Hjelpe notat til ELE111

Half adder

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

entity half_adder is
    port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
        carry_out :out std_logic
    );
end entity half_adder;

architecture RTL of half_adder is

begin
    sum <= a xor b;
    carry_out <= a and b;
end architecture RTL;
```

OR port

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

entity or2_port is
    port(
        x : in std_logic;
        y : in std_logic;
        z : out std_logic
    );
end entity or2_port;

architecture RTL of or2_port is

begin
    z <= x or y;
end architecture RTL;
```

Full adder med bruk av half addere

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

entity full_adder_top is
    port(
        a : in std_logic;
        b : in std_logic;
        carry_in : in std_logic;
        sum : out std_logic;
        carry_out: out std_logic
    );
end entity full_adder_top;

architecture struct of full_adder_top is
    component half_adder is
    port(
        a : in std_logic;
        b : in std_logic;
        sum : out std_logic;
        carry_out :out std_logic
    );
    end component half_adder;

    component or2_port is
    port(
        x : in std_logic;
        y : in std_logic;
        z : out std_logic
    );
    end component or2_port;

    signal sum1 : std_logic;
    signal carry1 : std_logic;
    signal carry2 : std_logic;
```

```
begin

    c_half_adder_1 : component half_adder
        port map(
            a         => a,
            b         => b,
            sum       => sum1,
            carry_out => carry1
        );

    c_half_adder_2 : component half_adder
        port map(
            a         => sum1,
            b         => carry_in,
            sum       => sum,
            carry_out => carry2
        );

    c_or2_port : component or2_port
        port map(
            x => carry1,
            y => carry2,
            z => carry_out
```

Mux 8 til 1

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


 ENTITY Mux8to1 IS

           PORT(s :IN std_logic_vector(2 downto 0);
           d0,d1,d2,d3,d4,d5,d6,d7: in std_logic_vector(1 downto 0);
           m: out std_logic_vector(1 downto 0)
           );
     END ;

     Architecture behavior OF Mux8to1 IS
     begin
     Multiplekser:
     WITH s SELECT
                 m <= d0 WHEN "000",
                        d1 WHEN "001",
                        d2 WHEN "010",
                        d2 WHEN "011",
                        d3 WHEN "100",
                        d5 WHEN "101",
                        d6 WHEN "110",
                        d7 WHEN "111",
                        "00" WHEN others;
END behavior;
```

Full adderer

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

entity full_adderer is
    port(
        A : in std_logic;
        B : in std_logic;
        C_in : in std_logic;
        sum : out std_logic;
        C_out : out std_logic
    );
end entity full_adderer;

architecture behavior of full_adderer is

begin
    sum <= (A xor B) xor C_in;
    C_out <= (A and B) or ((A xor B) and C_in);
end architecture behavior;
```

Konvertering av signaler

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

entity konvertering is
    port(
        SW : in std_logic_vector(17 downto 0);
        LEDG : out std_logic_vector(7 downto 0)
    );
end entity konvertering;

architecture RTL of konvertering is
    signal a,b :integer range 0 to 15;
    signal c : integer range 0 to 255;
    signal b_slv : std_logic_vector(3 downto 0);
    signal c_slv : std_logic_vector(7 downto 0);
begin
    a <= to_integer(unsigned(SW(3 downto 0)));
    b_slv <= SW(7 downto 4);
    b <= to_integer(unsigned(b_slv));
    c <= a+b;
    c_slv <= std_logic_vector(to_unsigned(c,8));

    LEDG <= c_slv;
end architecture RTL;
```

Ulike datatyper

```
architecture RTL of datatypar is
    type tabell is array (0 to 4) of std_logic_vector(2 downto 0);

    signal liste : tabell := ("000","001","010","011","100");
    signal c : std_logic_vector(2 downto 0);

    type tilstander is (T1,T2,T3);
    signal state : tilstander;
```

Fire Bit adderer m/ carry

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

entity fire_bit_adderer is
    port(
        A : in std_logic_vector(3 downto 0);
        B : in std_logic_vector(3 downto 0);
        sum : out std_logic_vector(4 downto 0)
    );
end entity fire_bit_adderer;

architecture struct of fire_bit_adderer is

    component full_adderer is
        port(
            A : in std_logic;
            B : in std_logic;
            C_in : in std_logic;
            sum : out std_logic;
            C_out : out std_logic
        );
    end component full_adderer;

    signal carry : std_logic_vector(4 downto 0);
begin
    carry(0) <= '0';
    g_adder : for i in 0 to 3 generate
    begin
        full_adderer_inst : component full_adderer
            port map(
                A     => A(i),
                B     => B(i),
                C_in  => carry(i),
                sum   => sum(i),
                C_out => carry(i+1)
            );
        end generate;
    sum(4) <= carry(4);


end architecture struct;
```

Vipper (Dvippe og Latch)

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

entity vipper is
    port(
        clk : in std_logic;
        rst : in std_logic;
        D : in std_logic;
        en : in std_logic;
        Q : out std_logic;
        Q_lacth : out std_logic
    );
end entity vipper;

architecture RTL of vipper is

begin

    pvippe: process(clk)
    begin
        --if clk'event and clk = '1' then
        if rising_edge(clk) then
            Q <= D;
        end if;
    end process;

    p_Latch : process(D,en)
    begin
        if en = '1' then
            Q_lacth <= D;
        end if;
    end process;

end architecture RTL;
```

Asynkron- og synkron reset

```
p_asynkron_reset : process(clk,rst_n) is
begin
    if rst_n = '0' then
        Q1 <= '0';
    elsif rising_edge(clk) then
        Q1 <= a;
    end if;
end process;

p_synkron_reset: process(clk)
begin
    if rising_edge(clk) then
        if rst_n = '0' then
            Q2 <= '0';
        else
            Q2 <= b;
        end if; -- rst_n
    end if; --clk
end process;
```

ROM

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

entity ROM_hallo is
    port(
        adresse : in std_logic_vector(2 downto 0);
        data_ut : out std_logic_vector(6 downto 0)
    );
end entity ROM_hallo;

architecture RTL of ROM_hallo is
    type ROM_ARRAY is array(0 to 7) of std_logic_vector(6 downto 0);
    constant HALLO_ROM : ROM_ARRAY := ("0001001","0001000","1000111",
        "1000111","1000000","1111111","1111111","1111111");

begin
    data_ut <= HALLO_ROM(to_integer(unsigned(adresse)));
end architecture RTL;
```

Ram 16x8 array

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

entity ram16x8 is
    port(
        clk : in std_logic;
        adresse : in std_logic_vector(3 downto 0);
        data_inn : in std_logic_vector(7 downto 0);
        data_ut : out std_logic_vector(7 downto 0);
        we_n,cs_n,oe_n : in std_logic
    );
end entity ram16x8;

architecture RTL of ram16x8 is
    type ram_array is array(0 to 15) of std_logic_vector(7 downto 0);
    signal memory : ram_array;
    signal adresse_int : integer range 0 to 15;

begin
    adresse_int <= to_integer(unsigned(adresse));

    p_ram: process(clk)
    begin
        if rising_edge(clk) then
            if cs_n = '0' then
                if we_n = '0' then
                    memory(adresse_int) <= data_inn;
                end if;
                if oe_n = '0' then
                    data_ut <= memory(adresse_int);
                end if;
            end if;
        end if;
    end process;

end architecture RTL;
```

N bit adderer

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

entity n_bit_adderer is
    generic( g_bredde : integer := 4);
    port(
        A : in std_logic_vector(g_bredde-1 downto 0);
        B : in std_logic_vector(g_bredde-1 downto 0);
        sum : out std_logic_vector(g_bredde downto 0)
    );
end entity n_bit_adderer;

architecture struct of n_bit_adderer is

    component full_adderer is
        port(
            A : in std_logic;
            B : in std_logic;
            C_in : in std_logic;
            sum : out std_logic;
            C_out : out std_logic
        );
    end component full_adderer;

    signal carry : std_logic_vector(g_bredde downto 0);
begin
    carry(0) <= '0';
    g_adder : for i in 0 to g_bredde-1 generate
    begin
        full_adderer_inst : component full_adderer
            port map(
                A     => A(i),
                B     => B(i),
                C_in  => carry(i),
                sum   => sum(i),
                C_out => carry(i+1)
            );
        end generate;
    sum(g_bredde) <= carry(g_bredde);


end architecture struct;
```

Reset synkroniser

```vhdl
library ieee;
use ieee.std_logic_1164.ALL;

entity Reset_synchronizer IS
Port ( clk: IN std_logic;
             reset_key3 : IN std_logic;
             reset_sync: OUT std_logic);
      End Entity;

ARCHITECTURE RTL OF Reset_synchronizer IS

            signal  dff: std_logic;
BEGIN

sync_reset: Process(clk, reset_key3)
            begin
                  if reset_key3 ='0' then
                        reset_sync <= '0';
                        dff <= '0';
                  elsif rising_edge(clk) then
                        reset_sync <= dff;
                        dff <= '1';
                  end if;
            end process;
END;
```

ROM 7 segment display

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

entity ROM_7_seg is
    port(
        adresse : in  std_logic_vector(3 downto 0);
        HEX     : out std_logic_vector(6 downto 0)
    );
end entity ROM_7_seg;

architecture behavior of ROM_7_seg is
    type ROM_array is array(0 to 15) of std_logic_vector(6 downto 0);
    constant ROM  : ROM_array := (
        "1000000", -- gfedcba -- 0 0x40
        "1111001", -- 1 0x79
        "0100100", -- 2 0x24
        "0110000", -- 3 0x30
        "0011001", -- 4 0x19
        "0010010", -- 5 0x12
        "0000010", -- 6 0x02
        "1111000", -- 7 0x78
        "0000000", -- 8 0x00
        "0010000", -- 9 0x10
        "0001000", -- A 0x08
        "0000011", -- b 0x03
        "1000110", -- C 0x46
        "0100001", -- d 0x21
        "0000110", -- E 0x06
        "0001110" -- F 0x0E
    );

begin
    HEX <= ROM(to_integer(unsigned(adresse)));
end architecture behavior;
```

Teller til 15

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

entity teller0_15 is
    port(
        clk : in std_logic;
        rst_n : in std_logic;
        count : out std_logic_vector(3 downto 0)
    );
end entity teller0_15;

architecture RTL of teller0_15 is
    signal teller : integer range 0 to 15;
begin
    p_tell : process (clk) is
    begin
        if rising_edge(clk) then
            if rst_n = '0' then
                teller <= 0;
            else
                if teller >= 15 then
                    teller <= 0;
                else
                    teller <= teller + 1;
                end if;
            end if;
        end if;
    end process p_tell;

    count <= std_logic_vector(to_unsigned(teller,4));

end architecture RTL;
```

Opp/ned teller til 15

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

entity teller0_15opp_ned is
    port(
        clk : in std_logic;
        rst_n : in std_logic;
        opp : in std_logic;
        count : out std_logic_vector(3 downto 0)
    );
end entity teller0_15opp_ned;

architecture RTL of teller0_15opp_ned is
    signal teller : integer range 0 to 15;
begin
    p_tell : process (clk) is
    begin
        if rising_edge(clk) then
            if rst_n = '0' then
                teller <= 0;
            else
                if opp = '1' then
                    if teller >= 15 then
                        teller <= 0;
                    else
                        teller <= teller + 1;
                    end if;
                else -- opp = '0'
                    if teller <= 0 then
                        teller <= 15;
                    else
                        teller <= teller -1;
                    end if;
                end if;
            end if;
        end if;
    end process p_tell;

    count <= std_logic_vector(to_unsigned(teller,4));

end architecture RTL;
```

Register for 4bit

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

entity register_4 is
    port(
        clk : in std_logic;
        D : in std_logic_vector(3 downto 0);
        Q : out std_logic_vector(3 downto 0)
    );
end entity register_4;

architecture RTL of register_4 is

begin
    p_register: process(clk) is
    begin
        if rising_edge(clk) then
            Q <= D;
        end if;
    end process;

end architecture RTL;
```

Skiftregister

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

entity skiftregister is
    port(
        clk : in std_logic;
        rst : in std_logic;
        SI  : in std_logic;
        SO  : out std_logic
    );
end entity skiftregister;

architecture RTL of skiftregister is
    signal Q : std_logic_vector(3 downto 0);
begin
    p_skiftreg: process(clk) is
    begin
        if rising_edge(clk) then
            if rst = '1' then
                Q <= (others => '0');
            else
                -- Q(0) <= Q(1);
                -- Q(1) <= Q(2);
                -- Q(2) <= Q(3);
                -- Q(3) <= SI;
                Q <= SI & Q(3 downto 1);
            end if;
        end if;
    end process;

    SO <= Q(0);
end architecture RTL;
```

Signal VS variable

```vhdl
architecture RTL of variableEksempel is
    signal A,B,C : integer range 0 to 15 := 1;
    signal E_int : integer range 0 to 31;

begin

    p_signal : process (clk) is
        variable vA,vB,vC :integer range 0 to 15 := 1;
    begin
```

Komparator

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

entity komparator is
    port(
        A, B : in std_logic_vector(3 downto 0);
        A_gt_B, A_lt_B, A_eq_B : out std_logic
    );
end entity komparator;

architecture behav of komparator is

begin
    pkomp: process(A,B)
    begin
        A_eq_B <= '0';
        A_lt_B <= '0';
        A_gt_B <= '0';

        for i in 3 downto 0 loop
            if A(i) = '1' and B(i) = '0' then -- A størst;
                A_gt_B <= '1';
                EXIT;
            elsif A(i) = '0' and B(i) = '1' then -- B størst;
                A_lt_B <= '1';
                EXIT;
            elsif i = 0 then -- A og B like
                A_eq_B <= '1';
            end if;
        end loop;
    end process;
end architecture behav;
```

SRAM

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

entity sram is
    port(
        SW : in std_logic_vector(17 downto 0);
        KEY : in std_logic_vector(3 downto 0);
        LEDR : out std_logic_vector(17 downto 0);
        SRAM_ADDR : out std_logic_vector(19 downto 0);
        SRAM_DQ : inout std_logic_vector(15 downto 0);
        SRAM_WE_N : buffer std_logic;
        SRAM_CE_N : out std_logic;
        SRAM_OE_N : out std_logic;
        SRAM_UB_N : out std_logic;
        SRAM_LB_N : out std_logic;
        HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,HEX6,HEX7 : out std_logic_vector(6 downto 0)
    );
end entity sram;
architecture RTL of sram is

    component ROM_7_seg is
    port(
        adresse : in  std_logic_vector(3 downto 0);
        HEX     : out std_logic_vector(6 downto 0)
    );
    end component ROM_7_seg;

    signal adr : std_logic_vector(4 downto 0);
    signal dataInn : std_logic_vector(7 downto 0);
    signal dataUt : std_logic_vector(7 downto 0);
begin
    SRAM_CE_N <= '0';
    SRAM_OE_N <= '0';
    SRAM_LB_N <= '0';
    SRAM_UB_N <= '0';
    SRAM_WE_N <= SW(17);

    adr <= SW(15 downto 11);
    dataInn <= SW(7 downto 0);

    SRAM_DQ <= "00000000" & dataInn when SRAM_WE_N = '0'
        else (others => 'Z');
    dataUt <= SRAM_DQ(7 downto 0);

    SRAM_ADDR(19 downto 5) <= (others => '0');
    SRAM_ADDR(4 downto 0) <= adr;

    datautLOW: ROM_7_seg port map (adresse => dataUt(3 downto 0), HEX => HEX0);
    datautHIGH: ROM_7_seg port map (adresse => dataUt(7 downto 4), HEX => HEX1);
    datainnLOW: ROM_7_seg port map (adresse => dataInn(3 downto 0), HEX => HEX4);
    datainnHIGH: ROM_7_seg port map (adresse => datainn(7 downto 4), HEX => HEX5);
    adresseLOW: ROM_7_seg port map (adresse => adr(3 downto 0), HEX => HEX6);
    adresseHIGH: ROM_7_seg port map (adresse => "000"&adr(4), HEX => HEX7);
    HEX3 <= ((others => '1'));
    HEX2 <= ((others => '1'));

end architecture RTL;
```

Enable_gen

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

        Entity Enable_gen Is
        Port ( clock_50 : in std_logic;
                 resetn : in std_logic;
                 velg_enable:in std_logic_vector(2 downto 0);
                 Enable: out std_logic);
        End;

        Architecture rtl of Enable_gen is

        type rom is array (0 to 7 ) of integer range 0 to 50000000;
        Constant RomEN: rom:=(50_000_000,12_500_000,500_000,50000,5000,500,50,5);
        Signal teller : integer range 0 to 50000000;

        Begin
        -- velg 001 peker på plass 1, der legges 12_500_000 så gjør enable 4 ganger i
sekundet
        process(clock_50)
        begin
                if rising_edge(clock_50) then
                        if resetn = '0' then
                                teller <= 0;
                        else
                                teller <= teller +1;
                                        if teller = romEN(to_integer(unsigned(velg_enable)))
then
                                        Enable <= '1';
                                        teller <= 0;
                                else
                                        Enable <= '0';
                                end if;
                        end if;
                end if;
        end process;
        End;
```

Fallende flanke deteketor m/ synkronisering

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

entity flankedetektor is
    port(
        clk : in std_logic;
        rst : in std_logic;
        dataInn : in std_logic;
        flankeUt : out std_logic
    );
end entity flankedetektor;

architecture RTL of flankedetektor is
    signal vippeA,vippeB,vippeC :std_logic;
begin

    p_sync_flanke: process(clk)
    begin
        if rising_edge(clk) then
            if rst ='0' then
                vippeA <= '0';
                vippeB <= '0';
                vippeC <= '0';
                flankeUt <= '0';
            else
                vippeA <= dataInn;
                vippeB <= vippeA;
                vippeC <= vippeB;
                flankeUt <= vippeC and (not vippeB); -- negativ flanke
            end if;
        end if;
    end process;

end architecture RTL;
```

For loop

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

entity tell_1 is
    port(
        input : in std_logic_vector(15 downto 0);
        count : out std_logic_vector(3 downto 0)
    );
end entity tell_1;

architecture Behavioral of tell_1 is

begin
    p_tell_1:process(input) is
      variable v_count : unsigned(3 downto 0):= "0000";

    begin
        for i in 0 to 15 loop
            if input(i) ='1' then
                v_count := v_count + 1;

            end if;
        end loop;
        count <= std_logic_vector(v_count);
        v_count:= ((others => '0'));
    end process;



end architecture Behavioral;
```