Lecture 9

Prepared by: Dr. Alireza Kavianpour

Example:16x4 ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity rom_16_4 is
port (Clk, en : in std_logic;
addr : in unsigned(3 downto 0);
data : out unsigned(3 downto 0));
end rom_16_4;
architecture imp of rom_16_4 is
type rom_type is array (0 to 15) of unsigned(3 downto 0);
constant ROM : rom_type :=
(X"1", X"2", X"3", X"4", X"5", X"6", X"7", X"8",
X"9", X"A", X"B", X"C", X"D", X"E", X"F", X"1");
```

Example:16x4 ROM

```
begin
process (Clk)
begin
if rising_edge(Clk) then
if en = '1' then
data <= ROM(TO_INTEGER(addr));
end if;
end if;
end process;
end imp;
```

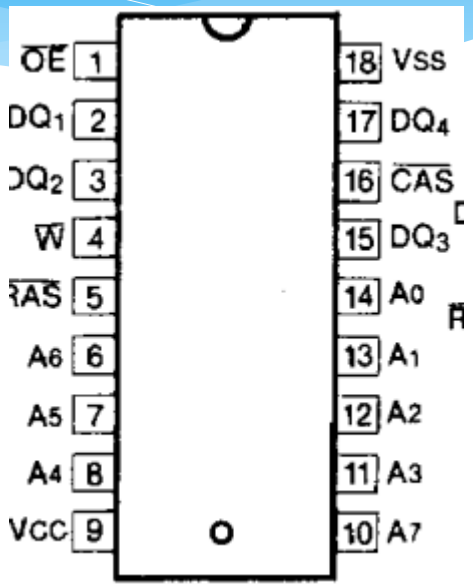
Function TO_INTEGER(addr) means:

TO_INTEGER(1001)=9

TO_INTEGER(1011)=11

Example:DRAM

Write entity portion of for 64k x 4 **DRAM**

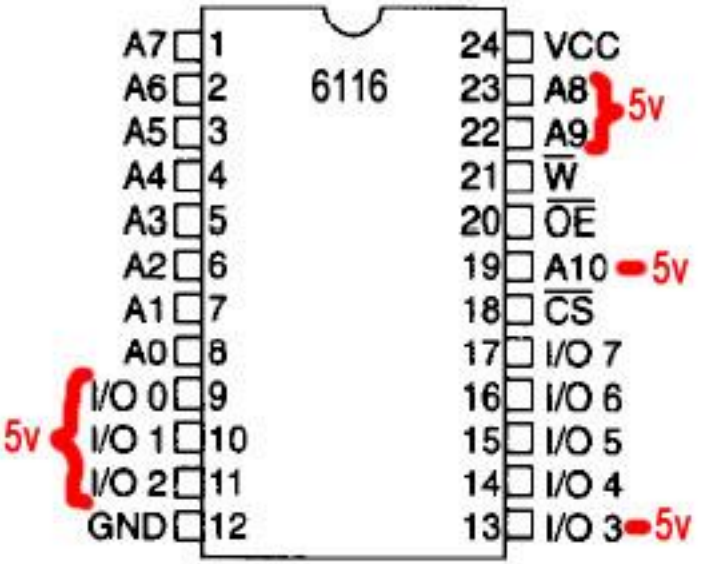



```
entity dram is
port (nOE: in STD_LOGIC;
nRAS: in STD_LOGIC;
nCAS: in STD_LOGIC;
nWE: in STD_LOGIC;
A: in STD_LOGIC_VECTOR (7 downto 0);
DQ: inout STD_LOGIC_VECTOR (3 downto 0));
end dram;
```

Example:SRAM

Write entity portion for 2k x 8 **SRAM**. Use minimum number of required inputs and outputs

(a)-Without generic constants.




```
entity sram is
port (nCS: in STD_LOGIC;
nOE: in STD_LOGIC;
nWE: in STD_LOGIC;
A: in STD_LOGIC_VECTOR (10 downto 0);
D: inout STD_LOGIC_VECTOR (7 downto 0));
end sram;
```

Example:SRAM

Write entity portion for 2k x 8 **SRAM**. Use minimum number of required inputs and outputs

(b)-With generic constants.

```
entity sram is
generic(w:positive:=8;    addr:positive:=11);
port (nCS: in STD_LOGIC;
nOE: in STD_LOGIC;
nWE: in STD_LOGIC;
A: in STD_LOGIC_VECTOR (addr-1 downto 0);
D: inout STD_LOGIC_VECTOR (w-1 downto 0));
end sram;
```

Example:File

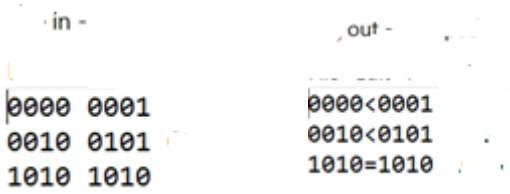
Read a file, assume two numbers/row separated by space.
Compare these two numbers
architecture Behavioral of file is
file myfile: text is in “C:/in.txt”;
file outfile: text is out “C:/out.txt”;
constant num_width: positive := 4;
signal num10,num20:bit_vector(num_width-1 downto 0);
signal compar1: character;
begin
process
variable num1,num2:bit_vector(num_width-1 downto 0) ;
variable space:character;
variable in, out: line;
begin
file_open(myfile, "C:/in.txt", read_mode);
file_open(outfile, "C:/out.txt", write_mode);

Example:File

```
while not endfile(myfile)loop
readline(myfile,in );
read(in ,num1);
read(in ,space);
read(in ,num2);
if (num1 > num2) then compare1  <= '>';
elsif (num1 = num2) then compare1 <= '=';
else compare1 <= '<';
end if;
num10 <= num1;
num21<= num2;
wait for 60ns;
```

Example:File

```
write(out, num10,right, num_width);
write(out, compar1,right,1);
write(out, num20,right,num_width);
writeline(outfile,out);
end loop;
file_close(myfile);
file_close(outfile);
wait;
end process;
end Behavioral;
```



Example

How to read images in VHDL in a way that the images can be loaded into the block memory of the FPGA during synthesis or simulation.
Since VHDL cannot read image files such as BMP, JPG, TIF, etc. directly, images are required to be converted into binary text files so that VHDL can read them using the TEXTIO VHDL package.
To convert images into binary text files, you can use Matlab or C. Once the image binary text files are ready, you can copy it to the project folder. Then, in VHDL, the binary text files can be read

Example

```
TYPE mem_type IS ARRAY(0 TO IMAGE_SIZE) OF std_logic_vector((DATA_WIDTH-1) DOWNTO 0)
variable temp_mem : mem_type;
variable buffer: line;
variable temp:  bit;

begin
    for i in mem_type'range loop
        readline(myfile, buffer);
        read(buffer, temp);
end loop;
```

Binary vs. Std_logic

Don’t assign binary to std_logic.

signal a : std_logic;

signal b : unsigned(7 **downto** 0);

a <= b = “01111111"; *BAD:*

result is a binary, not std_logic

a <= ’1’ **when** b = “01111111" **else** ’0’; *OK*

Default" values are convenient

OK

```
process (state, input)
begin
case state is
when S1 =>
if input = '1' then
output <= '0';
else
output <= '1';
end if;
when S2 =>
output <= '1';
end case;
end process;
```

Better

```
process (state, input)
begin
output <= '1';
case state is
when S1 =>
if input = '1' then
output <= '0';
end if;
end case;
end process;
```

Avoid asynchronous reset

OK
for external Reset

```
process (Clk, Reset)
begin
if Reset = '1' then
Q <= '0';
else
if rising_edge(Clk) then
Q <= D;
end if;
end if;
end process;
```

Better

```
process (Clk)
begin
if rising_edge(Clk) then
if Reset = '1' then
Q <= '0';
else
Q <= D;
end if;
end if;
end process;
```

Binary vs. Std_logic

Don’t test std_logic in a binarycontext.

signal a, b, foo : std_logic;

if a **then** *BAD:*

a is not binary

foo <= ’1’;

end if;

b <= ’0’ **when** a **else** ’1’; *BAD: a is not Binary*

if a = ’1’ **then** *OK*

foo <= ’1’;

end if;

b <= ’0’ **when** a = ’1’ **else** ’0’; *OK*

Output signal assigned

In a combinational process, make sure all output signals are always assigned.

```
process (x, y)
begin
if x = '1' then
y <= '0'; BAD: y not assigned when x = '0'
end if;
end process;
```

```
process (x, y)
begin
y <= '1'; OK: y is always assigned
if x = '1' then
y <= '0';
end if;
end process
```

Reading Output Port

```
library ieee;
use ieee.std_logic_1164.all;
entity dont_read_output is
port ( a : in std_logic; x, y : out std_logic );
end dont_read_output;
architecture BAD of dont_read_output is
begin
x <= not a;
y <= not x; Error: can't read an output port
end BAD;

architecture OK of dont_read_output is
signal x_sig : std_logic;
begin
x_sig <= not a;
x <= x_sig; x_sig just another name for x
y <= not x_sig; OK
end OK;
```

Complex Port Map

```
architecture BAD of bad_port_map is
component bar port (x : in unsigned(5 downto 0) );
end component;
signal a : unsigned(3 downto 0);
begin
mybar : bar port map ( x => "000" & a);          --BAD
end BAD;
```

```
architecture OK of bad_port_map is
component bar port (x : in unsigned(5 downto 0) );
end component;
signal a : unsigned(3 downto 0);
signal aa : unsigned(5 downto 0);
begin
aa <= "000" & a;
mybar : bar port map ( x => aa );       --OK
end OK;
```

Different type delay

One of VHDL's key points: can describe hardware and environment together.

Explicit delays are allowed

```
clk <= not clk after 50 ns;
process
begin
reset <= '0';
wait for 10 ns;          --Explicit delay
reset <= '1';
wait for a = '1';        --Delay for an event
assert b = '1' report "b did not rise" severity failure;
assert c = '1' report "c=0" severity warning;
wait for 50 ns;          --Delay for some time
wait;                    --Halt this process
end process;
```

Initialize variables

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity tlc_tb is
A testbench usually has no ports
end tlc_tb;
architecture tb of tlc_tb is
signal clk : std_logic := '0';          --Must initialize!
signal reset, cars, short, long : std_logic;
signal farm_red, start_timer : std_logic;
begin
clk <= not clk after 34.92 ns;

14 MHz
```

Three Modeling Styles

Combinational logic described by expressions

Simple case

```
a <= x and y;
```

When...else *selector*

```
a <= '1' when x = y else
'0';
```

With..*select selector*

```
with x select
a <='0' when '0',
'1' when '1',
'X' when others;
```

Package

Package declaration format:

```
package package_name is
   ... exported constant declarations
   ... exported type declarations
   ... exported subprogram declarations
end package_name;
```

Example:

```
package eecs31 is
   constant A: integer := 16 ;
   type arith is (signed, unsigned);
   function minimum(constant a, b: in integer) return integer;
end eecs31;
```

Standard VHDL Packages

```
library IEEE;
    use IEEE.std_logic_1164.all;
    use IEEE.std_logic_textio.all;
    use IEEE.std_logic_arith.all;
    use IEEE.numeric_bit.all;
    use IEEE.numeric_std.all;
    use IEEE.std_logic_signed.all;
    use IEEE.std_logic_unsigned.all;
    use IEEE.math_real.all;
    use IEEE.math_complex.all;
```

Standard VHDL Packages

* The package **_numeric_std_** provides numerical computation
* Types defined include: unsigned signed arrays of type std_logic for signals
* The package **_textio_** provides user input/output
* Types defined include:line , text
* Functions defined include: readline, read, writeline, write, endline
* The package **_std_logic_arith_** provides numerical computation.
* The package **_math_real_** provides numerical computation
* on type real.