt] = cache[tmp_set][cache_hit -
   1][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 + i];
549.                     end
550.                     #2
551.                     b1_data = tmp_data1;
552.                     vnt = 15;
553.                     for (i = 0; i < 16; i++) begin
554.                         tmp_data1[vnt] = cache[tmp_set][cache_hit -
   1][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 + i];
555.                     end
556.                     b1_data = tmp_data1;
557.                     #1;
558.                     b1_command = 3'bz;
559.                 end else begin
560.                     #(`CACHE_MISS);
561.                     miss++;
562.                     // go to mem
563.                     if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1]
   == 1'b0) begin
564.                         #1;
565.                         b2_command = `C2_READ_LINE;
566.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
567.                         #1;
568.                         b2_command = 2'bz;
569.                         b2_addr = 2'bz;
570.                         wait(b2_command == `C2_RESPONSE);
```

```
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][0][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                        end else if
   (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][1][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 1;
                        end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
                            // write back if needed
                            if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
                                #1;
                                b2_command = `C2_WRITE_LINE;
                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
                                b2_addr = tmp_addr2;

                                if (cnt < 3 + 8) begin
                                    vnt = 15;
                                    for (i = 0; i < 16; i += 1) begin
                                        tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
                                        vnt -= 1;
                                        #2;
                                    end
                                    shift += 16;
                                end
                                b2_command = 2'bz;
                                #1;
                                wait(b2_command == `C2_RESPONSE);
                            end
                            shift = 0;
```

```
623.                            #1;
624.                            b2_command = `C2_READ_LINE;
625.                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE];
626.                            #1;
627.                            b2_command = 2'bz;
628.                            b2_addr = 2'bz;
629.                            wait(b2_command == `C2_RESPONSE);
630.                            if (cnt < 3 + 8 + 8) begin
631.                                tmp_data2 = b2_data;
632.                                j = 15;
633.                                for (i = 0; i < 16; i++) begin
634.                                    cache[tmp_set][0][shift + i] =
    tmp_data2[j];
635.                                    j -= 1;
636.                                end
637.                                shift += 16;
638.                                cnt += 1;
639.                            end
640.                            num_of_line = 0;
641.                        end else begin
642.                            // write back if needed
643.                            // write back if needed
644.                            if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
    - 2] == 1'b1) begin
645.                                #1;
646.                                b2_command = `C2_WRITE_LINE;
647.                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
    cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
    `CACHE_LINE_SIZE];
648.                                tmp_addr2[`CACHE_OFFSET_SIZE +
    `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
649.                                b2_addr = tmp_addr2;
650.
651.                                if (cnt < 3 + 8) begin
652.                                    vnt = 15;
653.                                    for (i = 0; i < 16; i += 1) begin
654.                                        tmp_data2[vnt] =
    cache[tmp_set][1][i + shift];
655.                                        vnt -= 1;
656.                                        #2;
657.                                    end
658.                                    shift += 16;
659.                                end
660.                                b2_command = 2'bz;
661.                                #1;
662.                                wait(b2_command == `C2_RESPONSE);
663.                            end
664.                            shift = 0;
665.                            #1;
666.                            b2_command = `C2_READ_LINE;
667.                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE];
668.                            #1;
669.                            b2_command = 2'bz;
670.                            b2_addr = 2'bz;
671.                            wait(b2_command == `C2_RESPONSE);
672.                            if (cnt < 3 + 8 + 8) begin
673.                                tmp_data2 = b2_data;
674.                                j = 15;
675.                                for (i = 0; i < 16; i++) begin
676.                                    cache[tmp_set][1][shift + i] =
```

```
   tmp_data2[j];
677.                                j -= 1;
678.                            end
679.                            shift += 16;
680.                            cnt += 1;
681.                        end
682.                        num_of_line = 1;
683.                    end
684.                end
685.                #1;
686.                b1_command = `C1_RESPONSE;
687.                vnt = 15;
688.                for (i = 0; i < 16; i++) begin
689.                    tmp_data1[vnt] =
   cache[tmp_set][num_of_line][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 +
   i];
690.                end
691.                #2
692.                b1_data = tmp_data1;
693.                vnt = 15;
694.                for (i = 0; i < 16; i++) begin
695.                    tmp_data1[vnt] =
   cache[tmp_set][num_of_line][tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] * 8 +
   i];
696.                end
697.                b1_data = tmp_data1;
698.                #1;
699.                b1_command = 3'bz;
700.            end
701.            cnt = 0;
702.        end
703.
704.        (`C1_WRITE8) : begin
705.            if (cnt == 0) begin
706.                tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
   = a1_bus;
707.                tmp_data1 = b1_data;
708.                j = 0;
709.                shift = 0;
710.                for (i = 15; i >= 0; i -= 1) begin
711.                    tmp_buffer[shift + j] = tmp_data1[i];
712.                    j += 1;
713.                end
714.                shift += 16;
715.                cnt += 1;
716.            end if (cnt == 1) begin
717.                tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
   bug
718.                cnt += 1;
719.            end else begin
720.                // find cacheLine
721.                if (cnt == 2) begin
722.                    tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
   `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
723.                    if (
724.                        cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
725.                        tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
726.                    ) begin
727.                        cache_hit = 1;
728.                    end
729.
```

```
730.                     if (
731.                         cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
    : `CACHE_LINE_SIZE] ==
732.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
    - 1: `CACHE_OFFSET_SIZE]
733.                     ) begin
734.                         cache_hit = 2;
735.                     end
736.                     cnt += 1;
737.                 end
738.                 // cache_hit
739.                 if (cache_hit != 0) begin
740.                     // cache_hit += 1;
741.                     #(`CACHE_HIT - 1);
742.                     hit++;
743.                     shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
744.                     j = `CACHE_LINE_SIZE - 1;
745.                     for (vnt = 0; vnt < 2; vnt += 1) begin
746.                         for (i = 0; i < 16; i += 1) begin
747.                             cache[tmp_set][cache_hit - 1][shift + i]
    = tmp_buffer[j];
748.                             j -= 1;
749.                         end
750.                         #2;
751.                         shift -= 16;
752.                     end
753.                     cnt += 1;
754.                     #1;
755.                     b1_command = `C1_RESPONSE;
756.                     #1;
757.                     b1_command = 3'bz;
758.                 end else begin
759.                     #(`CACHE_MISS);
760.                     // go to mem
761.                     miss++;
762.                     if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1]
    == 1'b0) begin
763.                         #1;
764.                         b2_command = `C2_READ_LINE;
765.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE];
766.                         #1;
767.                         b2_command = 2'bz;
768.                         b2_addr = 2'bz;
769.                         wait(b2_command == `C2_RESPONSE);
770.                         if (cnt < 3 + 8) begin
771.                             tmp_data2 = b2_data;
772.                             j = 15;
773.                             for (i = 0; i < 16; i++) begin
774.                                 cache[tmp_set][0][shift + i] =
    tmp_data2[j];
775.                                 j -= 1;
776.                             end
777.                             shift += 16;
778.                             cnt += 1;
779.                         end
780.                     end else if
    (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
781.                         #1;
782.                         b2_command = `C2_READ_LINE;
783.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE];
784.                         #1;
```

```
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][1][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 1;
                        end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
                            // write back if needed
                            if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
                                #1;
                                b2_command = `C2_WRITE_LINE;
                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
                                tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
                                b2_addr = tmp_addr2;

                                if (cnt < 3 + 8) begin
                                    vnt = 15;
                                    for (i = 0; i < 16; i += 1) begin
                                        tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
                                        vnt -= 1;
                                        #2;
                                    end
                                    shift += 16;
                                end
                                b2_command = 2'bz;
                                #1;
                                wait(b2_command == `C2_RESPONSE);
                            end
                            shift = 0;
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][0][shift + i] =
   tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
```

```
                            end
                            num_of_line = 0;
                        end else begin
                            // write back if needed
                            // write back if needed
                            if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
    - 2] == 1'b1) begin
                                #1;
                                b2_command = `C2_WRITE_LINE;
                                tmp_addr2[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
    cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
    `CACHE_LINE_SIZE];
                                tmp_addr2[`CACHE_OFFSET_SIZE +
    `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
                                b2_addr = tmp_addr2;

                                if (cnt < 3 + 8) begin
                                    vnt = 15;
                                    for (i = 0; i < 16; i += 1) begin
                                        tmp_data2[vnt] =
    cache[tmp_set][1][i + shift];
                                        vnt -= 1;
                                        #2;
                                    end
                                    shift += 16;
                                end
                                b2_command = 2'bz;
                                #1;
                                wait(b2_command == `C2_RESPONSE);
                            end
                            shift = 0;
                            #1;
                            b2_command = `C2_READ_LINE;
                            b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
    `CACHE_OFFSET_SIZE];
                            #1;
                            b2_command = 2'bz;
                            b2_addr = 2'bz;
                            wait(b2_command == `C2_RESPONSE);
                            if (cnt < 3 + 8 + 8) begin
                                tmp_data2 = b2_data;
                                j = 15;
                                for (i = 0; i < 16; i++) begin
                                    cache[tmp_set][1][shift + i] =
    tmp_data2[j];
                                    j -= 1;
                                end
                                shift += 16;
                                cnt += 1;
                            end
                            num_of_line = 1;
                        end
                    end

                    shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
                    j = `CACHE_LINE_SIZE - 1;
                    for (vnt = 0; vnt < 1; vnt += 1) begin
                        for (i = 0; i < 16; i += 1) begin
                            cache[tmp_set][num_of_line][shift + i] =
    tmp_buffer[j];
                            j -= 1;
                            #2;
```

```
892.                     end
893.                     shift -= 16;
894.                 end
895.                 #1;
896.                 b1_command = `C1_RESPONSE;
897.                 #1;
898.                 b1_command = 3'bz;
899.             end
900.             cnt = 0;
901.         end
902.
903.         (`C1_WRITE16) : begin
904.             if (cnt == 0) begin
905.                 tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
    = a1_bus;
906.                 tmp_data1 = b1_data;
907.                 j = 0;
908.                 shift = 0;
909.                 for (i = 15; i >= 0; i -= 1) begin
910.                     tmp_buffer[shift + j] = tmp_data1[i];
911.                     j += 1;
912.                 end
913.                 shift += 16;
914.                 cnt += 1;
915.             end if (cnt == 1) begin
916.                 tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
    bug
917.                 cnt += 1;
918.             end else begin
919.                 // find cacheLine
920.                 if (cnt == 2) begin
921.                     tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
    `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
922.                     if (
923.                         cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
    : `CACHE_LINE_SIZE] ==
924.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
    - 1: `CACHE_OFFSET_SIZE]
925.                     ) begin
926.                         cache_hit = 1;
927.                     end
928.
929.                     if (
930.                         cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
    : `CACHE_LINE_SIZE] ==
931.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
    - 1: `CACHE_OFFSET_SIZE]
932.                     ) begin
933.                         cache_hit = 2;
934.                     end
935.                     cnt += 1;
936.                 end
937.                 // cache_hit
938.                 hit++;
939.                 if (cache_hit != 0) begin
940.                     // cache_hit += 1;
941.                     #(`CACHE_HIT - 1);
942.
943.                     shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
944.                     j = `CACHE_LINE_SIZE - 1;
945.                     for (vnt = 0; vnt < 2; vnt += 1) begin
946.                         for (i = 0; i < 16; i += 1) begin
947.                             cache[tmp_set][cache_hit - 1][shift + i]
```

```
   = tmp_buffer[j];
948.                            j -= 1;
949.                        end
950.                        #2;
951.                        shift -= 16;
952.                    end
953.                    cnt += 1;
954.                    #1;
955.                    b1_command = `C1_RESPONSE;
956.                    #1;
957.                    b1_command = 3'bz;
958.                end else begin
959.                    #(`CACHE_MISS);
960.                    // go to mem
961.                    miss++;
962.                    if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1]
   == 1'b0) begin
963.                        #1;
964.                        b2_command = `C2_READ_LINE;
965.                        b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
966.                        #1;
967.                        b2_command = 2'bz;
968.                        b2_addr = 2'bz;
969.                        wait(b2_command == `C2_RESPONSE);
970.                        if (cnt < 3 + 8) begin
971.                            tmp_data2 = b2_data;
972.                            j = 15;
973.                            for (i = 0; i < 16; i++) begin
974.                                cache[tmp_set][0][shift + i] =
   tmp_data2[j];
975.                                j -= 1;
976.                            end
977.                            shift += 16;
978.                            cnt += 1;
979.                        end
980.                    end else if
   (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
981.                        #1;
982.                        b2_command = `C2_READ_LINE;
983.                        b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
984.                        #1;
985.                        b2_command = 2'bz;
986.                        b2_addr = 2'bz;
987.                        wait(b2_command == `C2_RESPONSE);
988.                        if (cnt < 3 + 8) begin
989.                            tmp_data2 = b2_data;
990.                            j = 15;
991.                            for (i = 0; i < 16; i++) begin
992.                                cache[tmp_set][1][shift + i] =
   tmp_data2[j];
993.                                j -= 1;
994.                            end
995.                            shift += 16;
996.                            cnt += 1;
997.                        end
998.                        num_of_line = 1;
999.                    end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
1000.                       // write back if needed
1001.                       if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
```

```
1002.                               #1;
1003.                               b2_command = `C2_WRITE_LINE;
1004.                               tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
1005.                               tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
1006.                               b2_addr = tmp_addr2;
1007.
1008.                               if (cnt < 3 + 8) begin
1009.                                   vnt = 15;
1010.                                   for (i = 0; i < 16; i += 1) begin
1011.                                       tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
1012.                                       vnt -= 1;
1013.                                       #2;
1014.                                   end
1015.                                   shift += 16;
1016.                               end
1017.                               b2_command = 2'bz;
1018.                               #1;
1019.                               wait(b2_command == `C2_RESPONSE);
1020.                           end
1021.                           shift = 0;
1022.                           #1;
1023.                           b2_command = `C2_READ_LINE;
1024.                           b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1025.                           #1;
1026.                           b2_command = 2'bz;
1027.                           b2_addr = 2'bz;
1028.                           wait(b2_command == `C2_RESPONSE);
1029.                           if (cnt < 3 + 8 + 8) begin
1030.                               tmp_data2 = b2_data;
1031.                               j = 15;
1032.                               for (i = 0; i < 16; i++) begin
1033.                                   cache[tmp_set][0][shift + i] =
   tmp_data2[j];
1034.                                   j -= 1;
1035.                               end
1036.                               shift += 16;
1037.                               cnt += 1;
1038.                           end
1039.                           num_of_line = 0;
1040.                       end else begin
1041.                           // write back if needed
1042.                           // write back if needed
1043.                           if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
1044.                               #1;
1045.                               b2_command = `C2_WRITE_LINE;
1046.                               tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
1047.                               tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
1048.                               b2_addr = tmp_addr2;
1049.
1050.                               if (cnt < 3 + 8) begin
1051.                                   vnt = 15;
1052.                                   for (i = 0; i < 16; i += 1) begin
```

```
1053.                                       tmp_data2[vnt] =
   cache[tmp_set][1][i + shift];
1054.                                       vnt -= 1;
1055.                                   end
1056.                                   #2;
1057.                                   shift += 16;
1058.                               end
1059.                               b2_command = 2'bz;
1060.                               #1;
1061.                               wait(b2_command == `C2_RESPONSE);
1062.                           end
1063.                           shift = 0;
1064.                           #1;
1065.                           b2_command = `C2_READ_LINE;
1066.                           b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1067.                           #1;
1068.                           b2_command = 2'bz;
1069.                           b2_addr = 2'bz;
1070.                           wait(b2_command == `C2_RESPONSE);
1071.                           if (cnt < 3 + 8 + 8) begin
1072.                               tmp_data2 = b2_data;
1073.                               j = 15;
1074.                               for (i = 0; i < 16; i++) begin
1075.                                   cache[tmp_set][1][shift + i] =
   tmp_data2[j];
1076.                                   j -= 1;
1077.                               end
1078.                               shift += 16;
1079.                               cnt += 1;
1080.                           end
1081.                           num_of_line = 1;
1082.                       end
1083.                   end
1084.
1085.                   shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
1086.                   j = `CACHE_LINE_SIZE - 1;
1087.                   for (vnt = 0; vnt < 2; vnt += 1) begin
1088.                       for (i = 0; i < 16; i += 1) begin
1089.                           cache[tmp_set][num_of_line][shift + i] =
   tmp_buffer[j];
1090.                           j -= 1;
1091.                           #2;
1092.                       end
1093.                       shift -= 16;
1094.                   end
1095.                   #1;
1096.                   b1_command = `C1_RESPONSE;
1097.                   #1;
1098.                   b1_command = 3'bz;
1099.               end
1100.               cnt = 0;
1101.           end
1102.
1103.           (`C1_WRITE32) : begin //must-have
1104.               if (cnt == 0) begin
1105.                   tmp_addr[`CACHE_ADDR_SIZE - 1 : `CACHE_OFFSET_SIZE]
   = a1_bus;
1106.                   tmp_data1 = b1_data;
1107.                   j = 0;
1108.                   shift = 0;
1109.                   for (i = 15; i >= 0; i -= 1) begin
1110.                       tmp_buffer[shift + j] = tmp_data1[i];
```

```
1111.                     j += 1;
1112.                 end
1113.                 shift += 16;
1114.                 cnt += 1;
1115.             end if (cnt == 1) begin
1116.                 tmp_addr[`CACHE_OFFSET_SIZE - 1 : 0] = a1_bus; // mb
   bug
1117.                 tmp_data1 = b1_data;
1118.                 j = 0;
1119.                 for (i = 15; i >= 0; i -= 1) begin
1120.                     tmp_buffer[shift + j] = tmp_data1[i];
1121.                     j += 1;
1122.                 end
1123.                 shift += 16;
1124.                 cnt += 1;
1125.             end else begin
1126.                 // find cacheLine
1127.                 if (cnt == 2) begin
1128.                     tmp_set = tmp_addr[`CACHE_ADDR_SIZE - 1 -
   `CACHE_TAG_SIZE : `CACHE_OFFSET_SIZE];
1129.                     if (
1130.                         cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
1131.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
1132.                     ) begin
1133.                         cache_hit = 1;
1134.                     end
1135.
1136.                     if (
1137.                         cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 4
   : `CACHE_LINE_SIZE] ==
1138.                         tmp_addr[`CACHE_ADDR_SIZE - `CACHE_TAG_SIZE
   - 1: `CACHE_OFFSET_SIZE]
1139.                     ) begin
1140.                         cache_hit = 2;
1141.                     end
1142.                     cnt += 1;
1143.                 end
1144.                 // cache_hit
1145.                 if (cache_hit != 0) begin
1146.                     // cache_hit += 1;
1147.                     hit++;
1148.                     #(`CACHE_HIT - 1);
1149.
1150.                     shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
1151.                     j = `CACHE_LINE_SIZE - 1;
1152.                     for (vnt = 0; vnt < 4; vnt += 1) begin
1153.                         for (i = 0; i < 16; i += 1) begin
1154.                             cache[tmp_set][cache_hit - 1][shift + i]
   = tmp_buffer[j];
1155.                             j -= 1;
1156.                             #2;
1157.                         end
1158.                         shift -= 16;
1159.                     end
1160.                     cnt += 1;
1161.                     #1;
1162.                     b1_command = `C1_RESPONSE;
1163.                     #1;
1164.                     b1_command = 3'bz;
1165.                 end else begin
1166.                     #(`CACHE_MISS);
```

```
1167.                     // go to mem
1168.                     miss++;
1169.                     if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 1]
   == 1'b0) begin
1170.                         #1;
1171.                         b2_command = `C2_READ_LINE;
1172.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1173.                         #1;
1174.                         b2_command = 2'bz;
1175.                         b2_addr = 2'bz;
1176.                         wait(b2_command == `C2_RESPONSE);
1177.                         if (cnt < 3 + 8) begin
1178.                             tmp_data2 = b2_data;
1179.                             j = 15;
1180.                             for (i = 0; i < 16; i++) begin
1181.                                 cache[tmp_set][0][shift + i] =
   tmp_data2[j];
1182.                                 j -= 1;
1183.                             end
1184.                             shift += 16;
1185.                             cnt += 1;
1186.                         end
1187.                     end else if
   (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE - 1] == 1'b0) begin
1188.                         #1;
1189.                         b2_command = `C2_READ_LINE;
1190.                         b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1191.                         #1;
1192.                         b2_command = 2'bz;
1193.                         b2_addr = 2'bz;
1194.                         wait(b2_command == `C2_RESPONSE);
1195.                         if (cnt < 3 + 8) begin
1196.                             tmp_data2 = b2_data;
1197.                             j = 15;
1198.                             for (i = 0; i < 16; i++) begin
1199.                                 cache[tmp_set][1][shift + i] =
   tmp_data2[j];
1200.                                 j -= 1;
1201.                             end
1202.                             shift += 16;
1203.                             cnt += 1;
1204.                         end
1205.                         num_of_line = 1;
1206.                     end else if
   (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE - 3] == 1'b1) begin
1207.                         // write back if needed
1208.                         if (cache[tmp_set][0][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
1209.                             #1;
1210.                             b2_command = `C2_WRITE_LINE;
1211.                             tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
1212.                             tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
1213.                             b2_addr = tmp_addr2;
1214.
1215.                             if (cnt < 3 + 8) begin
1216.                                 vnt = 15;
1217.                                 for (i = 0; i < 16; i += 1) begin
```

```
1218.                                       tmp_data2[vnt] =
   cache[tmp_set][0][i + shift];
1219.                                       vnt -= 1;
1220.                                       #2;
1221.                                   end
1222.                                   shift += 16;
1223.                               end
1224.                               b2_command = 2'bz;
1225.                               #1;
1226.                               wait(b2_command == `C2_RESPONSE);
1227.                           end
1228.                           shift = 0;
1229.                           #1;
1230.                           b2_command = `C2_READ_LINE;
1231.                           b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1232.                           #1;
1233.                           b2_command = 2'bz;
1234.                           b2_addr = 2'bz;
1235.                           wait(b2_command == `C2_RESPONSE);
1236.                           if (cnt < 3 + 8 + 8) begin
1237.                               tmp_data2 = b2_data;
1238.                               j = 15;
1239.                               for (i = 0; i < 16; i++) begin
1240.                                   cache[tmp_set][0][shift + i] =
   tmp_data2[j];
1241.                                   j -= 1;
1242.                               end
1243.                               shift += 16;
1244.                               cnt += 1;
1245.                           end
1246.                           num_of_line = 0;
1247.                       end else begin
1248.                           // write back if needed
1249.                           // write back if needed
1250.                           if (cache[tmp_set][1][`CACHE_FULL_LINE_SIZE
   - 2] == 1'b1) begin
1251.                               #1;
1252.                               b2_command = `C2_WRITE_LINE;
1253.                               tmp_addr2[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE + `CACHE_SETS_SIZE] =
   cache[tmp_set][0][`CACHE_LINE_SIZE + `CACHE_TAG_SIZE - 1 :
   `CACHE_LINE_SIZE];
1254.                               tmp_addr2[`CACHE_OFFSET_SIZE +
   `CACHE_SETS_SIZE - 1 : `CACHE_OFFSET_SIZE] = tmp_set;
1255.                               b2_addr = tmp_addr2;
1256.
1257.                               if (cnt < 3 + 8) begin
1258.                                   vnt = 15;
1259.                                   for (i = 0; i < 16; i += 1) begin
1260.                                       tmp_data2[vnt] =
   cache[tmp_set][1][i + shift];
1261.                                       vnt -= 1;
1262.                                       #2;
1263.                                   end
1264.                                   shift += 16;
1265.                               end
1266.                               b2_command = 2'bz;
1267.                               #1;
1268.                               wait(b2_command == `C2_RESPONSE);
1269.                           end
1270.                           shift = 0;
1271.                           #1;
```

```
1272.                           b2_command = `C2_READ_LINE;
1273.                           b2_addr = tmp_addr[`CACHE_ADDR_SIZE - 1 :
   `CACHE_OFFSET_SIZE];
1274.                           #1;
1275.                           b2_command = 2'bz;
1276.                           b2_addr = 2'bz;
1277.                           wait(b2_command == `C2_RESPONSE);
1278.                           if (cnt < 3 + 8 + 8) begin
1279.                               tmp_data2 = b2_data;
1280.                               j = 15;
1281.                               for (i = 0; i < 16; i++) begin
1282.                                   cache[tmp_set][1][shift + i] =
   tmp_data2[j];
1283.                                   j -= 1;
1284.                               end
1285.                               shift += 16;
1286.                               cnt += 1;
1287.                           end
1288.                           num_of_line = 1;
1289.                       end
1290.                   end
1291.
1292.                   shift = tmp_addr[`CACHE_OFFSET_SIZE : 0];
1293.                   j = `CACHE_LINE_SIZE - 1;
1294.                   for (vnt = 0; vnt < 4; vnt += 1) begin
1295.                       for (i = 0; i < 16; i += 1) begin
1296.                           cache[tmp_set][num_of_line][shift + i] =
   tmp_buffer[j];
1297.                           j -= 1;
1298.                           #2;
1299.                       end
1300.                       shift -= 16;
1301.                   end
1302.                   #1;
1303.                   b1_command = `C1_RESPONSE;
1304.                   #1;
1305.                   b1_command = 3'bz;
1306.               end
1307.               cnt = 0;
1308.           end
1309.       endcase
1310.   end
1311.
1312.   endmodule
```

mem.sv

```
1. module mem(
2.     input wire CLK,
3.     input wire M_DUMP,
4.     input wire RESET,
5.
6.     input wire [14: 0] a2_bus,
7.     inout wire [15: 0] d2_bus,
8.     inout wire [1: 0] c2_bus
9.     );
10.
11.`define ADDR2_BUS_SIZE 15
```

```
12.`define DATA2_BUS_SIZE 16
13.`define CTR2_BUS_SIZE 2
14.`define CACHE_TAG_SIZE 10
15.`define CACHE_SETS_SIZE 5
16.`define CACHE_ADDR_SIZE 19
17.`define CACHE_LINE_SIZE 128
18.`define CACHE_OFFSET_SIZE 4
19.`define MEM_SIZE 524288
20.
21.`define MEM_WORKING_TIME 100
22.
23.`define C2_NOP 2'b00
24.`define C2_READ_LINE 2'b10
25.`define C2_WRITE_LINE 2'b11
26.`define C2_RESPONSE 2'b01
27.
28.reg [`ADDR2_BUS_SIZE - 1: 0] addr;
29.reg [`DATA2_BUS_SIZE - 1: 0] data;
30.reg [`CTR2_BUS_SIZE - 1: 0] command;
31.
32.assign a2_bus = addr;
33.assign d2_bus = data;
34.assign c2_bus = command;
35.
36.integer SEED = 225526;
37.reg[7:0] a[0:99];
38.reg [7 : 0] memory[0 : 524287];
39.integer i = 0;
40.integer k = 0;
41.integer j = 0;
42.integer cnt = 0;
43.reg [(`CACHE_TAG_SIZE + `CACHE_SETS_SIZE - 1) + 4 : 0] tmp_addr;
44.reg [`DATA2_BUS_SIZE - 1 : 0] tmp_data;
45.reg [`CACHE_OFFSET_SIZE - 1 : 0] offset = 4'b0000;
46.reg [7 : 0] new_a;
47.
48.initial begin
49.    addr[`ADDR2_BUS_SIZE - 1: 0] = 15'bz;
50.    data[`DATA2_BUS_SIZE - 1: 0] = 16'bz;
51.    command[`CTR2_BUS_SIZE - 1: 0] = 2'bz;
52.
53.    for (i = 0; i < `MEM_SIZE; i += 1) begin
54.        memory[i] = $random(SEED)>>16;
55.    end
56.
57.    /*for (i = 0; i < 100; i += 1) begin
58.        $display("[%d] %d", i, a[i]);
59.    end*/
60.end
61.
62.always @(RESET) begin
63.    addr[`ADDR2_BUS_SIZE - 1: 0] = 15'bz;
64.    data[`DATA2_BUS_SIZE - 1: 0] = 16'bz;
65.    command[`CTR2_BUS_SIZE - 1: 0] = 2'bz;
66.
67.    for (i = 0; i < `MEM_SIZE; i += 1) begin
68.        memory[i] = $random(SEED)>>16;
69.    end
70.
71.    /*for (i = 0; i < 100; i += 1) begin
72.        $display("[%d] %d", i, a[i]);
73.    end*/
74.end
```

```
75.
76.always @(M_DUMP) begin
77.   $dumpfile("output.txt");
78.end
79.
80.
81.always @(posedge CLK) begin
82.   case(command)
83.       `C2_READ_LINE : begin //C2_READ_LINE
84.           tmp_addr[(`CACHE_TAG_SIZE + `CACHE_SETS_SIZE - 1)] =
   a2_bus;
85.           offset = 4'b1111;
86.           #(`MEM_WORKING_TIME * 2 - (`CACHE_LINE_SIZE /
   `DATA2_BUS_SIZE) * 2 - 1);
87.           command = `C2_RESPONSE;
88.           for (i = 0; i < (`CACHE_LINE_SIZE / `DATA2_BUS_SIZE); i +=
   1) begin
89.               new_a = memory[(tmp_addr << 4) + offset];
90.               offset -= 1;
91.               data[15 : 8] = new_a;
92.               new_a = memory[(tmp_addr << 4) + offset];
93.               offset -= 1;
94.               data[7 : 0] = new_a;
95.               #2;
96.           end
97.           #1;
98.           command = 2'bz;
99.       end
100.
101.          `C2_WRITE_LINE : begin //C2_WRITE_LINE
102.              if (cnt == 0) begin
103.                  tmp_addr[(`CACHE_TAG_SIZE + `CACHE_SETS_SIZE - 1)] =
   a2_bus;
104.                  offset = 4'b1111;
105.              end
106.              if (cnt < 8) begin
107.                  tmp_data = d2_bus;
108.                  memory[(tmp_addr << 4) + offset] = tmp_data[15 : 8];
109.                  offset = offset - 4'b0001;
110.                  memory[(tmp_addr << 4) + offset] = tmp_data[7 : 0];
111.                  offset = offset - 4'b0001;
112.                  cnt += 1;
113.              end
114.              if (cnt == 7) begin
115.                  cnt = 0;
116.                  #(`MEM_WORKING_TIME * 2 - (`CACHE_LINE_SIZE /
   `DATA2_BUS_SIZE) * 2 - 1);
117.                  command = `C2_RESPONSE;
118.                  #1;
119.                  command = 2'bz;
120.              end
121.          end
122.
123.     endcase
124.  end
125.  endmodule
```