Exercises

Mahdi Haghverdi

October 31, 2023

1 3.1

Write an entity declaration for a memory circuit whose input and output ports are shown below. Use only the `std_logic` or `std_logic_vector` data types.

- addr: 12-bit address input
- wra: 1-bit write-enable control signal
- oen: l-bit output-enable control signal
- bit: bidirectional data bus

1.1 Answer

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity MemoryCiruit is
5  port (
6      addr: in std_logic_vector(11 downto 0);
7      wra : in std_logic;
8      oen : in std_logic;
9      bit : inout std_logic_vector(11 downto 0)
10 );
```

2 3.5

Assume that `a` is a 10-bit signal with the `std_logic_vector(9 downto 0)` data type. List the 10 bits assigned to the a signal.

(a) `a <= (others=>'1');`

(b) `a <= (1|3|5|7|9=>'1', others=>'0');`

(c) `a <= (9|7|2=>'1', 6=>'0', 0=>'1', 1|5|8=>'0', 3|4=>'0');`

2.1 Answer

(a) `"1111111111"`

(b) `"0101010101"`

(c) `"1010000101"`

3 3.6

Assume that `a` and `y` are 8-bit signals with the `std_logic_vector(7 downto 0)` data type. If the signals are interpreted as unsigned numbers, the following assignment statement performs `a / 8`. Explain.

```
1 y <= "000" & a(7 downto 3);
```

3.1 Answer

Shifting the binary numbers does multiplication or division by 2 depending on the direction of the shift operation. • Left: $number \times 2$ • right: $\frac{number}{2}$

In the question, we've got division so we must do a right shift, but how many shifts are needed? $\log_2 8 = 3$. So we have to do the right shift 3 times.

In shifting the unsigned numbers, we don't care about the last digit because it is not the *sign* of the number, so we insert zero for each shift.

In the question, we have concatenated 3 zeros with the rest of the number (5 digits that are still present and not thrown away).

4 3.7

Assume the same `a` and `y` signals in Problem 3.6. We want to perform `a mod 8` and assign the result to `y`. Rewrite the previous signal assignment statement using only the & operator.

4.1 Answer

Taking `number mod` 2^n is equivalent to stripping off all but the n lowest-order (right-most) bits of `number`.

For example

number	number % 4
00000001	00000001
00000010	00000010
00000011	00000011
00000100	00000000
00000101	00000001
00000110	00000010

In order to do it only with concatenation in VHDL, we have to keep 3 bits ($log_2 8 = 3$) and make the rest bits to be zero.

```
1 y <= "00000" & a(2 downto 0)
```

5 3.8

Assume that the following double-quoted strings are with the `std_logic_vector` data type. Determine whether the relational operation is syntactically correct. If yes, what is the result (i.e., `true` or `false`)?

(a) `"0110" > "1001"`

(b) `"0110" > "0001001"`

(c) `2#1010# > "1010"`

(d) `1010 > "1010"`

5.1 Answer

(a) `false`

(b) `false`

(c) **type mismath**

(d) **type mismath**

6 3.11

Determine whether the following signal assignment is syntactically correct. If not, use the proper conversion function and type casting to correct the problem.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  ...
5  signal s1, s2, s3, s4, s5, s6, s7: std_logic_vector(3 downto
       0);
6  signal u1, u2, u3, u4, u5, u6, u7: unsigned(3 downto 0);
7  signal sg: signed(3 downto 0);
8  ...
9  u1 <= 2#0001#;
10 u2 <= u3 and u4;
11 u5 <= s1 + 1;
12 u6 <= u3 + u4 + 3;
13 u7 <= (others=>'1');
14 s2 <= s3 + s4 - 1;
15 s5 <= (others=>'1');
16 s6 <= u3 and u4;
17 sg <= u3 - 1;
18 s7 <= not sg;
```

6.1 Answer

- `u1 <= 2#0001#` → `u1` is `unsigned` but `2#0001#` is `integer`.

  Correct: `to_unsigned(2#0001#, 4)`

- `u2 <= u3 and u4` → **Correct**
- `u5 <= s1 + 1` → `std_logic_vector` does not support `+`

  Correct: `u5 <= unsigned(s1) + 1`
- `u6 <= u3 + u4 + 3` → **Correct**
- `u7 <= (others=>'1')` → **Correct**
- `s2 <= s3 + s4 - 1` → `std_logic_vector` does not support `+`

  Correct: `s2 <= std_logic_vector(unsigned(s3) + unsigned(s4) - 1)`
- `s5 <= (others=>'1')` → **Correct**
- `s6 <= u3 and u4` → `unsigned` type does not support `and`

  Correct: `s6 <= std_logic_vector(u3) and std_logic_vector(u4)`
- `sg <= u3 - 1` → `unsigned` does not support minus operator AND the result should be `signed`

  Correct: `sg <= signed(u3) - 1`
- `s7 <= not sg` → `singed` does not support `not` operator

  Correct: `s7 <= not std_logic_vector(sg)`

7 3.12

For the following VHDL segment, correct the type mismatch with proper conversion function(s).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 ...
5 signal src, dest: std_logic_vector(15 downto 0);
6 signal amount: std_logic_vector(3 downto 0);
7 ...
8 dest <= shift_left(src, amount);
```

7.1 Answer

8 3.13

For the following VHDL segment, correct the type mismatch with proper conversion function(s).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4 ...
5 signal src, dest: std_logic_vector(15 downto 0);
6 signal amount: std_logic_vector(3 downto 0);
7 ...
8 dest <= src sll amount;
```

8.1 Answer

9 3.14

For the following VHDL segment, correct the type mismatch with proper conversion function(s).

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_arith.all;
4 use ieee.numeric_std.all;
5 ...
6 signal src, dest: std_logic_vector(15 downto 0);
7 signal amount: std_logic_vector(3 downto 0);
8 ...
9 dest <= src sll amount;
```