Digitalkonstruktion I 2007-08-27

1 a) - c) se kursboken "VHDL konstruktion", L Lindh.

2) Förenkla uttrycket

f <= (a and b) or (c and (w xor z))

så långt det är möjligt.

$f = ab + c(w \oplus z)$

3) Skriv VHDL-kod för en flanktriggad DFF med synkron reset.

```
entity dff is
  port ( d, clk, reset : in bit ;
         q : out bit);
end dff;

architecture rtl of dff is
begin
  process (clk)
  begin
    if (clk'event and clk='1') then
      if reset ='1' then q <= '0';
      else q <= d;
      end if;
    end if;
  end process;
end rtl;
```

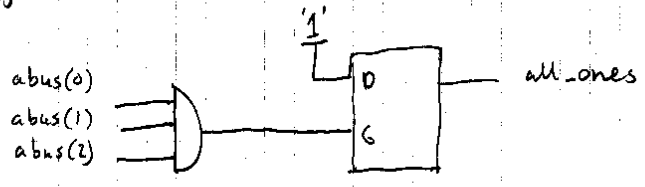
4) Schemaläggning:

tid	a, b, z	drivet	Kommentar
0ns	1, 0, 0	(0, 0ns)(1, 0ns+Δ)	p6 körs pga initiering
0ns Δ	1, 0, 1	(1, 0ns+Δ)	
10ns	1, 1, 1	(1, 10ns) ~~(1, 10ns+Δ)~~ (0, 10ns+Δ)	p6 körs p.g.a att b ändras
10ns Δ	1, 1, 0	(0, 10ns+Δ)	

5.

kod	värde
my_type1'left	16
my_type1'right	2
my_type2'low	2
my_type2'length	15
my_type3'succ(Adam)	Bertil
my_type4'pred(Adam)	ger fel!

6. Syntesresultatet blir



```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sevenseg_decoder is

  port (
    b     : in  std_logic_vector(2 downto 0);
    ssgmt : out std_logic_vector(6 downto 0));


-- ssgmt(0) <=> a
-- ssgmt(1) <=> b
-- ssgmt(2) <=> c
-- ssgmt(3) <=> d
-- ssgmt(4) <=> e
-- ssgmt(5) <=> f
-- ssgmt(6) <=> g

end sevenseg_decoder;

architecture rtl of sevenseg_decoder is

begin  -- rtl

  decode : process (b)
  begin  -- process decode

    case b is
      when "000" => ssgmt <= "1000000";
      when "001" => ssgmt <= "1111001";
      when "010" => ssgmt <= "0100100";
      when "011" => ssgmt <= "0110000";
      when "100" => ssgmt <= "0011001";
      when "101" => ssgmt <= "0010010";
      when "110" => ssgmt <= "0000010";
      when "111" => ssgmt <= "1111000";
      when others => ssgmt <= "1111111";
    end case;
  end process decode;
end rtl;
```

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity max is

  port (
    A, B : in  std_logic_vector(6 downto 0);
    CLK  : in std_logic;
    M : out std_logic_vector(6 downto 0)
    );
end max;

architecture rtl of max is

begin  -- rtl

  compare : process (CLK)
  begin  -- process compare
    if clk'event and clk = '1' then
      if (A > B) then
        M <= A;
      else
        M<= B;
      end if;
    end if;
  end process compare;
end rtl;
```

(9.1)


```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use work.all;

entity memctrl is

  port (
    read_write, ready, clk : in  bit;
    oe, we                 : out bit);
end memctrl;

ARCHITECTURE
  rtl of memctrl IS
  subtype statetype is std_logic_vector(1 downto 0);

  -- state encoding
  constant idle : statetype := "00";
  constant decision : statetype := "01";
  constant write_state : statetype := "10";
  constant read_state : statetype := "11";

  signal present_state , next_state : statetype;

  begin
    state_comb: process (present_state , read_write, ready)
    begin  -- process
      case present_state is
        when idle => oe <= '0'; we <= '0';
                     if ready = '1' then
                       next_state <= decision;
                       else
                         next_state <= idle;
                     end if;
        when decision => oe <= '0'; we <= '0';
                         if read_write = '1' then
                           next_state <= read_state;
                           else
                             next_state <= write_state;
                         end if;
        when read_state => oe <= '1'; we <= '0';
                           if ready = '1' then
                             next_state <= idle;
                             else
                               next_state <= read_state;
                           end if;
        when write_state => oe <= '0'; we <= '1';
                            if ready = '1' then
                              next_state <= idle;
                              else
                                next_state <= write_state;
                            end if;
        when others => null;
      end case;
    end process;
    state_clocked: process(clk)
```

```
      begin
        if (clk'event and clk = '1') then
          present_state <= next_state;
        end if;
      end process;
  end rtl;
```