--- 2018 RSRC new "Multimaster RSRC VGA Testbench" VHDL Code

--- Current file name: testbench.vhd

--- Last Revised: 10/12/2018; 2:51 p.m.

--- Author: WDR

--- Copyright: William D. Richard, Ph.D.

LIBRARY IEEE ;

USE IEEE.STD\_LOGIC\_1164.ALL ;

USE IEEE.STD\_LOGIC\_ARITH.ALL ;

ENTITY testbench IS

PORT(clk : IN STD\_LOGIC ;

reset\_l : IN STD\_LOGIC ;

read : INOUT STD\_LOGIC ;

write : INOUT STD\_LOGIC ;

done : INOUT STD\_LOGIC ;

r : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

g : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

b : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

hs : OUT STD\_LOGIC ;

vs : OUT STD\_LOGIC) ;

END testbench ;

ARCHITECTURE structure OF testbench IS

COMPONENT clk\_wiz\_0

PORT (clk\_out1 : OUT STD\_LOGIC;

clk\_out2 : OUT STD\_LOGIC;

clk\_in1 : IN STD\_LOGIC);

END COMPONENT;

COMPONENT arbiter

PORT (clk : IN STD\_LOGIC ;

request0 : IN STD\_LOGIC ;

request1 : IN STD\_LOGIC ;

request2 : IN STD\_LOGIC ;

reset\_l : IN STD\_LOGIC ;

grant0 : OUT STD\_LOGIC ;

grant1 : OUT STD\_LOGIC ;

grant2 : OUT STD\_LOGIC) ;

END COMPONENT ;

COMPONENT rsrc

PORT(clk : IN STD\_LOGIC ;

reset\_l : IN STD\_LOGIC ;

d : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0) ;

address : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0) ;

read : OUT STD\_LOGIC ;

write : OUT STD\_LOGIC ;

request : OUT STD\_LOGIC ;

grant : IN STD\_LOGIC ;

done : IN STD\_LOGIC) ;

END COMPONENT ;

COMPONENT eprom

PORT(d : OUT STD\_LOGIC\_VECTOR(31 DOWNTO 0) ;

address : IN STD\_LOGIC\_VECTOR(9 DOWNTO 0) ;

ce\_l : IN STD\_LOGIC ;

oe\_l : IN STD\_LOGIC) ;

END COMPONENT ;

COMPONENT sram

PORT (d : INOUT STD\_LOGIC\_VECTOR(31 DOWNTO 0) ;

address : IN STD\_LOGIC\_VECTOR(9 DOWNTO 0) ;

ce\_l : IN STD\_LOGIC ;

oe\_l : IN STD\_LOGIC ;

we\_l : IN STD\_LOGIC ;

clk : IN STD\_LOGIC) ;

END COMPONENT ;

COMPONENT vga

PORT(src\_clk : IN STD\_LOGIC ;

ena : IN STD\_LOGIC ;

wea : IN STD\_LOGIC\_VECTOR(0 DOWNTO 0) ;

addra : IN STD\_LOGIC\_VECTOR(18 DOWNTO 0) ;

dina : IN STD\_LOGIC\_VECTOR(8 DOWNTO 0) ;

vga\_clk : IN STD\_LOGIC ;

r : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

g : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

b : OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0) ;

hs : OUT STD\_LOGIC ;

vs : OUT STD\_LOGIC);

END COMPONENT ;

SIGNAL reset\_l\_temp : STD\_LOGIC ;

SIGNAL reset\_l\_sync : STD\_LOGIC ;

SIGNAL src\_clk : STD\_LOGIC ;

SIGNAL vga\_clk : STD\_LOGIC ;

SIGNAL d : STD\_LOGIC\_VECTOR(31 DOWNTO 0):= "00000000000000000000000000000000" ;

SIGNAL address : STD\_LOGIC\_VECTOR(31 DOWNTO 0):= "00000000000000000000000000000000" ;

---SIGNAL read : STD\_LOGIC;

---SIGNAL write : STD\_LOGIC;

---SIGNAL done : STD\_LOGIC;

SIGNAL eprom\_ce\_l : STD\_LOGIC ;

SIGNAL eprom\_oe\_l : STD\_LOGIC ;

SIGNAL sram\_ce\_l : STD\_LOGIC ;

SIGNAL sram\_oe\_l : STD\_LOGIC ;

SIGNAL sram\_we\_l : STD\_LOGIC ;

SIGNAL vga\_ena : STD\_LOGIC;

SIGNAL vga\_wea : STD\_LOGIC\_VECTOR(0 DOWNTO 0) ;

SIGNAL request0 : STD\_LOGIC;

SIGNAL request1 : STD\_LOGIC;

SIGNAL request2 : STD\_LOGIC;

SIGNAL grant0 : STD\_LOGIC;

SIGNAL grant1 : STD\_LOGIC;

SIGNAL grant2 : STD\_LOGIC;

BEGIN

arb1:arbiter

PORT MAP(clk => src\_clk,

request0 => request0,

request1 => request1,

request2 => request2,

reset\_l => reset\_l,

grant0 => grant0,

grant1 => grant1,

grant2 => grant2);

mydcm1:clk\_wiz\_0

PORT MAP(clk\_out1 => src\_clk ,

clk\_out2 => vga\_clk ,

clk\_in1 => clk) ;

syncprocess:PROCESS(src\_clk)

BEGIN

IF (src\_clk = '1' AND src\_clk'event) THEN

reset\_l\_temp <= reset\_l ;

reset\_l\_sync <= reset\_l\_temp ;

END IF;

END PROCESS syncprocess ;

------------------------------------------------------------------------

rsrc0:rsrc

PORT MAP(clk => src\_clk,

reset\_l => reset\_l\_sync,

d => d,

address => address,

read => read,

write => write,

request => request0,

grant => grant0,

done => done);

rsrc1:rsrc

PORT MAP(clk => src\_clk,

reset\_l => reset\_l\_sync,

d => d,

address => address,

read => read,

write => write,

request => request1,

grant => grant1,

done => done);

rsrc2:rsrc

PORT MAP(clk => src\_clk,

reset\_l => reset\_l\_sync,

d => d,

address => address,

read => read,

write => write,

request => request2,

grant => grant2,

done => done);

------------------------------------------------------------------------

---done <= '1' WHEN (eprom\_ce\_l = '0' OR sram\_ce\_l = '0' OR vga\_ena = '1') ELSE 'Z' ;

------------------------------------------------------------------------

eprom\_ce\_l <= '0' WHEN (address(31 DOWNTO 12) = "00000000000000000000" AND read = '1') ELSE '1' ;

eprom\_oe\_l <= '0' WHEN read = '1' ELSE '1' ;

done <= '1' WHEN (eprom\_ce\_l = '0') ELSE 'Z' ;

erpom1:eprom

PORT MAP(d => d,

address => address(11 DOWNTO 2),

ce\_l => eprom\_ce\_l,

oe\_l => eprom\_oe\_l);

------------------------------------------------------------------------

sram\_ce\_l <= '0' WHEN (address(31 DOWNTO 12) = "00000000000000000001" AND (read = '1' OR write = '1')) ELSE '1' ;

sram\_oe\_l <= '0' WHEN read = '1' ELSE '1' ;

sram\_we\_l <= '0' WHEN write = '1' ELSE '1' ;

done <= '1' WHEN (sram\_ce\_l = '0') ELSE 'Z' ;

sram1:sram

PORT MAP(d => d,

address => address(11 DOWNTO 2),

ce\_l => sram\_ce\_l,

oe\_l => sram\_oe\_l,

we\_l => sram\_we\_l,

clk => src\_clk);

------------------------------------------------------------------------

vga\_ena <= '1' WHEN address(31 DOWNTO 21) = "11111111111" ELSE '0' ;

vga\_wea <= CONV\_STD\_LOGIC\_VECTOR(write,1) ;

done <= '1' WHEN (vga\_ena = '1') ELSE 'Z' ;

vga1:vga

PORT MAP(src\_clk => src\_clk,

ena => vga\_ena,

wea => vga\_wea,

addra => address(20 DOWNTO 2),

dina => d(8 DOWNTO 0),

vga\_clk => vga\_clk,

r => r,

g => g,

b => b,

hs => hs,

vs => vs);

------------------------------------------------------------------------

END structure;

--- Current "arbiter" VHDL Code

--- Current file name: arbiter.vhd

--- Last Revised: 10/23/2008; 10:15 p.m.

--- Author: WDR

--- Copyright: William D. Richard, Ph.D., 2008

LIBRARY IEEE ;

USE IEEE.STD\_LOGIC\_1164.ALL ;

USE IEEE.STD\_LOGIC\_ARITH.ALL ;

ENTITY arbiter IS

PORT (clk : IN STD\_LOGIC ;

request0 : IN STD\_LOGIC ;

request1 : IN STD\_LOGIC ;

request2 : IN STD\_LOGIC ;

reset\_l : IN STD\_LOGIC ;

grant0 : OUT STD\_LOGIC ;

grant1 : OUT STD\_LOGIC ;

grant2 : OUT STD\_LOGIC) ;

END arbiter ;

ARCHITECTURE behavioral of arbiter IS

TYPE states IS (idle012, idle021, idle102, idle120, idle201, idle210,

gr00, gr01, gr02, gr10, gr11, gr12) ;

SIGNAL state : states ;

SIGNAL nxt\_state : states ;

BEGIN

clkd:PROCESS(clk)

BEGIN

IF (clk'EVENT AND clk='1') THEN

IF (reset\_l = '0') THEN

state <= idle012;

ELSE

state <= nxt\_state;

END IF;

END IF;

END PROCESS clkd;

state\_trans:PROCESS(request0,request1,request2,state)

BEGIN

CASE state IS

when idle012 => if (request0 = '1') then nxt\_state <= gr00;

elsif (request1 = '1') then nxt\_state <= gr01;

elsif (request2 = '1') then nxt\_state <= gr02;

else nxt\_state <= idle012;

end if;

when idle021 => if (request0 = '1') then nxt\_state <= gr10;

elsif (request2 = '1') then nxt\_state <= gr02;

elsif (request1 = '1') then nxt\_state <= gr01;

else nxt\_state <= idle021;

end if;

when idle102 => if (request1 = '1') then nxt\_state <= gr01;

elsif (request0 = '1') then nxt\_state <= gr00;

elsif (request2 = '1') then nxt\_state <= gr12;

else nxt\_state <= idle102;

end if;

when idle120 => if (request1 = '1') then nxt\_state <= gr11;

elsif (request2 = '1') then nxt\_state <= gr12;

elsif (request0 = '1') then nxt\_state <= gr00;

else nxt\_state <= idle120;

end if;

when idle201 => if (request2 = '1') then nxt\_state <= gr02;

elsif (request0 = '1') then nxt\_state <= gr10;

elsif (request1 = '1') then nxt\_state <= gr11;

else nxt\_state <= idle201;

end if;

when idle210 => if (request2 = '1') then nxt\_state <= gr12;

elsif (request1 = '1') then nxt\_state <= gr11;

elsif (request0 = '1') then nxt\_state <= gr10;

else nxt\_state <= idle210;

end if;

when gr00 => if (request0 = '1') then nxt\_state <= gr00;

else nxt\_state <= idle120;

end if;

when gr01 => if (request1 = '1') then nxt\_state <= gr01;

else nxt\_state <= idle021;

end if;

when gr02 => if (request2 = '1') then nxt\_state <= gr02;

else nxt\_state <= idle012;

end if;

when gr10 => if (request0 = '1') then nxt\_state <= gr10;

else nxt\_state <= idle210;

end if;

when gr11 => if (request1 = '1') then nxt\_state <= gr11;

else nxt\_state <= idle201;

end if;

when gr12 => if (request2 = '1') then nxt\_state <= gr12;

else nxt\_state <= idle102;

end if;

END CASE ;

END PROCESS state\_trans ;

output:PROCESS(state)

BEGIN

CASE state IS

when gr00 | gr10 =>

grant0 <= '1';

grant1 <= '0';

grant2 <= '0';

when gr01 | gr11 =>

grant0 <= '0';

grant1 <= '1';

grant2 <= '0';

when gr02 | gr12 =>

grant0 <= '0';

grant1 <= '0';

grant2 <= '1';

when others =>

grant0 <= '0';

grant1 <= '0';

grant2 <= '0';

END CASE ;

END PROCESS output ;

END behavioral ;