|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2 | M3136 | 2022 |
| Моделирование схем в Verilog | Корнилович Михиал Антонович | |
|

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

**Инструментарий:** весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 10 и новее (полезные материалы: [Verilog.docx](https://docs.google.com/document/u/0/d/179Da-G5IvQp12NpAxpFa0MLwzdFsBTVp/edit)). В отчёте нужно указать, какой версией вы пользовались (можно также приложить ссылку на онлайн-платформу). Использовать SystemVerilog допустимо, главное, чтобы код компилился под Icarus 10, 11 или 12. Далее в этом документе Verilog+SystemVerilog обозначается как Verilog. В работе использовался Icarus Verilog 12.

**Описание**

Необходимо построить систему “процессор-кэш-память”. Система должна обладать следующими параметрами (см. таблицу 1).

|  |  |  |  |
| --- | --- | --- | --- |
| **CPU** | | | |
| Команды | CPU → Cache | 0 – **C1\_NOP**  1 – **C1\_READ8**  2 – **C1\_READ16**  3 – **C1\_READ32**  4 – **C1\_INVALIDATE\_LINE**  5 – **C1\_WRITE8**  6 – **C1\_WRITE16**  7 – **C1\_WRITE32** | Команда 4 означает инвалидацию всей кэш-линии, содержащей указанный адрес.  Число в командах означает кол-во бит данных, запрашиваемое данной командой.  Команды, запрашивающие несколько байт, не могут пересекать кэш-линию.  NOP – no operation.  Response – ответ на команду. |
| CPU ← Cache | 0 – **C1\_NOP**  7 – **C1\_RESPONSE** |
| **Кэш (look-through write-back)** | | | |
| Политика вытеснения | | LRU | |
| Команды | Cache  → Mem | 0 – **C2\_NOP**  2 – **C2\_READ\_LINE**  3 – **C2\_WRITE\_LINE** | Команды пишут и читают порциями, равными размеру кэш-линии. |
| Cache  ← Mem | 0 – **C2\_NOP**  1 – **C2\_RESPONSE** |
| Служебные биты | | V (valid), D (dirty) | Если valid установлен в 0, то данная кэш-линия свободна и состояние остальных битов не важно.  dirty означает, что кэш-линия хранит изменённые данные, которые ещё не записаны в память. |

Таблица №1 – общие параметры системы.

Также необходимо решить задачу аналитическим методом, а потом с помощью построенной модели.

**Вариант**

Предоставленные мне параметры (см. таблицу 2):

|  |  |  |
| --- | --- | --- |
| **Кэш (продолжение)** | | |
| Размер кэша | 2 Кб – **CACHE\_SIZE** | Размер полезных данных. |
| Размер кэш-линии | 16 байта – **CACHE\_LINE\_SIZE** | Размер полезных данных. |
| Кол-во бит под тэг адреса | 8 бита – **CACHE\_TAG\_SIZE** |  |
| **Память** | | |
| Размер памяти | 256 Кбайт – **MEM\_SIZE** | (старое значение 128 Кбайт – **MEM\_SIZE**) |

Таблица №2 – параметры (вариант 2)

**Расчёт параметров**

Найдём недостающие параметры системы (см. таблицу 3).

|  |  |
| --- | --- |
| **CACHE\_SIZE** | 16384 |
| **CACHE\_LINE\_SIZE** | 128 |
| **CACHE\_TAG\_SIZE** | 8 |
| **MEM\_SIZE** | 2097152 |
| **CACHE\_LINE\_COUNT** | 128 |
| **CACHE\_WAY** | 2 |
| **CACHE\_SETS\_COUNT** | 64 |
| **CACHE\_SET\_SIZE** | 6 |
| **CACHE\_OFFSET\_SIZE** | 7 |
| **CACHE\_ADDR\_SIZE** | 21 |
| **DATA1\_BUS\_SIZE** | 16 |
| **DATA2\_BUS\_SIZE** | 16 |
| **ADDR1\_BUS\_SIZE** | 14 |
| **ADDR2\_BUS\_SIZE** | 14 |
| **CTR1\_BUS\_SIZE** | 3 |
| **CTR2\_BUS\_SIZE** | 2 |

Таблица №3 – Все параметры системы

**Аналитическое решение задачи**

Для начала хотелось бы рассказать, как работает 2-ух ассоциативный кэш. Ассоциативность – это количество кэш-линий в одном сете. В моём варианте кэш делится н 64 сета, и в каждом сете находится 2 кэш-линии. Адресация происходит так: нам даны номер сета и тег, с помощью номера можно найти первую кэш-линию в сете, умножив номер на ассоциативность. После находим нужную кэш-линию, пройдясь по всем линиям в сете и сравнивая их теги с данным тегом. Также процессор даёт сдвиг, который показывает с какого бита нужно начинать читать или записывать.

Для решения воспользуемся языком Python. Для начала надо прописать все параметры в глобальные переменные. После чего нужно реализовать контроллер памяти. Для этого создадим класс с двумя методами для чтения линии из памяти и для записи линии в память. (см. листинг 1)

class Memory:

def \_\_init\_\_(self, data=[[0] \* CACHE\_LINE\_SIZE] \* CACHE\_SIZE):

self.data = data

self.tag = [0] \* CACHE\_SIZE

for i in range(CACHE\_SIZE):

self.tag[i] = i

def read\_line(self, tag):

for i in range(len(self.data)):

if self.tag[i] == tag:

return self.data[i]

def write\_line(self, tag, line):

for i in range(len(self.data)):

if self.tag[i] == tag:

self.data[i] = line

return

Листинг №1 – класс контроллера памяти

Теперь можно перейти к основной части задания – к кэшу. Для его реализации необходимо хранить теги линий, бит валидности линий, бит актуальности в памяти (dirty), данные для каждой кэш-линии, а также, так как в условии указана политика вытеснения LRU, нужно для каждой кэш-линии сохранять частоту обращения к ней. Чтобы посчитать количество cache hit-ов и cache miss-ов создадим соответствующие счётчики. Напишем код (см. листинг 2).

class Cache:

def \_\_init\_\_(self, data=[[0] \* CACHE\_LINE\_SIZE] \* CACHE\_SIZE):

self.cache\_hits = 0

self.cache\_misses = 0

self.valid = [0] \* CACHE\_LINE\_COUNT

self.dirty = [0] \* CACHE\_LINE\_COUNT

self.data = [[0] \* CACHE\_LINE\_SIZE] \* CACHE\_LINE\_COUNT

self.tag = [0] \* CACHE\_LINE\_COUNT

self.lru = [0] \* CACHE\_LINE\_COUNT

self.mem\_ctr = Memory(data)

Листинг №2 – инициализация кэша

Рассмотрим операцию чтения. Для этого создадим метод, который на вход принимает размер, который нужно прочитать (data\_size). Есть несколько случаев, которые нужно обработать:

1. Искомая кэш-линия находится в кэше, то есть происходит cache hit. Значит нужно также увеличить количество запросов (нужно для LRU)

if self.tag[addr\_set \* CACHE\_WAY] == addr\_tag:

self.cache\_hits += 1

self.lru[addr\_set \* CACHE\_WAY] = 1

self.lru[addr\_set \* CACHE\_WAY + 1] = 0

return self.data[addr\_set \* CACHE\_WAY][addr\_offset:addr\_offset + data\_size]

elif self.tag[addr\_set \* CACHE\_WAY + 1] == addr\_tag:

self.cache\_hits += 1

self.lru[addr\_set \* CACHE\_WAY + 1] = 1

self.lru[addr\_set \* CACHE\_WAY] = 0

return self.data[addr\_set \* CACHE\_WAY + 1][addr\_offset:addr\_offset + data\_size]

Листинг №3 – 1 случай чтения

1. Одна из кэша линий в сете имеет состояние invalid. То есть мы можем занять её. Для этого запишем в неё значение искомой кэш-линии из памяти. Здесь произошёл cache miss. (Незабываем про LRU)

elif self.valid[addr\_set \* CACHE\_WAY] == 0:

self.cache\_misses += 1

self.valid[addr\_set \* CACHE\_WAY] = 1

self.lru[addr\_set \* CACHE\_WAY] = 1

self.lru[addr\_set \* CACHE\_WAY + 1] = 0

self.tag[addr\_set \* CACHE\_WAY] = addr\_tag

self.data[addr\_set \* CACHE\_WAY] = self.mem\_ctr.read\_line(addr\_tag)

return self.data[addr\_set \* CACHE\_WAY][addr\_offset:addr\_offset + data\_size]

elif self.valid[addr\_set \* CACHE\_WAY + 1] == 0:

self.cache\_misses += 1

self.valid[addr\_set \* CACHE\_WAY + 1] = 1

self.lru[addr\_set \* CACHE\_WAY + 1] = 1

self.lru[addr\_set \* CACHE\_WAY] = 0

self.tag[addr\_set \* CACHE\_WAY + 1] = addr\_tag

self.data[addr\_set \* CACHE\_WAY + 1] = self.mem\_ctr.read\_line(addr\_tag)

return self.data[addr\_set \* CACHE\_WAY + 1][addr\_offset:addr\_offset + data\_size]

Листинг №4 – 2 случай чтения

1. Так как все линии в сете заняты, необходимо воспользоваться политикой вытеснения LRU. Для этого найдём кэш-линию с минимальным количеством запросов, если она dirty, то мы должны записать её в память. После чего нужно также прочитать из памяти искомую кэш-линию.

elif self.lru[addr\_set \* CACHE\_WAY] < self.lru[addr\_set \* CACHE\_WAY + 1]:

self.cache\_misses += 1

if self.dirty[addr\_set \* CACHE\_WAY] == 1:

self.mem\_ctr.write\_line(self.tag[CACHE\_WAY \* addr\_set], self.data[addr\_set \* CACHE\_WAY])

self.dirty[addr\_set \* CACHE\_WAY] = 0

self.tag[addr\_set \* CACHE\_WAY] = addr\_tag

self.lru[addr\_set \* CACHE\_WAY] = 1

return self.data[CACHE\_WAY \* addr\_set][addr\_offset:addr\_offset + data\_size]

else:

self.cache\_misses += 1

if self.dirty[addr\_set \* CACHE\_WAY + 1] == 1:

self.mem\_ctr.write\_line(self.tag[CACHE\_WAY \* addr\_set + 1], self.data[CACHE\_WAY \* addr\_set + 1])

self.dirty[addr\_set \* CACHE\_WAY + 1] = 0

self.tag[addr\_set \* CACHE\_WAY + 1] = addr\_tag

self.lru[addr\_set \* CACHE\_WAY + 1] = 1

return self.data[addr\_set \* CACHE\_WAY + 1][addr\_offset:addr\_offset + data\_size]

Листинг №5 – 3 случай чтения

Теперь наш кэш умеет обрабатывать запросы чтения. Запросы на запись практически аналогично, поэтому просто перечислим случаи, которые могут быть:

1. Искомая кэш-линия находится в кэше, то есть происходит cache hit.
2. Искомой кэш-линии нет в кэше, но одна из кэш-линий, которая находится в том же сете, имеет состояние invalid.
3. Искомой кэш-линии нет в кэше, и все кэш-линии в том же сете имеют состояние valid. Как и в read воспользуемся политикой вытеснения LRU.

Теперь можно приступить к написанию задачи, также этот кусок кода будет выполнять роль процессора. Важно обратить внимание на то, что переменные pa, pb, pc, y, x, k, s находятся не в кэше, а в регистрах процессора. Поэтому в кэш нам нужно записать только массивы a, b и c. Для этого подсчитаем сколько кэш-линий будет занимать каждый массив. Для a это будет. 128 линий, для b 240 линий, для c 960 линий.

Теперь можно инициализировать память.

data = [[0] \* CACHE\_LINE\_SIZE] \* CACHE\_SIZE

n = 0

m = 0

k = 0

for I in range(M):

for j in range(K):

if k >= CACHE\_LINE\_SIZE:

m += 1

k = 0

number = bin(random.randint(-128, 127) & 127)

binary\_form\_of\_number = list(map(int, number[2::])) # int list equals binary form of random number

for l in range(0, 8):

if l < len(binary\_form\_of\_number):

data[m][k] = binary\_form\_of\_number[l]

k += 1

# a: values in cache\_line = 128 / 8 = 16, cache\_lines\_count = K \* M / 16 = 128 lines

# tag = 0...128

n = 0

k = 0

m = 0

for I in range(K):

for j in range(N):

if n >= CACHE\_LINE\_SIZE:

k += 1

n = 0

number = bin(random.randint(-32768, 32767) & 32767)

binary\_form\_of\_number = list(map(int, number[2::]))

for l in range(0, 16):

if l < len(binary\_form\_of\_number):

data[129 + k][n] = binary\_form\_of\_number[l]

n += 1

# b: values in cache\_line = 128 / 16 = 8, cache\_lines\_count = N \* K / 8 = 240 lines

# tag = 129..369

n = 0

m = 0

k = 0

for I in range(M):

for j in range(N):

if n >= CACHE\_LINE\_SIZE:

m += 1

n = 0

number = bin(random.randint(-2147483648, 2147483647) & 2147483647)

binary\_form\_of\_number = list(map(int, number[2::]))

for l in range(0, 32):

if l < len(binary\_form\_of\_number):

data[370 + m][n] = binary\_form\_of\_number[l]

n += 1

# c: values in cache\_line = 128 / 32 = 4, cache\_lines\_count = M \* N / 4 = 960 lines

# tag = 370..1330

Листинг №6 – инициализация памяти

Теперь надо просто переписать код из условия с некоторыми изменениями. Теперь надо обращаться к кэшу причём в качестве номера сета указывать pa % CACHE\_SETS\_COUNT, а в качестве тега pa.

a = 0

b = 0

c = 0

for y in range(M):

x = 0

while x < N \* 32:

pb = 0

s = 0

k = 0

m = 0

while k < K \* 8 and m < K \* 16:

# print(k \* 8)

if a - pa \* CACHE\_LINE\_SIZE >= 0:

pa += 1

if b - pb \* CACHE\_LINE\_SIZE >= 0:

pb += 1

#print(k \* 8)

a\_n = ''.join(str(i) for i in cache.read(pa % CACHE\_SETS\_COUNT, pa, a % CACHE\_LINE\_SIZE, 8))

b\_n = ''.join(str(i) for i in cache.read(pb % CACHE\_SETS\_COUNT, 129 + pb, b % CACHE\_LINE\_SIZE, 16))

s += int(a\_n, 2) + int(b\_n, 2)

k += 8

m += 16

a += 8

b += 16

data = [0] \* 32

binary\_form\_of\_number = list(map(int, bin(s & 2147483647)[2::]))

for l in range(32):

if l < len(binary\_form\_of\_number):

data[l] = binary\_form\_of\_number[l]

if c - pc \* CACHE\_LINE\_SIZE >= 0:

pc += 1

cache.write(pc % CACHE\_SETS\_COUNT, 370 + pc, c % CACHE\_LINE\_SIZE, data)

x += 32

c += 32

cache.cache\_info()

Листинг №7 – задача на языке Python

После запуска программы мы получим такой результат:

|  |  |
| --- | --- |
| Requests | 249600 |
| Cache hits | 240635 |
| Cache misses | 8965 |
| Cache hit ratio | 0.964082532051282 |
| Cache miss ratio | 0.035917467948717946 |
| Tick count | 3373598 |

Таблица №5 – Результаты аналитического решения

Получается наш Cache hit ration = 96.4%. А количество тактов равно 3373598.

**Моделирование заданной системы на Verilog**

Сначала нужно определить наш модуль cache

module cache #(

parameter MEM\_SIZE = 2097152,

parameter CACHE\_SIZE = 16384,

parameter CACHE\_LINE\_SIZE = 128,

parameter CACHE\_LINE\_COUNT = 128,

parameter CACHE\_WAY = 2,

parameter CACHE\_SETS\_COUNT = 64,

parameter CACHE\_TAG\_SIZE = 8,

parameter CACHE\_SET\_SIZE = 6,

parameter CACHE\_OFFSET\_SIZE = 7,

parameter CACHE\_ADDR\_SIZE = 21,

parameter DATA1\_BUS\_SIZE = 16,

parameter DATA2\_BUS\_SIZE = 16,

parameter ADDR1\_BUS\_SIZE = 14,

parameter ADDR2\_BUS\_SIZE = 14,

parameter CTR1\_BUS\_SIZE = 3,

parameter CTR2\_BUS\_SIZE = 2,

parameter C1\_NOP = 0,

parameter C1\_READ8 = 1,

parameter C1\_READ16 = 2,

parameter C1\_READ32 = 3,

parameter C1\_INVALIDATE\_LINE = 4,

parameter C1\_WRITE8 = 5,

parameter C1\_WRITE16 = 6,

parameter C1\_WRITE32 = 7,

parameter C1\_RESPONSE = 7,

parameter C2\_NOP = 0,

parameter C2\_READ\_LINE = 2,

parameter C2\_WRITE\_LINE = 3,

parameter C2\_RESPONSE = 1

) (

input CLK,

input RESET,

input C\_DUMP,

input M\_DUMP,

input wire[ADDR1\_BUS\_SIZE-1:0] A1,

inout wire[DATA1\_BUS\_SIZE-1:0] D1,

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

);

Листинг №8 – определения модуля cache на Verilog

Так же, как и в аналитическом решении надо инициализировать переменных. Здесь практически всё то же самое, кроме шин и регистров для них.

int cache\_hits\_count = 0;

int cache\_miss\_count = 0;

bit valid[CACHE\_LINE\_COUNT];

bit dirty[CACHE\_LINE\_COUNT];

reg[CACHE\_LINE\_SIZE-1:0] data[CACHE\_LINE\_COUNT];

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

bit lru[CACHE\_LINE\_COUNT];

mem\_ctr mem(CLK, RESET, M\_DUMP, A2, D2, C2);

reg[ADDR1\_BUS\_SIZE-1:0] a1='bz; // регистры, чтобы можно было посылать данные по шине

reg[DATA1\_BUS\_SIZE-1:0] d1='bz;

reg[CTR1\_BUS\_SIZE-1:0] c1='bz;

reg[ADDR2\_BUS\_SIZE-1:0] a2='bz;

reg[DATA2\_BUS\_SIZE-1:0] d2='bz;

reg[CTR2\_BUS\_SIZE-1:0] c2='bz;

assign A1 = a1; // Здесь мы устанавливаем соответствие между регистрами и шинами

assign D1 = d1;

assign C1 = c1;

assign A2 = a2;

assign D2 = d2;

assign C2 = c2;

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

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

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

int one\_tick = 2;

int index1 = 0;

int index2 = 0;

int cur\_command = 0;

Листинг №9 – инициализация переменных на Verilog

Как происходит чтение в Verilog? Процессор посылает запрос по шине, и, как только он был получен кэшом, кэш перехваытвает владение шиной и смотрит существует ли линия с адресом, который послал процессор по шине A1. Если такая линия есть, то произошёл cache hit, мы должны обновить LRU,

И прочитать данные с линии.

Если же линия не нашлась в кэше, то мы ищем в нашем сете линию с флагом valid равным 0. Если такая есть, то мы обновляем информацию об этой кэш линии (valid, tag), и делаем запрос к mem\_ctr (контроллер памяти) на получение данных этой кэш-линии.

Если и invalid кэш-линий е оказалось, то мы пользуемся политикой вытеснения LRU. LRU определяет какую кэш-линию надо выкинуть: кэш-линия, к которой мы дольше всего не обращались выкидывается. Пользуясь LRU, мы выбираем линию, в которую мы запишем новую кэш-линию. Для этого, если линия было до этого обновлена в кэше и не было записана в память, то мы записываем её в память (Write back). Потом мы делаем ещё один запрос к памяти, но теперь, чтобы прочитать линию с новым адресом.

В конце мы отправляем процессору RESPONSE и данные вместе с ответом.

Запись происходит аналогичным образом, только теперь в конце мы пишем в кэш линию новые данные, меняем бит dirty на 1 и отправляем процессору RESPONSE.

**Воспроизведение задачи на Verilog.**

Теперь напишем mem\_ctr, который умеет обробатывать два запроса. На C2\_READ\_LINE мы будем искать по всем кэш-линиям нужную, после чего отправлять данные по шине d2, заранее передав владение ею mem\_ctr.

Аналогично происходит запись в память. После любой из этих двух операций мы отправляем C2\_RESPONSE.

Также реализуем процессор и саму программу, между каждой командой поставим задержку в 300 тактов (Verilog тактов). Делается процессор аналогично процессору на Python, кроме того, что теперь мы посылаем запросы по шинам, а не через аргументы функций.

Запустив программу, получим такой результат.

|  |  |
| --- | --- |
| Requests | 248533 |
| Cache hits | 239567 |
| Cache misses | 8966 |
| Cache hit ratio | 0.963924 |
| Ticks | 3467350 |

Таблица №6 – Результаты на Verilog

**Сравнение полученных результатов.**

Как видим результат практически сошёлся. Это разность результатов связана с тем, что на Verilog были неправильны проставлены задержки в некоторых местах.

`include "mem\_ctr.sv"

module cache #(

parameter MEM\_SIZE = 2097152,

parameter CACHE\_SIZE = 16384,

parameter CACHE\_LINE\_SIZE = 128,

parameter CACHE\_LINE\_COUNT = 128,

parameter CACHE\_WAY = 2,

parameter CACHE\_SETS\_COUNT = 64,

parameter CACHE\_TAG\_SIZE = 8,

parameter CACHE\_SET\_SIZE = 6,

parameter CACHE\_OFFSET\_SIZE = 7,

parameter CACHE\_ADDR\_SIZE = 21,

parameter DATA1\_BUS\_SIZE = 16,

parameter DATA2\_BUS\_SIZE = 16,

parameter ADDR1\_BUS\_SIZE = 14,

parameter ADDR2\_BUS\_SIZE = 14,

parameter CTR1\_BUS\_SIZE = 3,

parameter CTR2\_BUS\_SIZE = 2,

parameter C1\_NOP = 0,

parameter C1\_READ8 = 1,

parameter C1\_READ16 = 2,

parameter C1\_READ32 = 3,

parameter C1\_INVALIDATE\_LINE = 4,

parameter C1\_WRITE8 = 5,

parameter C1\_WRITE16 = 6,

parameter C1\_WRITE32 = 7,

parameter C1\_RESPONSE = 7,

parameter C2\_NOP = 0,

parameter C2\_READ\_LINE = 2,

parameter C2\_WRITE\_LINE = 3,

parameter C2\_RESPONSE = 1

) (

input CLK,

input RESET,

input C\_DUMP,

input M\_DUMP,

input wire[ADDR1\_BUS\_SIZE-1:0] A1,

inout wire[DATA1\_BUS\_SIZE-1:0] D1,

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

);

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

inout wire[DATA2\_BUS\_SIZE-1:0] D2;

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

int cache\_hits\_count = 0;

int cache\_miss\_count = 0;

bit valid[CACHE\_LINE\_COUNT];

bit dirty[CACHE\_LINE\_COUNT];

reg[CACHE\_LINE\_SIZE-1:0] data[CACHE\_LINE\_COUNT];

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

bit lru[CACHE\_LINE\_COUNT];

mem\_ctr mem(CLK, RESET, M\_DUMP, A2, D2, C2);

reg[ADDR1\_BUS\_SIZE-1:0] a1='bz;

reg[DATA1\_BUS\_SIZE-1:0] d1='bz;

reg[CTR1\_BUS\_SIZE-1:0] c1='bz;

reg[ADDR2\_BUS\_SIZE-1:0] a2='bz;

reg[DATA2\_BUS\_SIZE-1:0] d2='bz;

reg[CTR2\_BUS\_SIZE-1:0] c2='bz;

assign A1 = a1;

assign D1 = d1;

assign C1 = c1;

assign A2 = a2;

assign D2 = d2;

assign C2 = c2;

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

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

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

int one\_tick = 2;

int index1 = 0;

int index2 = 0;

int cur\_command = 0;

int count = 0;

initial begin

//$monitor("%b", D1);

$dumpfile("dump\_cache.vcd");

$dumpvars(0, cache);

$dumpoff;

end

function int get\_count();

return count;

endfunction

function void reset();

cache\_hits\_count = 0;

cache\_miss\_count = 0;

for (int i = 0; i < CACHE\_LINE\_COUNT; i++) begin

valid[i] = 0;

dirty[i] = 0;

data[i] = 0;

tag[i] = 'bz;

lru[i] = 0;

end

endfunction

function void cache\_info();

$display(count);

$display("Requests = %d", cache\_miss\_count + cache\_hits\_count);

$display("Cache hits = %d", cache\_hits\_count);

$display("Cache misses = %d", cache\_miss\_count);

$display("Cache hit ratio = %f", (1.0 \* cache\_hits\_count) / (1.0 \* (cache\_hits\_count + cache\_miss\_count)));

endfunction

task read\_from\_cache(int index, int data\_size);

//$display(data\_size);

case(data\_size)

8: begin

d1[7:0] = data[index][addr\_offset +: 8];

//$display(D1);

end

16: begin

d1 = data[index][addr\_offset +: 16];

end

32: begin

d1 = data[index][addr\_offset +: 16];

#one\_tick d1 = data[index][addr\_offset+16+:16];

end

endcase

endtask

task write\_in\_cache(int index, int data\_size);

case(data\_size)

8: begin

data[index][addr\_offset +: 8] = D2[7:0];

//$display(123);

end

16: begin

data[index][addr\_offset +: 16] = D2;

end

32: begin

data[index][addr\_offset +: 16] = D2;

#one\_tick data[index][addr\_offset+16 +: 16] = D2;

end

endcase

dirty[index] = 1;

endtask

task write\_data\_in\_mem(int index);

c2 = C2\_WRITE\_LINE;

a2 = {addr\_tag, addr\_set};

//wait (C2 == C2\_RESPONSE);

for (int i = 0; i < CACHE\_LINE\_SIZE; i += DATA2\_BUS\_SIZE) begin

#(i > 0 ? one\_tick : 0) d2 = data[index][i +: DATA2\_BUS\_SIZE];

end

#200 c2 = 'bz;

dirty[index] = 0;

d2 = 'bz;

count += 100;

endtask

task read\_data\_from\_mem(int index);

if (valid[index] == 1 && dirty[index] == 1) begin

write\_data\_in\_mem(index);

end

c2 = C2\_READ\_LINE;

a2 = {addr\_tag, addr\_set};

#200 c2 = 'bz;

// $display("ok");

//wait (C2 == C2\_RESPONSE);

// $display("ok");

valid[index] = 1;

tag[index] = addr\_tag;

lru[index] = 1;

for (int i = 0; i < CACHE\_LINE\_SIZE; i += DATA2\_BUS\_SIZE) begin

// $display("Take = %0b", D2);

#(i > 0 ? one\_tick : 0) data[index][i +: DATA2\_BUS\_SIZE] = D2;

end

count += 100;

//$display("%b", data[index]);

endtask

task read(int data\_size);

index1 = addr\_set \* CACHE\_WAY;

index2 = addr\_set \* CACHE\_WAY + 1;

if (valid[index1] == 1 && tag[index1] == addr\_tag) begin

lru[index1] = 1;

lru[index2] = 0;

cache\_hits\_count++;

c1 = C1\_RESPONSE;

count += 6;

#12 read\_from\_cache(index1, data\_size);

end

else if (valid[index2] == 1 && tag[index2] == addr\_tag) begin

lru[index2] = 1;

lru[index1] = 0;

cache\_hits\_count++;

c1 = C1\_RESPONSE;

count += 6;

#12 read\_from\_cache(index2, data\_size);

end

else if (valid[index1] == 0) begin

count += 4;

cache\_miss\_count++;

lru[index2] = 0;

#8 read\_data\_from\_mem(index1);

//$display("1234");

c1 = C1\_RESPONSE;

read\_from\_cache(index1, data\_size);

end

else if (valid[index2] == 0) begin

count += 4;

cache\_miss\_count++;

lru[index1] = 0;

#8 read\_data\_from\_mem(index2);

c1 = C1\_RESPONSE;

read\_from\_cache(index2, data\_size);

end

else if (lru[index1] < lru[index2]) begin

count += 4;

cache\_miss\_count++;

#8 read\_data\_from\_mem(index1);

c1 = C1\_RESPONSE;

read\_from\_cache(index1, data\_size);

end

else begin

count += 4;

cache\_miss\_count++;

#8 read\_data\_from\_mem(index2);

c1 = C1\_RESPONSE;

read\_from\_cache(index2, data\_size);

end

#2 c1 = 'bz;

endtask

task write(int data\_size);

index1 = addr\_set \* CACHE\_WAY;

index2 = addr\_set \* CACHE\_WAY + 1;

if (valid[index1] == 1 && tag[index1] == addr\_tag) begin

count += 6;

#12 cache\_hits\_count++;

lru[index1] = 1;

lru[index2] = 0;

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index1, data\_size);

end

else if (valid[index2] == 1 && tag[index2] == addr\_tag) begin

count += 6;

#12 cache\_hits\_count++;

lru[index2] = 1;

lru[index1] = 0;

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index2, data\_size);

end

else if (valid[index1] == 0) begin

count += 4;

#8 cache\_miss\_count++;

lru[index2] = 0;

read\_data\_from\_mem(index1);

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index1, data\_size);

end

else if (valid[index2] == 0) begin

count += 4;

#8 cache\_miss\_count++;

lru[index1] = 0;

read\_data\_from\_mem(index2);

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index2, data\_size);

end

else if (lru[index1] < lru[index2]) begin

count += 4;

#8 cache\_miss\_count++;

read\_data\_from\_mem(index1);

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index1, data\_size);

end

else begin

count += 4;

#8 cache\_miss\_count++;

read\_data\_from\_mem(index2);

c1 = C1\_RESPONSE;

#1 write\_in\_cache(index2, data\_size);

end

#2 c1 = 'bz;

endtask

task invalidate\_line;

static int index1 = addr\_set \* CACHE\_WAY;

static int index2 = addr\_set \* CACHE\_WAY + 1;

if (tag[index1] == addr\_tag) begin

if (dirty[index1] == 1) begin

write\_data\_in\_mem(index1);

end

c1 = C1\_RESPONSE;

valid[index1] = 0;

tag[index1] = 0;

data[index1] = 0;

end

else if (tag[index2] == addr\_tag) begin

if (dirty[index2] == 1) begin

write\_data\_in\_mem(index2);

end

c1 = C1\_RESPONSE;

valid[index2] = 0;

tag[index2] = 0;

data[index2] = 0;

end

#2 c1 = 'bz;

endtask

always @(posedge CLK or posedge RESET or posedge C\_DUMP) begin

//y(123);

//$display("%d", C1);

//$display(C1);

if (RESET) begin

reset();

end

if (C\_DUMP) begin

$dumpon;

end

else begin

$dumpoff;

end

if (C1 != 0) begin

cur\_command = C1;

// $display("time = %d C1 = %d", $time(), C1);

addr\_tag = A1[7:0];

addr\_set = A1[13:8];

#one\_tick addr\_offset = A1[6:0];

case (cur\_command)

C1\_READ8: read(8);

C1\_READ16: read(16);

C1\_READ32: read(32);

C1\_INVALIDATE\_LINE: invalidate\_line();

C1\_WRITE8: write(8);

C1\_WRITE16: write(16);

C1\_WRITE32: write(32);

endcase

// $display("%0d %0d", $time(), C1);

//$display($time());

end

end

endmodule : cache

  Листинг №10 – another\_cache.sv

module mem\_ctr #(

parameter \_SEED = 225526,

parameter MEM\_SIZE = 2097152,

parameter CACHE\_LINE\_SIZE = 128,

parameter CACHE\_LINE\_COUNT = 16384,

parameter MEM\_TAG\_SIZE = 14,

parameter MEM\_OFFSET\_SIZE = 7,

parameter ADDR2\_BUS\_SIZE = 14,

parameter DATA2\_BUS\_SIZE = 16,

parameter CTR2\_BUS\_SIZE = 2,

parameter C2\_NOP = 0,

parameter C2\_READ\_LINE = 2,

parameter C2\_WRITE\_LINE = 3,

parameter C2\_RESPONSE = 1

) (

input CLK,

input RESET,

input M\_DUMP,

input wire[ADDR2\_BUS\_SIZE-1:0] A2,

inout wire[DATA2\_BUS\_SIZE-1:0] D2,

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

);

int SEED = 225526;

reg[DATA2\_BUS\_SIZE-1:0] d2='bz;

reg[CTR2\_BUS\_SIZE-1:0] c2='bz;

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

bit[CACHE\_LINE\_SIZE-1:0] data[CACHE\_LINE\_COUNT];

assign D2 = d2;

assign C2 = c2;

byte command;

reg[MEM\_TAG\_SIZE-1:0] addr\_tag;

// program variables

localparam M = 64;

localparam N = 60;

localparam K = 32;

reg[7:0] a;

reg[15:0] b;

reg[31:0] c;

function void random\_init();

for (int i = 0; i < CACHE\_LINE\_COUNT; i++) begin

data[i] = $random(SEED);

tag[i] = i;

end

endfunction

function void init();

int n = 0;

int m = 0;

int k = 0;

for (int i = 0; i < M; i++) begin

for (int j = 0; j < K; j++) begin

if (k >= 128) begin

m++;

k = 0;

end

a = $random(SEED);

data[m][k +: 8] = a;

k += 8;

end

end

n = 0;

k = 0;

m = 0;

for (int i = 0; i < K; i++) begin

for (int j = 0; j < N; j++) begin

if (m >= 128) begin

k++;

m = 0;

end

b = $random(SEED);

data[k][m +: 16] = b;

m += 16;

end

end

n = 0;

k = 0;

m = 0;

for (int i = 0; i < M; i++) begin

for (int j = 0; j < N; j++) begin

if (n >= 128) begin

m++;

n = 0;

end

c = $random(SEED);

data[m][n +: 32] = c;

n += 32;

end

end

endfunction

initial begin // generate random memory

random\_init();

$dumpfile("dump\_mem\_ctr.vcd");

$dumpvars(0, mem\_ctr);

$dumpoff;

end

always @(posedge CLK or posedge RESET) begin

if (M\_DUMP) begin

$dumpon;

end

else begin

$dumpoff;

end

if (RESET) begin

random\_init();

end

else begin

command = C2;

//$display(command);

if (command == C2\_READ\_LINE) begin

addr\_tag = A2;

#200

c2 = C2\_RESPONSE;

for (int i = 0; i < CACHE\_LINE\_COUNT; i++) begin

if (tag[i] == addr\_tag) begin

for (int j = 0; j < CACHE\_LINE\_SIZE; j += DATA2\_BUS\_SIZE) begin

//$display("Sent: %0d", $time());

//$display("%d %d", i, j);

#(j > 0 ? 2 : 0) d2 = data[i][j +:DATA2\_BUS\_SIZE];

end

end

end

#2 c2 = 'bz;

d2 = 'bz;

end

end

end

always @(negedge CLK) begin

if (command == C2\_WRITE\_LINE) begin

//$display(1234);

addr\_tag = A2;

for (int i = 0; i < CACHE\_LINE\_COUNT; i++) begin

if (tag[i] == addr\_tag) begin

for (int j = 0; j < CACHE\_LINE\_SIZE; j += DATA2\_BUS\_SIZE) begin

//$display("Sent: %0d", $time());

//$display("%d %d", i, j);

// #(j > 0 ? 2 : 0) data[i][j +:DATA2\_BUS\_SIZE] = D2;

end

end

end

#200 c2 = C2\_RESPONSE;

#2 c2 = 'bz;

d2 = 'bz;

//$display(123);

end

end

endmodule : mem\_ctr

Листинг №11 – mem\_ctr.sv

`include "another\_cache.sv"

module cache\_tb #(parameter \_SEED = 225526) ();

input wire CLK;

input wire RESET;

input wire[13:0] A1;

inout wire[15:0] D1;

inout wire[2:0] C1;

input wire C\_DUMP;

input wire M\_DUMP;

cache ch(CLK, RESET, C\_DUMP, M\_DUMP, A1, D1, C1);

int SEED = \_SEED;

reg clk=0;

reg[2:0] c1='bz;

reg[13:0] a1='bz;

reg[15:0] d1='bz;

assign CLK=clk;

assign C1=c1;

assign A1=a1;

assign D1=d1;

localparam M = 64;

localparam N = 60;

localparam K = 32;

int pa = 0;

int pb = 0;

int pc = 0;

int s = 0;

reg[7:0] a;

reg[15:0] b;

reg[31:0] c;

int i = 0;

int a\_i = 0;

int b\_i = 0;

int c\_i = 0;

int count = 0;

initial begin

$display("Start");

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

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

pb = 0;

s = 0;

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

if (a\_i - pa \* 128 >= 0) begin

pa++;

end

if (b\_i - pb \* 128 >= 0) begin

pb++;

end

d1 = 'bz;

c1 = 1;

a1[7:0] = pa;

a1[13:8] = pa % 64;

#2 a1[6:0] = a\_i % 128;

c1 = 'bz;

//count += 1;

#300

a = D1[7:0];

#2 d1 = 'bz;

c1 = 2;

a1[7:0] = pb + 129;

a1[13:8] = pb % 64;

#2 a1[6:0] = b\_i % 128;

c1 = 'bz;

// count += 1;

#300

b = D1[15:0];

#2 d1 = 'bz;

s += a + b;

a\_i += 8;

b\_i += 16;

count += 5 + 1 + 1 + 1;

end

c = s;

if (c\_i - pc \* 128 >= 0) begin

pc++;

end

#300 c1 = 7;

a1[7:0] = pc + 370;

a1[13:8] = pc % 64;

#2 a1[6:0] = c\_i % 128;

c1 = 'bz;

#300

c\_i += 32;

count += 2;

end

count += 1;

$display("$d", y);

end

$display(count + ch.get\_count());

ch.cache\_info();

$finish;

end

always #1 if ($time() < 100000000) begin

clk = ~clk;

end

endmodule

Листинг №12 – testbench.sv