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

**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 |

**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. 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. 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

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

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

Листинг №3 – 1 случай чтения

1. Она из кэш линий в сете invalid. То есть мы должны записать в неё значение из памяти. Значит произошёл cache miss

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

self.cache\_misses += 1

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

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.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.