| Лабораторная работа №2 | М3136 | 2022 |
|---|---|---|
| Моделирование схем в Verilog | Рязанова Екатерина Владимировна | |

**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 11. Для написания кода используется Visual Studio Code, куда установлены необходимые расширения (плагины).

**Описание**

Нам необходимо эмулировать работу системы процессор-кэш-память. Нужно прописать полностью кэш, а память и кэш – «заглушки», которые просто умеют посылать и принимать команды. Причем инициализация памяти уже дана в условии. Затем необходимо выполнить аналитическое решение задачи, посчитав количество тактов и процент кэш-попаданий. После этого нужно проверить посчитанные данные с помощью написанной программы.

**Вариант**

У меня 1 вариант.

**Расчет параметров системы**

1) Mem_size = $2^{19}$ байт = $2^{22}$ бит – размер памяти
2) Cache_size = $2^{10}$ байт = $2^{13}$ бит – размер кэша
3) Cache_line_size = $2^4$ байт = $2^7$ бит – размер кэш-линий
4) Way = 2 – ассоциативность*
5) Cache_line_count = $2^6$ – количество кэш-линий
   Cache_line_count = Cache_size/ Cache_line_size, значит, Cache_size = Cache_line_size* Cache_line_count = $2^{10}$ байт
6) Cache_set_count = Cache_line_count/ Way = $2^5$ – количество кэш-наборов(блоков)
7) Cache_addr_size = $\log_2$(Mem_size) = 19 – минимальное количество битов для адреса, чтобы по нему обращаться к памяти
8) Cache_offset_size = 4 = $\log_2$(Cache_line_size) – смещение внутри кэш-линии (адрес внутри самой кэш-линии), младшие биты

9) Cache_set_size = $\log_2$(Cache_set_count) = 5 – кодировка номера линии блока, средние биты
10) Cache_tag_size = 19 – 5 – 4 =10 – Старшие биты, необходимо помнить при поиске кэша

* Ассоциативность кэша – это когда мы разделяем кэш на n множеств, в каждом из которых хранится m линий. Блок в памяти сначала соотносится с множеством, а потом заносится в любую кэш-линию этого множества.

| valid | dirty | tag | data |
|---|---|---|---|
| 1 | 1 | 10 = Cache_tag_size | Cache_line_size = 128 |

Таблица 1 – Устройство кэш-линии (все в битах)

| tag | set | offset |
|---|---|---|
| 10 = Cache_tag_size | 5 = Cache_set_size | Cache_offset_size = 4 |

Таблица 2 – Интерпретация адреса кэшем (тоже в битах)

Самый точный адрес – 19 бит, то есть у шины A1 размерность 15 бит Max(Cache_set_size+ Cache_tag_size, Cache_offset_size) = 15 бит. По шине A2 передается адрес без части offset 15 бит, потому что эти данные передаются в 2 такта. У шины C1 размерность 3 бита, так как по ней передаются 8 команд максимум, 8 = $2^3$. Аналогично у шины C2 размерность 2 бита, так как 4 команды – 0, 1, 2, 3.

**Аналитическое решение**

Для подсчета процента кэш-попаданий и тактов я написала программу на Pascal. Mwrite, mread – имитация работы кэша, чтение и запись. Основная работа происходит в этом куске кода:

```
begin
  tactcount:=4; //три инициалиции pa,pc,y и выход – сразу же
автоматически добавляется 4 такта – из задачи. Это строчки int8 *pa
= a; int32 *pc = c; for (int y = 0 ….. и выход из этой процедуры.
  hitcount:=0;//количество попаданий
  accesscount:=0; //количество доступов

  for i:=0 to 31 do begin //цикл сбрасывает кэш ,дальше начинается
эмуляция работы алгоритма. Сами действия не выполняются, считаются
только сами такты.
    cache[i].valid[0]:=false;
    cache[i].valid[1]:=false;
    cache[i].todel:=0;
  end;
```

```
  pa:=abeg;pc:=cbeg;
  for y:=0 to M-1 do begin
      tactcount:=tactcount+5; // инициализация x; y++, увеличение pa
и pc; условный переход на новую итерацию
      for x:=0 to N-1 do begin
          tactcount:=tactcount+5; // инициализация pb, s, k; x++ ;
условный переход на новую итерацию
          pb:=bbeg;
          for k:=0 to KK-1 do begin
              tactcount:=tactcount+5; // k++;сложение; увеличение
pb; умножение(5) ; условный переход на новую итерацию
              mread(pa+k,1);
              mread(pb+x,2);
              pb:=pb+N;
              end;
          mwrite(pc+x,4);
          end;
      pa:=pa+KK;
      pc:=pc+N;
           end;
```

В любой процедуре mread или mwrite идет увеличение accesscount на 1, а дальше условие: if (hit) then begin – если попали, то отвечаем из кэша. Например, вот процедура mread (mwrite почти не отличается).

```
procedure mread(addr,plen:integer);
var csetln,ctag,hitset:integer;
  hit:boolean;
begin
     accesscount:=accesscount+1;
     csetln:=(addr shr 4) and $1F;
     ctag:=addr shr 9;
     hitset:=5;
     if cache[csetln].tag[0]=ctag then hitset:=0;
     if cache[csetln].tag[1]=ctag then hitset:=1;
     hit:=(hitset<5);
     if (hit) then begin
        cache[csetln].todel:=hitset xor 1; // todel – какую строку
из двух удалять, а hitset – куда мы попали, то есть удалять надо
другую, поэтому и xor. Это принцип lru.
        hitcount:=hitcount+1; //попадание в кэш – hitcount
увеличивается
        tactcount:=tactcount+7; // +7, так как в условии написано
про +6, но это расстояние от такта команды до такта ответа, то есть
всего 7 тактов.
        if plen>2 then tactcount:=tactcount+1; // если данных больше
2 байт передать, то тогда еще +такт на передачу второй части.
        exit;
        end;
     hitset:=cache[csetln].todel;
```

```
     if (cache[csetln].valid[hitset] and
cache[csetln].dirty[hitset]) then begin // если в кэш не попали,
строка проверяется валидная и грязная – значит, ее надо записывать.
        cache[csetln].dirty[hitset]:=false; // эмулируем работу
кэша, причем в mwrite этой строки нет , так как тут надо сбрасывать
грязность, а тут нет.
        cache[csetln].tag[hitset]:=ctag; // эмулируем работу кэша
        cache[csetln].todel:=hitset xor 1; // эмулируем работу кэша
        tactcount:=tactcount+5+100+100+8; // 5 тактов по условию,
что на 4 такте чтение памяти начинается, 100 тактов на запись, 100
на чтение, 8 на передачу линии.
        if plen>2 then tactcount:=tactcount+1;
        exit;
        end;
     cache[csetln].valid[hitset]:=true;
     cache[csetln].tag[hitset]:=ctag;
     cache[csetln].todel:=hitset xor 1;
     tactcount:=tactcount+5+100+8;
     if plen>2 then tactcount:=tactcount+1;
end;
```

Таким образом, результат работы программы -

```
tacts=4222252
hitcount=234162 accesscount=249600
hitrate=93.815%
```

### Моделирование заданной системы на Verilog

У кэша имеется несколько политик чтения и записи. В данной лабораторной look-through и write-back. А также есть политики замещения, у нас LRU.

Look-through означает, что CPU отправляет запрос на чтение в кэш. Если у кэша есть нужная линия, он отвечает. Если нет, то запрашивает линию у основной памяти, а её ответ сохраняет себе. Преимуществом этого метода является возможность создать большую шину до процессора, так как они находятся на одном кристалле.

Write-back означает, что данные пишутся только в кэш. Данные из кэша пишутся в оперативную память, только когда вытесняются в кэш новые, грязные данные, которые были изменены. Если же была просто копия, то просто выкидывается. (Для этого варианта нужно знать, dirty или clean – являются данные измененными или копией памяти, поэтому реализация нетривиальная).

Чтобы при кэш-промахе записать куда-то новую линию, нужно освободить место. LRU означает, что выбрасывается самая давно использованная линия (last-recently used).

Далее отдельные отрывки кода будут поясняться в таблицах.

| | |
|---|---|
| Модуль процессора<br><br>(просто перечисление входов-выходов) | `module CPU (`<br>`  input CLK, //вход синхронизации`<br>`  output [ADDR1_BUS_SIZE-1:0] A1, //шина A1`<br>`  inout [DATA1_BUS_SIZE-1:0] D1, //шина D1`<br>`  inout [CTR1_BUS_SIZE-1:0] CTR1 //шина C1`<br>`);`<br>`endmodule;` |

Также стоит разобрать модуль памяти. Его инициализация взята из тз, и, соответственно, поменян размер. Также добавлены команды чтения и записи. В строке `always@(posedge CLK or posedge RESET or posedge M_DUMP)` начинается блок always. Вначале если сигнал сброса Reset, то выполняем сброс, если M_Dump, то выводим содержимое памяти. И в противном случае (то есть если сигнал тактовый), выполняем цикл по памяти (с внутренним автоматом состояний *– что это такое, подробнее написано в пояснениях к кэшу,* там аналогичная конструкция). Если получили какую-то команду – чтение или запись строки, то начинаем реагировать.

| | |
|---|---|
| Команда прочитать строчку | `C2_READ_LINE:begin`<br>`LineAddr=A2;`<br>`ICount=99;`<br>`IState=MS_Wait;`<br>`LCount=8;`<br>`NewState=MS_RDAnswer;`<br>`answer_cmd=C2_NOP;`<br>`D2Ctrl<=1;`<br>`end` |
| Команда записать строчку | `C2_WRITE_LINE:begin`<br>`LineAddr=A2;`<br>`ICount=99;`<br>`IState=MS_WRLine;`<br>`LCount=7;`<br>`Mem[(LineAddr<<4)+1]=D2[15:8];`<br>`Mem[(LineAddr<<4)]=D2[7:0];`<br>`LineAddr+=2;`<br>`end` |
| Команда ожидания | `MS_Wait:begin`<br>`ICount-=1;`<br>`if (ICount==0) IState=NewState;`<br>`end` |

| | |
|---|---|
| Команда ответа – выдать содержимое прочитанной строчки | MS_RDAnswer: begin<br>answer_cmd=C2_RESPONSE;<br>answer_word=(Mem[(LineAddr<<4)+1]<<8)\|Mem[(LineAddr<<4)];<br>LCount-=1;<br>if(LCount==0) begin<br>IState=MS_Idle;<br>D2Ctrl<=0;<br>end<br>end |
| Команда записать строку в память | MS_WRLine: begin<br>Mem[(LineAddr<<4)+1]=D2[15:8];<br>Mem[(LineAddr<<4)]=D2[7:0];<br>LineAddr+=2;<br>LCount-=1;<br>if(LCount==0) begin<br>ICount=99-8;<br>IState=MS_Wait;<br>NewState=MS_WREnd;<br>end<br>end |
| Ожидание положенных тактов и ответ, что все записано в память | MS_WREnd: begin<br>answer_cmd=C2_RESPONSE;<br>answer_word=0;<br>IState=MS_Idle;<br>D2Ctrl<=0;<br>end |

Мы завели 3 перечисления – enum, где перечислили команды процессор->кэш, кэш->процессор, кэш<->память. Далее объявляем параметры – из наших расчетов, а так же битности шин.

| | |
|---|---|
| Перечисление входов и выходов. Начался сам модуль кэша. | module Cache (<br>input CLK, //вход синхронизации<br>input RESET, //вход сброса<br>input [ADDR1_BUS_SIZE-1:0] A1, //шина A1<br>output [ADDR2_BUS_SIZE-1:0] A2, //шина A2<br>inout [DATA1_BUS_SIZE-1:0] D1, //шина D1<br>inout [DATA2_BUS_SIZE-1:0] D2, //шина D2<br>inout [CTR1_BUS_SIZE-1:0] CTR1, //шина C1<br>inout [CTR2_BUS_SIZE-1:0] CTR2, //шина C2<br>input C_DUMP //сигнал сохранения состояния кэша<br>); |

Далее объявляем константы состояния автомата, вводим два счетчика – Icount (счетчик на задержку), Lcount (количество передач, когда линия выдается в память и из памяти). Потом создаются массивы данных, тегов, валидности, признак какую строку удалять. Также дальше идут внутренние регистры, объявлены провода `wire hit0,hit1,hit` и как они формируются(сигналы, туда прописано логическое условие). Потом прописано D1 – шина данных, она двунаправленная inout, `D1Ctrl` – внутренний регистр из 1 бита. Он означает, что если 1 – шиной управляем мы, если 0 – мы ей не управляем, мы с нее будем читать. И в зависимости от `D1Ctrl` либо выдаем регистр answer_word, либо если `D1Ctrl = 0,` состояние dZ, неопределенное, мы не управляем. Аналогично и с D2. A2 – выход, она формируется старшие разряды берутся из внутреннего регистра A2tag, а младшие из разряда cset, это то, что мы получили от процессора и с чем работаем. A1 нет это входная шина, ей всегда управляет процессор, поэтому ее тут нет.

Затем идут последние два внутренних регистра.

```
reg [5:0] IState,NewState,RetState; // IState - внутреннее состояние. NewState - состояние в которое перейти по окончанию ожидания. RetState - состояние в которое вернуться после записи строки в память.

reg [CTR1_BUS_SIZE-1:0] cmd;
```

Первый это регистр для хранения состояний. 6 разрядов это значения не имеет, потому что состояний гораздо меньше. А в cmd мы запоминаем, какая команда у нас была от процессора, чтобы потом помнить, какая была команда и подготовить ответ.

| | |
|---|---|
| Процедура инициализации кэша. Прописываются биты занятости и грязности. | `integer i = 0;`<br>`  initial begin`<br>`  IState = RS_Idle;`<br>`  D1Ctrl='b0;`<br>`  for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin`<br>`  Valid0[i]=0;`<br>`  Valid1[i]=0;`<br>`  Dirty0[i]=0;`<br>`  Dirty1[i]=0;`<br>`  end;`<br>`  end;` |

Затем начинается блок always. Прописываем ему, что делать в той или иной ситуации.

```
always@(posedge CLK or posedge RESET or posedge C_DUMP)
```

| | |
|---|---|
| Сброс всех параметров (если Reset). | `if(RESET)`<br>`  begin`<br>`  IState = RS_Idle;`<br>`  D1Ctrl='b0;`<br>`  for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin`<br>`  Valid0[i]=0;`<br>`  Valid1[i]=0;`<br>`  Dirty0[i]=0;`<br>`  Dirty1[i]=0;`<br>`  end;`<br>`  end` |
| (Как в памяти) – вывод в консоль для отладки. | `else if (C_DUMP) begin`<br>`  for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin`<br>`  $display("Line [%d]: ToDel=%b; Set0: Valid=%b Dirty=%b Tag=%h Data=%h; Set1: Valid=%b Dirty=%b Tag=%h Data=%h;", i, ToFree[i], Valid0[i], Dirty0[i], TagSet0[i], DataSet0[i], Valid1[i], Dirty1[i], TagSet1[i], DataSet1[i]);`<br>`  end` |

И третий случай, если синхронизация. Тут учитываем автомат состояний. Суть автомата состояний заключается в следующем. Изначально кэш ничего не делает (Idle), просто ждет команд. При каждом такте CLK смотрится команда на шине C1. Далее написано действие на команды чтения - запомнить части адреса, передаваемого процессором и перейти в состояние Read2nd для чтения второй части адреса. При следующем сигнале мы будем в состоянии Read2nd и считаем остаток адреса. Дальше надо включать логику кэша и отвечать процессору.

```
case(IState)//а это тот самый автомат состояний IState = текущее состояние
```

Если «холостое» состояние Idle (анализируем шину C1) – какая команда от процессора.

| | |
|---|---|
| Команда инвалидации линии. Считали разряды, пришли в RS_Invalidate и делаем сброс, вставляем паузу. (Пропускаем 5 тактов и перейдем в состояние RS_Invalidate, RS_Invalidate ответит C1_Response процессору, что мы все сделали и снова придем в Idle, но при этом надо еще сбросить – в Rs_Invalidate) | `C1_INVALIDATE_LINE:begin`<br>`  ctag=A1[14:5];`<br>`  cset=A1[4:0];`<br>`  ICount=5;`<br>`  IState=RS_Wait;`<br>`  NewState=RS_Invalidate;`<br>`  end` |
| Для всех команд чтения в 1 такте выполняем чтение старших | `C1_READ8,C1_READ16,C1_READ32:begin`<br>`  ctag=A1[14:5];`<br>`  cset=A1[4:0];` |

| | |
|---|---|
| разрядов адреса от процессора и переходим в состояние RS_Read2nd для чтения оставшихся разрядов. | IState=RS_Read2nd;<br>cmd=CTR1;<br>end |

| | |
|---|---|
| Тут прописаны сбросы валидности в 1 и 0 сетах. (Инвалидация линий кэша). | RS_Invalidate:begin<br>D1Ctrl='b1;<br>answer_word=0;<br>D1Ctrl<='b0;<br>answer_cmd=C1_RESPONSE;<br>if (hit0) Valid0[cset]=0;<br>if (hit1) Valid1[cset]=0;<br>IState=RS_Idle;<br>end |
| Чтение младших битов адреса. (offset) | RS_Read2nd: begin<br>coffset=A1[4:0];<br>IState=RS_RDLogic;<br>D1Ctrl='b1;<br>answer_cmd=C1_NOP;<br>answer_word='d0;<br>end |

Затем начинается большой блок RS_RDLogic. Это логика работы кэша. Сначала смотрит на сигнал hit, попали ли мы в кэш.

| | |
|---|---|
| Если в кэш попали (соответственно, линия валидная, а грязная или нет – не важно), мы из нее читаем. | if (hit==1'd1) begin<br>ICount=3; // прошло два такта, пропустить еще 3 до ответа<br>IState=RS_Wait;<br>NewState=RS_RDAnswer;<br>ToFree[cset]=hit0; //кого выкидывать при промахе<br>end |

В соответствии со стратегией LRU записываем номер блока, который подлежит удалению (в случае необходимости) – в строчке ToFree[cset]=hit0;. Если мы попали в кэш, то маркируем, что к этому был доступ (попали в строку). И в tofree записываем hit0. Если мы попали в 0 set, то hit0 = 1, и соответственно в tofree запишется 1, то есть удалить надо будет 1 ,так как последний доступ получали к 0. Если придется удалить какую-то из строк, то удалять будем первую. А если попали в 1 set, то наоборот.

Далее идет случай кэш-промаха. Надо читать память и куда-то записывать данные, которые читаем из памяти. Смотрим, откуда из памяти будем читать (если она грязная, ее надо сначала сбросить в память ,потом читаем из памяти по новому адресу)

В строке `if (ToFree[cset]==1'b0) begin //выбрасываем линию из set0` проверяется, какую линию надо выбрасывать 0 или 1. Например, если определили, что 0. Но сначала нужно на нее посмотреть. Если эта линия валидная и грязная, то тогда эту линию сначала надо сохранить в память. А в противном случае (не валидная или не грязная) просто забываем про нее, она не нужна (данные из нее не надо записывать в память). Затем аналогично поступаем с 1 линией.

Блок записи строки в память состоит из трех «подблоков». Сначала выставляется адрес на A2 – куда писать (в RS_WriteLine – он берет младшие 16 бит из линии, дает их шине данных, выставляет команду – инициализация вывода в память), затем выдается вся линия (в RS_WriteLineData – по циклу сдвиг, снова младшие 16 бит дает на D2, уменьшает счетчик, и если линия кончилась, переходим в состояние ожидания, `D2Ctrl=0` - прекращаем управление шиной D2), потом ждем ответ от памяти - C2_Response (RS_WaitMem – когда память выдает Nop, так и будем ждать, пока память выдаст Response). После ответа возвращаемся обратно в состояние RD_Logic, снова будет промах – только ввиду того, что мы записали, мы сбросили грязный бит в строке `if (ReadToSet==0) Dirty0[cset]=0.` И поэтому условие в строке `if ((Valid0[cset]&&Dirty0[cset])==1'b1) begin` уже не выполнилось, линия уже не грязная. Поэтому мы попадаем в `if ((Valid0[cset]==0)||(Dirty0[cset]==0)) begin` и начинаем чтение из памяти.

Дальше идет блок чтения строки из памяти (практически аналогично). Сначала выставляем памяти адрес, даем команду чтения, потом ждем (RS_ReadLine). Потом ждем, пока память ответит Response (RS_ReadWait). Как память начала отвечать, считываем всю линию. Нужно поместить ее в кэш и линию отметить как чистую и валидную. Потом опять возвращаемся в состояние R2_Logic. Там уже срабатывает сигнал hit, что попали в кэш и оно прочитается штатным образом (ответ берется из кэша) ответит процессору (RS_RdAnswer).

| | |
|---|---|
| Универсальное состояние ожидания, сколько тактов необходимо пропустить. Перед тем, как перейти в это состояние, в регистр ICount нужно записать, сколько тактов пропустить, потом записать в регистр NewState состояние, в которое надо перейти. И в итоге этот RS_Wait отсчитывает нужное количество тактов и переходит в нужное состояние. | RS_Wait:begin<br>ICount-=1;<br>if (ICount==0)<br>IState=NewState; |

*при промахе эта задержка еще повторится. Но протокол не ломается. Возможны ошибки во времени, это займет немного больше тактов, чем положено.

Блок RS_RDAnswer (ответ процессору) имеет в себе описание команд чтения - C1_READ8, C1_READ16, C1_READ32, которые похожи между собой. Это чтение за первый такт. (Обработка 8, 16 и первых 16 бит из 32). А вот RS_RDAnswer2nd – чтение второй части из 16 бит 32-битных данных, второй такт ответа, куда мы переходим из C1_READ32. На этом модуль кэша заканчивается.

Команды write мало отличаются от команд read, поэтому нет смысла их так же подробно описывать. Там есть подобные RS_Write2nd – записывать вторую часть адреса и данных, RS_WRAnswer такое же как и RS_RdAnswer, только RS_RdAnswer отвечает данными, а RS_WRAnswer отвечает командой, что все выполнено. Блок RS_WRLogic аналогичен RS_RDLogic. Запись отличается только тем, что при записи мы ставим признак, что строка теперь грязная - Dirty0[cset]=1 и в зависимости от того, сколько данных надо записать по команде – столько и записываем.

Еще одна последняя команда - C1_Invalidate_Line. Просто нужно сбросить линию.

```
C1_INVALIDATE_LINE:begin
                ctag=A1[14:5];
                cset=A1[4:0];
                ICount=5;
                IState=RS_Wait;
                NewState=RS_Invalidate;
            end
```

Приняли адрес, ждем 5 тактов, приходим в rs_invalidate и там сбрасываем (было выше). `if (hit0) Valid0[cset]=0;`

```
            if (hit1) Valid1[cset]=0;
```

В этих строчках сам сброс.

## Листинг кода

```
enum {C1_NOP, C1_READ8, C1_READ16, C1_READ32, C1_INVALIDATE_LINE, C1_WRITE8,
C1_WRITE16, C1_WRITE32} C1_ASK;   //команды процессор->кэш
enum {C1_NOP2, C1_RESPONSE = 7}
C1_ANSWER;
//команды кэш->процессор
enum {C2_NOP, C2_RESPONSE, C2_READ_LINE, C2_WRITE_LINE}
C2_B;                                                  //команды кэш<->память

parameter MEM_SIZE = 524288;         //размер памяти
parameter CACHE_SIZE = 1024;         //размер кэша
parameter CACHE_LINE_SIZE = 16;      //размер кэш-линии
parameter CACHE_LINE_COUNT = 64;     //кол-во кэш-линий
parameter CACHE_WAY = 2;             //ассоциативность
parameter CACHE_SETS_COUNT = 32;     //кол-во наборов кэш-линий
parameter CACHE_TAG_SIZE = 10;       //размер тэга адреса
parameter CACHE_SET_SIZE = 5;        //размер индекса в наборе кэш-линий
parameter CACHE_OFFSET_SIZE = 4;     //размер смещения
parameter CACHE_ADDR_SIZE = 19;      //размер адреса

parameter ADDR1_BUS_SIZE = CACHE_TAG_SIZE+CACHE_SET_SIZE;   // ширина шины A1
parameter ADDR2_BUS_SIZE = CACHE_TAG_SIZE+CACHE_SET_SIZE;   // ширина шины A2
parameter DATA1_BUS_SIZE = 16;                              // ширина шины D1
parameter DATA2_BUS_SIZE = 16;                              // ширина шины D2
parameter CTR1_BUS_SIZE = 3;                                // ширина шины C1
parameter CTR2_BUS_SIZE = 2;                                // ширина шины C2

module MemCTR (
    input CLK,                      //вход синхронизации
    input RESET,                    //вход сброса
    input [ADDR2_BUS_SIZE-1:0] A2,  //шина A2
    inout [DATA2_BUS_SIZE-1:0] D2,          //шина D2
    inout [CTR2_BUS_SIZE-1:0] CTR2,         //шина C2
    input M_DUMP                    //сигнал сохранения состояния памяти
 );

reg [7:0] Mem [MEM_SIZE-1:0];

reg D2Ctrl; // 1 - управлять шинами D2 CTR2 0 - не управлять
integer ICount = 0, LCount = 0;
reg [5:0] IState,NewState; // IState - внутреннее состояние. NewState -
состояние в которое перейти по окончанию ожидания.
reg [ADDR2_BUS_SIZE-1:0] LineAddr;

reg [DATA2_BUS_SIZE-1:0] answer_word;
reg [CTR2_BUS_SIZE-1:0] answer_cmd;
```

```
assign D2 = (D2Ctrl) ? answer_word : 16'dZ;
assign CTR2 = (D2Ctrl) ? answer_cmd : 2'dZ;


localparam MS_Idle = 0;
localparam MS_Wait = 1;
localparam MS_RDAnswer = 2;
localparam MS_WRLine = 3;
localparam MS_WREnd = 4;


integer SEED = 225526;
integer i = 0;
  initial begin
    for (i = 0; i < MEM_SIZE; i += 1) begin
      Mem[i] = $random(SEED)>>16;
    end

    for (i = 0; i < MEM_SIZE; i += 1) begin
      $display("[%d] %d", i, Mem[i]);
    end

    IState = MS_Idle;
    D2Ctrl='b0;

//    $finish;
  end

always@(posedge CLK or posedge RESET or posedge M_DUMP)
begin
    if(RESET)
    begin
    IState = MS_Idle;
    D2Ctrl='b0;
    for (i = 0; i < MEM_SIZE; i += 1) begin
      Mem[i] = $random(SEED)>>16;
    end

    for (i = 0; i < MEM_SIZE; i += 1) begin
      $display("[%d] %d", i, Mem[i]);
    end
    end
    else if (M_DUMP) begin
     for (i = 0; i < MEM_SIZE; i += 1) begin
      $display("[%d] %d", i, Mem[i]);
     end
    end
    else begin
        case(IState)//автомат состояний IState = текущее состояние
```

```
MS_Idle:begin
    case(CTR2)
    C2_READ_LINE:begin
        LineAddr=A2;
        ICount=99;
        IState=MS_Wait;
        LCount=8;
        NewState=MS_RDAnswer;
        answer_cmd=C2_NOP;
        D2Ctrl<=1;
    end
    C2_WRITE_LINE:begin
        LineAddr=A2;
        ICount=99;
        IState=MS_WRLine;
        LCount=7;
        Mem[(LineAddr<<4)+1]=D2[15:8];
        Mem[(LineAddr<<4)]=D2[7:0];
        LineAddr+=2;
    end
endcase
end
MS_Wait:begin
    ICount-=1;
    if (ICount==0) IState=NewState;
end
MS_RDAnswer: begin
    answer_cmd=C2_RESPONSE;
    answer_word=(Mem[(LineAddr<<4)+1]<<8)|Mem[(LineAddr<<4)];
    LCount-=1;
    if(LCount==0) begin
        IState=MS_Idle;
        D2Ctrl<=0;
    end
end
MS_WRLine: begin
    Mem[(LineAddr<<4)+1]=D2[15:8];
    Mem[(LineAddr<<4)]=D2[7:0];
    LineAddr+=2;
    LCount-=1;
    if(LCount==0) begin
        ICount=99-8;
        IState=MS_Wait;
        NewState=MS_WREnd;
    end
end
MS_WREnd: begin
    answer_cmd=C2_RESPONSE;
```

```
            answer_word=0;
            IState=MS_Idle;
            D2Ctrl<=0;
        end

    endcase


    end
end

endmodule;

module Cache (
    input CLK,                      //вход синхронизации
    input RESET,                    //вход сброса
    input [ADDR1_BUS_SIZE-1:0] A1,  //шина A1
    output [ADDR2_BUS_SIZE-1:0] A2, //шина A2
    inout [DATA1_BUS_SIZE-1:0] D1,          //шина D1
    inout [DATA2_BUS_SIZE-1:0] D2,          //шина D2
    inout [CTR1_BUS_SIZE-1:0] CTR1,         //шина C1
    inout [CTR2_BUS_SIZE-1:0] CTR2,         //шина C2
    input C_DUMP                    //сигнал сохранения состояния кэша
 );

localparam RS_Idle = 0;
localparam RS_Read2nd = 1;
localparam RS_Wait = 2;
//localparam RS_WriteWait = 3;
localparam RS_RDLogic = 4;
localparam RS_RDAnswer = 5;
localparam RS_RDAnswer2nd = 6;
localparam RS_WriteLine = 7;
localparam RS_WriteLineData = 8;
localparam RS_WaitMem = 9;
localparam RS_ReadLine = 10;
localparam RS_ReadWait = 11;
localparam RS_ReadLineData = 12;
localparam RS_Write2nd = 13;
localparam RS_WRLogic = 14;
localparam RS_WRAnswer = 15;

localparam RS_Invalidate = 21;


integer ICount = 0, LCount = 0;

reg [CACHE_LINE_SIZE-1:0] DataSet0 [0:CACHE_SETS_COUNT-1];
reg [CACHE_LINE_SIZE-1:0] DataSet1 [0:CACHE_SETS_COUNT-1];
```

```
reg [CACHE_TAG_SIZE-1:0] TagSet0 [0:CACHE_SETS_COUNT-1];
reg [CACHE_TAG_SIZE-1:0] TagSet1 [0:CACHE_SETS_COUNT-1];
reg Valid0 [0:CACHE_SETS_COUNT-1];
reg Valid1 [0:CACHE_SETS_COUNT-1];
reg Dirty0 [0:CACHE_SETS_COUNT-1];
reg Dirty1 [0:CACHE_SETS_COUNT-1];
reg ToFree [0:CACHE_SETS_COUNT-1];

reg [CACHE_TAG_SIZE-1:0] ctag,A2tag;
reg [CACHE_SET_SIZE-1:0] cset;
reg [CACHE_OFFSET_SIZE-1:0] coffset;

reg [DATA1_BUS_SIZE-1:0] answer_word;
reg [CTR1_BUS_SIZE-1:0] answer_cmd;
reg [DATA2_BUS_SIZE-1:0] mem_word;
reg [CTR2_BUS_SIZE-1:0] mem_cmd,mem_answer;
reg [DATA1_BUS_SIZE-1:0] wr_word0,wr_word1;

reg [CACHE_LINE_SIZE-1:0] ActLine;
reg ReadToSet;


reg D1Ctrl; // 1 - управлять шинами D1 CTR1 0 - не управлять
reg D2Ctrl; // 1 - управлять шинами D2 CTR2 0 - не управлять

wire hit0,hit1,hit;

assign hit0 = (TagSet0[cset]==ctag)&&Valid0[cset];
assign hit1 = (TagSet1[cset]==ctag)&&Valid1[cset];
assign hit = hit0 | hit1;

assign D1 = (D1Ctrl) ? answer_word : 16'dZ;
assign CTR1 = (D1Ctrl) ? answer_cmd : 3'dZ;
assign D2 = (D2Ctrl) ? mem_word : 16'dZ;
assign CTR2 = (D2Ctrl) ? mem_cmd : 2'dZ;
assign A2=(A2tag<<CACHE_SET_SIZE)|cset;

reg [5:0] IState,NewState,RetState; // IState - внутреннее состояние. NewState
- состояние в которое перейти по окончанию ожидания. RetState - состояние в
которое вернуться после записи строки в память.
reg [CTR1_BUS_SIZE-1:0] cmd;

integer i = 0;
  initial begin
    IState = RS_Idle;
    D1Ctrl='b0;
    for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin
        Valid0[i]=0;
        Valid1[i]=0;
```

```
        Dirty0[i]=0;
        Dirty1[i]=0;
    end;
  end;

always@(posedge CLK or posedge RESET or posedge C_DUMP)
begin
    if(RESET)
    begin
    IState = RS_Idle;
    D1Ctrl='b0;
    for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin
        Valid0[i]=0;
        Valid1[i]=0;
        Dirty0[i]=0;
        Dirty1[i]=0;
    end;
    end
    else if (C_DUMP) begin
     for (i = 0; i < CACHE_LINE_COUNT; i += 1) begin
       $display("Line [%d]: ToDel=%b; Set0: Valid=%b Dirty=%b Tag=%h Data=%h;
Set1: Valid=%b Dirty=%b Tag=%h Data=%h;", i, ToFree[i], Valid0[i], Dirty0[i],
TagSet0[i], DataSet0[i], Valid1[i], Dirty1[i], TagSet1[i], DataSet1[i]);
     end

    end
    else begin
        case(IState)//а это тот самый автомат состояний IState = текущее
состояние
        RS_Idle:begin
            case(CTR1)
            C1_INVALIDATE_LINE:begin
                ctag=A1[14:5];
                cset=A1[4:0];
                ICount=5;
                IState=RS_Wait;
                NewState=RS_Invalidate;
            end
            C1_READ8,C1_READ16,C1_READ32:begin
                ctag=A1[14:5];
                cset=A1[4:0];
                IState=RS_Read2nd;
                cmd=CTR1;
            end
            C1_WRITE8,C1_WRITE16,C1_WRITE32:begin
                ctag=A1[14:5];
                cset=A1[4:0];
                wr_word0=D1[15:0];
```

```
                IState=RS_Write2nd;
                cmd=CTR1;
            end
            endcase
        end
        RS_Invalidate:begin
            D1Ctrl='b1;
            answer_word=0;
            D1Ctrl<='b0;
            answer_cmd=C1_RESPONSE;
            if (hit0) Valid0[cset]=0;
            if (hit1) Valid1[cset]=0;
            IState=RS_Idle;
        end
        RS_Read2nd: begin
            coffset=A1[4:0];
            IState=RS_RDLogic;
            D1Ctrl='b1;
            answer_cmd=C1_NOP;
            answer_word='d0;
            end
        RS_RDLogic: begin
            if (hit==1'd1) begin
                ICount=3; // прошло два такта, пропустить еще 3 до ответа
                IState=RS_Wait;
                NewState=RS_RDAnswer;
                ToFree[cset]=hit0; //кого выкидывать при промахе
            end
            else begin //промах - надо читать память
              if (ToFree[cset]==1'b0) begin //выбрасываем линию из set0
                if ((Valid0[cset]&&Dirty0[cset])==1'b1) begin
                //линия валидна и грязная - сохранить линию в память
                    A2tag=TagSet0[cset];
                    ActLine=DataSet0[cset];
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_WriteLine;
                    ReadToSet=0;
                    RetState=RS_RDLogic; //после записи опять вернемся в это
состояние
                   end
                else begin
                    //линию можно выбросить и читать новую из памяти
                    A2tag=ctag;
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
```

```
                    NewState=RS_ReadLine;
                    ReadToSet=0;
                    RetState=RS_RDLogic; //после чтения опять вернемся в это
состояние
                  end
                end
                else begin
                //выбрасываем линию set1
                  if ((Valid1[cset]&Dirty1[cset])==1'b1) begin
                  //линия валидна и грязная - сохранить линию в память
                    A2tag=TagSet1[cset];
                    ActLine=DataSet1[cset];
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_WriteLine;
                    ReadToSet=1;
                    RetState=RS_RDLogic; //после записи опять вернемся в это
состояние
                   end
                  else begin
                    //линию можно выбросить и читать новую из памяти
                    A2tag=ctag;
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_ReadLine;
                    ReadToSet=1;
                    RetState=RS_RDLogic; //после чтения опять вернемся в это
состояние
                  end
                end
             end
          end
          RS_WriteLine:begin
              mem_word=ActLine[15:0];
              mem_cmd=C2_WRITE_LINE;
              D2Ctrl=1;
              LCount=(CACHE_LINE_SIZE/DATA2_BUS_SIZE);
          end
          RS_WriteLineData:begin
              ActLine>>=16;
              mem_word=ActLine[15:0];
              LCount-=1;
              if (LCount==0) begin
                  //кончилась линия
                  D2Ctrl<=0;
                  IState=RS_WaitMem;
```

```
    end
end
RS_WaitMem:begin
    if (CTR2==C2_RESPONSE) begin
            if (ReadToSet==0) Dirty0[cset]=0; //сбрасываем Dirty
                        else Dirty1[cset]=0;
        IState=RetState;
        mem_cmd=C2_NOP;
        D2Ctrl=1;
    end
end
RS_ReadLine:begin
    mem_cmd=C2_READ_LINE;
    D2Ctrl=1;
    LCount=(CACHE_LINE_SIZE/DATA2_BUS_SIZE);
end
RS_ReadWait:begin
    D2Ctrl=0;
    IState=RS_ReadLineData;
end
RS_ReadLineData:begin
    if (CTR2==C2_RESPONSE) begin
        ActLine<<=16;
        ActLine=ActLine|D2;
        LCount-=1;
        if (LCount==0) begin
        //прочитали всю линию
            if (ReadToSet==0) begin // Линия валидная и чистая
                DataSet0[cset]=ActLine;
                TagSet0[cset]=ctag;
                Valid0[cset]=1;
                Dirty0[cset]=0;
            end
            else begin
                DataSet1[cset]=ActLine;
                TagSet0[cset]=ctag;
                Valid1[cset]=1;
                Dirty1[cset]=0;
            end
            IState=RetState;
            mem_cmd=C2_NOP;
            D2Ctrl=1;
            end
    end
end
RS_Wait:begin
    ICount-=1;
    if (ICount==0) IState=NewState;
```

```
end
RS_RDAnswer: begin
    answer_cmd=C1_RESPONSE;
    case (cmd)
       C1_READ8:begin
          answer_word[15:8]=8'd0;
          if (hit0) answer_word[7:0]=DataSet0[cset][coffset];
          else answer_word[7:0]=DataSet1[cset][coffset];
          IState=RS_Idle;
          D1Ctrl<=0;
         end
           C1_READ16:begin
              if (hit0) begin
                 answer_word[7:0]=DataSet0[cset][coffset];
                 answer_word[15:8]=DataSet0[cset][coffset+1];
              end
              else begin
                answer_word[7:0]=DataSet1[cset][coffset];
                answer_word[15:8]=DataSet1[cset][coffset+1];
              end
              IState=RS_Idle;
              D1Ctrl<=0;
           end
           C1_READ32:begin
              if (hit0) begin
                 answer_word[7:0]=DataSet0[cset][coffset];
                 answer_word[15:8]=DataSet0[cset][coffset+1];
              end
              else begin
                answer_word[7:0]=DataSet1[cset][coffset];
                answer_word[15:8]=DataSet1[cset][coffset+1];
              end
              IState=RS_RDAnswer2nd;
           end
        endcase;
end
RS_RDAnswer2nd: begin
              if (hit0) begin
                 answer_word[7:0]=DataSet0[cset][coffset+2];
                 answer_word[15:8]=DataSet0[cset][coffset+2];
              end
              else begin
                answer_word[7:0]=DataSet1[cset][coffset+2];
                answer_word[15:8]=DataSet1[cset][coffset+3];
              end
              IState=RS_Idle;
              D1Ctrl<=0;
end
```

```
RS_Write2nd: begin
    coffset=A1[4:0];
    wr_word1=D1[15:0];
    IState=RS_WRLogic;
    D1Ctrl='b1;
    answer_cmd=C1_NOP;
    answer_word='d0;
    end
RS_WRAnswer: begin
    answer_cmd=C1_RESPONSE;
    IState=RS_Idle;
    D1Ctrl<=0;
end
RS_WRLogic: begin
    if (hit==1'd1) begin
        if (hit0) begin
            Dirty0[cset]=1;
            case (cmd)
            C1_WRITE8:DataSet0[coffset]=wr_word0[7:0];
            C1_WRITE16:begin
                DataSet0[coffset]=wr_word0[7:0];
                DataSet0[coffset+1]=wr_word0[15:8];
            end
            C1_WRITE16:begin
                DataSet0[coffset]=wr_word0[7:0];
                DataSet0[coffset+1]=wr_word0[15:8];
                DataSet0[coffset+2]=wr_word1[7:0];
                DataSet0[coffset+3]=wr_word1[15:8];
            end
            endcase
        end
        else begin
            Dirty1[cset]=1;
            case (cmd)
            C1_WRITE8:DataSet1[coffset]=wr_word0[7:0];
            C1_WRITE16:begin
                DataSet1[coffset]=wr_word0[7:0];
                DataSet1[coffset+1]=wr_word0[15:8];
            end
            C1_WRITE16:begin
                DataSet1[coffset]=wr_word0[7:0];
                DataSet1[coffset+1]=wr_word0[15:8];
                DataSet1[coffset+2]=wr_word1[7:0];
                DataSet1[coffset+3]=wr_word1[15:8];
            end
            endcase
        end
        ICount=3; // прошло два такта, пропустить еще 3 до ответа
```

```
                IState=RS_Wait;
                NewState=RS_WRAnswer;
                ToFree[cset]=hit0; //кого выкидывать при промахе
            end
            else begin //промах - надо читать память
              if (ToFree[cset]==1'b0) begin //выбрасываем линию из set0
                if ((Valid0[cset]&&Dirty0[cset])==1'b1) begin
                //линия валидна и грязная - сохранить линию в память
                    A2tag=TagSet0[cset];
                    ActLine=DataSet0[cset];
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_WriteLine;
                    ReadToSet=0;
                    RetState=RS_WRLogic; //после записи опять вернемся в это
состояние
                   end
                else begin
                    //линию можно выбросить и читать новую из памяти
                    A2tag=ctag;
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_ReadLine;
                    ReadToSet=0;
                    RetState=RS_WRLogic; //после чтения опять вернемся в это
состояние
                end
              end
              else begin
              //выбрасываем линию set1
                if ((Valid1[cset]&Dirty1[cset])==1'b1) begin
                //линия валидна и грязная - сохранить линию в память
                    A2tag=TagSet1[cset];
                    ActLine=DataSet1[cset];
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_WriteLine;
                    ReadToSet=1;
                    RetState=RS_WRLogic; //после записи опять вернемся в это
состояние
                   end
                else begin
                    //линию можно выбросить и читать новую из памяти
                    A2tag=ctag;
```

```
                    ICount=1; // пропустить такт до работы с памятью по
условиям 4й
                    IState=RS_Wait;
                    NewState=RS_ReadLine;
                    ReadToSet=1;
                    RetState=RS_WRLogic; //после чтения опять вернемся в это
состояние
                end
              end
            end
        end
        endcase
    end
end

endmodule;
/*
module CPU  (
    input CLK,                      //вход синхронизации
    output [ADDR1_BUS_SIZE-1:0] A1, //шина A1
    inout [DATA1_BUS_SIZE-1:0] D1,           //шина D1
    inout [CTR1_BUS_SIZE-1:0] CTR1          //шина C1
 );
endmodule;
*/
```

Аналитическое решение

```
program project1;

type cacheline=record
    tag:array[0..1]of integer;
    valid:array[0..1]of boolean;
    dirty:array[0..1]of boolean;
    todel:integer;
  end;

var
  tactcount,hitcount,accesscount:Int64;
  pa,pb,pc:integer;
  cache:array[0..31]of cacheline;
  i,y,x,k:integer;

const abeg=0;bbeg=2048;cbeg=5888;
      M=64;N=60;KK=32;

procedure mread(addr,plen:integer);
var csetln,ctag,hitset:integer;
  hit:boolean;
```

```
begin
     accesscount:=accesscount+1;
     csetln:=(addr shr 4) and $1F;
     ctag:=addr shr 9;
     hitset:=5;
     if cache[csetln].tag[0]=ctag then hitset:=0;
     if cache[csetln].tag[1]=ctag then hitset:=1;
     hit:=(hitset<5);
     if (hit) then begin
        cache[csetln].todel:=hitset xor 1;
        hitcount:=hitcount+1;
        tactcount:=tactcount+7;
        if plen>2 then tactcount:=tactcount+1;
        exit;
        end;
     hitset:=cache[csetln].todel;
     if (cache[csetln].valid[hitset] and
cache[csetln].dirty[hitset]) then begin
        cache[csetln].dirty[hitset]:=false;
        cache[csetln].tag[hitset]:=ctag;
        cache[csetln].todel:=hitset xor 1;
        tactcount:=tactcount+5+100+100+8;
        if plen>2 then tactcount:=tactcount+1;
        exit;
        end;
     cache[csetln].valid[hitset]:=true;
     cache[csetln].tag[hitset]:=ctag;
     cache[csetln].todel:=hitset xor 1;
     tactcount:=tactcount+5+100+8;
     if plen>2 then tactcount:=tactcount+1;
end;

procedure mwrite(addr,plen:integer);
var csetln,ctag,hitset:integer;
  hit:boolean;
begin
     accesscount:=accesscount+1;
     csetln:=(addr shr 4) and $1F;
     ctag:=addr shr 9;
     hitset:=5;
     if cache[csetln].tag[0]=ctag then hitset:=0;
     if cache[csetln].tag[1]=ctag then hitset:=1;
     hit:=(hitset<5);


     if (hit) then begin
        cache[csetln].todel:=hitset xor 1;
        cache[csetln].dirty[hitset]:=true;
        hitcount:=hitcount+1;
```

```
        tactcount:=tactcount+7;
        exit;
        end;
     hitset:=cache[csetln].todel;
     cache[csetln].dirty[hitset]:=true;
     if (cache[csetln].valid[hitset] and
cache[csetln].dirty[hitset]) then begin
        cache[csetln].tag[hitset]:=ctag;
        cache[csetln].todel:=hitset xor 1;
        tactcount:=tactcount+5+100+100+8;
        exit;
        end;
     cache[csetln].valid[hitset]:=true;
     cache[csetln].tag[hitset]:=ctag;
     cache[csetln].todel:=hitset xor 1;
     tactcount:=tactcount+5+100+8;
end;

begin
  tactcount:=4; //три инициалиции pa,pc,y и выход
  hitcount:=0;
  accesscount:=0;

  for i:=0 to 31 do begin
    cache[i].valid[0]:=false;
    cache[i].valid[1]:=false;
    cache[i].todel:=0;
  end;

  pa:=abeg;pc:=cbeg;
  for y:=0 to M-1 do begin
      tactcount:=tactcount+5; // инициализация x; y++, увеличение pa
и pc; условный переход на новую итерацию
      for x:=0 to N-1 do begin
          tactcount:=tactcount+5; // инициализация pb, s, k; x++ ;
условный переход на новую итерацию
          pb:=bbeg;
          for k:=0 to KK-1 do begin
              tactcount:=tactcount+5; // k++;сложение; увеличение
pb; умножение(5) ; условный переход на новую итерацию
              mread(pa+k,1);
              mread(pb+x,2);
              pb:=pb+N;
              end;
          mwrite(pc+x,4);
          end;
      pa:=pa+KK;
      pc:=pc+N;
      end;
```

```
    writeln('tacts=',tactcount);
    writeln('hitcount=',hitcount,' accesscount=',accesscount);
    writelnformat('hitrate={0:f3}%',hitcount/accesscount*100);
    readln;
end.
```