10/18/2016

Ayush Sharma

VIT UNIVERSITY, CHENNAI CAMPUS

CSE2001: COMPUTER ARCHITECTURE AND ORGANIZATION

Simulated Direct mapping of Cache using VHDL

**Introduction:**

I have created a cache simulation using the VHDL and simulated the concept of direct Mapping and the Write-through policy using the created cache. ModelSim Simulator was used to simulate the VHDL code. Each block of cache consists of 8-bit word.

**I.** The Code for a 2X4 setup (2 Cache Blocks and 4 Memory Blocks) is given below:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

USE ieee.numeric\_std.ALL;

entity cache2block is port(

--Main ports

write\_data : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

addr\_to\_be\_read : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

read\_write : INOUT STD\_LOGIC;

clk : INOUT STD\_LOGIC;

read\_data : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

--Cache 0 ports

d0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld0 : INOUT STD\_LOGIC; -- load/enable.

output0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr0 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

addr0 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

ctr0 : INOUT STD\_LOGIC;

--Cache 1 port

d1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld1 : INOUT STD\_LOGIC;

output1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr1 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

addr1 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

ctr1 : INOUT STD\_LOGIC;

--Memory 0 port

mem\_d0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld0 : INOUT STD\_LOGIC;

mem\_output0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr0 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

--Memory 1 port

mem\_d1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld1 : INOUT STD\_LOGIC;

mem\_output1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr1 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

--Memory 2 port

mem\_d2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld2 : INOUT STD\_LOGIC;

mem\_output2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr2 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0);

--Memory 3 port

mem\_d3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld3 : INOUT STD\_LOGIC;

mem\_output3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr3 : INOUT STD\_LOGIC\_VECTOR(1 DOWNTO 0)

);

end cache2block;

ARCHITECTURE main\_cache\_2 OF cache2block IS

signal addr\_00:STD\_LOGIC\_VECTOR(1 DOWNTO 0):="00";

signal addr\_01:STD\_LOGIC\_VECTOR(1 DOWNTO 0):="01";

signal addr\_02:STD\_LOGIC\_VECTOR(1 DOWNTO 0):="10";

signal addr\_03:STD\_LOGIC\_VECTOR(1 DOWNTO 0):="11";

BEGIN

process(clk)

begin

if read\_write='1' then

if (addr\_to\_be\_read=addr\_00) or (addr\_to\_be\_read=addr\_02) then

if (addr\_to\_be\_read=addr0) then

read\_data<=output0;

elsif(addr\_to\_be\_read=mem\_addr0) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output0;

output0<=d0;

read\_data<=d0;

elsif(addr\_to\_be\_read=mem\_addr2) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output2;

output0<=d0;

read\_data<=d0;

end if;

elsif (addr\_to\_be\_read=addr\_01) or (addr\_to\_be\_read=addr\_03) then

if (addr\_to\_be\_read=addr1) then

read\_data<=output1;

elsif(addr\_to\_be\_read=mem\_addr1) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr1;

ld1<='1';

d1<=mem\_output1;

output1<=d1;

read\_data<=d1;

elsif(addr\_to\_be\_read=mem\_addr3) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr1;

ld1<='1';

d1<=mem\_output3;

output1<=d1;

read\_data<=d1;

end if;

end if;

else

if (addr\_to\_be\_read=addr0) then

ld0<='1';

d0<=write\_data;

end if;

if (mem\_addr0=addr0) then

mem\_ld0<='1';

mem\_d0<=write\_data;

mem\_output0<=write\_data;

elsif (mem\_addr2=addr0) then

mem\_ld2<='1';

mem\_d2<=write\_data;

mem\_output2<=write\_data;

end if;

if (addr\_to\_be\_read=addr1) then

ld1<='1';

d1<=write\_data;

end if;

if (mem\_addr1=addr1) then

mem\_ld1<='1';

mem\_d1<=write\_data;

mem\_output1<=write\_data;

elsif (mem\_addr3=addr1) then

mem\_ld3<='1';

mem\_d3<=write\_data;

mem\_output3<=write\_data;

end if;

end if;

end process;

END main\_cache\_2;

II. The code for the Larger 4X16 setup (4 Cache block and 16 Memory Blocks) is shown below:

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_arith.ALL;

USE ieee.std\_logic\_unsigned.ALL;

USE ieee.numeric\_std.ALL;

entity cache is port(

--Main ports

write\_data : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

addr\_to\_be\_read : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

read\_write : INOUT STD\_LOGIC;

clk : INOUT STD\_LOGIC;

read\_data : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

--Cache 0 ports

d0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld0 : INOUT STD\_LOGIC; -- load/enable.

output0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr0 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

addr0 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

ctr0 : INOUT STD\_LOGIC;

--Cache 1 port

d1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld1 : INOUT STD\_LOGIC;

output1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr1 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

addr1 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

ctr1 : INOUT STD\_LOGIC;

--Cache 2 ports

d2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld2 : INOUT STD\_LOGIC; -- load/enable.

output2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr2 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

addr2 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

ctr2 : INOUT STD\_LOGIC;

--Cache 3 port

d3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

ld3 : INOUT STD\_LOGIC;

output3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

src\_addr3 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

addr3 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

ctr3 : INOUT STD\_LOGIC;

--Memory 0 port

mem\_d0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld0 : INOUT STD\_LOGIC;

mem\_output0 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr0 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 1 port

mem\_d1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld1 : INOUT STD\_LOGIC;

mem\_output1 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr1 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 2 port

mem\_d2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld2 : INOUT STD\_LOGIC;

mem\_output2 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr2 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 3 port

mem\_d3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld3 : INOUT STD\_LOGIC;

mem\_output3 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr3 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 4 port

mem\_d4 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld4 : INOUT STD\_LOGIC;

mem\_output4 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr4 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 5 port

mem\_d5 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld5 : INOUT STD\_LOGIC;

mem\_output5 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr5 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 6 port

mem\_d6 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld6 : INOUT STD\_LOGIC;

mem\_output6 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr6 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 7 port

mem\_d7 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld7 : INOUT STD\_LOGIC;

mem\_output7 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr7 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 8 port

mem\_d8 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld8 : INOUT STD\_LOGIC;

mem\_output8 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr8 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 9 port

mem\_d9 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld9 : INOUT STD\_LOGIC;

mem\_output9 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr9 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 10 port

mem\_d10 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld10 : INOUT STD\_LOGIC;

mem\_output10 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr10 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 11 port

mem\_d11 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld11 : INOUT STD\_LOGIC;

mem\_output11 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr11 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 12 port

mem\_d12 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld12 : INOUT STD\_LOGIC;

mem\_output12 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr12 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 13 port

mem\_d13 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld13 : INOUT STD\_LOGIC;

mem\_output13 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr13 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 14 port

mem\_d14 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld14 : INOUT STD\_LOGIC;

mem\_output14 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr14 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0);

--Memory 15 port

mem\_d15 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_ld15 : INOUT STD\_LOGIC;

mem\_output15 : INOUT STD\_LOGIC\_VECTOR(7 DOWNTO 0);

mem\_addr15 : INOUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)

);

end cache;

ARCHITECTURE main\_cache OF cache IS

signal addr\_00:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0000";

signal addr\_01:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0001";

signal addr\_02:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0010";

signal addr\_03:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0011";

signal addr\_04:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0100";

signal addr\_05:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0101";

signal addr\_06:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0110";

signal addr\_07:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="0111";

signal addr\_08:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1000";

signal addr\_09:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1001";

signal addr\_10:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1010";

signal addr\_11:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1011";

signal addr\_12:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1100";

signal addr\_13:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1101";

signal addr\_14:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1110";

signal addr\_15:STD\_LOGIC\_VECTOR(3 DOWNTO 0):="1111";

BEGIN

process(clk)

begin

if read\_write='1' then

if (addr\_to\_be\_read=addr\_00) or (addr\_to\_be\_read=addr\_04)or (addr\_to\_be\_read=addr\_08)or (addr\_to\_be\_read=addr\_12) then

if (addr\_to\_be\_read=addr0) then

read\_data<=output0;

elsif(addr\_to\_be\_read=mem\_addr0) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output0;

output0<=d0;

read\_data<=d0;

elsif(addr\_to\_be\_read=mem\_addr4) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output4;

output0<=d0;

read\_data<=d0;

elsif(addr\_to\_be\_read=mem\_addr8) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output8;

output0<=d0;

read\_data<=d0;

elsif(addr\_to\_be\_read=mem\_addr12) then

src\_addr0<=addr\_to\_be\_read;

ctr0<='1'; --addr line

addr0<=src\_addr0;

ld0<='1';

d0<=mem\_output12;

output0<=d0;

read\_data<=d0;

end if;

elsif (addr\_to\_be\_read=addr\_01) or (addr\_to\_be\_read=addr\_05)or (addr\_to\_be\_read=addr\_09)or (addr\_to\_be\_read=addr\_13) then

if (addr\_to\_be\_read=addr1) then

read\_data<=output1;

elsif(addr\_to\_be\_read=mem\_addr1) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr1;

ld1<='1';

d1<=mem\_output1;

output1<=d1;

read\_data<=d1;

elsif(addr\_to\_be\_read=mem\_addr5) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr1;

ld1<='1';

d1<=mem\_output5;

output1<=d1;

read\_data<=d1;

elsif(addr\_to\_be\_read=mem\_addr9) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr0;

ld1<='1';

d1<=mem\_output9;

output1<=d1;

read\_data<=d1;

elsif(addr\_to\_be\_read=mem\_addr13) then

src\_addr1<=addr\_to\_be\_read;

ctr1<='1'; --addr line

addr1<=src\_addr1;

ld1<='1';

d1<=mem\_output13;

output0<=d1;

read\_data<=d1;

end if;

elsif (addr\_to\_be\_read=addr\_02) or (addr\_to\_be\_read=addr\_06)or (addr\_to\_be\_read=addr\_10)or (addr\_to\_be\_read=addr\_14) then

if (addr\_to\_be\_read=addr3) then

read\_data<=output2;

elsif(addr\_to\_be\_read=mem\_addr2) then

src\_addr2<=addr\_to\_be\_read;

ctr2<='1'; --addr line

addr2<=src\_addr2;

ld2<='1';

d2<=mem\_output2;

output2<=d2;

read\_data<=d2;

elsif(addr\_to\_be\_read=mem\_addr6) then

src\_addr2<=addr\_to\_be\_read;

ctr2<='1'; --addr line

addr2<=src\_addr2;

ld2<='1';

d2<=mem\_output6;

output2<=d2;

read\_data<=d2;

elsif(addr\_to\_be\_read=mem\_addr10) then

src\_addr2<=addr\_to\_be\_read;

ctr2<='1'; --addr line

addr2<=src\_addr2;

ld2<='1';

d2<=mem\_output10;

output2<=d2;

read\_data<=d2;

elsif(addr\_to\_be\_read=mem\_addr14) then

src\_addr2<=addr\_to\_be\_read;

ctr2<='1'; --addr line

addr2<=src\_addr2;

ld2<='1';

d2<=mem\_output14;

output2<=d2;

read\_data<=d2;

end if;

elsif (addr\_to\_be\_read=addr\_03) or (addr\_to\_be\_read=addr\_07)or (addr\_to\_be\_read=addr\_11)or (addr\_to\_be\_read=addr\_15) then

if (addr\_to\_be\_read=addr3) then

read\_data<=output3;

elsif(addr\_to\_be\_read=mem\_addr3) then

src\_addr3<=addr\_to\_be\_read;

ctr3<='1'; --addr line

addr3<=src\_addr3;

ld3<='1';

d3<=mem\_output3;

output3<=d3;

read\_data<=d3;

elsif(addr\_to\_be\_read=mem\_addr7) then

src\_addr3<=addr\_to\_be\_read;

ctr3<='1'; --addr line

addr3<=src\_addr3;

ld3<='1';

d3<=mem\_output5;

output3<=d1;

read\_data<=d3;

elsif(addr\_to\_be\_read=mem\_addr11) then

src\_addr3<=addr\_to\_be\_read;

ctr3<='1'; --addr line

addr3<=src\_addr3;

ld3<='1';

d3<=mem\_output11;

output3<=d3;

read\_data<=d3;

elsif(addr\_to\_be\_read=mem\_addr15) then

src\_addr3<=addr\_to\_be\_read;

ctr3<='1'; --addr line

addr3<=src\_addr3;

ld3<='1';

d3<=mem\_output15;

output3<=d3;

read\_data<=d3;

end if;

end if;

else

if (addr\_to\_be\_read=addr0) then

ld0<='1';

d0<=write\_data;

if (mem\_addr0=addr0) then

mem\_ld0<='1';

mem\_d0<=write\_data;

mem\_output0<=write\_data;

elsif (mem\_addr4=addr0) then

mem\_ld4<='1';

mem\_d4<=write\_data;

mem\_output4<=write\_data;

elsif (mem\_addr8=addr0) then

mem\_ld8<='1';

mem\_d8<=write\_data;

mem\_output8<=write\_data;

elsif (mem\_addr12=addr0) then

mem\_ld12<='1';

mem\_d12<=write\_data;

mem\_output12<=write\_data;

end if;

elsif (addr\_to\_be\_read=addr1) then

ld1<='1';

d1<=write\_data;

if (mem\_addr1=addr1) then

mem\_ld1<='1';

mem\_d1<=write\_data;

mem\_output1<=write\_data;

elsif (mem\_addr5=addr1) then

mem\_ld3<='1';

mem\_d3<=write\_data;

mem\_output3<=write\_data;

elsif (mem\_addr9=addr1) then

mem\_ld9<='1';

mem\_d9<=write\_data;

mem\_output9<=write\_data;

elsif (mem\_addr13=addr1) then

mem\_ld13<='1';

mem\_d13<=write\_data;

mem\_output13<=write\_data;

end if;

elsif (addr\_to\_be\_read=addr2) then

ld2<='1';

d2<=write\_data;

if (mem\_addr2=addr2) then

mem\_ld2<='1';

mem\_d2<=write\_data;

mem\_output2<=write\_data;

elsif (mem\_addr6=addr2) then

mem\_ld6<='1';

mem\_d6<=write\_data;

mem\_output6<=write\_data;

elsif (mem\_addr10=addr2) then

mem\_ld10<='1';

mem\_d10<=write\_data;

mem\_output10<=write\_data;

elsif (mem\_addr14=addr2) then

mem\_ld14<='1';

mem\_d14<=write\_data;

mem\_output14<=write\_data;

end if;

elsif (addr\_to\_be\_read=addr3) then

ld3<='1';

d3<=write\_data;

if (mem\_addr3=addr3) then

mem\_ld3<='1';

mem\_d3<=write\_data;

mem\_output3<=write\_data;

elsif (mem\_addr7=addr3) then

mem\_ld7<='1';

mem\_d7<=write\_data;

mem\_output7<=write\_data;

elsif (mem\_addr11=addr3) then

mem\_ld11<='1';

mem\_d11<=write\_data;

mem\_output11<=write\_data;

elsif (mem\_addr15=addr3) then

mem\_ld15<='1';

mem\_d15<=write\_data;

mem\_output15<=write\_data;

end if;

end if;

end if;

end process;

END main\_cache;

**OUTPUT OF SIMULATED CACHE**









