Лабораторная работа №2	М3139	2022
Моделирование схем в Verilog	Яковлев Илья Игоревич	

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

Инструментарий и требования к работе: весь код пишется на языке Verilog, компиляция и симуляция – Icarus Verilog 10 и новее (полезные материалы: Verilog.docx). 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

В качестве варианта вам даны некоторые параметры системы и задача, которую надо сначала решить аналитически на основании параметров системы, а затем промоделировать и сравнить полученные результаты.

Вариант

Вариант №125.

Параметры системы: 3 (на рисунке ниже)

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`)

Вычисление недостающих параметров

Параметры, которые мы знаем:

Название параметра	Значения параметра
CACHE_SIZE	2^{11} байт
CACHE_LINE_SIZE	2^4 байт
CACHE_TAG_SIZE	8 бит
MEM_SIZE	2^{18} байт

Полная таблица параметров

Название параметра	Формула	Значение параметра
MEM_SIZE	Дано	2^{18} байт
CACHE_SIZE	Дано	2^{11} байт
CACHE_LINE_SIZE	Дано	2^4 байт
CACHE_WAY	CACHE_LINE_COUNT/ CACHE_SETS_COUNT	2
CACHE_LINE_COUNT	CACHE_SIZE/ CACHE_LINE_SIZE	2^7
CACHE_SETS_COUNT	$2^{CACHE\_SET\_SIZE}$	2^6
CACHE_ADDR_SIZE	Log_2(MEM_SIZE)	18 бит
CACHE_OFFSET_SIZE	Log_2(CACHE_LINE_SIZE)	4 бит
CACHE_SET_SIZE	CACHE_ADDR_SIZE– CACHE_TAG_SIZE– CACHE_OFFSET_SIZE	6 бит
CACHE_TAG_SIZE	Дано	8 бит

Пояснения:

CACHE_WAY – это ассоциативность кэша. То есть это число кэш линий в одном set. Из-за этого

CACHE_WAY = CACHE_LINE_COUNT / CACHE_SETS_COUNT.

Адрес каждой ячейки памяти представим в виде

Tag	Set	Offset
CACHE_TAG_SIZE	CACHE_SET_SIZE	CACHE_OFFSET_SIZE

Отсюда получаем формулу:

CACHE_ADDR_SIZE = CACHE_TAG_SIZE + CACHE_SET_SIZE + CACHE_OFFSET_SIZE

Причем offset это часть адреса, которую не нужно хранить в кэше. Ведь в кэше данные хранятся в кэш линиях, из-за чего часть адреса, отвечающую за определение положения данных в этой кэш линии, можно не хранить и передавать всю кэш линию целиком.

Отсюда формула CACHE_OFFSET_SIZE = Log_2(CACHE_LINE_SIZE)

Set – это адрес set в кэше.

Tag – часть адреса не вошедшая ни в set, ни в offset. Именно она хранится в каждой кэш линии.

Вычислим размер шин

Шина	Обозначение	Размерность
A1	ADDR1_BUS_SIZE	14 бит
A2	ADDR2_BUS_SIZE	14 бит
D1	DATA1_BUS_SIZE	16 бит
D2	DATA2_BUS_SIZE	16 бит
C1	CTR1_BUS_SIZE	3 бита
C2	CTR2_BUS_SIZE	2 бита

Размерности D1 и D2 даны.

Размерность A1 и A2 =

= CACHE_ADDR_SIZE - CACHE_OFFSET_SIZE = 14 бит. Так как в протоколе обмена данными по шине сказано: “По шине A1 адрес передаётся за 2 такта: в первый такт tag+set, во второй - offset. По шине A2 передаётся адреса без части offset за 1 такт. ”

Размерность C1 равна 3 битам. Номера команд от 0 до 7, а значит, их можно закодировать 3 битами.

Размерность C2 равна 2 битам. Номера команд от 0 до 3, а значит, их можно закодировать 2 битами.

Аналитическое решение задачи

Задача:

Имеется следующее определение глобальных переменных и функций:

```
#define M 64
#define N 60
#define K 32
int8 a[M][K];
int16 b[K][N];
int32 c[M][N];

void mmul()
{
  int8 *pa = a;
  int32 *pc = c;
  for (int y = 0; y < M; y++)
  {
    for (int x = 0; x < N; x++)
    {
      int16 *pb = b;
      int32 s = 0;
      for (int k = 0; k < K; k++)
      {
        s += pa[k] * pb[x];
        pb += N;
      }
      pc[x] = s;
    }
    pa += K;
    pc += N;
  }
}
```

Сложение, инициализация переменных и переход на новую итерацию цикла, выход из функции занимают 1 такт. Умножение – 5 тактов. Обращение к памяти вида pc[x] считается за одну команду.

Массивы последовательно хранятся в памяти, и первый из них начинается с 0.

Все локальные переменные лежат в регистрах процессора.

По моделируемой шине происходит только обмен данными (не командами).

Определите процент попаданий (число попаданий к общему числу обращений) для кэша и общее время (в тактах), затраченное на выполнение этой функции.

Аналитика была проведена с помощью языка высокого уровня C++:

```
#include <iostream>

#define M 64
#define N 60
#define K 32
#define a_size M * K
#define b_size K * N * 2
#define c_size M * N * 4

#define Cache_set_count 64
#define Cache_line_count 128
#define Cache_way 2
#define Cache_line_size 16
#define Mem_size 262144
#define Cache_offset_size 4
#define Cache_set_size 6
#define Cache_tag_size 8

//int8 a[M][K];
int a[M][K];
int pa;

//int16 b[K][N];
int b[K][N];
int pb;

//int32 c[M][N];
int c[M][N];
int pc;

long long Time = 0,
Count_cache_hit = 0,
Count_memory_access = 0;

int Cache_mem[Cache_set_count][Cache_way][16];
bool Cache_isValid[Cache_set_count][Cache_way];
int Cache_tag[Cache_set_count][Cache_way];

using namespace std;

bool isCacheHit(int index) {
   int ofset = index % (1 << Cache_offset_size),
       set = index / (1 << Cache_offset_size) % (1 << Cache_set_size),
       tag = index / (1 << (Cache_offset_size + Cache_set_size));
   return Cache_isValid[set][0] && Cache_tag[set][0] == tag || Cache_isValid[set][1] &&
Cache_tag[set][1] == tag;
}

void countDelayRead(int index) {
   int ofset = index % (1 << Cache_offset_size),
       set = index / (1 << Cache_offset_size) % (1 << Cache_set_size),
       tag = index / (1 << (Cache_offset_size + Cache_set_size));
   if (isCacheHit(index)) {
       Count_cache_hit++;
       Time += 6;
       if (Cache_tag[set][1] == tag)
          swap(Cache_tag[set][0], Cache_tag[set][1]);
       return;
   }
   else {
```

```
        Time += 4;
        Time += 100;
        Time += 8;
        Cache_tag[set][1] = Cache_tag[set][0];
        Cache_isValid[set][1] = Cache_isValid[set][0];
        Cache_tag[set][0] = tag;
        Cache_isValid[set][0] = true;
        return;
    }
}

void countDelayWrite(int index) {
    int ofset = index % (1 << Cache_offset_size),
        set = index / (1 << Cache_offset_size) % (1 << Cache_set_size),
        tag = index / (1 << (Cache_offset_size + Cache_set_size));
    if (isCacheHit(index)) {
        Count_cache_hit++;
        Time += 6;
        if (Cache_tag[set][1] == tag)
            swap(Cache_tag[set][0], Cache_tag[set][1]);
        return;
    }
    else {
        Time += 4;
        Time += 100;
        Cache_tag[set][1] = Cache_tag[set][0];
        Cache_isValid[set][1] = Cache_isValid[set][0];
        Cache_tag[set][0] = tag;
        Cache_isValid[set][0] = true;
        return;
    }
}

void main()
{
    //int8* pa = a;
    pa = 0; Time++;

    //int32* pc = c;
    pc = 0; Time++;

    Time += 2;
    for (int y = 0; y < M; y++)
    { Time++;
        Time += 2;
        for (int x = 0; x < N; x++)
        { Time++;

            //int16* pb = b;
            pb = 0; Time++;

            int s = 0; Time++;

            Time += 2;
            for (int k = 0; k < K; k++)
            { Time++;

                Count_memory_access += 2;
                countDelayRead(pa + k); Time++;
                countDelayRead(2 * (pb + x) + a_size); Time++;
                //s += pa[k] * pb[x];
                s += a[pa / K][k] * b[pb / N][x]; Time += 5; Time++;
```

```
            pb += N; Time++;

            Time += 2;
        }

        Count_memory_access += 1;
        countDelayWrite(4 * (pc + x) + a_size + b_size);
        //pc[x] = s;
        c[pc / N][x] = s; Time += 2;

        Time += 2;
    }

    pa += K; Time++;
    pc += N; Time++;

    Time += 2;
  }

  cout << "Worktime: " << Time << " takts\n"
    << "Percent Cache hit: " << (double)100 * Count_cache_hit / Count_memory_access << "%\n"
    << "Count Cache hit: " << Count_cache_hit << "\n"
    << "Count Memory access: " << Count_memory_access;
}
```

Вывод программы:

“Worktime: 5000224 takts

Percent Cache hit: 92.4271%

Count Cache hit: 230698

Count Memory access: 249600”

Пояснение к программе

Исходный код был переписан на аналогичный по логике. Для наглядности в комментариях оставлены части изначального кода.

В начале программы с помощью define заданы параметры системы.

Глобальная переменная Time – это счетчик тактов.

a_size, b_size и c_size – это количество байтов, занимаемое массивом a, b и c соответственно.

pa, pb и pc – это индексы в глобальном массиве, которые указывают на одномерные массивы a, b и c соответственно.

Count_cache_hit – количество кэш попаданий.

Count_memory_access – количество обращений к памяти.

Функция isCacheHit(int index) возвращает true, если ячейка памяти с индексом index находится в кэше (происходит кэш попадание), и возвращает false в противном случае.

Как она работает?

Вначале происходит разбиения index на tag, set и offset. Затем проверяется не лежит ли в нулевой кэш линии соответствующего cache_set нужный tag (при условии, что эта кэш линия valid (valid == 1), т е не мусор). Далее проверяется то же для первой кэш линии. Если или в нулевой, или в первой найден нужный valid tag, то true, иначе false.

Функция countDelayRead(int index) ничего не возвращает. Она считает задержку от чтения из памяти ячейки памяти с индексом index и обновляет кэш.

Как она работает?

Аналогично с isCacheHit разбивает index. Далее проверяет с помощью isCacheHit было ли кэш попадание. Если да, то (1*), иначе (2*).

(1*): Увеличивается счетчик кэш попаданий. Увеличивается счетчик тактов на 6, так как согласно ТЗ 6 тактов – это время, через которое в результате кэш попадания, кэш начинает отвечать. Затем, если наша ячейка найдена в кэш линии 1, то меняем кэш линии 0 и 1 местами. Это сделано, чтобы недавно использованные данные оставались в кэше дольше (Согласно политике вытеснения LRU).

(2*): Увеличивается счетчик тактов на 4, так как согласно ТЗ 4 такта – э время, через которое в результате кэш промаха, кэш посылает запрос к памяти. Далее счетчик увеличивается на 100, так как чтение из памяти 100 тактов. Далее счетчик увеличивается на 8, так как размер кэш линии 2^4 байт, а размер шины данных 2 байта. (В данной версии кэша будем ждать записи в кэш, а только потом отправлять в процессор данные). Затем перемещаем кэш линию 0 в кэш линию 1 и записываем в кэш линию 0 полученные данные.

Функция countDelayWrite(int index) считает задержку от записи в память ячейки памяти с индексом index и обновляет кэш.

Реализация мало чем отличается от countDelayRead, так что отмечу только различия: При кэш попадании ничего не меняется, а при кэш промахе нам

не нужно ждать дополнительные 8 тактов. 100 тактов мы ждем, чтобы записать данные в память для решения когерентности.

Разбор главной функции main():

Время инициализации pa и pc – по 1 такту.

Перед входом в новый цикл учитываем расходы на создание переменной-счетчика для цикла и проверки условия. Это еще 2 такта.

Далее в каждом цикле при входе в новую итерацию счетчик увеличивается на 1 (так сказано в условии). При выходе будем увеличивать на 2 такта – инкремент переменной счетчика и проверка условия.

Инициализация pb и s – по 1 такту.

Внутри третьего цикла увеличиваем счетчик обращений к памяти на 2 (т к далее обращение к памяти дважды). И дважды считаем задержку чтения данных с помощью функции countDelayRead. В первый раз от pa + k, второй раз от 2 * (pb + x) + a_size (так как массив b лежит после a и элементы b – двухбайтовые числа). Также добавляется еще 1 такт сверху от каждого вызова функции для передачи данных от кэша процессору.

Далее происходит умножение и инкремент переменной на это значение. Умножение 5 тактов, а инкремент будем считать за 1.

После выхода из третьего цикла увеличиваем счетчик обращений памяти на 1 и считаем задержку записи в память с помощью функции countDelayWrite от 4 * (pc + x) + a_size + b_size (так как массив c лежит после a и b, а элементы c – четырехбайтовые числа). Также добавляется 2 такта – это передача четырехбайтовые числа s по двухбайтовой шине.

После выхода из второго цикла происходят 2 инкремента по 1 такту.

Далее выводится ответ.

Итого:

Процент попаданий 92.4271%, что довольно много, несмотря на одноуровневый кэш. Это вполне похоже на правду, ведь наша программа довольно последовательно обращалась к данным, т е обладала свойством пространственно временной локальности.

Трудно оценить количество тактов на правдоподобность. Однако можно с уверенностью сказать, что количество тактов можно заметно уменьшить с

помощью более умного кэша. Ведь имитированный в данной программе кэш можно назвать тривиальным.

Моделирование заданной системы на Verilog

Mem:

```
module Mem #(parameter MEM_SIZE = 262144, _SEED = 225526)(input wire CLK, input wire[17:0] A2, inout wire[15:0] D2, inout wire[1:0] C2, input wire Reset);
  integer SEED = _SEED;
  reg[7:0] Mem[MEM_SIZE - 1:0];
  reg dataFlag = 0, ctrlFlag = 0;
  integer memData;
  integer memComand;
  assign D2 = dataFlag == 1 ? memData : D2;
  assign C2 = ctrlFlag == 1 ? memComand : C2;

  initial begin
    _Reset();
  end

  always @(Reset) begin
    _Reset();
  end

  task _Reset();
  for (integer i = 0; i < MEM_SIZE; i++) begin
    Mem[i] = $random(SEED)>>16;
    $display("[%d] %d", i, Mem[i]);
  end
  endtask

  always @(C2) begin
    memComand = C2;
    if (memComand == 2) begin
```

```
        #99
        dataFlag = 1;
        memData = Mem[A2];
        #1
        dataFlag = 0;
    end
    else begin
        Mem[A2] = memData;
        #99
        ctrlFlag = 1;
        memComand = 1;
        #1
        ctrlFlag = 0;
    end
  end
endmodule
```

Cache:

```
`include "Mem.v"

module Cache#(parameter _SEED = 225526, Cache_set_count = 64, Cache_line_size = 128, cache_tag_size = 8)(input wire CLK, input wire[17:0] A1, inout wire[15:0] D1, inout wire[2:0] C1, input wire Reset);
  integer SEED = _SEED;
  reg[138 - 1:0] cache[Cache_set_count - 1:0][1:0];
  reg dataFlag1 = 0, ctrlFlag1 = 0,
    dataFlag2 = 0, ctrlFlag2 = 0;
  integer memData1, memData2;
  integer memComand1, memComand2;
  wire[15:0] D2;
  wire[1:0] C2;
  reg[17:0] A2;
  assign D1 = dataFlag1 == 1 ? memData1 : D1;
```

```
assign C1 = ctrlFlag1 == 1 ? memComand1 : C1;
assign D2 = dataFlag2 == 1 ? memData2 : D2;
assign C2 = ctrlFlag2 == 1 ? memComand2 : C2;
Mem _MEM(CLK, A2, D2, C2, Reset);

initial begin
  _Reset();
end

always @(Reset) begin
  _Reset();
end

task _Reset();
for (integer i = 0; i < Cache_set_count; i++) begin
  cache[i][0] = $random(SEED)>>16;
  cache[i][1] = $random(SEED)>>16;
  $display("[%d] %d", i, cache[i][0], cache[i][1]);
end
endtask

always @(C1) begin
  memComand1 = C1;
  if (memComand1 == 1) begin
      #1
      ctrlFlag2 = 1;
      memComand2 = 2;
      A2 = A1;
      #1
      ctrlFlag2 = 0;
      dataFlag1 = 1;
      ctrlFlag1 = 1;
      #1
      memData1 = memData2;
```

```
    memComand2 = 1;
    #1
    dataFlag1 = 0;
    ctrlFlag1 = 0;
end
else if (memComand1 == 2) begin
    #1
    ctrlFlag2 = 1;
    memComand2 = 2;
    A2 = A1;
    #1
    ctrlFlag2 = 0;
    dataFlag1 = 1;
    ctrlFlag1 = 1;
    #1
    memData1 = memData2;
    memComand2 = 1;
    #1
    dataFlag1 = 0;
    ctrlFlag1 = 0;
end
else if (memComand1 == 3) begin
    #1
    ctrlFlag2 = 1;
    memComand2 = 2;
    A2 = A1;
    #1
    ctrlFlag2 = 0;
    dataFlag1 = 1;
    ctrlFlag1 = 1;
    #1
    memData1 = memData2;
    memComand2 = 1;
    #1
```

```
    dataFlag1 = 0;
    ctrlFlag1 = 0;
end
else if (memComand1 == 4) begin

end
else if (memComand1 == 5) begin
    #1
    ctrlFlag2 = 1;
    dataFlag2 = 1;
    memComand2 = 3;
    memData2 = D1;
    A2 = A1;
    #1
    ctrlFlag2 = 0;
    dataFlag2 = 0;
    dataFlag1 = 1;
    ctrlFlag1 = 1;
    #1
    memData1 = memData2;
    memComand1 = memComand2;
    #1
    dataFlag1 = 0;
    ctrlFlag1 = 0;
end
else if (memComand1 == 6) begin
    #1
    ctrlFlag2 = 1;
    dataFlag2 = 1;
    memComand2 = 3;
    memData2 = D1;
    A2 = A1;
    #1
    ctrlFlag2 = 0;
```

```
        dataFlag2 = 0;
        dataFlag1 = 1;
        ctrlFlag1 = 1;
        #1
        memData1 = memData2;
        memComand1 = memComand2;
        #1
        dataFlag1 = 0;
        ctrlFlag1 = 0;
    end
    else if (memComand1 == 7) begin
        #1
        ctrlFlag2 = 1;
        dataFlag2 = 1;
        memComand2 = 3;
        memData2 = D1;
        A2 = A1;
        #1
        ctrlFlag2 = 0;
        dataFlag2 = 0;
        dataFlag1 = 1;
        ctrlFlag1 = 1;
        #1
        memData1 = memData2;
        memComand1 = memComand2;
        #1
        dataFlag1 = 0;
        ctrlFlag1 = 0;
    end
  end
endmodule
```

CPU:

```
module CPU();
```

```
  wire[2:0] outComand, inComand;
  wire[4:0] outAddr, inAddr;
  wire[15:0] outData, inData;

  Cache cache(.outComand1(outComand), .outAddr1(outAddr), .outData1(outData),
.inComand1(inComand), .inAddr1(inAddr), .inData1(inData));

  initial begin

  end

  always @(inComand != 0) begin
    tag = inAddr[17:10];
    set = inAddr[9:4];
    offset = inAddr[3:0];
    case(inComand)
      1: begin

      end
      2: begin

      end
      3: begin

      end
      endcase
  end

endmodule
```