|  |  |  |
| --- | --- | --- |
| ***Лабораторная работа №2*** | ***M3137*** | ***2022*** |
| ***Моделирование схем в Verilog*** | ***Булавко Тимофей***  ***Евгеньевич*** | |
|

**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.

**Задача:** построить кэш и смоделировать систему на языке описания Verilog. Определить процент попаданий (число попаданий к общему числу обращений) для кэша и общее время (в тактах), затраченное на выполнение заданной функции. Параметры кэша и памяти заданы в моем варианте (3).

***Параметры системы***

*По условию мне данный были следующие параметры кэша:*

***CACHE\_SIZE* = 2 кб**

***CACHE\_LINE\_SIZE* = 16 байта**

***CACHE\_TAG\_SIZE* = 8 бита**

***MEM\_SIZE* = 256 кб**

*Недостающий параметры были вычислены по следующим уравнениям:*

***CACHE\_LINE\_COUN*T = *CACHE\_SIZE* / *CACHE\_LINE\_SIZE***

***CACHE\_OFFSET\_SIZE* = log2(*CACHE\_LINE\_SIZE*)**

***CACHE\_ADDR\_SIZE* = log2(*MEM\_SIZE*)**

***CACHE\_SET\_SIZE* = *CACHE\_ADDR\_SIZE* - *CACHE\_TAG\_SIZE* - *CACHE\_OFFSET\_SIZE***

***CACHE\_SETS\_COUNT* = 2^*CACHE\_SET\_SIZE***

***CACHE\_WAY* = *CACHE\_LINE\_COUNT* / *CACHE\_SETS\_COUNT***

*Для удобства подсчета я перевел все данные в биты:*

***CACHE\_SIZE* = 16 384**

***CACHE\_LINE\_SIZE* = 128**

***CACHE\_TAG\_SIZE* = 8**

***MEM\_SIZE* = 2 097 152**

***CACHE\_LINE\_COUNT* = 128**

***CACHE\_OFFSET\_SIZE* = 7**

***CACHE\_ADDR\_SIZE* = 21**

***CACHE\_SET\_SIZE* = 6**

***CACHE\_SETS\_COUNT* = 64**

***CACHE\_WAY* = 2**

*Данные по условию параметры шин:*

***DATA1\_BUS\_SIZE*** **= 16**

***DATA2\_BUS\_SIZE* = 16**

*Остальные параметры я вычислил по следующим уравнениям:*

***ADDR1\_BUS\_SIZE* = max (*CACHE\_TAG\_SIZE* + *CACHE\_SET\_SIZE*, *CACHE\_OFFSET\_SIZE*)**

***ADDR2\_BUS\_SIZE* = *CACHE\_TAG\_SIZE* + *CACHE\_SET\_SIZE***

***CTR1\_BUS\_SIZE* = 3** *(так как коды команд от 0 до 7)*

***CTR2\_BUS\_SIZE* = 2** *(так как коды команд от 0 до 3)*

***Принцип работы системы***

***“процессор-кэш-память”***

Процессор подает команды на кэш, в команде присутствует ***tag*** и что от него требуется (прочитать или записать какое-то количество битов). Кэш проверяет есть ли у него в памяти нужные значения, если они отсутствуют, то он отправляет запрос в память на получение новой кэш-линии. После эта линия записывает в кэш и подается процессору как ответ на команду, вытеснение старых кэш-линий работает по методу ***LRU*** (алгоритм, при котором вытесняются значения, которые дольше всего не запрашивались).

***Аналитическое решение задачи***

Задачу я решал, написав код на языке высокого уровня ***Java***, симулирующего работу системы. В моем коде присутствуют 4 класса: ***Cpu***, ***Cache***, ***Mem*** и ***Counter*** (считает такты) – которые так называются по довольно понятным причинам. Для удобства понимания я начну объяснение с конца.

***Counter***

***Counter*** – самая простая часть кода, его задача считать такты и возвращать их количество. Добавление тактов происходит в методе ***add***, а ответ возвращает ***getAnswer***.

public void add(double x) {  
 cnt += x;  
}  
public double getAnswer() {  
 return cnt;  
}

***Mem***

Все данные памяти в моем коде хранятся в трёхмерном массиве, это симулирует разбиение нашей памяти на блоки (***tag***), которые разбиты на линии (***set***), ну и линии разбитые на ячейки памяти (***offset***).

static final int *CACHE\_TAG\_SIZE* = 8;  
static final int *CACHE\_SET\_SIZE* = 6;  
static final int *CACHE\_OFFSET\_SIZE* = 7;  
static int[][][] *MEM\_ARRAY* = new int[(int)Math.*pow*(2, *CACHE\_TAG\_SIZE*)]  
 [(int)Math.*pow*(2, *CACHE\_SET\_SIZE*)][(int)Math.*pow*(2, *CACHE\_OFFSET\_SIZE*)];

В память данные можно записать и прочитать, чтение происходит в методе ***READ\_LINE***.

Изначально по шине ***А2*** поступает информация о тегах за **1** такт, после из памяти получаем информацию о линии(***s***) и возвращаем ее по шине ***D2*** за ***CACHE\_LINE\_SIZE*/16** тактов.

public String READ\_LINE(int tag, int set) {  
 counter.add(1);  
 StringBuilder s = new StringBuilder();  
 for (int i = 0; i < *MEM\_ARRAY*[tag][set].length; i++) {  
 s.append((char)(*MEM\_ARRAY*[tag][set][i] + '0'));  
 }  
 counter.add(100);  
 counter.add(128/16);  
 return String.*valueOf*(s);  
}

Запись в память происходит в методе ***WRITE\_LINE***.

Информация о тегах поступает за **1** такт, а информация о линии которую надо записать за ***CACHE\_LINE\_SIZE*/16** , так как информация по шинам поступает одновременно передача информации о тегах в посчете не учитывается. Далее ***line*** записывается в память.

public void WRITE\_LINE(int tag, int set, String line) {  
 //counter.add(1);  
 counter.add(128/16);  
 for (int i = 0; i < line.length(); i++) {  
 *MEM\_ARRAY*[tag][set][i] = (int)(line.charAt(i) - '0');  
 }  
 counter.add(100);  
}

***Cache***

В кэш поступают запросы от процессора вида прочитать или записать какое-то количество бит, в случае кэш-попадания кэш возвращает ответ. В противном случае кэш отправляет запрос памяти на получение информации, и происходит перезапись кэш-линии. После этого из только что записанной линии получаем ответ и отправляем его процессору.

При вызове метода ***answer*** кэш возвращает процент кэш-попаданий к общему вызову кэша.

public double answer() {  
 double x = *cntP*;  
 double y = *cntA*;  
 return x / y;  
}

***Значение переменных:***

***CACHE\_VALID*** – хранит **1** или **0**, что означает в кэш-линии хранится нужная информация или мусор, соответственно.

***CACHE\_DIRTY*** – хранит **1** или **0**, что означает была изменена или не была изменена кэш-линия соответственно.

***CACHE\_LINE\_USABILITY*** – хранит **0** или **1**, что означает – дольше всех не использовалась строка в **0** или **1** потоке соответственно.

***CACHE\_TAG*** – хранит тэги кэш линий.

***CACHE\_LINE*** – хранит значение кэш линий.

***cntP*** – количество кэш попаданий.

***cntA*** – количество обращений к кэшу.

static int[][] *CACHE\_VALID* = new int[*CACHE\_WAY*][*CACHE\_SETS\_COUNT*];  
static int[][] *CACHE\_DIRTY* = new int[*CACHE\_WAY*][*CACHE\_SETS\_COUNT*];  
static int[][][] *CACHE\_TAG* = new int[*CACHE\_WAY*][*CACHE\_SETS\_COUNT*][*CACHE\_TAG\_SIZE*];  
static int[][][] *CACHE\_LINE* = new int[*CACHE\_WAY*][*CACHE\_SETS\_COUNT*][*CACHE\_LINE\_SIZE*];  
static int[] *CACHE\_LINE\_USABILITY* = new int[*CACHE\_SETS\_COUNT*];  
static int *cntP* = 0;  
static int *cntA* = 0;

Изначально разберем вспомогательные методы.

Метод ***parseInt*** из битовой строки возвращает число.

private int parseInt(String s) {  
 int x = 0;  
 for (int i = 0; i < s.length(); i++) {  
 x = x \* 2 + (int)(s.charAt(i) - '0');  
 }  
 return x;  
}

Метод ***check*** проверяет по way и set кэш-линии, является ли она той которую мы ищем.

private boolean check(int way, int set, String tag) {  
 if (*CACHE\_VALID*[way][set] == 0) {  
 return false;  
 }  
 for (int i = 0; i < tag.length(); i++) {  
 if ((int)(tag.charAt(i) - '0') != *CACHE\_TAG*[way][set][i]) {  
 return false;  
 }  
 }  
 return true;  
}

Метод ***set*** устанавливает в определенный поток линию и позицию какое-то количество битов записанных в ***line***, и помечает в ***CACHE\_DIRTY*** что кэш-линия была изменена.

private void set(int way, int set, int offset, int bits, String line) {  
 *CACHE\_DIRTY*[way][set] = 1;  
 for (int i = 0; i < bits; i++) {  
 *CACHE\_LINE*[way][set][i + offset] = (int)(line.charAt(i));  
 }  
}

В методе ***get*** всё почти точно так же, как и в ***set***, только он возвращает какое-то количество битов кэш-линии с определенного потока, линии и позиции.

private int get(int way, int set, int offset, int bits) {  
 StringBuilder x = new StringBuilder();  
 for (int i = 0; i < bits; i++) {  
 x.append((char)(*CACHE\_LINE*[way][set][i + offset]));  
 }  
 return parseInt(String.*valueOf*(x));  
}

Метод ***getLine*** возвращает всю кэш-линию из определенного потока.

private String getLine(int way, int set) {  
 StringBuilder x = new StringBuilder();  
 for (int i = 0; i < *CACHE\_LINE*[way][set].length; i++) {  
 x.append((char)(*CACHE\_LINE*[way][set][i] + '0'));  
 }  
 return String.*valueOf*(x);  
}

Метод ***getTag*** возвращает ***tag*** кэш-линии.

private int getTag(int way, int set) {  
 int x = 0;  
 for (int i = 0; i < *CACHE\_TAG*[way][set].length; i++) {  
 x = x \* 2 + *CACHE\_TAG*[way][set][i];  
 }  
 return x;  
}

В методе ***getNewLine*** при кэш промахе происходит общение кэша и памяти, и строка, которая не использовалась дольше всего заменяется на нужную прочитанную из памяти.

Изначально нам нужно проверить, если заменяемая кэш-линия изменялась нужно записать ее в память, после этого мы уже можем эту кэш линию поменять так как она точно находится в памяти. После замены метод возвращает ***way*** (поток, в который была записана кэш-линия)

private int getNewLine(String tag, String setString) {  
 String line = mem.READ\_LINE(parseInt(tag), parseInt(setString));  
 int set = parseInt(setString);  
 int way = *CACHE\_LINE\_USABILITY*[set];  
 if (*CACHE\_DIRTY*[way][set] == 1) {  
 mem.WRITE\_LINE(getTag(way, set), set, getLine(way, set));  
 }  
 *CACHE\_LINE\_USABILITY*[set] = 1 - way;  
 *CACHE\_VALID*[way][set] = 1;  
 *CACHE\_DIRTY*[way][set] = 0;  
 for (int i = 0; i < line.length(); i++) {  
 *CACHE\_LINE*[way][set][i] = (int)(line.charAt(i) - '0');  
 }  
 for (int i = 0; i < tag.length(); i++) {  
 *CACHE\_TAG*[way][set][i] = (int)(tag.charAt(i) - '0');  
 }  
 return way;  
}

Теперь можем перейти к основным методам кэша это ***READ*** и ***WRITE***

Метод ***READ*** в его подается информация о первом бите, который нужно прочитать и количество битов, информация о битах подается за **2** такта (первый такт – ***tag*** и ***set,*** второй такт - **offset**). Счетчик обращений к кэшу сразу же увеличивается на **1**. После происходит проверка находится ли нужная информация в **0**-м потоке, в случае ее нахождения происходит кэш-попадание, счетчик попаданий увеличивается и к counter записывается **6** тактов(по условию после кэш-попадания кэш отвечает через 6 тактов), проверка **1**-го потока точно такая же. Если нужной информации не найдено, произошел кэш-промах (по условию после кэш-промаха кэш отвечает через **4** такта). Вызывается метод g***etNewLine*** и в котором записывается в поток way нужная мне информация из памяти. Информация о полученной строке возвращается за ***bits*/16** тактов.

public int READ(String tag, String set, String offset, int bits) {  
 counter.add(2);  
 *cntA*++;  
 int intSet = parseInt(set);  
 int intOffset = parseInt(offset);  
 if (check(0, intSet, tag)) {  
 *cntP*++;  
 counter.add(6);  
 return get(0, intSet, intOffset, bits);  
 }  
 if (check(1, intSet, tag)) {  
 *cntP*++;  
 counter.add(6);  
 return get(1, intSet, intOffset, bits);  
 }  
 counter.add(4);  
 int way = getNewLine(tag, set);  
 counter.add(bits/16.);  
 return get(way, intSet, intOffset, bits);

}

В метод ***WRITE*** информация о тегах поступает за **2** такта, а информация о line за ***bits*/16**. Mаксимально возможное значение ***bits*** – **32,** максимально возможная затрата времени – **2** такта. Так как информация по шинам поступает одновременно то учитывать я буду только время поступления тегов. В остальном метод ***WRITE*** идентичен методу **READ**, то, что он производит запись ***line*** с помощью метода ***set***.

public void WRITE(String tag, String set, String offset, int bits, String line) {  
 counter.add(2);  
 //counter.add(bits / 16);  
 *cntA*++;  
 int intSet = parseInt(set);  
 int intOffset = parseInt(offset);  
 if (check(0, intSet, tag)) {  
 *cntP*++;  
 counter.add(6);  
 set(0, intSet, intOffset, bits, line);  
 return;  
 }  
 if (check(1, intSet, tag)) {  
 *cntP*++;  
 counter.add(6);  
 set(1, intSet, intOffset, bits, line);  
 return;  
 }  
 counter.add(4);  
 int way = getNewLine(tag, set);  
 set(way, intSet, intOffset, bits, line);  
}

**Cpu**

В ***Cpu*** находит функция, заданная в ТЗ, и подаются команды в кэш.

***Значения переменных:***

***A\_BIT\_SIZE*** – размер массива ***A*** в битах

***B\_BIT\_SIZE*** – размер массива ***B*** в битах

static final int *A\_BIT\_SIZE* = *M* \* *K* \* 8;  
static final int *B\_BIT\_SIZE* = *K* \* *N* \* 16;

Как и в случае с кэшем начну со вспомогательных методов

Метод ***parseBits*** переводит число в типе ***int*** в битовую строку установленной длинны.

static String parseBits(int x, int sz) {  
 StringBuilder s = new StringBuilder();  
 for (int i = 0; i < sz; i++) {  
 s.insert(0, (char) (x % 2 + '0'));  
 x /= 2;  
 }  
 return s.toString();  
}

***READ*** переводит указатель в ***tag***, ***set*** и ***offset***, после передает запрос на чтение кэшу Метод.

static int READ(int x, int bits) {  
 int tag = x / (128 \* 64);  
 int set = (x / 128)%64;  
 int offset = x % 128;  
 return *cache*.READ(*parseBits*(tag, 8),  
 *parseBits*(set, 6), *parseBits*(offset, 7), bits);  
}

Метод ***WRITE*** переводит указатель в ***tag***, ***set*** и ***offset***, после передает данные на запись кэшу.

static void WRITE(int x, int bits, int y) {  
 int tag = x / (128 \* 64);  
 int set = (x / 128)%64;  
 int offset = x % 128;  
 *cache*.WRITE(*parseBits*(tag, 8), *parseBits*(set, 6),  
 *parseBits*(offset, 7), bits, *parseBits*(y, bits));  
}

И ***main*** в нем находится основная функция, заданная в ТЗ, сразу происходят **2** инициализации указателей, которые происходят каждая по **1** такту:

В первом цикле – **2** прибавления и **1** переход на новую итерацию цикла (**3** такта).

Во втором цикле – **2** инициализации и переход на новую итерацию (**3** такта).

В третьем цикле – **2** прибавления, **1** переход на новую итерацию и **1** умножение (**8**) тактов.

После выполнения всех операций выводится ответ.

public static void main(String[] args) {  
 int pa = 0;  
 int pc = *A\_BIT\_SIZE* + *B\_BIT\_SIZE*;  
 *counter*.add(1 + 1);  
 for (int y = 0; y < *M*; y++)  
 {  
 for (int x = 0; x < *N*; x++)  
 {  
 int pb = *A\_BIT\_SIZE*;  
 int s = 0;  
 for (int k = 0; k < *K*; k++)  
 {  
 s += *READ*(pa + k\*8, 8) \* *READ*(pb + x\*16, 16);  
 pb += *N*\*16;  
 *counter*.add(1 + 1 + 1 + 5);  
 }  
 *WRITE*(pc + x\*32, s, 32);  
 *counter*.add(1 + 1 + 1);  
 }  
 pa += *K*\*8;  
 pc += *N*\*32;  
 *counter*.add(1 + 1 + 1);  
 }  
 System.*out*.println(*cache*.answer() \* 100 + "% " + *counter*.getAnswer() + " tacts");  
}

**Ответ:**

