Лабораторная работа №2	M3138	2023
Моделирование схем в Verilog	Попович Виталий Сергеевич	

Цель работы

Построение кэша и моделирование системы “процессор-кэш-память” на языке описания Verilog

Инструментарий и требования к работе

Компиляция и симуляция – Icarus Verilog 10.1

Аналитическое решение - Python 3.10.4

Описание

Требуется построить и смоделировать систему “процессор-кэш-память”. Схема работы кэша – “**look-through write-back”**

Описание работы Кэша

Look-through: процессор делает запрос к памяти: если данные есть в кэше, то берётся из кэша, иначе данные записываются в кэш из памяти, после чего процессор берёт данные из кэша.

Write-back: процессор записывает данные в кэш(нужную кэш-линию).

Если же произойдет “кэш промах”, то в кэше сохранится нужная кэш-линия, на её же место и будут записаны данные. Данные запишутся в память, после “вытеснения” этой кэш-линии.

Стоит отметить, что для каждой кэш-линии соответствует набор из несколько линий(некоторый сет).С помощью этого в кэше можно хранить

несколько линий параллельно(в зависимости от ассоциативности кэша(кол-во линий в одном сете))

Least Recently Used(LRU) – метод вытеснения из кэша:
Если сет занят, но требуется залить в него ещё одну линию, то эту линию запишем на место той линии, которая использовалась раньше всех(и конечно, ещё находится в сете)

Вариант

Вариант – 1.

Ассоциативность	2 – **CACHE_WAY**
Размер тэга адреса	10 бит – **CACHE_TAG_SIZE**
Размер кэш-линии	16 байта – **CACHE_LINE_SIZE**
Кол-во кэш-линий	64 – **CACHE_LINE_COUNT**
Размер памяти	512 Кбайт – **MEM_SIZE**

Рисунок 1 – Параметры системы

Вычисление недостающих параметров системы

CACHE_SETS_COUNT = CACHE_LINE_COUNT / CACHE_WAY = 32

CACHE_SET_SIZE = LOG2(CACHE_SETS_COUNT) = 5

CACHE_SIZE = CACHE_LINE_COUNT * CACHE_LINE_SIZE = 1024

ADDR_SIZE = LOG2(MEM_SIZE) = 19

CACHE_OFFSET_SIZE =

ADDR_SIZE – CACHE_TAG_SIZE – CACHE_SET_SIZE = 4

Аналитическое решение

ЯП для аналитического решения – Python(файл – “analytic.py”).
Решение симулирует запросы, а также состояния кэш-линий.

Моделирование заданной системы(Verilog)

Всего есть 3 главных части: CPU, CACHE, MEMORY. Коммутация сигнала(reset) и генерации(clk) происходит с помощью модуля testbench.

Модель состоит из 3 элементов: CPU, CACHE и MEMORY. Так же есть модуль testbench, однако он нужен только для коммутации остальных модулей вместе, сигнала reset и генерации clk.

Родоначальником всех действий является CPU. Для каждой команды существует отдельный task, образующий ассемблер(с алгоритмом и тестами). Запуски совершаются из блока initial.

CPU и CACHE соединены шинами: A1(15 бит), D1(16 бит), C1(3 бита).

MEM - оперативная память на 512Кбайт. Этот модуль симулирует задержки, присущие настоящей памяти.

CACHE содержит в себе память для кэш-линий и доп. Информации(tag, LRU, dirty, valid). Ещё CACHE считает кол-во попаданий и промахов.

Через команду cache_dump информация выводится в файл и консоль.

Замечание: на шине C1 команды C1_RESPONSE и C1_WRITE32 имеют одинаковые коды, но пишутся разными агентами. В коде эта команда называется C1_WRITE32_RESP.

В блоках always находится все логика кэша и памяти(работающий по сигналу clk). Для синхронизации в каждом модуле есть таски задержек.

Шины cpu-cache содержит CPU, а cache-mem содержит CACHE. Также “владение” можно передать с помощью присваивания на шину сигнала z содержащей стороной.

Это может использовать при:

Процессор отдал кэшу команду C1_READ[8|16|32] или C1_WRITE[8|16|32] и ждёт от него C1_RESPONSE.

CACHE отдал памяти команду C2_READ/C2_WRITE и ждёт C2_RESPONSE для начала передачи информации.

У каждой шины и модуля есть свой буфер – переменная типа reg с суффиксом _buff. В буфер записывается значение, которые будет присвоено шине в следующий такт Verilog’a при помощи assign, которые располагаются внизу.

Данные по шине D1 пересылаются за 1 или 2 такта(1 такт для 8, 16 битных команд)(2 такта для 32 битных)

По шине D2 передаются кэш-линии целиком. В CACHE и MEM одновременно работают 2 цикла, один из которых отправляет данные, а другой читает.

Тестировка

Несколько тестов содержатся в CPU(вынесены в таски): read_write_test, invalidate_test и eviction_test. Тесты запускаются по окончании симуляции и дампинга кэша и памяти в файлы. Результат работы выписывается в консоль.

Воспроизведение задачи на Verilog

В модуле CPU таска “matrix_mull_sim” симулирует заданный алгоритм(умножения двух мтриц)Там присутствуют команды READ8, READ16, WRITE32

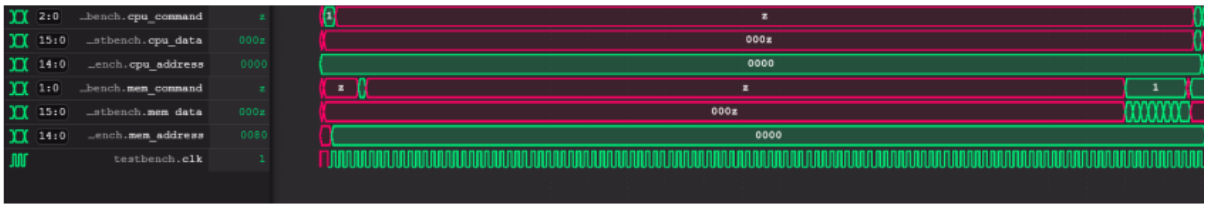

Рисунок 2 - Кэш промаха

Поступает команда на чтение от CPU, CACHE читает адрес, фиксирует промах и через 4 такта отдаёт команду в MEMORY. Через 200 тактов приходит ответ от памяти и начинается передача кэш-линии, которая занимает 8 тактов. В конце CACHE отдаёт ответ CPU и передаёт информацию за 1 такт.

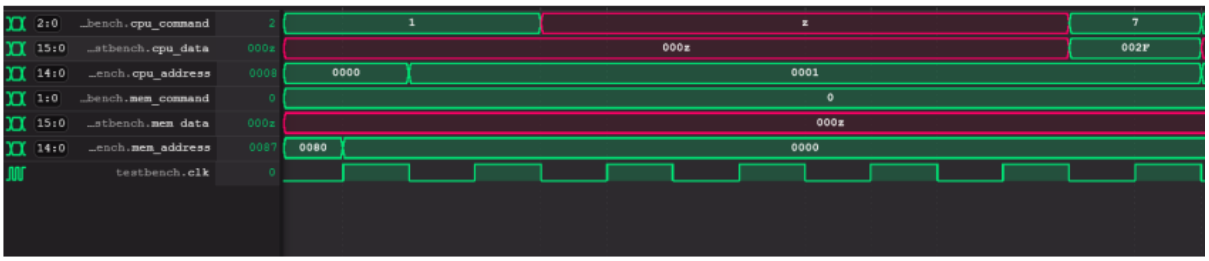

Рисунок 3 - Кэш попадания

Как и в прошлом примере поступает команда на чтение от cpu, в этот раз запрашиваемая кэш-линия оказывается в памяти и через 6 тактов cache даёт ответ и передаёт нужные данные.

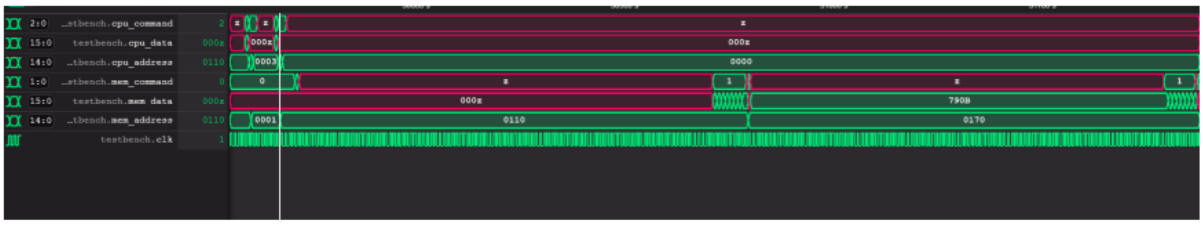

Рисунок 4 - Кэш промаха с вытеснением

От cpu поступает запрос, запрашиваемой кэш-линии не оказывает в кэше и cache отправляет запрос на чтение в память. Но места для этой линии в кэше нет, т. к. весь подходящий сет занят другими линиями, да ещё и линия, которую нужно вытеснить (least recently used), содержит изменённые данные (dirty). Поэтому cache начинает запись это кэш линии в memory. Суммарно мы имеем 4 на обращение к памяти + 200 на ответ от памяти + 8 на передачу новой линии + 200 на ожидания ответа от памяти для записи + 8 циклов на передачи старый dirty линии в память.

Сравнение полученных результатов

Результаты Verilog сохраняет в файл “hit_stat.dump”(симуляция запросов и кэш попаданий). Далее запускаем симуляцию на Python “analytic.py”,вычисляющая свой результат, после чего считает из файла результат Verilog’a и сравнит их. В ходе работы, решение добивается полного совпадения значений запросов (249600) и кэш попаданий (228080). Не менее 91% запросов были кэш попаданиями. Такой процент получился благодаря последовательному расположению данных в памяти. Так же данные разных массивов не конкурировали за одни CACHE сеты в силу не большого размера по сравнению с размером всей памяти.Кол-во тактов: 5–7 миллионов. Это порядка $2m$ суммирований во внешнем цикле, $2mnk$ суммирований во внутреннем, mnk умножений ($5mnk$ циклов), 228080 кэш попаданий по 7 циклов, и 21520 промахов по ~200-400 циклов. При симуляции насчитывается 5099575 циклов.

Запуск

При запуске команд последовательно из корневой директории проекта:

iverilog -g2012 -o testbench.out testbench.sv

vvp testbench.out

python analytic.py

то будет запущена симуляция алгоритма на Verilog, автоматические тесты на Verilog, симуляция на Python, сравнение результатов. Тесты можно отключить, удалив: read_write_test, invalidate_test и eviction_test. из CPU (строчки 225–229)

Листинг кода

analytic.py

```
import numpy as np
import math

N = 60
K = 32
M = 64

CACHE_SET_COUNT = 32
CACHE_SET_SIZE = int(math.log2(CACHE_SET_COUNT))
CACHE_LINE_SIZE = 16
CACHE_OFFSET_SIZE = 4
```

```
aStart = 0

aIntSize = 1

aSize = K*M*aIntSize



bStart = aSize + aStart;

bIntSize = 2

bSize = N*K*bIntSize



cStart = bStart + bSize

cIntSize = 4

cSize = N*M*cIntSize



class Cache:

    def __init__(self):

        self.tag_array = np.zeros((CACHE_SET_COUNT, 2), dtype=int)

        self.valid_array = np.zeros((CACHE_SET_COUNT, 2), dtype=bool)

        self.lru_array = np.zeros(CACHE_SET_COUNT, dtype=bool)
```

```
    self.reqCount = 0
    self.hitCount = 0

def req(self, addr: int):
    setNum = Cache.getSet(addr)
    tag = Cache.getTag(addr)
    return self.checkHit(setNum, tag)

def checkHit(self, setNum: int, tag: int):
    self.reqCount += 1
    for i in range(2):
        if (self.valid_array[setNum, i] and self.tag_array[setNum, i] == tag):
            self.hit(setNum, i)
            return True
    else:
        self.miss(setNum, tag)
        return False

def hit(self, setNum: int, i: int):
    self.hitCount += 1
    self.lru_array[setNum] = (i == 0)

def miss(self, setNum: int, tag: int):
```

```
            lru_index = int(self.lru_array[setNum])
            self.tag_array[setNum, lru_index] = tag
            self.valid_array[setNum, lru_index] = True
            self.lru_array[setNum] = not self.lru_array[setNum]

        @ staticmethod
        def getTag(address: int) -> int:
            return (address >> (CACHE_OFFSET_SIZE + CACHE_SET_SIZE))

        @ staticmethod
        def getSet(address: int) -> int:
            return (address >> CACHE_OFFSET_SIZE) % CACHE_SET_COUNT



def simulate(cache):
    pa = aStart
    pc = cStart
    for i in range(M):
        for j in range(N):
            pb = bStart
            for k in range(K):
                cache.req(pa + k*aIntSize)  # a
                cache.req(pb + j*bIntSize)  # b
```

```
            pb += N*bIntSize
        cache.req(pc + j*cIntSize)  # c
    pa += K*aIntSize
    pc += N*cIntSize



def validate(reqCount: int, hitCount: int):
    with open("hit_stat.dump", "r") as hit_stat:
        givenReqCount = int(hit_stat.readline())
        givenHitCount = int(hit_stat.readline())

        if (givenReqCount != reqCount or givenHitCount != hitCount):
            raise Exception(
                f"{cl.OKBLUE}Results missmatch{cl.ENDC}
{cl.OKGREEN}Analytic{cl.ENDC}:   reqs {reqCount}, hits {hitCount}, rate
{round(hitCount/reqCount, 6)}
{cl.WARNING}Simulation{cl.ENDC}: reqs {givenReqCount}, hits
{givenHitCount}, rate {round(givenHitCount/givenReqCount, 6)}\n")



def main():
    cache = Cache()
    simulate(cache)
```

```
    try:
        validate(cache.reqCount, cache.hitCount)
    except Exception as e:
        print("\nData validation" + cl.FAIL + " failed" + cl.ENDC)
        print(e)
    else:
        print("\nData validation" + cl.OKGREEN + " passed" + cl.ENDC)
        print(
            f'{cl.HEADER}{cl.BOLD}Requests{cl.ENDC}:
{cache.reqCount}\n{cl.HEADER}{cl.BOLD}Hits{cl.ENDC}:
{cache.hitCount}')
        print(f'{cl.UNDERLINE}{cl.OKBLUE}HIT RATE{cl.ENDC}:
{cl.WARNING}{round(cache.hitCount/cache.reqCount, 6)}{cl.ENDC}\n')



if __name__ == '__main__':
    class cl:
        HEADER = '\033[95m'
        OKBLUE = '\033[94m'
        OKCYAN = '\033[96m'
        OKGREEN = '\033[92m'
        WARNING = '\033[93m'
        FAIL = '\033[91m'
        ENDC = '\033[0m'
        BOLD = '\033[1m'
```

```
    UNDERLINE = '\033[4m'
main()
```

cache.sv

```
module cache#(
    parameter BUS_SIZE = 16 ,
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_LINE_SIZE   = 16
    ) (
    input clk,
    input reset,
    input dump,
    input [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] cpu_address,
    inout [BUS_SIZE-1:0] cpu_data,
    inout [3-1:0] cpu_command,

    output [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address,
    inout [BUS_SIZE-1:0] mem_data,
    inout [2-1:0] mem_command
);

    parameter CACHE_WAY         = 2;
```

```
    parameter CACHE_LINE_COUNT  = 64;


    parameter CACHE_SET_SIZE    = $clog2(CACHE_SETS_COUNT);
    parameter CACHE_TAG_SIZE    = 10;
    parameter CACHE_SIZE        = CACHE_LINE_COUNT *
CACHE_LINE_SIZE;
    parameter CACHE_SETS_COUNT  =
CACHE_LINE_COUNT/CACHE_WAY;


    localparam  C1_NOP          = 3'd0,
                C1_READ8        = 3'd1,
                C1_READ16       = 3'd2,
                C1_READ32       = 3'd3,
                C1_INV_LINE     = 3'd4,
                C1_WRITE8       = 3'd5,
                C1_WRITE16      = 3'd6,
                C1_WRITE32_RESP = 3'd7;


    localparam  C2_NOP          = 2'd0,
                C2_RESPONSE     = 2'd1,
                C2_READ         = 2'd2,
                C2_WRITE        = 2'd3;


    // STORAGE
```

```
    reg valid_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];

    reg dirty_array [CACHE_SETS_COUNT-1:0][CACHE_WAY-1:0];

    reg LRU_array [CACHE_SETS_COUNT-1:0];

    reg [CACHE_TAG_SIZE-1:0] tag_array [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0];

    reg [CACHE_LINE_SIZE*8-1:0] data_array [CACHE_SETS_COUNT-
1:0][CACHE_WAY-1:0]; // stores lines



    reg [CACHE_TAG_SIZE-1:0] cpu_tag_buff;

    reg [CACHE_SET_SIZE-1:0] cpu_set_buff;

    reg [CACHE_OFFSET_SIZE-1:0] cpu_offset_buff;

    reg [BUS_SIZE-1:0] cpu_data_bus_buff;

    reg [BUS_SIZE*2-1:0] cpu_data_to_write;

    reg [3-1:0] cpu_command_buff;



    reg [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address_buff;

    reg [CACHE_LINE_SIZE*8-1:0] mem_line_buff;

    reg [BUS_SIZE-1:0] mem_data_buff;

    reg [2-1:0] mem_command_buff;



    // Analytic

    real req;

    real hit;
```

```
// Tasks
task delay;
    begin
        @(negedge clk);
    end
endtask

task read_bus_delay;
    begin
        @(posedge clk);
    end
endtask

task hit_resp_delay;
    repeat(4) begin
        delay;
    end
endtask

task miss_req_delay;
    repeat(3) begin
        delay;
    end
```

```
    endtask


    task wait_for_resp;
        while (mem_command !== C2_RESPONSE) begin
            read_bus_delay;
        end
    endtask


    task evict_if_dirty;
        if (dirty_array[cpu_set_buff][index_in_set] == 1) begin
            mem_address_buff = {tag_array[cpu_set_buff][index_in_set],
cpu_set_buff};
            write_to_MM;
        end
    endtask


    task replace_from_MM;
        // command
        mem_command_buff = C2_READ;
        delay;
        mem_command_buff = 'z;
        wait_for_resp;


        // data
```

```
    for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
        mem_line_buff[BUS_SIZE*i +: BUS_SIZE] = mem_data;
        read_bus_delay;
    end

    // restore
    mem_command_buff = C2_NOP;

    if (valid_array[cpu_set_buff][0] == 0) begin
        index_in_set = 0;
        store;
    end else if (valid_array[cpu_set_buff][1] == 0) begin
        index_in_set = 1;
        store;
    end else begin
        // evict if no empty space
        index_in_set = LRU_array[cpu_set_buff];
        evict_if_dirty;

        index_in_set = LRU_array[cpu_set_buff];
        store;
    end
endtask
```

```
    task write_to_MM;
        // command
        mem_command_buff = C2_WRITE;
        delay;
        mem_command_buff = 'z;
        mem_data_buff = data_array[cpu_set_buff][index_in_set][0 +:
BUS_SIZE];
        wait_for_resp;


        // data
        delay;
        for (int i=1; i<CACHE_LINE_SIZE/2; i=i+1) begin
            mem_data_buff = data_array[cpu_set_buff][index_in_set][BUS_SIZE*i
+: BUS_SIZE];
            delay;
        end
        // restore
        mem_command_buff = C2_NOP;
        mem_data_buff = 'z;
    endtask


    reg index_in_set;
    task store;
```

```
        data_array[cpu_set_buff][index_in_set] = mem_line_buff;
        tag_array[cpu_set_buff][index_in_set] = cpu_tag_buff;
        valid_array[cpu_set_buff][index_in_set] = 1;
        dirty_array[cpu_set_buff][index_in_set] = 0;
        LRU_array[cpu_set_buff] = ~index_in_set;
    endtask

    task read_cpu_address;
        mem_address_buff = cpu_address;
        cpu_tag_buff = cpu_address[CACHE_TAG_SIZE+CACHE_SET_SIZE-
1:CACHE_SET_SIZE];
        cpu_set_buff = cpu_address[CACHE_SET_SIZE-1:0];

        delay;

        cpu_offset_buff = cpu_address[3:0];
    endtask

    reg [3-1:0] cur_cpu_command;
    task read_from_storage;
        delay;
        if (cur_cpu_command == C1_READ8) begin
            cpu_data_bus_buff =
data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 8];
```

```
        end else if (cur_cpu_command == C1_READ16 || cur_cpu_command ==
C1_READ32) begin

            cpu_data_bus_buff =
data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 16];

        end

        LRU_array[cpu_set_buff] = ~index_in_set;

    endtask



    task write_to_storage;

        delay;

        if (cur_cpu_command == C1_WRITE8) begin

            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 8] =
cpu_data_to_write;

        end else if (cur_cpu_command == C1_WRITE16) begin

            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 16] =
cpu_data_to_write;

        end else if (cur_cpu_command == C1_WRITE32_RESP) begin

            data_array[cpu_set_buff][index_in_set][cpu_offset_buff*8 +: 32] =
cpu_data_to_write;

        end

        dirty_array[cpu_set_buff][index_in_set] = 1;

        LRU_array[cpu_set_buff] = ~index_in_set;

    endtask



    int hit_stat_file;
```

```
task dump_hit_stat;
    $display("\nHIT statistic:");
    $display("Requests: %d\nHits: %d\nHIT RATE: %f", req, hit, hit/req);
    hit_stat_file = $fopen("hit_stat.dump", "w");
    if (hit_stat_file) begin
        $fdisplay(hit_stat_file, "%d\n%d\n", req, hit);
        $display("HIT stat dumped to hit_stat.dump.\n");
    end else begin
        $display("Error while hit stat dump");
    end
    $fclose(hit_stat_file);
endtask

int dump_f;
task dump_to_file;
    dump_f = $fopen("cache.dump", "w");
    if (dump_f) begin
        $fdisplay(dump_f,"$$$$$$ CACHE DUMP $$$$$$");
        for (int i=0; i<CACHE_SETS_COUNT; i=i+1) begin
            $fdisplay(dump_f,"== SET 0x%0H\t==", i);

            $fdisplay(dump_f,"WAY %0d\nvalid: %b", 0, valid_array[i][0]);
            if (valid_array[i][0]) begin
```

```
            $fdisplay(dump_f,"dirty: %b", dirty_array[i][0]);
            $fdisplay(dump_f,"tag:   0x%0H", tag_array[i][0]);
            $fdisplay(dump_f,"data:  0x%h", data_array[i][0]);
        end

        $fdisplay(dump_f,"WAY %0d\nvalid: %b", 1, valid_array[i][1]);
        if (valid_array[i][1]) begin
            $fdisplay(dump_f,"dirty: %b", dirty_array[i][1]);
            $fdisplay(dump_f,"tag:   0x%0H", tag_array[i][1]);
            $fdisplay(dump_f,"data:  0x%h", data_array[i][1]);
        end
        $fdisplay(dump_f,"");
      end

      $display("Cache dumped successful. Check cache.dump");
   end else begin
      $display("Error while cache dump");
   end
   $fclose(dump_f);
endtask


always @(posedge clk or posedge reset) begin
```

```
        if (reset) begin
            for (int i=0; i<CACHE_SETS_COUNT; i=i+1) begin
                valid_array[i][0] =  0;
                dirty_array[i][0] =  0;
                tag_array[i][0]   = 'z;
                data_array[i][0]  = 'z;
                valid_array[i][1] =  0;
                dirty_array[i][1] =  0;
                tag_array[i][1]   = 'z;
                data_array[i][1]  = 'z;
            end
            mem_line_buff     =  0;
            cpu_data_bus_buff = 'z;
            cpu_command_buff  = 'z;
            cur_cpu_command   =  0;
            mem_command_buff  = 'z;
            mem_data_buff     = 'z;
        end else if (dump) begin
            dump_to_file;
            dump_hit_stat;
        end else if (cpu_command == C1_READ8 || cpu_command ==
C1_READ16 || cpu_command == C1_READ32) begin
            req = req + 1;
```

```
        cur_cpu_command = cpu_command;

        read_cpu_address;


        if (valid_array[cpu_set_buff][0] == 1 && tag_array[cpu_set_buff][0] ==
cpu_tag_buff) begin

            hit_resp_delay;

            hit = hit + 1;

            index_in_set = 0;

            read_from_storage;

        end else if (valid_array[cpu_set_buff][1] == 1 &&
tag_array[cpu_set_buff][1] == cpu_tag_buff) begin

            hit_resp_delay;

            hit = hit + 1;

            index_in_set = 1;

            read_from_storage;

        end else begin

            miss_req_delay;

            replace_from_MM;

            read_from_storage;

        end


        cpu_command_buff = C1_WRITE32_RESP;


        if (cur_cpu_command == C1_READ32) begin
```

```
            cpu_offset_buff += 2;
            read_from_storage;
        end
        delay;
        cpu_command_buff = 'z;
        cpu_data_bus_buff = 'z;
        cur_cpu_command = 'z;

    end else if (cpu_command == C1_WRITE8 || cpu_command ==
C1_WRITE16 || cpu_command == C1_WRITE32_RESP) begin
        req = req + 1;

        cur_cpu_command = cpu_command;
        cpu_data_to_write[0 +: BUS_SIZE] = cpu_data;

        read_cpu_address;

        if (cpu_command == C1_WRITE32_RESP) begin
            cpu_data_to_write[BUS_SIZE +: BUS_SIZE] = cpu_data;
        end

        if (valid_array[cpu_set_buff][0] == 1 && tag_array[cpu_set_buff][0] ==
cpu_tag_buff) begin
            hit_resp_delay;
```

```
            hit = hit + 1;
            index_in_set = 0;
            write_to_storage;
        end else if (valid_array[cpu_set_buff][1] == 1 &&
tag_array[cpu_set_buff][1] == cpu_tag_buff) begin
            hit_resp_delay;
            hit = hit + 1;
            index_in_set = 1;
            write_to_storage;
        end else begin
            miss_req_delay;
            replace_from_MM;
            write_to_storage;
            delay;
        end

        cpu_command_buff = C1_WRITE32_RESP;
        delay;
        cpu_command_buff = 'z;
        cur_cpu_command  = 'z;

    end else if (cpu_command == C1_INV_LINE) begin
        read_cpu_address;
        if (tag_array[cpu_set_buff][0] == cpu_tag_buff) begin
```

```
            index_in_set = 0;
            evict_if_dirty;
            valid_array[cpu_set_buff][0] = 0;
        end else if (tag_array[cpu_set_buff][1] == cpu_tag_buff) begin
            index_in_set = 0;
            evict_if_dirty;
            valid_array[cpu_set_buff][1] = 0;
        end

        cpu_command_buff = C1_WRITE32_RESP;
        delay;
        cpu_command_buff = 'z;
        cur_cpu_command  = 'z;

    end
end

assign mem_address = mem_address_buff;
assign mem_data = mem_data_buff;
assign mem_command = mem_command_buff;
assign cpu_data = cpu_data_bus_buff;
assign cpu_command = cpu_command_buff;
```

```
endmodule
```

cpu.sv

```
module cpu #(
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter BUS_SIZE = 16,
    parameter CACHE_OFFSET_SIZE = 4
    ) (

    input clk,
    output cache_dump,
    output mem_dump,
    output [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address,
    inout [BUS_SIZE-1:0] data,
    inout [3-1:0] command
);

    localparam  C1_NOP          = 3'd0,
                C1_READ8        = 3'd1,
                C1_READ16       = 3'd2,
                C1_READ32       = 3'd3,
                C1_INV_LINE     = 3'd4,
                C1_WRITE8       = 3'd5,
                C1_WRITE16      = 3'd6,
```

```
            C1_WRITE32_RESP = 3'd7;


reg [MEM_ADDR_SIZE-1:0] cpu_address_buff;
reg [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address_bus_buff;
reg [BUS_SIZE*2-1:0] data_to_write;
reg [BUS_SIZE-1:0] data_buff;
reg [3-1:0] cpu_command_buff;
reg [BUS_SIZE-1:0] recieved_data;
reg [BUS_SIZE*2-1:0] local_storage;

reg cache_dump_buff;
reg mem_dump_buff;

task delay;
    begin
        @(negedge clk);
    end
endtask

task read_bus_delay;
    begin
        @(posedge clk);
    end
```

```
    endtask


    task send_address;
        address_bus_buff = cpu_address_buff[MEM_ADDR_SIZE-
1:CACHE_OFFSET_SIZE];
        delay;
        address_bus_buff = cpu_address_buff[CACHE_OFFSET_SIZE-1:0];
        delay;
    endtask
    task wait_for_resp;
        while (command !== C1_WRITE32_RESP) begin
            read_bus_delay;
        end
    endtask


    task READ8;
        cpu_command_buff = C1_READ8;
        READ;
        local_storage = recieved_data[8-1:0];
        delay;
    endtask
    task READ16;
        cpu_command_buff = C1_READ16;
        READ;
```

```
    local_storage = recieved_data;
    delay;
endtask
task READ32;
    cpu_command_buff = C1_READ32;
    READ;
    local_storage[BUS_SIZE-1:0] = recieved_data;
    read_bus_delay;
    recieved_data = data;
    local_storage[BUS_SIZE*2-1:BUS_SIZE] = recieved_data;
    delay;
endtask
task READ;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    recieved_data = data;
endtask

task WRITE8;
    cpu_command_buff = C1_WRITE8;
    WRITE;
    delay;
```

```
    endtask
    task WRITE16;
        cpu_command_buff = C1_WRITE16;
        WRITE;
        delay;
    endtask
    task WRITE32;
        cpu_command_buff = C1_WRITE32_RESP;


        data_buff = data_to_write[BUS_SIZE-1:0];
        address_bus_buff = cpu_address_buff[MEM_ADDR_SIZE-
1:CACHE_OFFSET_SIZE];
        delay;


        data_buff = data_to_write[BUS_SIZE*2-1:BUS_SIZE];
        address_bus_buff = cpu_address_buff[CACHE_OFFSET_SIZE-1:0];
        delay;


        cpu_command_buff = 'z;
        wait_for_resp;
        data_buff = 'z;
        delay;
    endtask
    task WRITE;
```

```
    data_buff = data_to_write;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    data_buff = 'z;
endtask

task INV;
    cpu_command_buff = C1_INV_LINE;
    send_address;
    cpu_command_buff = 'z;
    wait_for_resp;
    delay;
endtask

initial begin
    address_bus_buff = 'z;
    cpu_command_buff = 'z;
    recieved_data = 0;
    data_buff = 'z;
    cache_dump_buff = 0;
    mem_dump_buff = 0;
end
```

```
task dump_cache;
    cache_dump_buff = 1;
    delay;
    cache_dump_buff = 0;
endtask

task dump_mem;
    mem_dump_buff = 1;
    delay;
    mem_dump_buff = 0;
endtask

task dump_all;
    delay;
    dump_cache;
    dump_mem;
endtask

parameter M = 19'd64;
parameter N = 19'd60;
parameter K = 19'd32;
```

```
parameter aStart = 19'd0;
parameter aIntSize = 19'd1;
parameter aSize = M*K*aIntSize;

parameter bStart = aStart + aSize;
parameter bIntSize = 19'd2;
parameter bSize = K*N*bIntSize;

parameter cStart = bStart + bSize;
parameter cIntSize = 19'd4;
parameter cSize = M*N*cIntSize;

int pa;
int pb;
int pc;
int s;
int j;
int k;
int prev_val;

task matrix_mull_sim;
    $display("Simulation started");
    pa = aStart;
```

```
pc = cStart;
for (int i=0; i<M; ++i) begin
    for (j=0; j<N; ++j) begin
        pb = bStart;
        s = 0;
        for (k=0; k<K; ++k) begin
            // a
            cpu_address_buff = pa + k*aIntSize;
            READ8;
            prev_val = local_storage[7:0];

            //b
            cpu_address_buff = pb + j*bIntSize;
            READ16;
            s += local_storage[15:0] * prev_val;
            repeat(6) begin
                delay;
            end

            pb += N*bIntSize;
            delay;
        end
        // c
```

```
            cpu_address_buff = pc + j*cIntSize;
            data_to_write = s;
            WRITE32;
        end
        pa += K*aIntSize;
        delay;
        pc += N*cIntSize;
        delay;
    end
    $display("Simulation finished");
endtask

// Place for test calls
initial begin
    delay;
    matrix_mull_sim;

    dump_all;

    read_write_test;

    invalidate_test;
```

```
    eviction_test;

    $finish();
end

assign address = address_bus_buff;
assign data = data_buff;
assign command = cpu_command_buff;
assign cache_dump = cache_dump_buff;
assign mem_dump = mem_dump_buff;

task pass;
    $display("%c[5;32mPASS%c[0m",27,27);
endtask

task fail;
    $display("%c[1;31mFAIL%c[0m",27,27);
endtask

reg [19-1:0] test_addr;
reg [32-1:0] test_data;
task read_write_test;
    $display("\n##################################################");
```

```
$display("#############  READ/WRITE TEST  #############");
$display("################################################\n");

$display("@@@@@@@@@@@@@@@@");
$display("@@@  8 bit @@@");
$display("@@@@@@@@@@@@@@@@");
test_addr = 19'b0000000000_01110_0000;
test_data = 8'b1111_0000;

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ8;
$display("data      %b", local_storage[8-1:0]);
$display("----------------");

cpu_address_buff = test_addr;
data_to_write = test_data;
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write[8-1:0]);
WRITE8;
$display("----------------");

cpu_address_buff = test_addr;
```

```
$display("read from %b", cpu_address_buff);
READ8;
$display("data      %b", local_storage[8-1:0]);
$display("----------------");

$display("\nREAD/WRITE 8 bit");
if (test_data == local_storage[8-1:0]) begin
   pass;
end else begin
   fail;
   $display("expected  %b", test_data[8-1:0]);
   $display("actual    %b", local_storage[8-1:0]);
end

$display("\n@@@@@@@@@@@@@@@@");
$display("@@@ 16 bit @@@");
$display("@@@@@@@@@@@@@@@@");
test_addr = 19'b0000000000_01110_0010;
test_data = 16'b1111_1111_0000_0000;

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ16;
```

```
$display("data      %b", local_storage[16-1:0]);
$display("----------------");

cpu_address_buff = test_addr;
data_to_write = test_data;
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write[16-1:0]);
WRITE16;
$display("----------------");

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ16;
$display("data      %b", local_storage[16-1:0]);
$display("----------------");

$display("\nREAD/WRITE 16 bit");
if (test_data == local_storage[16-1:0]) begin
    pass;
end else begin
    fail;
    $display("expected  %b", test_data[16-1:0]);
    $display("actual    %b", local_storage[16-1:0]);
```

```
end

$display("\n@@@@@@@@@@@@@@@@");
$display("@@@ 32 bit @@@");
$display("@@@@@@@@@@@@@@@@");
test_addr = 19'b0000000000_01110_0000;
test_data = 32'b0101_0101_0101_0101_0101_0101_0101_0101;



$display("----------------");
cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage[32-1:0]);
$display("----------------");

cpu_address_buff = test_addr;
data_to_write = test_data;
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write[32-1:0]);
WRITE32;
$display("----------------");
```

```
    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage[32-1:0]);
    $display("----------------");

    $display("\nREAD/WRITE 32 bit");
    if (test_data == local_storage[32-1:0]) begin
        pass;
    end else begin
        fail;
        $display("expected  %b", test_data[32-1:0]);
        $display("actual    %b", local_storage[32-1:0]);
    end
    $display("");
endtask

reg [19-1:0] evtest_addr[3-1:0];
reg [32-1:0] evtest_data[3-1:0];
task eviction_test;
    $display("\n##############################################");
    $display("#############  EVICTION TEST  #############");
    $display("##############################################\n");
```

```
$display("~~~~~~~~~~~~~~");
$display("~~~ FIRST ~~~");
$display("~~~~~~~~~~~~~~");
evtest_addr[0] = 19'b0000000000_01110_0000;
evtest_data[0] = 32'b1111_0000_1111_0000_0000_1111_0000_1111;

$display("----------------");
cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[0];
data_to_write = evtest_data[0];
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");

cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
```

```
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~~");
$display("~~~ SECOND ~~~");
$display("~~~~~~~~~~~~~~~");
evtest_addr[1] = 19'b0000000001_01110_0000;
evtest_data[1] = 32'b1111_1111_1111_1111_0000_1111_1111_0000;

$display("----------------");
cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[1];
data_to_write = evtest_data[1];
$display("write to  %b", cpu_address_buff);
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");
```

```
cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~");
$display("~~~ THIRD ~~~");
$display("~~~~~~~~~~~~~~");
evtest_addr[2] = 19'b0000000010_01110_0000;
evtest_data[2] = 32'b1111_1111_0001_1001_1001_1000_1111_1111;

$display("----------------");
cpu_address_buff = evtest_addr[2];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = evtest_addr[2];
data_to_write = evtest_data[2];
$display("write to  %b", cpu_address_buff);
```

```
$display("data      %b", data_to_write);
WRITE32;
$display("----------------");

cpu_address_buff = evtest_addr[2];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("~~~~~~~~~~~~~~~~~~~");
$display("~~~ FIRST AGAIN ~~~");
$display("~~~~~~~~~~~~~~~~~~~");
$display("----------------");
cpu_address_buff = evtest_addr[0];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("EVICTION FIRST");
if (evtest_data[0] == local_storage[32-1:0]) begin
    pass;
```

```
end else begin
    fail;
    $display("expected  %b", evtest_data[0][32-1:0]);
    $display("actual    %b", local_storage[32-1:0]);
end

$display("\n~~~~~~~~~~~~~~~~~~~~");
$display("~~~ SECOND AGAIN ~~~");
$display("~~~~~~~~~~~~~~~~~~~~");
$display("----------------");
cpu_address_buff = evtest_addr[1];
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------\n");

$display("EVICTION SECOND");
if (evtest_data[1] == local_storage[32-1:0]) begin
    pass;
end else begin
    fail;
    $display("expected  %b", evtest_data[1][32-1:0]);
    $display("actual    %b", local_storage[32-1:0]);
```

```
    end


    $display("\n~~~~~~~~~~~~~~~~~~~");
    $display("~~~ THIRD AGAIN ~~~");
    $display("~~~~~~~~~~~~~~~~~~~");
    $display("----------------");
    cpu_address_buff = evtest_addr[2];
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage);
    $display("----------------\n");


    $display("EVICTION THIRD");
    if (evtest_data[2] == local_storage[32-1:0]) begin
        pass;
    end else begin
        fail;
        $display("expected  %b", evtest_data[2][32-1:0]);
        $display("actual    %b", local_storage[32-1:0]);
    end
    $display("");
endtask
```

```
task invalidate_test;
    $display("\n##############################################");
    $display("#############  INVALIDATE TEST  #############");
    $display("##############################################\n");
    test_addr = 19'b0000000000_10001_0000;
    test_data = 32'b0101_0101_0101_0101_0101_0101_0101_0101;

    $display("----------------");
    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
    READ32;
    $display("data      %b", local_storage);
    $display("----------------");

    cpu_address_buff = test_addr;
    data_to_write = test_data;
    $display("write to  %b", cpu_address_buff);
    $display("data      %b", data_to_write);
    WRITE32;
    $display("----------------");

    cpu_address_buff = test_addr;
    $display("read from %b", cpu_address_buff);
```

```
READ32;
$display("data      %b", local_storage);
$display("----------------");

cpu_address_buff = test_addr;
$display("inv_line  %b", cpu_address_buff);
INV;
$display("----------------");

cpu_address_buff = test_addr;
$display("read from %b", cpu_address_buff);
READ32;
$display("data      %b", local_storage);
$display("----------------");

$display("\nINV");
if (test_data[8-1:0] == local_storage[8-1:0]) begin
    pass;
end else begin
    fail;
    $display("expected  %b", test_data[32-1:0]);
    $display("actual    %b", local_storage[32-1:0]);
end
```

```
    endtask


endmodule
```

mem.sv

```
module mem #(
    parameter MEM_ADDR_SIZE = 10 + 9,
    parameter BUS_SIZE = 16 ,
    parameter CACHE_OFFSET_SIZE = 4,
    parameter CACHE_LINE_SIZE = 16
    ) (
    input clk,
    input reset,
    input dump,
    input [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] address,
    inout [BUS_SIZE-1:0] data,
    inout [2-1:0] command
);


    // 512KB = 2^9 * 2^10 = 2^19 = 2^15 lines * 2^4 bits in each line (16 8-bit words)
    parameter MEM_SIZE = 1 << (MEM_ADDR_SIZE-CACHE_OFFSET_SIZE); // 2^15 cache lines
    parameter RESPONSE_TIME = 100;
```

```
    localparam  C2_NOP      = 3'd0,
                C2_RESPONSE = 3'd1,
                C2_READ     = 3'd2,
                C2_WRITE    = 3'd3;


    reg [CACHE_LINE_SIZE*8-1:0] storage [MEM_SIZE-1:0]; // 2^15 16-byte
lines
    reg [BUS_SIZE-1:0] data_buff; // single bus
    reg [2-1:0] command_buff;


    task delay;
        begin
            @(negedge clk);
        end
    endtask


    task read_bus_delay;
        begin
            @(posedge clk);
        end
    endtask


    task wait_and_response;
        repeat(RESPONSE_TIME) begin
```

```
        delay;
    end
    command_buff = C2_RESPONSE;
endtask

int dump_f;
task dump_to_console;
    dump_f = $fopen("mem.dump", "w");
    if (dump_f) begin

        $fdisplay(dump_f, "$$$$$$ MEM DUMP $$$$$$");
        $fdisplay(dump_f, "TAG     SET     DATA");
        for (int i = 0; i < MEM_SIZE; i=i+1) begin
            $fdisplay(dump_f, "0x%0H \t0x%0H \t0x%h\n", (i >> 5), i%32,
storage[i]);
        end

        $display("Mem dumped successful. Check mem.dump");
    end else begin
        $display("Error while mem dump");
    end
    $fclose(dump_f);
endtask
```

```
integer SEED = 225526;
int j;
always @(posedge clk or posedge reset) begin
    if (reset) begin
        $display("Filling the MEM...");
        for (int i = 0; i < MEM_SIZE; i=i+1) begin
            for (j=0; j<CACHE_LINE_SIZE; ++j) begin
                storage[i][8*j +: 8] = $random(SEED)>>16;
            end
        end
        $display("MEM filled");
        data_buff = 'z;
        command_buff = 'z;
    end else if (dump) begin
        dump_to_console;
    end else begin
        if (command == C2_READ) begin
            command_buff = 'z;
            wait_and_response;

            // READ
            for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
```

```
            data_buff = storage[address][BUS_SIZE*i +: BUS_SIZE];
            delay;
        end
        command_buff = 'z;
    end else if (command == C2_WRITE) begin
        data_buff = 'z;
        command_buff = 'z;
        wait_and_response;

        // WRITE
        for (int i=0; i<CACHE_LINE_SIZE/2; i=i+1) begin
            read_bus_delay;
            storage[address][BUS_SIZE*i +: BUS_SIZE] = data;
        end
        command_buff = 'z;
    end else begin
        data_buff = 'z;
    end
  end
end

assign data = data_buff;
assign command = command_buff;
```

```
endmodule
```

testbench.sv

```
`include "cache.sv"
`include "cpu.sv"
`include "mem.sv"

module testbench ();
    parameter BUS_SIZE = 16;
    parameter MEM_ADDR_SIZE = 10 + 9;   // log2(MEM_SIZE)
    parameter CACHE_OFFSET_SIZE = $clog2(CACHE_LINE_SIZE);
    parameter CACHE_LINE_SIZE = 16;
    wire [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] cpu_address;
    wire [BUS_SIZE-1:0] cpu_data;
    wire [3-1:0] cpu_command;
    wire [MEM_ADDR_SIZE-CACHE_OFFSET_SIZE-1:0] mem_address;
    wire [BUS_SIZE-1:0] mem_data;
    wire [2-1:0] mem_command;
    wire cache_dump;
    wire mem_dump;

    reg clk;
    reg reset;
```

```
    cpu #(MEM_ADDR_SIZE, BUS_SIZE, CACHE_OFFSET_SIZE)
    cpu (
        .clk(clk),
        .cache_dump(cache_dump),
        .mem_dump(mem_dump),
        .address(cpu_address),
        .data(cpu_data),
        .command(cpu_command)
        );

    cache #(BUS_SIZE, MEM_ADDR_SIZE, CACHE_OFFSET_SIZE,
CACHE_LINE_SIZE)
    cache (
        .clk(clk),
        .reset(reset),
        .dump(cache_dump),

        .cpu_address(cpu_address),
        .cpu_data(cpu_data),
        .cpu_command(cpu_command),

        .mem_address(mem_address),
        .mem_data(mem_data),
```

```
        .mem_command(mem_command)
        );

    mem #(MEM_ADDR_SIZE, BUS_SIZE, CACHE_OFFSET_SIZE,
CACHE_LINE_SIZE)
    mem(
        .clk(clk),
        .reset(reset),
        .dump(mem_dump),
        .address(mem_address),
        .data(mem_data),
        .command(mem_command)
        );

    int cyclesCount = 1;
    initial begin
        #2 forever begin
            #1 if (clk) begin
                ++cyclesCount;
            end
            if (mem_dump) begin
                $display("Total clock cycles: %0d", cyclesCount);
                repeat(2) begin
                    @(posedge clk);
```

```
            end
        end
    end
end

initial begin
    reset = 0;
    #1 reset = 1;
    #1 reset = 0;
    clk = 1'd0;

    forever begin
        #1 clk = ~clk;
    end
end

endmodule
```