Lösningsförslag:
Tentamen Digitalkonstruktion I, 3p. 2001-06-01

2.

```
a <= not ((x xnor y) and (w or (not z)))
```

3.

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

entity ndff_a is
  port (
    D    : in  std_logic;
    CLK  : in  std_logic;
    CLR  : in  std_logic;
    Q    : out std_logic);
end ndff_a;

architecture rtl of ndff_a is
begin  -- rtl
  process (D, CLK, CLR)
  begin
    if CLR = '1' then
      Q <= '0';
    elsif (CLK'event and CLK = '0') then
      Q <= D;
    end if;
  end process;
end rtl;
```

4.

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

entity my_parity is
  port (
    a_bus: in  std_logic_vector(3 downto 0);
    x: out std_logic);
end my_parity;

architecture will_work of my_parity is

begin
  process (a_bus)
    variable x_v: std_logic;
  begin
    x_v := ‘0’;
    for i in a_bus’range loop
      x_v := a_bus(i) xor x_v;
    end loop;  -- i
    x <= x_v;
  end process;
end will_work;
```

5.

Svar: z_state

6.

Svar: 10 stycken

7.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
    generic (
    N : integer := 64);

  port (
    A, B : in  std_logic_vector(N-1 downto 0);
    S    : in  std_logic_vector(2 downto 0);
    CIN  : in  std_logic;
    F    : out std_logic_vector(N-1 downto 0);
    COUT : out std_logic);
end ALU;

architecture rtl of ALU is
begin  -- rtl

  sel_func : process (A, B, S, CIN)
  variable F_var: std_logic_vector(N downto 0);
  variable zero_vector : std_logic_vector(N-1 downto 0) :=
(others=>'0');

  begin  -- process sel_func
    case S is
      when "000" => F_var := (others => '0');
     when "001" => F_var := ('0' & B) - ('0' & A) - (zero_vector&"1")
+ (zero_vector&CIN);
     when "010" => F_var := ('0' & A) - ('0' & B) - (zero_vector&"1")
+  (zero_vector&CIN);
      when "011" => F_var := ('0' & A) + ('0' & B) +
(zero_vector&CIN);
      when "100" => F_var := ('0' & A) xor ('0' & B);
      when "101" => F_var := ('0' & A) and ('0' & B);
      when "110" => F_var := ('0' & A) or ('0' & B);
      when others => null;
    end case;

    COUT <= F_var(N);

    F <= F_var(N-1 downto 0);

  end process sel_func;
end rtl;
```

8.

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

entity load_count16 is
  port (
    ld, rst, clk: in std_logic;
    c: in std_logic_vector(15 downto 0);
    q : out std_logic_vector(15 downto 0));
end load_count16;

architecture rtl of load_count16 is
  signal i_count : std_logic_vector(15 downto 0);

  begin  -- rtl
  process (clk, rst)
  begin
    if rst = '0' then                   -- asynchronous reset (active
low)
      i_count <= (others => '0');
    elsif clk'event and clk = '1' then    -- rising clock edge
      if ld = '1' then
        i_count <= c;
      else
        i_count <= i_count +1;
      end if;
    end if;
  end process;

  q <= i_count;
end rtl;
```

9.

```
entity bus_controller is
    port (
    clk, BR1, BR2, BB, LOCK, LOCKE    : in  std_logic;
    BG1, BG2: out std_logic);
end bus_controller;

architecture rtl of bus_controller is
  type state_type is (S0, S1, S2, S3);
  signal current_state, next_state : state_type;
begin  -- rtl
  transition_function: process (BR1, BR2, BB, LOCK, LOCKE, current_state)
  begin  -- process
    case current_state is
      when S0 =>
        if (BB='0' and LOCK='0' and LOCKE='1') then next_state <= S1;
        elsif (BB='0') then next_state <= S0;
        elsif BB='1' then next_state <= S3;
        end if;
      when S1 =>
        if BR1='0' then next_state <= S0;
        else
          next_state <= S1;
        end if;
      when S2 =>
        if (BB='0' and LOCK='0' and LOCKE='1') then next_state <= S3;
        elsif BB='1' then next_state <= S1;
        else
          next_state <= S2;
        end if;
      when S3 =>
        if BR2='0' then next_state <= S2;
        else
          next_state <= S3;
        end if;
      when others => null;
    end case;
  end process;

  output_function: process (current_state)
  begin  -- process
    case current_state is
      when S0 => BG1 <= '0'; BG2 <= '1';
      when S1 => BG1 <= '1'; BG2 <= '0';
      when S2 => BG1 <= '1'; BG2 <= '0';
      when S3 => BG1 <= '0'; BG2 <= '1';
      when others => null;
    end case;
  end process;

  process (clk)
  begin  -- process
    if clk'event and clk = '1' then  -- rising clock edge
      current_state <= next_state;
    end if;
  end process;
end rtl;
```