|  |  |  |
| --- | --- | --- |
| 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.

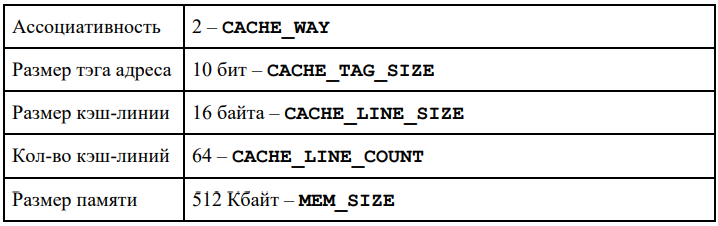


Рисунок 1 – Параметры системы

**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, СACHE, 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)

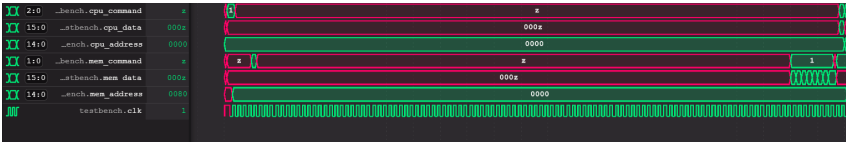
По шине D2 передаются кэш-линии целиком. В CACHE и MEM одновременно работают 2 цикла, один из которых отправляет данные, а другой читает.

**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.

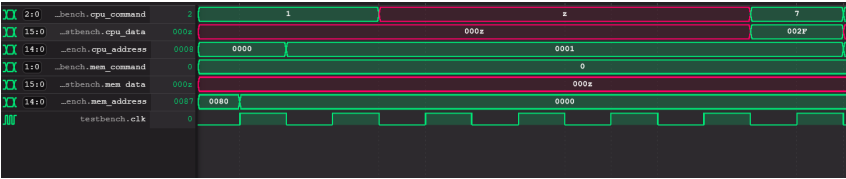
**Воспроизведение задачи на Verilog**

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



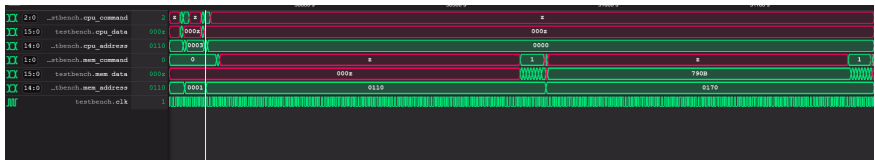
**Рисунок 2 - Кэш промаха**

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’а 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