31000
ELEKTRONIK

Counters

Purpose	Acquire practical skills in developing behavioural VHDL of counters.
Assignment	Develop behavioural VHDL models of the digital circuits specified in the exercise. Check the correctness of the models through a simulation.
Contents	• Counter. • Counter with asynchronous RESET. • Binary coded decimals (BCD) counter. • Up/Down counter. • Up/Down BCD counter. • Up/Down counter with a parallel load.

Before starting the exercise answer the following questions:

- What is a counter.
- Draw a 4-bit ripple counter and its time diagram. Explain how it works.
- Draw a synchronous 4-bit counter with T flip-flops (see Wakerly, section 8.4.2). Make sure that you understand how it works.
- Draw a 4-bit synchronous counter using D flip-flops (hint: see the slides from the lectures in course 31000). Write the truth table of the half-adders used in the design.
- Draw a 4-bit up/down counter using D flip-flops (hint: exchange the the half adders with half adder/subtractors). What is the difference between a half adder and a half subtractor.
- What are binary coded decimals ?
- How do you sum BCD numbers ? Give an example.

Counter

The ports of a 4-bit counter are given in Table 1.

The VHDL model of the counter is given in Fig. 1. Consider the following features of the model:

- An additional signal *TMP* was used. Its value is assigned to the output through a concurrent assignment operator. This is necessary because the port *COUNT* is an output, and cannot be read internally, i.e. an expression of the type `COUNT <= COUNT + 1` is not possible. That is why, the computations are done using the internal signal *TMP*, and the result is sent to the output port.

Signal	Type	Direction	Function
CLK	**`std_logic`**	Input	Clock
COUNT	**`std_logic_vector`**(3 **`downto`** 0)	Output	Counter's output

Table 1: Description of the ports of a counter.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BCNT is port (
  CLK: in std_logic;
  COUNT: out std_logic_vector(3 downto 0));
end BCNT;

architecture simple of BCNT is

signal TMP: unsigned(3 downto 0) := (others=>'0');

begin

  increment: process (CLK) begin
    if rising_edge(CLK) then
      TMP <= TMP + 1;
    end if;
  end process;

  COUNT <= std_logic_vector(TMP);

end simple;
```

Figure 1: Behavioural description of a Counter.

- *TMP* is defined as **`unsigned`** in order to participate in arithmetic operations. The type **`unsigned`**, and respective functions for manipulating data of this type are given in the package **`numeric_std`** from the library **`ieee`**.
- An initial value of `'0'` is assigned for all the bits of the vector *TMP* when it is declared. This initial value is necessary *only* for the simulation. During the synthesis it is *ignored*.

Simulate the VHDL model by using the following input signals in the simulator:

```
wave /*
force clk 0 0, 1 50 -repeat 100
run 2000
```

The results from the simulations are given in Fig. 2.

Counter with asynchronous RESET

Add an input port *RST* for asynchronous reset of the counter implemented in the previous section. This port should be active low.

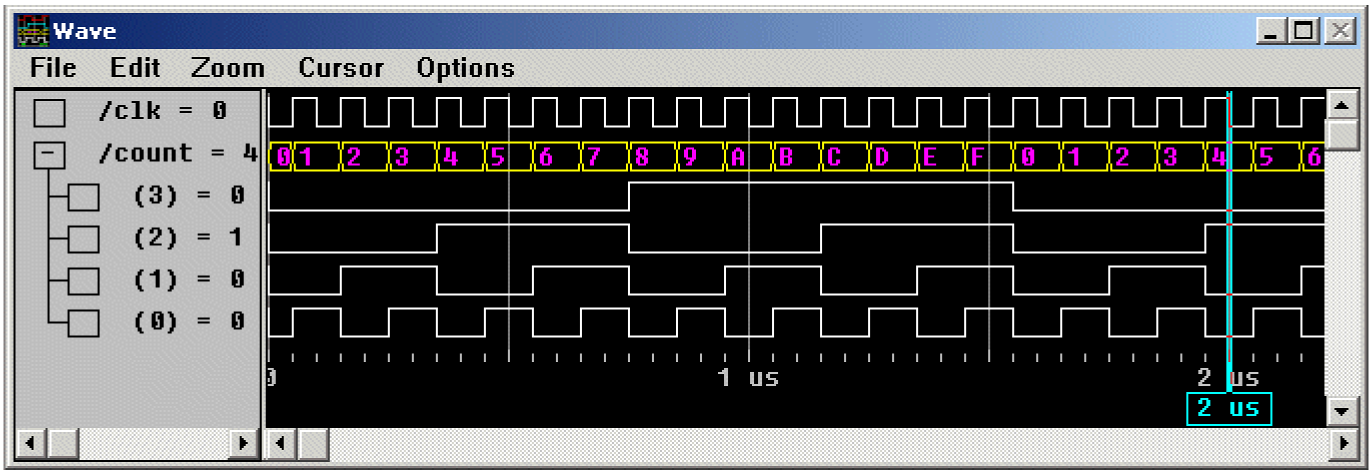
Wave
File Edit Zoom Cursor Options
/clk = 0
/count = 4
(3) = 0
(2) = 1
(1) = 0
(0) = 0
0 1 2 3 4 5 6 7 8 9 A B C D E F 0 1 2 3 4 5 6
1 us
2 us
2 us

Figure 2: A simulation of a 4-bit counter.

Directions:

In order to implement an asynchronous reset, the check for the level of the signal *RST* must be done prior to the check for the rising edge of the clock.

BCD counter

The ports of a BCD counter with asynchronous reset are given in Table 2.

Signal	Type	Direction	Function
CLK	**`std_logic`**	Input	Clock
RST	**`std_logic`**	Input	Asynchronous RESET. Active low
COUNT	**`std_logic_vector`**(3 **`downto`** 0)	Output	Counter's output

Table 2: Description of the ports of a counter.

The VHDL model of the BCD counter is given in Fig. 3.
Simulate the model using the following input signals:

```
wave /*
force clk 0 0, 1 50 -repeat 100
force rst 0 0, 1 30, 0 1500
run 2000
```

The results of the simulation are given in Fig. 4:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BCDCNT is port (
  CLK: in std_logic;
  RST: in std_logic;
  COUNT: out std_logic_vector(3 downto 0));
end BCDCNT;

architecture simple of BCDCNT is

signal TMP: unsigned(3 downto 0);

begin

  increment: process (CLK,RST) begin
     if RST = ’0’ then
        TMP <= "0000";
     elsif rising_edge(CLK) then
       if TMP = "1001" then
          TMP <= "0000";
       else
          TMP <= TMP + 1;
       end if;
     end if;
  end process;

  COUNT <= std_logic_vector(TMP);

end simple;
```

Figure 3: Behavioural description of a BCD counter.

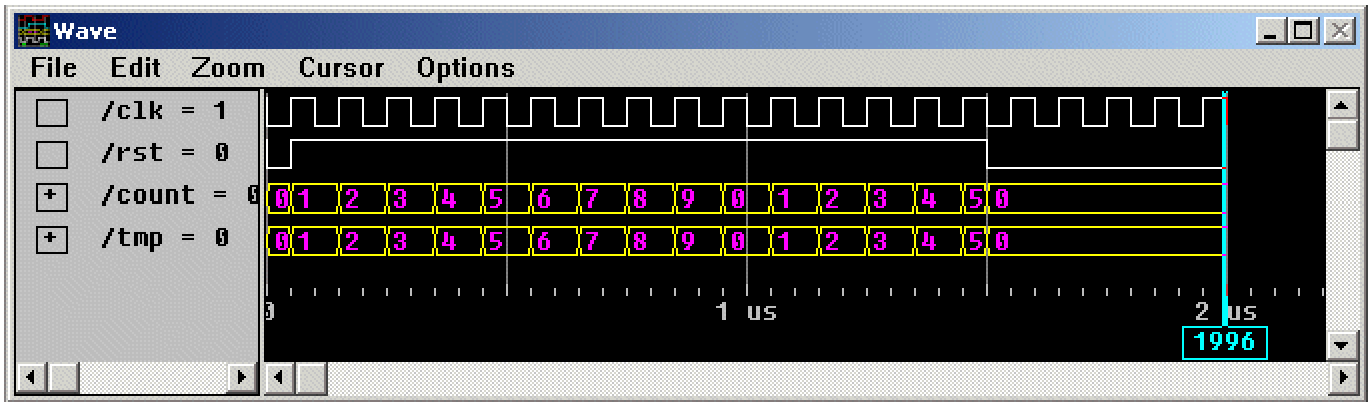

Figure 4: A simulation of a BCD counter.

Up/Down counter

The ports of a 4-bit Up/Down counter with asynchronous reset are given in Table 3.

Signal	Type	Direction	Function
CLK	**std_logic**	Input	Clock
RST	**std_logic**	Input	Asynchronous RESET. Active low
UP	**std_logic**	Input	Control of the counting direction *UP*='1' - increment *UP*='0' - decrement
COUNT	**std_logic_vector**(3 **downto** 0)	Output	Counter's output

Table 3: Description of the ports of a 4-bit Up/Down counter.

The VHDL model of the counter is given in Fig. 5
Simulate the model by choosing the input signals such as to get a result from the simulation as the one shown in Fig. 6.

Up/Down BCD Counter

Develop a behavioural VHDL model of a 4-bit Up/Down BCD counter. The ports of the counter are described in Table 4

Signal	Type	Direction	Function
CLK	**std_logic**	Input	Clock
RST	**std_logic**	Input	Asynchronous RESET. Active low
UP	**std_logic**	Input	Control of the counting direction *UP*='1' - increment *UP*='0' - decrement
COUNT	**std_logic_vector**(3 **downto** 0)	Output	Counter's output (BCD code)

Table 4: Description of the ports of a 4-bit Up/Down BCD counter.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UDCNT is port (
  CLK: in std_logic;
  RST: in std_logic;
  UP: in std_logic;
  COUNT: out std_logic_vector(3 downto 0));
end UDCNT;

architecture rtl of UDCNT is

signal TMP: unsigned(3 downto 0);

begin

  increment: process (CLK,RST) begin
     if RST = ’0’ then
        TMP <= "0000";
     elsif rising_edge(CLK) then
       if UP = ’1’ then
          TMP <= TMP + 1;
       else
          TMP <= TMP - 1;
       end if;
     end if;
  end process;

  COUNT <= std_logic_vector(TMP);

end rtl;
```

Figure 5: Behavioural description of a 4-bit Up/Down counter with asynchronous RESET.

Directions:

Use for a base the models of the Up/Down counter and BCD counter.

Simulate the VHDL model by choosing the input stimuli such as to get the simulation result shown in Fig. 7

Up/Down counter with parallel load

The VHDL model of the counter is given in Fig. 8.
Simulate the VHDL model by using the following input stimuli:

```
wave /*
force clk 0 0, 1 50 -repeat 100
force rst 0 0, 1 50
force up 1 0
```

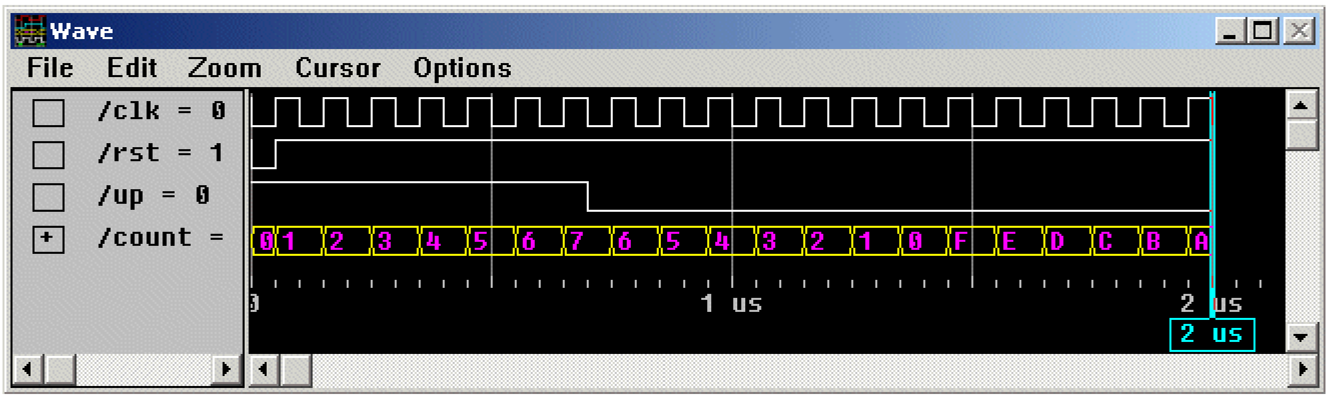
Wave
File Edit Zoom Cursor Options
/clk = 0
/rst = 1
/up = 0
/count =
0 1 2 3 4 5 6 7 6 5 4 3 2 1 0 F E D C B A
1 us
2 us
2 us

Figure 6: A simulation of a 4-bit Up/Down counter.

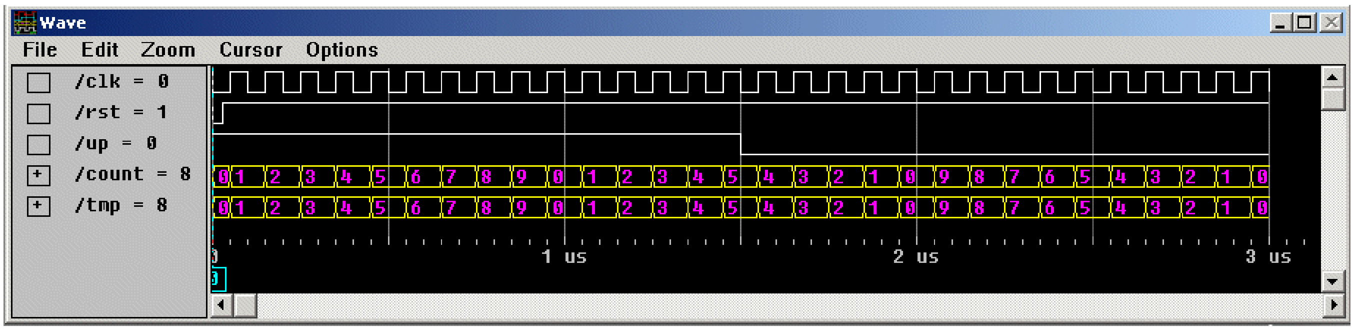

Figure 7: The result of a simulation of a 4-bit Up/Down BCD counter.

```
force load 0 0, 1 800, 0 1200
force din 1010
run 2000
```

The results of the simulation are given in Fig. 9.

Signal	Type	Direction	Function
CLK	**std_logic**	Input	Clock
RST	**std_logic**	Input	Asynchronous RESET. Active low
UP	**std_logic**	Input	Control of the counting direction *UP*='1' - increment *UP*='0' - decrement
LOAD	**std_logic**	Input	Control of the load/count mode *LOAD*='1' - loading mode *LOAD*='0' - counting mode
DIN	**std_logic_vector**(3 **downto** 0)	Input	Parallel load data input
COUNT	**std_logic_vector**(3 **downto** 0)	Output	Counter's output (BCD code)

Table 5: Description of the ports of a 4-bit Up/Down counter with a parallel load.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UDCNTLOAD is port (
  CLK: in std_logic;
  RST: in std_logic;
  UP: in std_logic;
  LOAD: in std_logic;
  DIN: in std_logic_vector(3 downto 0);
  COUNT: out std_logic_vector(3 downto 0));
end UDCNTLOAD;

architecture rtl of UDCNTLOAD is

signal TMP: unsigned(3 downto 0);

begin

  increment: process (CLK,RST) begin
     if RST = ’0’ then
        TMP <= "0000";
     elsif rising_edge(CLK) then
        if LOAD = ’1’ then
           TMP <= unsigned(DIN);
        elsif UP = ’1’ then
           TMP <= TMP + 1;
        else
           TMP <= TMP - 1;
       end if;
     end if;
  end process;

  COUNT <= std_logic_vector(TMP);

end rtl;
```

Figure 8: Behavioural description of a 4-bit Up/Down counter with a parallel load.

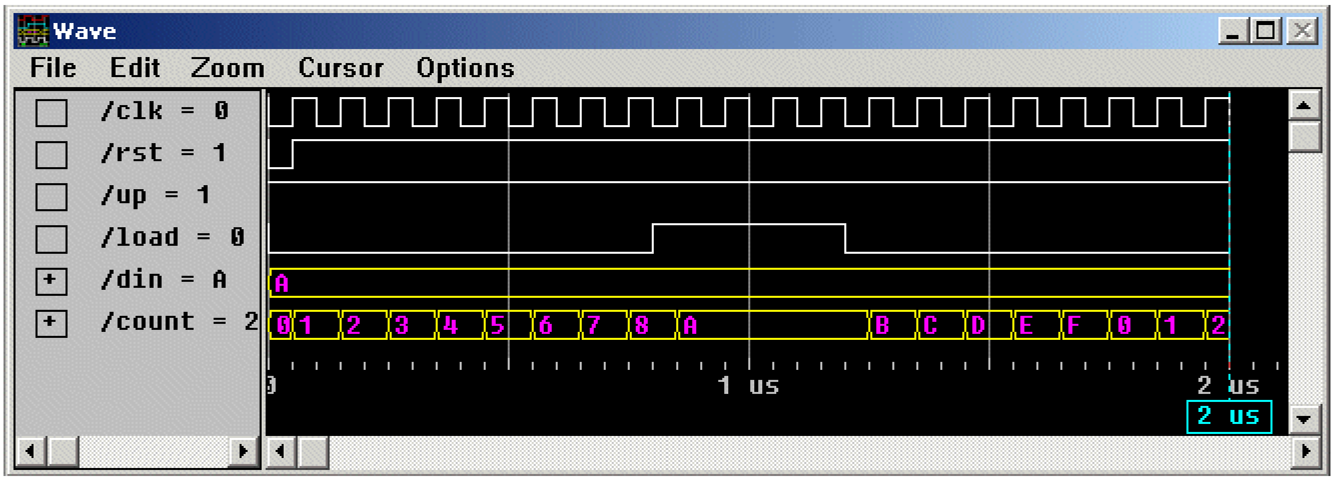

Figure 9: The result of a simulation of a 4-bit Up/Down counter with a parallel load.