**Circuit modeling in Verilog**

**The purpose of the work:** cache construction and modeling of the processor-cache-memory system in the Verilog description language.

Toolkit: all code is written in Verilog, compilation and simulation – Icarus Verilog 10 and newer (useful materials: Verilog.docx ). In the report, you need to specify which version you used (you can also attach a link to the online platform). It is acceptable to use SystemVerilog, the main thing is that the code is compiled under Icarus 10, 11 or 12. Later in this document, Verilog+SystemVerilog is referred to as Verilog.

What I need

1. Analytical solution of the problem (the solution of the form to write code in a high-level language that emulates the operation of the system will be counted).
2. Simulation of a given system on Verilog.
3. Reproducing the task on Verilog.
4. Comparison of the results obtained.

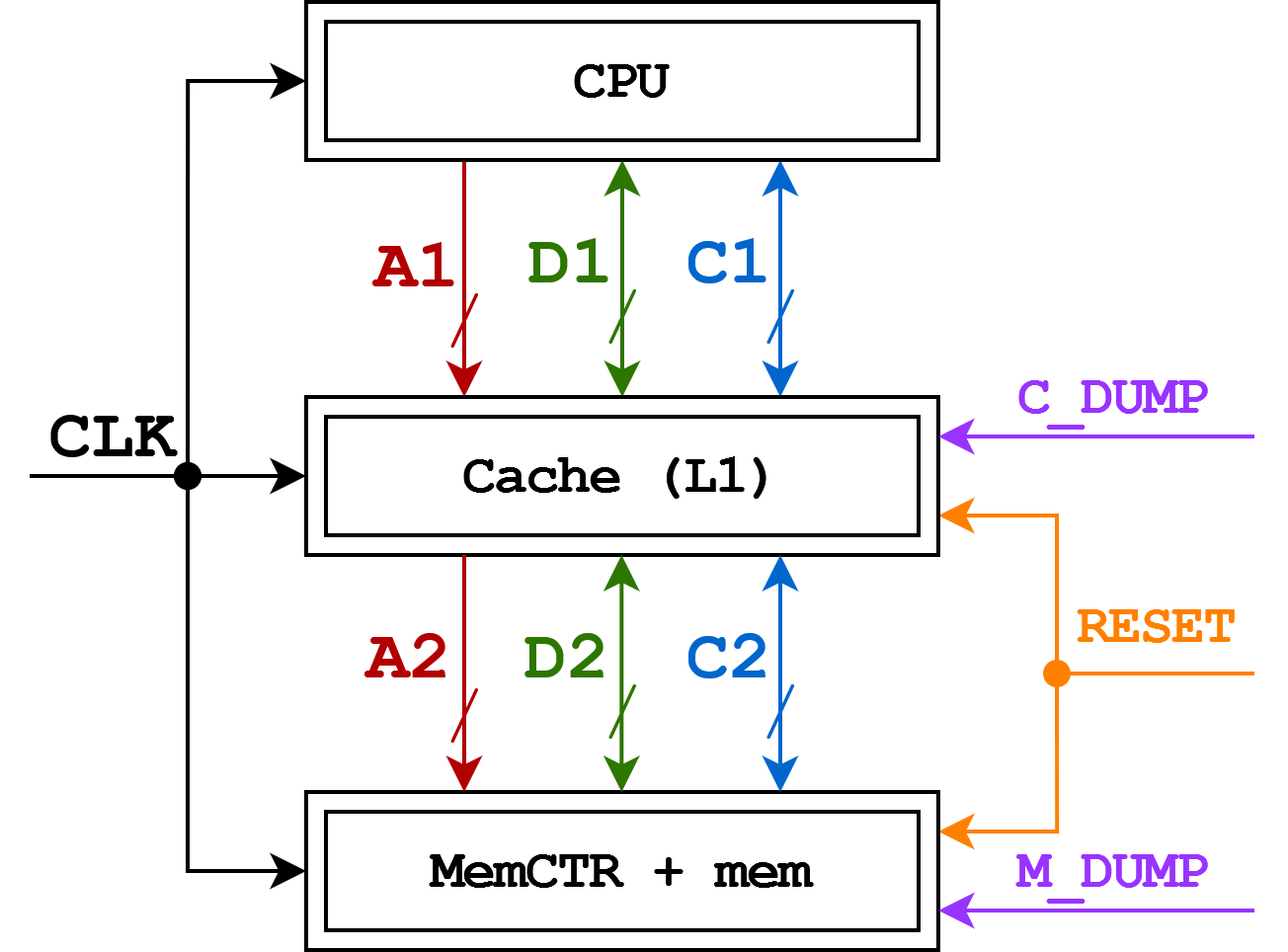
Explanations:

As an option, you are given some system parameters and a task that must first be solved analytically based on the system parameters, and then simulate and compare the results obtained.

In paragraph 2, describe how the model issued to you works. What are its constituent elements (which later become modules), what they are (for example, read, write, replace policies are set for the cache - you need to describe how they work). How you described the system from the condition using the Verilog hardware description language. How they work (when this or that operation is triggered, when data is read, written). You can illustrate the explanations with partial code listings (for example, the necessary block always shows and explains what it does).

Item 3. In a test environment, you have to simulate the task. Here it is desirable to give a time diagram or a log with time output.

**Processor-cache-memory system**



Symbols:

CACHE\_SIZE – names of variables/parameters that are recommended to be used in the work. The full list is in the section Parameter designations in the code.

Signals:

CLK – synchronization of the entire circuit

RESET – reset to the initial state

\*\_DUMP – saving the current state to a file/output to the console for debugging

Modules:

CPU – processor model for verifying cache operation

Cache – a single-level cache

MemCTR (+ mem) is a model of a memory controller + memory modules for verifying cache operation

Изображение выглядит как стол

Автоматически созданное описание

Сache line

Изображение выглядит как текст, стол

Автоматически созданное описание

Interpretation of the address by the cache

Везде, где не указаны значения, означает, что их нужно вычислить в ходе выполнения работы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CPU** | | | | | | |
| Commands | 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** | | Command 4 means invalidation of the entire cache line containing the specified address.  The number in commands means the number of data bits requested by this command.  Commands requesting multiple bytes cannot cross the cache line.  NOP – no operation.  Response – response to the command. | |
| CPU ← Cache | | 0 – **C1\_NOP**  7 – **C1\_RESPONSE** | |
| **Cache (look-through write-back)** | | | | | | |
| The policy of displacement | | | LRU | | | |
| Commands | Cache  → Mem | | 0 – **C2\_NOP**  2 – **C2\_READ\_LINE**  3 – **C2\_WRITE\_LINE** | | Commands are written and read in chunks equal to the size of the cache line. | |
| Cache  ← Mem | | 0 – **C2\_NOP**  1 – **C2\_RESPONSE** | |
| Service bits | | | V (valid), D (dirty) | | If valid is set to 0, then this cache line is free and the state of the remaining bits is not important.  dirty means that the cache line stores modified data that has not yet been written to memory. | |
| **Cache (continued)** | | | | | |
| Associativity | | 2 – **CACHE\_WAY** | | | |
| Address tag size | | 10 bit – **CACHE\_TAG\_SIZE** | | | |
| Cache line size | | 16 bytes – **CACHE\_LINE\_SIZE** | | The size of the payload. | |
| Number of cache lines | | 64 – **CACHE\_LINE\_COUNT** | |  | |
| **Memory** | | | | | |
| Memory size | | 512 Kbytes – **MEM\_SIZE** | |  | |

Tire dimension

|  |  |  |
| --- | --- | --- |
| The tire | Designation | Dimension |
| A1, A2 | **ADDR1\_BUS\_SIZE**, **ADDR2\_BUS\_SIZE** | Calculate it yourself |
| D1, D2 | **DATA1\_BUS\_SIZE, DATA2\_BUS\_SIZE** | 16 bit |
| C1, C2 | **CTR1\_BUS\_SIZE**, **CTR2\_BUS\_SIZE** | Calculate it yourself |

Response time

6 clock cycles – the time after which, as a result of a cache hit, the cache begins to respond.

4 clock cycles – the time after which, as a result of a cache miss, the cache sends a request to memory.

MemCTR – 100 clock cycles

Response time is the distance in clock cycles from the first clock cycle of the command to the first clock cycle of the response.

Bus data exchange protocol

Commands and responses to them are transmitted over the buses for several cycles in a row. But there can be an arbitrary number of idle cycles between the command and the response.

Over the A1 bus, the address is transmitted in 2 cycles: in the first cycle, the tag + set, in the second, the offset. Parts without offset are transmitted over the A2 address bus in 1 clock cycle.

On the D buses (D1 and D2), 16 bits of data are transmitted in each clock cycle, starting with the lowest, the small end.

On the C command lines (C1 and C2), the value is kept all the time the command or response is transmitted.

At the initial moment of time (or after the Reset), bus 1 is the processor, and bus 2 is the Cache. After submitting the command and before the end of sending the response, ownership of the bus passes to the Cache and MemCTR, respectively. Bus ownership means which device sets the logic levels on the bus wires.

Parameter designations in the code

In the code, commands can be indicated as comments near the designation of the command code, or they can be values in the enumeration (enumeration, more details):

C1\_NOP

C1\_READ8

C1\_READ16

C1\_READ32

C1\_INVALIDATE\_LINE

C1\_WRITE8

C1\_WRITE16

C1\_WRITE32

C1\_RESPONSE

C2\_NOP

C2\_READ\_LINE

C2\_WRITE\_LINE

C2\_RESPONSE (C2\_RESPONSE RESPONSE)

Variables, parameters, constants:

MEM\_SIZE – memory size

CACHE SIZE – cache size

CACHE LINE\_SIZE – the size of the cache line

CASH\_LINE\_COUNT – number of cache lines

CASH\_WAY – associativity

CACHE\_SETS\_COUNT – number of sets of cache lines

CACHE TAG\_SIZE – the size of the address tag

CACHE SET\_SIZE – the size of the index in the set of cache lines

CACHE\_OFFSET\_SIZE – the size of the offset

CACHE ADDR\_SIZE – the size of the address

**Implementation Details**

* You can use any constructions and variables of the Verilog and SystemVerilog description language.
* Default initialization (also RESET):
* • all cache lines in the invalid state
* • data in memory is initialized according to the following rule:

|  |
| --- |
| module test #(parameter \_SEED = 225526);    integer SEED = \_SEED;    reg[7:0] a[0:99];    integer i = 0;    initial begin      for (i = 0; i < 100; i += 1) begin        a[i] = $random(SEED)>>16;      end        for (i = 0; i < 100; i += 1) begin        $display("[%d] %d", i, a[i]);      end        $finish;    end  endmodule |

The initialization algorithm (including SEED) must not be changed.

**Task**

There is the following definition of global variables and functions:

|  |
| --- |
| #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;    }  } |

Addition, initialization of variables and transition to a new iteration of the loop, exit from the function take 1 clock cycle. Multiplication – 5 cycles. Accessing the memory of the type pc[x] is considered to be a single command.

Arrays are sequentially stored in memory, and the first one starts with 0.

All local variables are stored in processor registers.

Only data exchange (not commands) takes place over the simulated bus.

Determine the percentage of hits (the number of hits to the total number of hits) for the cache and the total time (in clock cycles) spent on performing this function.