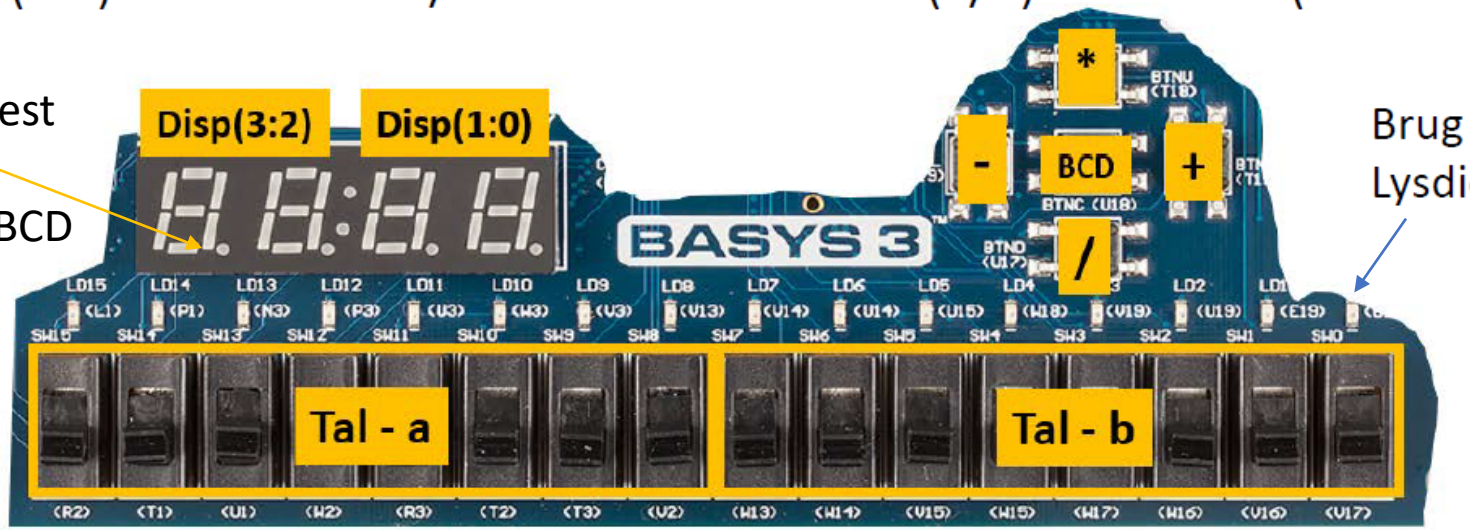
Oplæg til øvelse Calculator

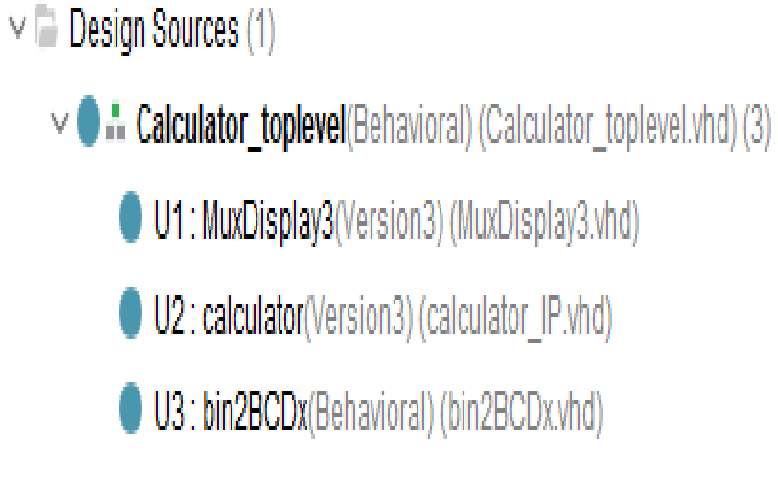
Det anbefales at man nøjes med en binær (hex) version – BCD funktionen kan vente

	Disp(3:2)	Disp(1:0)
Op(4:0) = 0 0 0 0 0 = Binær =>	a	b
Op(4:0) = 0 0 0 0 1 = BCD =>	BCD(a)	BCD(b)
Op(4:0) = 1 0 0 0 0 = + =>	a+b	
Op(4:0) = 1 0 0 0 1 = + BCD =>	BCD (a+b)	
Op(4:0) = 0 1 0 0 0 = - =>	a-b	
Op(4:0) = 0 1 0 0 1 = - BCD =>	BCD (a-b)	
Op(4:0) = 0 0 1 0 0 = * =>	a*b	
Op(4:0) = 0 0 1 0 1 = * BCD =>	BCD (a*b)	
Op(4:0) = 0 0 0 1 0 = / =>	a/b	a MOD b
Op(4:0) = 0 0 0 1 1 = / BCD =>	BCD(a/b)	BCD(a MOD b)

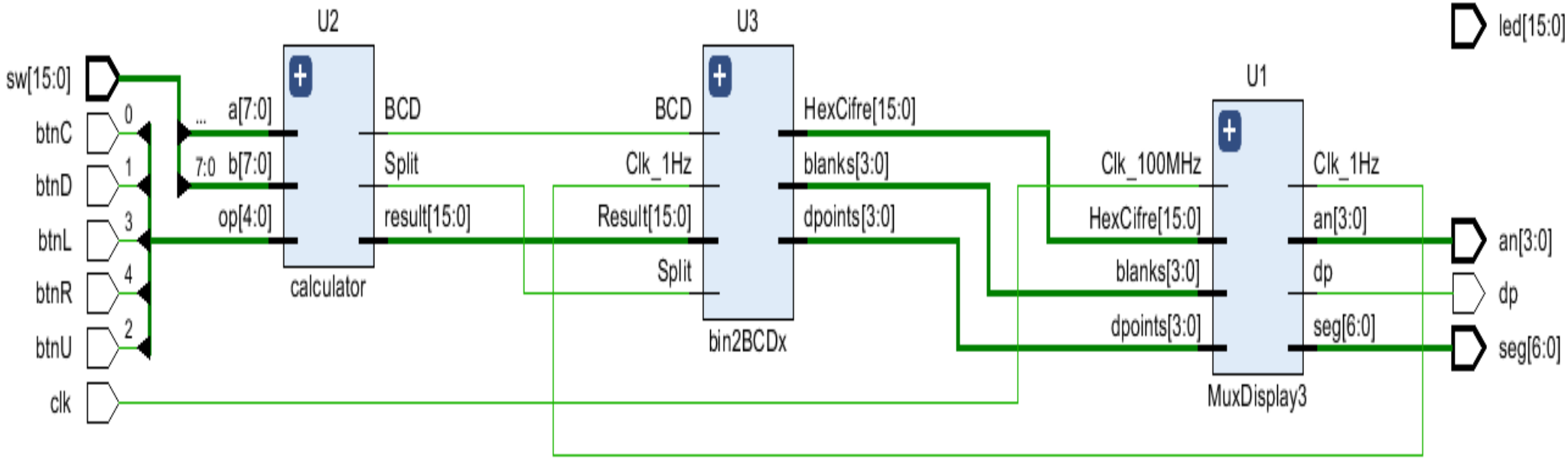
Brug evt. dp til mest betydende cifre i forbindelse med BCD



Brug eventuelt Lysdioder også



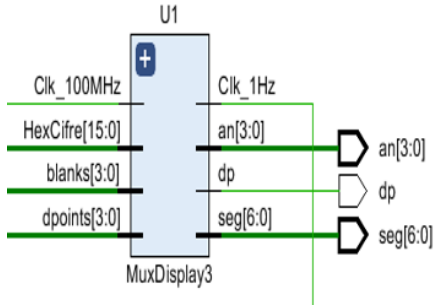

```
entity Calculator_toplevel is
    Port ( clk : in STD_LOGIC;
           sw :  in STD_LOGIC_VECTOR (15 downto 0);
           btnL,btnR,btnC,btnU,btnD : in STD_LOGIC;
           led : out STD_LOGIC_VECTOR (15 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           dp :  out STD_LOGIC;
           an :  out STD_LOGIC_VECTOR (3 downto 0));
end Calculator_toplevel;
```



```
architecture Behavioral of Calculator_toplevel is
 component MuxDisplay3
       port (Clk_100MHz : in std_logic;
             HexCifre   : in std_logic_vector (15 downto 0);
             dpoints    : in std_logic_vector (3 downto 0);
             blanks     : in std_logic_vector (3 downto 0);
             Clk_1kHz   : out std_logic;
             Clk_1Hz    : out std_logic;
             an         : out std_logic_vector (3 downto 0);
             seg        : out std_logic_vector (6 downto 0);
             dp         : out std_logic);
   end component;
   signal HexCifre   : std_logic_vector (15 downto 0);
   signal dpoints    : std_logic_vector (3 downto 0);
   signal blanks     : std_logic_vector (3 downto 0);
   signal Clk_1kHz   : std_logic;
   signal Clk_1Hz    : std_logic;

   component calculator
       port (a      : in std_logic_vector (7 downto 0);
             b      : in std_logic_vector (7 downto 0);
             op     : in std_logic_vector (4 downto 0);
             BCD    : out std_logic;
             Split  : out std_logic;
             result : out std_logic_vector (15 downto 0));
   end component;
   signal a      : std_logic_vector (7 downto 0);
   signal b      : std_logic_vector (7 downto 0);
   signal op     : std_logic_vector (4 downto 0);
   signal BCD    : std_logic;
   signal Split  : std_logic;
   signal result : std_logic_vector (15 downto 0);

   component bin2BCDx
       port (Split    : in std_logic;
             Result   : in std_logic_vector (15 downto 0);
             BCD      : in std_logic;
             Clk_1Hz  : in std_logic;
             HexCifre : out std_logic_vector (15 downto 0);
             dpoints  : out std_logic_vector (3 downto 0);
             blanks   : out std_logic_vector (3 downto 0));
   end component;
```

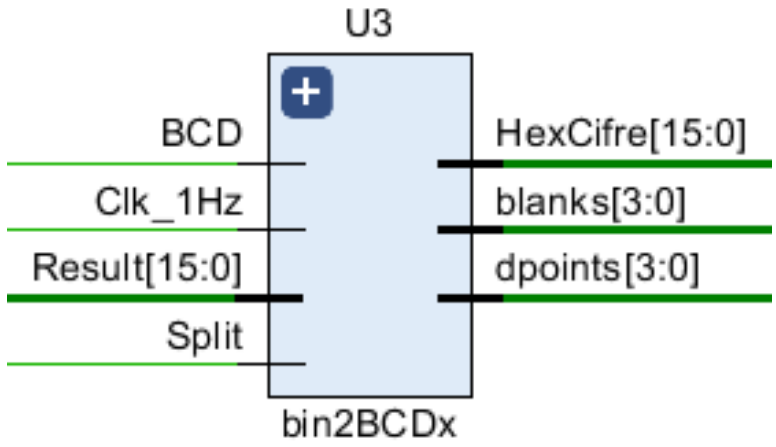
```
begin
    op  <= btnR & btnL & btnU & btnD & btnC;

    U1 : MuxDisplay3
    port map (Clk_100MHz => clk,
              HexCifre   => HexCifre,
              dpoints    => dpoints,
              blanks     => blanks,
              Clk_1kHz   => Clk_1kHz,
              Clk_1Hz    => Clk_1Hz,
              an         => an,
              seg        => seg,
              dp         => dp);

    U2 : calculator
    port map (a      => sw(15 downto 8),
              b      => sw( 7 downto 0),
              op     => op,
              BCD    => BCD,
              Split  => Split,
              result => Result);

    U3 : bin2BCDx
    port map (Split    => Split,
              Result   => Result,
              BCD      => BCD,
              Clk_1Hz  => Clk_1Hz,
              HexCifre => HexCifre,
              dpoints  => dpoints,
              blanks   => blanks);
end Behavioral;
```

```
ENTITY MuxDisplay3 is
      Port (Clk_100MHz: in  STD_LOGIC;
            HexCifre:  in  STD_LOGIC_VECTOR (15 downto 0);
            dpoints:   in  STD_LOGIC_VECTOR (3 downto 0);
            blanks:    in  STD_LOGIC_VECTOR (3 downto 0);
            Clk_1kHz:  out STD_LOGIC;
            Clk_1Hz:   out STD_LOGIC;
            an:        out STD_LOGIC_VECTOR (3 downto 0);
            seg:       out STD_LOGIC_VECTOR (6 downto 0);
            dp:        out STD_LOGIC);
end MuxDisplay3;
```




```
ARCHITECTURE Version3 of MuxDisplay3 is
BEGIN
    Mux_Display3:
    process( Clk_100MHz,HexCifre,blanks,dpoints)
        variable Scale100000: integer range 0 to 100000;
        variable Scale1023:   std_logic_vector( 9 downto 0);
        variable X:           std_logic_vector( 1 downto 0) := "00";
        variable Xi:          integer range 0 to 3;
        variable HexDig:      std_logic_vector( 3 downto 0);

        type ROM_array is array (0 to 15) of std_logic_vector (1 to 7);
        constant Hex27Segm: ROM_array := (
        "1000000","1111001","0100100","0110000", -- 0123
        "0011001","0010010","0000010","1111000", -- 4567
        "0000000","0010000","0001000","0000011", -- 89Ab
        "1000110","0100001","0000110","0001110");-- CdEF
    begin
    ---------------------------------------------- Sequential logic
    if rising_edge( Clk_100MHz) then
        if Scale100000 <100000 then
            Scale100000 := Scale100000+ 1;
            Clk_1kHz   <= '0';
        else
            Scale100000 := 1;
            Clk_1kHz   <= '1';
            Scale1023  := Scale1023+1;
        --  Clk_1Hz    <= Scale1023(9) or Scale1023(8); -- 75% Duty cycle
            Clk_1Hz    <= Scale1023(9);    -- approx. 1 Hz 50% Duty cycle
            X          := X+1;
            Xi         := conv_integer(X);
        end if;
    end if;
    ---------------------------------------------- Combinatorial logic
    -- Note! HexCifre and dpoints must now be added to the sensivity list
    HexDig   := HexCifre( Xi*4+3 downto Xi*4);
    if Blanks(Xi)='1' then
        seg <= "1111111";
        dp  <= '1';
    else
        seg <= Hex27Segm( Conv_integer(HexDig));
        dp  <= not dpoints(Xi);
    end if;
    an       <= "1111";
    an(Xi)   <= '0';
    end process;
end Version3;
```

```
entity bin2BCDx is
    Port ( Split :    in STD_LOGIC;
           Result :   in STD_LOGIC_VECTOR (15 downto 0);
           BCD :      in STD_LOGIC;
           Clk_1Hz :  in STD_LOGIC;
           HexCifre : out STD_LOGIC_VECTOR (15 downto 0);
           dpoints :  out STD_LOGIC_VECTOR (3 downto 0);
           blanks :   out STD_LOGIC_VECTOR (3 downto 0)
           );
end bin2BCDx;

architecture Behavioral of bin2BCDx is
    signal ax:     STD_LOGIC_VECTOR (7 downto 0);
    signal bx:     STD_LOGIC_VECTOR (7 downto 0);
    signal rx:     STD_LOGIC_VECTOR (15 downto 0);

    signal aBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal bBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal rBCD:   STD_LOGIC_VECTOR (19 downto 0);
begin
    ax <= Result( 15 downto 8);
    bx <= Result( 7  downto 0);
    rx <= Result;
```



```
architecture Behavioral of bin2BCDx is
    signal ax:     STD_LOGIC_VECTOR (7 downto 0);
    signal bx:     STD_LOGIC_VECTOR (7 downto 0);
    signal rx:     STD_LOGIC_VECTOR (15 downto 0);

    signal aBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal bBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal rBCD:   STD_LOGIC_VECTOR (19 downto 0);
begin
    ax <= Result( 15 downto 8);
    bx <= Result( 7  downto 0);
    rx <= Result;
```

```
BCD_a: process(ax)
    variable bcd_data : integer;
    variable huns_data: integer;
    variable tens_data: integer;
    variable ones_data: integer;
begin
    bcd_data  := conv_integer(ax);
    ones_data := bcd_data mod 10;
    bcd_data  := bcd_data / 10;
    tens_data := bcd_data mod 10;
    huns_data := bcd_data / 10;

    aBCD(11 downto 8)  <= conv_std_logic_vector(huns_data,4);
    aBCD( 7 downto 4)  <= conv_std_logic_vector(tens_data,4);
    aBCD( 3 downto 0)  <= conv_std_logic_vector(ones_data,4);
end process;
```

```
BCD_b: process(bx)
    variable bcd_data : integer;
    variable huns_data: integer;
    variable tens_data: integer;
    variable ones_data: integer;
begin
    bcd_data  := conv_integer(bx);
    huns_data := bcd_data / 100;
    bcd_data  := bcd_data mod 100;
    tens_data := bcd_data / 10;
    bcd_data  := bcd_data mod 10;
    ones_data := bcd_data;

    bBCD(11 downto 8)  <= conv_std_logic_vector(huns_data,4);
    bBCD( 7 downto 4)  <= conv_std_logic_vector(tens_data,4);
    bBCD( 3 downto 0)  <= conv_std_logic_vector(ones_data,4);
end process;
```

```
BCD_r: process(rx)
    variable bcd_data : integer;
    variable th10_data: integer;
    variable thos_data: integer;
    variable huns_data: integer;
    variable tens_data: integer;
    variable ones_data: integer;
begin
    bcd_data  := conv_integer(rx);
    th10_data := bcd_data / 10000;
    bcd_data  := bcd_data mod 10000;
    thos_data := bcd_data / 1000;
    bcd_data  := bcd_data mod 1000;
    huns_data := bcd_data / 100;
    bcd_data  := bcd_data mod 100;
    tens_data := bcd_data / 10;
    bcd_data  := bcd_data mod 10;
    ones_data := bcd_data;

    rBCD(19 downto 16) <= conv_std_logic_vector(th10_data,4);
    rBCD(15 downto 12) <= conv_std_logic_vector(thos_data,4);
    rBCD(11 downto 8)  <= conv_std_logic_vector(huns_data,4);
    rBCD( 7 downto 4)  <= conv_std_logic_vector(tens_data,4);
    rBCD( 3 downto 0)  <= conv_std_logic_vector(ones_data,4);
end process;
```

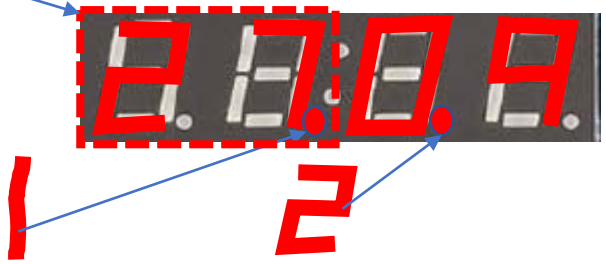
```
architecture Behavioral of bin2BCDx is
    signal ax:     STD_LOGIC_VECTOR (7 downto 0);
    signal bx:     STD_LOGIC_VECTOR (7 downto 0);
    signal rx:     STD_LOGIC_VECTOR (15 downto 0);

    signal aBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal bBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal rBCD:   STD_LOGIC_VECTOR (19 downto 0);
begin
    ax <= Result( 15 downto 8);
    bx <= Result( 7  downto 0);
    rx <= Result;
```

```
Multiplekser: process( Result, BCD, Split)
begin
   HexCifre <= Result;
   dpoints  <= "0000";
   blanks   <= "0000";
   if BCD='1' then
      if Split='1' then
         HexCifre <= aBCD( 7 downto 0) & bBCD( 7 downto 0);
         dpoints  <= aBCD( 9) & aBCD( 8) & bBCD( 9) & bBCD(8);
         blanks   <= Clk_1Hz & Clk_1Hz & "00";
      else
         HexCifre <= rBCD( 15 downto 0);
         dpoints  <= rBCD( 19 downto 16);
         if   rBCD(19 downto 4)="0000000000000000" then
            blanks   <= "1110";
         elsif rBCD(19 downto 8)="000000000000" then
            blanks   <= "1100";
         elsif rBCD(19 downto 12)="00000000" then
            blanks   <= "1000";
         end if;
      end if;
   end if;
end process;
```

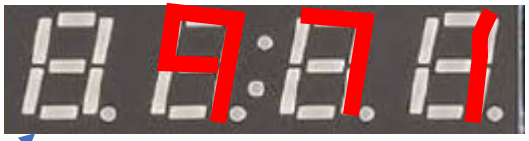
```
architecture Behavioral of bin2BCDx is
    signal ax:     STD_LOGIC_VECTOR (7 downto 0);
    signal bx:     STD_LOGIC_VECTOR (7 downto 0);
    signal rx:     STD_LOGIC_VECTOR (15 downto 0);

    signal aBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal bBCD:   STD_LOGIC_VECTOR (11 downto 0);
    signal rBCD:   STD_LOGIC_VECTOR (19 downto 0);
begin
    ax <= Result( 15 downto 8);
    bx <= Result( 7  downto 0);
    rx <= Result;
```

blink

aBCD=127
bBCD=209
split='1'

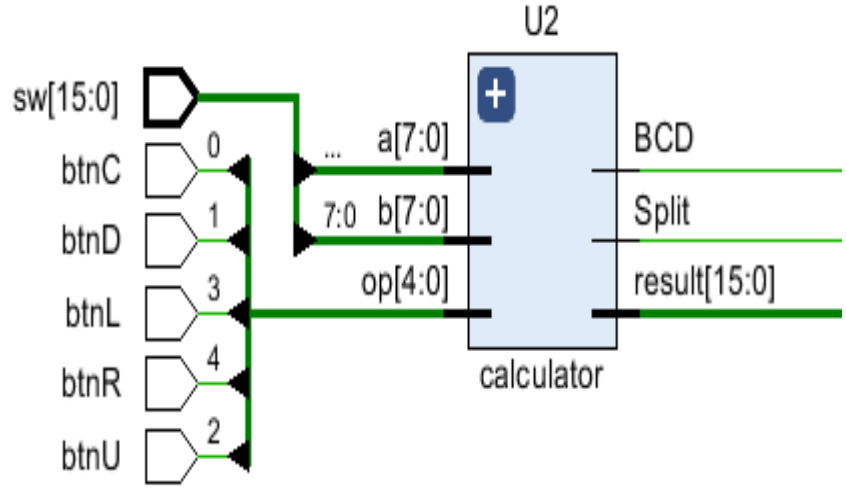


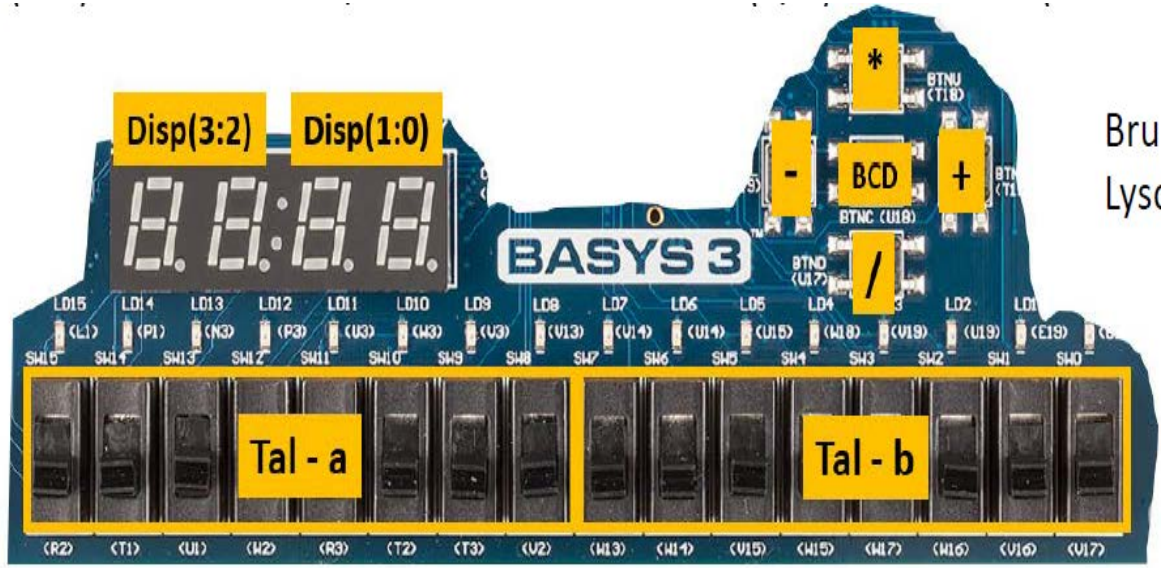
rBCD=971
split='0'



blank

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
--use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY calculator is
    Port ( a  :     in  STD_LOGIC_VECTOR (7 downto 0);
           b  :     in  STD_LOGIC_VECTOR (7 downto 0);
           op :     in  STD_LOGIC_VECTOR (4 downto 0); -->  + - * / B
           BCD :    out STD_LOGIC;
           Split :  out STD_LOGIC;
           result : out STD_LOGIC_VECTOR (15 downto 0));
end calculator;

architecture Version3 of calculator is
      signal ax,bx:     std_logic_vector(7 downto 0);
begin
    BCD <= op(0);

    ALU: process(a,b,op)
        variable ai, bi, ri: integer := 0;
    begin
        ai := conv_integer(a);
        bi := conv_integer(b);
        Split <= '0';
        case op(4 downto 1) is
          when "1000" =>
               ri := ai + bi;
               result <= conv_std_logic_vector(ri,16);
          when "0100" =>
```





Løsning på næste side

```
architecture Version3 of calculator is
      signal ax,bx:     std_logic_vector(7 downto 0);
begin
    BCD <= op(0);

    ALU: process(a,b,op)
        variable ai, bi, ri: integer := 0;
    begin
        ai := conv_integer(a);
        bi := conv_integer(b);
        Split <= '0';
        case op(4 downto 1) is
          when "1000" =>
               ri := ai + bi;
               result <= conv_std_logic_vector(ri,16);
          when "0100" =>
                ri := ai - bi;
                result <= conv_std_logic_vector(ri,16);
          when "0010" =>
               ri := ai * bi;
               result <= conv_std_logic_vector(ri,16);
          when "0001" =>
                ax <= conv_std_logic_vector( ai/bi,8);
                bx <= conv_std_logic_vector( ai rem bi,8);
                result <= ax & bx;
                split <= '1';
          when others =>
                result <= a & b;
                split  <= '1';
        end case;
    end process;
end Version3;
```