Electrical and Computer Engineering

Solutions to Digital System Projects Final Practice Problems

By Prawat Nagvajara

Problem 1

```
-- Problem 1. Design an up/down counter
-- using arith and unsigned package
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity counter is
  generic( N: natural := 4);
  port (ck, reset, up_down: in std_logic;
        z : out std_logic_vector(n-1 downto 0)
      );
end counter;
architecture beh of counter is
  signal temp: std_logic_vector(n-1 downto 0);
begin
process(ck)
begin
if ck='1' and ck'event then
  if reset = '1' then
    temp <= (others => '0');
  else
    if up_down = '0' then
      temp <= temp + 1;
    else
      temp <= temp - 1;
    end if;
  end if;
end if;
end process;
z <= temp;
end beh;
```

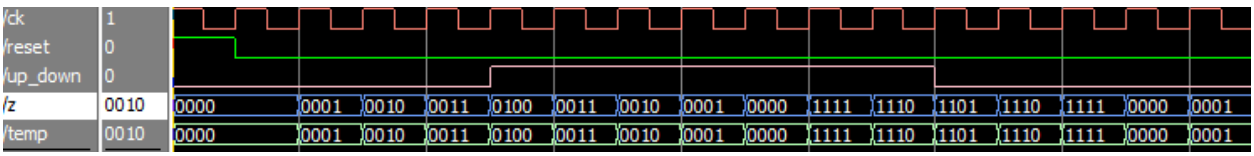

Fig. 1 Simulation Wave

Figure 1 shows simulation wave of the up/down counter where the contents of the counter turns over from “0000” to “1111” = -1 when it is counting down.

Problem 2 Up-down counter IP from IP Catalog

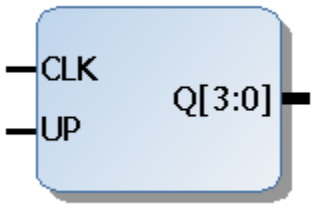

Fig. 2 Up Down Counter IP Schematic (4-bit)

```
--------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- IP Catalog binary counter
--------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity updown_count is
port( CK, updown : IN STD_LOGIC;
Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
end updown_count;

architecture Behavioral of updown_count is
COMPONENT c_counter_binary_0
  PORT (
    CLK : IN STD_LOGIC;
    UP : IN STD_LOGIC;
    Q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END COMPONENT;
begin
U : c_counter_binary_0
PORT MAP (CLK => CK,UP => UPdown,Q => Q);
end Behavioral;
```

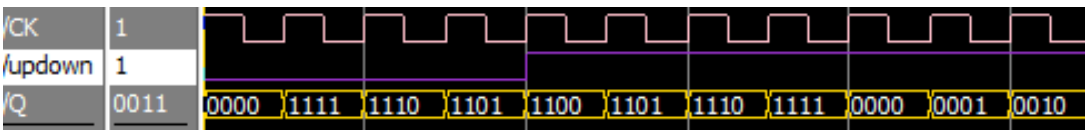

Fig. 3 Up Down Counter Wave

Problem 3

Write a test bench for the factorial sequential circuit for n = 1, …, 10

Factorial sequential calculator code

```
----------------------------------------------------
-- Factorial Sequential Calculator
-- After a reset each clock cycle n,
-- n = 1, 2, ... the output is n!
----------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity fact is
generic (K: natural := 32); -- K number of bits
port(ck, reset: in std_logic;
     LED: out std_logic_vector(K-1 downto 0));
end fact;

architecture Behavioral of fact is
signal temp: std_logic_vector(2*K - 1 downto 0);
signal n: std_logic_vector(K-1 downto 0);

begin
process(ck)
begin
if ck='1' and ck'event then
   if reset = '1' then
   -- initialize to vector "00..01"
  temp <= (0=>'1',others=>'0');
  n <= (0=>'1',others=>'0');
  else
  -- K by K product is 2K bits
  temp <= n * temp(K-1 downto 0);
  n <= n+1;
  end if;
end if;
end process;
led <= temp(K-1 downto 0); -- K-bit data
end Behavioral;
```

```
--------------------------------
-- factorial circuit test bench
--------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity fact_tb is
end fact_tb;
architecture Beh of fact_tb is
component fact
generic (K: natural := 32);--K number of bits
port(ck, reset: in std_logic;
     LED: out std_logic_vector(K-1 downto 0));
end component;
signal z: std_logic_vector(31 downto 0);
signal n, n_factorial: integer := 1;
signal ck: std_logic := '1';
type my_state is (init,run_test,done);
signal n_s: my_state := init;
signal reset: std_logic := '1';
begin
ck <= not ck after 5 ns;
dut: fact generic map(32)
          port map(ck, reset, z);                    \
process(ck)
variable n, n_factorial: integer := 0;
begin
if ck = '1' then
  case n_s is
  when init => reset <= '1';
               n := 1;
         n_factorial := 1;
         n_s <= run_test;
  when run_test => reset <= '0';
            n_factorial := n*N_factorial;
      n := n + 1;
  assert conv_integer(unsigned(z)) = n_factorial
         report "incorrect" severity ERROR;
            if n > 11 then
         n_s <= done;
      end if;
  when done => assert FALSE
       report "test completed" severity FAILURE;
  end case;
end if;
end process;
end beh;
```

Fig. 4 Factorial Test Bench Wave

Problem 4

```
-- Problem 4 Design a combinational block to add
-- two N-digit decimal numbers x and y and
-- implement on FPGA for N = 2.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity decimal_N_digit_add is
generic(N: natural := 3);
port ( x, y : in std_logic_vector(4*N - 1 downto 0);
           z: out std_logic_vector(4*N - 1 downto 0)
);
end decimal_N_digit_add;

architecture beh of decimal_N_digit_add is
begin
process(x,y)
variable carry: std_logic;
variable temp, temp_x, temp_y: std_logic_vector(4 downto 0);
begin
-- reset carry
carry := '0';
for i in 0 to N-1 loop
  temp_x := '0'&x(4*(i+1)-1 downto 4*i);
  temp_y := '0'&y(4*(i+1)-1 downto 4*i);
  temp := temp_x + temp_y;
if carry = '1' then
  temp := temp + 1;
end if;
if temp > 9 then
  carry := '1';
  temp := temp + 6;
end if;
-- assign temp to output
z(4*(i+1)-1 downto 4*i) <= temp(3 downto 0);
end loop;
end process;
end beh;
```

Fig. 5 Decimal N-digit Adder

Problem 5

```
-- Problem 5 Design a signed accumulator similar to
-- the IP catalog core with generic parameters,
-- n-bit data and L clock-cycles latency.
-- Use a shift register with depth L-1 to buffer
-- the input data to emulate the latency.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity accumulator_latency_L is
generic( N: natural := 4;
         L: natural := 3);
port (ck, reset : in std_logic;
       x : in  std_logic_vector(n-1 downto 0);
        z : out std_logic_vector(2*n-1 downto 0)
      );
end accumulator_latency_L;

architecture beh of accumulator_latency_L is
type vector_array is
array (natural range <>) of std_logic_vector(n-1 downto 0);
signal latency_temp: vector_array(0 to L-2);
signal zero_vector, one_vector: std_logic_vector(n-1 downto 0);
signal temp, temp_x: std_logic_vector(2*n-1 downto 0);
begin
```

```
begin
-- process for sign extension n-bit to 2*n-bit
zero_vector <= (others => '0');
one_vector <= (others => '1');
process(latency_temp(L-2))
begin
    if signed(latency_temp(L-2)) < 0 then
      temp_x <= one_vector&latency_temp(L-2);
    else
      temp_x <= zero_vector&latency_temp(L-2);
  end if;
end process;
-- process for 2*n-bit accumulate
-- with L-1 stage shift register emulating latency
process(ck)
begin
if ck='1' and ck'event then
  if reset = '1' then
    for i in 0 to L-2 loop
      latency_temp(i) <= (others => '0');
    end loop;
    temp <= (others => '0');
  else
    temp <= temp + temp_x;
    latency_temp <= x&latency_temp(0 to L-3);
  end if;
end if;
end process;
z <= temp;
end beh;
```

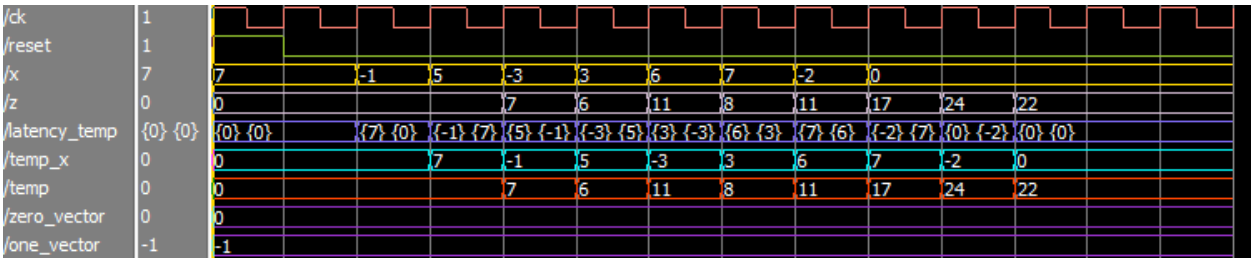

Fig. 6 Accumulator Simulation Wave