DSD # 5 - BEHAVIORAL-STYLE COMBINATORIAL DESIGNS IN VHDL

Gruppe # 17

Nicolaj Zefting 201204718

Robert Gudbjerg 202010534

30. November 2022

Contents

1 Øvelse 5

1.1 Binary to 7-Segment Decoder Using "CASE"

1.1.1 Introduktion

Vi vil her implementere en binær til syv segments dekoder ved brug af case statements. Den skal kunne vise hexadecimale tal, det vil sige fra 0 op til F.

1.1.2 Design og Implementering

vi har implementeret det ovenstående på følgende måde;

```
process(bin) is
BEGIN

case bin is
    when "0000" =>
       seg <= "1000000";--0
    when "0001"  =>
       seg <= "1111001";--1
    when "0010" =>
       seg <= "0100100";--2
    when "0011" =>
       seg <="0110000" ;--3
    when "0100" =>
       seg <= "0011001" ;--4
    when "0101" =>
       seg <= "0010010" ;--5
    when "0110" =>
       seg <= "0000010" ;--6
    when "0111" =>
       seg <= "1111000" ;--7
    when "1000" =>
       seg <= "0000000" ;--8
    when "1001" =>
       seg <= "0011000" ;--9
    when "1010" =>
       seg <= "0001000" ;--A
    when "1011" =>
       seg <= "0000011" ;--b
    when "1100" =>
       seg <= "1000110" ;--C
    when "1101" =>
       seg <= "0100001" ;--d
    when "1110" =>
       seg <= "0000110" ;--E
    when "1111" =>
       seg <= "0001110" ;--F
    when others =>
       seg <= "0111110" ;--U


end case;
end process;
```

Figure 1.1: implementering af bin2hex

Vi har håndteret latches ved en "when others". Vi har også implementeret en tester;

```
ENTITY bin2hex_tester IS
   PORT
   (
      -- Input ports
      SW     : IN STD_LOGIC_VECTOR(3    DOWNTO 0);

      -- Output ports
      HEX0   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0)

   );
END bin2hex_tester;

ARCHITECTURE bin2hex_tester_impl OF bin2hex_tester IS
BEGIN

   uut : ENTITY work.bin2hex
   PORT MAP
      (
         bin => SW(3 downto 0),
         seg => HEX0(6 downto 0)

      );


END bin2hex_tester_impl;
```

Figure 1.2: implementering af bin2hex tester

1.1.3 Resultater

Vi har i opgaven dannet et RTL-view

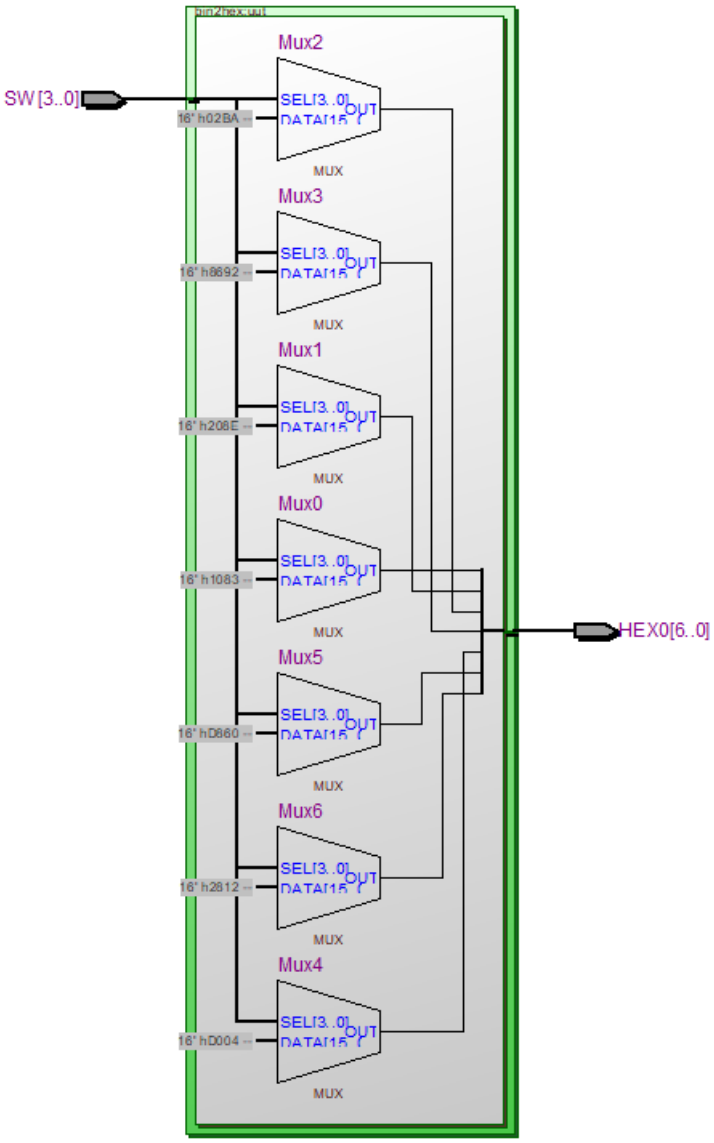

Figure 1.3: RTL fra opgave 4

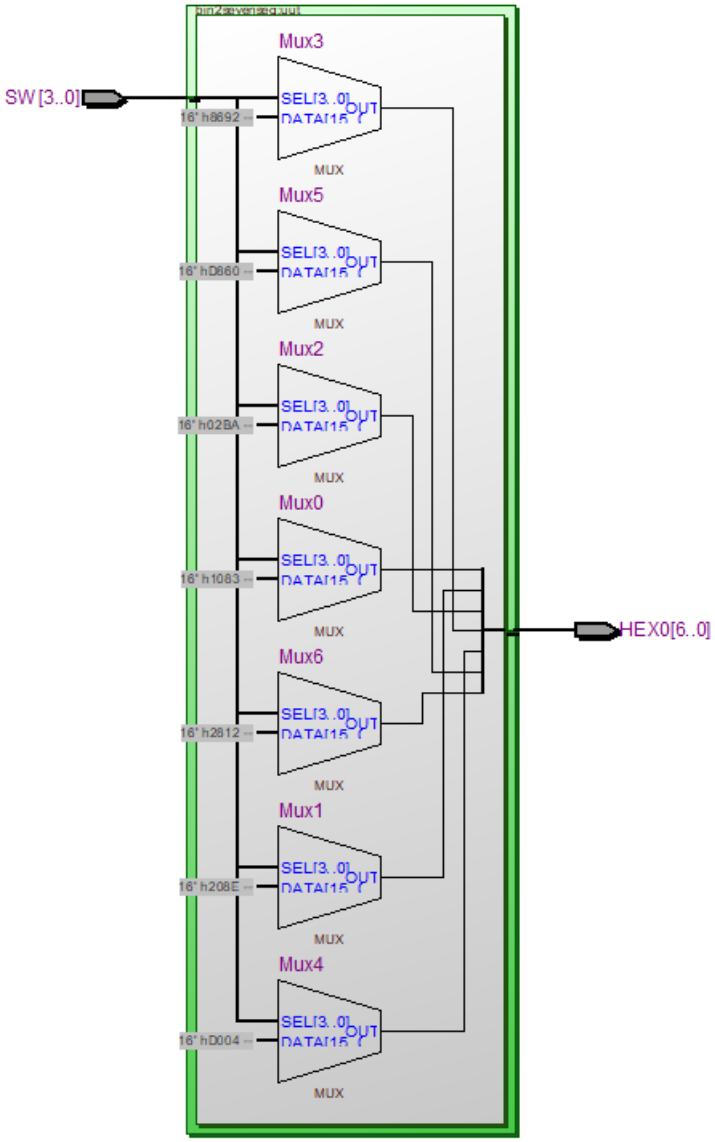

Figure 1.4: RTL fra opgave 4

Vi ser, at de to implimenteringer er ens. Vi har taget højde for latches med en with others.

1.1.4 Test på DE2-Board

Vi har testet på samme måde som i øvelse 4. Vi får de samme resultater på tværs af de to opgaver.

1.1.5 Konklusion

Vi har lavet en binær til syv segment display. Den er opsat i henhold til "ibd Multi_counter_tester"

1.2 Guess game

1.2.1 Introduktion

Vi skal lave et system hvor man skal gætte et hemmeligt nummer. alt efter om man gætter det hemmelige nummer skal "Hi", "Lo" or "–" outputtes.

1.2.2 Design og Implementering

Vi har implementeret de forskellige dele på IBD'et:

```
entity mux1 is
port (
          show : in std_logic;
          secret_value : in std_logic_vector(7 downto 0);
          input : in std_logic_vector(7 downto 0);
          bin: out std_logic_vector(7 downto 0)

);
end;


architecture mux1_impl OF mux1 is
BEGIN
   muxer : PROCESS(show, secret_value, input)
   BEGIN

      -- show = 1 fwd input -- show = 0 fwd secret_value
      if show = '0' then
         bin <= secret_value;
      else
         bin <= input;
      END if;
   END PROCESS muxer;
END mux1_impl;
```

Figure 1.5: Mux1

```
entity mux2 is
port (
          compare : in std_logic_vector(1 downto 0);
          input : in std_logic_vector(13 downto 0);
          hex: out std_logic_vector(13 downto 0)

);
end;

architecture mux2_impl OF mux2 is
BEGIN
   muxer : PROCESS(compare, input)
   BEGIN

if compare = "11" then
   hex <= input;
elsif compare = "10" then
      hex <= "00010011101111";
   elsif compare = "01" then
      hex <= "10001110100011";
   elsif compare = "00" then
      hex <= "01111110111111";
   else
      hex <= input;
end if;


   END PROCESS muxer;
END mux2_impl;
```

Figure 1.6: Mux2

```
entity mylatch is
port(
input : in std_logic_vector(7 downto 0);
set : in std_logic; -- Set predefined value

preDef : out std_logic_vector(7 downto 0)
);
end;

architecture mylatch_impl OF mylatch is
-- signal preDef : std_logic_vector(7 downto 0);
BEGIN


preDef <=
   input when set = '0';



END mylatch impl;
```

Figure 1.7: mylatch

```
architecture bin2hex_impl OF bin2hex is
BEGIN
process(bin) is
BEGIN

case bin is
   when "0000" =>
      seg <= "1000000";--0
   when "0001"  =>
      seg <= "1111001";--1
   when "0010" =>
      seg <= "0100100";--2
   when "0011" =>
      seg <="0110000" ;--3
   when "0100" =>
      seg <= "0011001" ;--4
   when "0101" =>
      seg <= "0010010" ;--5
   when "0110" =>
      seg <= "0000010" ;--6
   when "0111" =>
      seg <= "1111000" ;--7
   when "1000" =>
      seg <= "0000000" ;--8
   when "1001" =>
      seg <= "0011000" ;--9
   when "1010" =>
      seg <= "0001000" ;--A
   when "1011" =>
      seg <= "0000011" ;--b
   when "1100" =>
      seg <= "1000110" ;--C
   when "1101" =>
      seg <= "0100001" ;--d
   when "1110" =>
      seg <= "0000110" ;--E
   when "1111" =>
      seg <= "0001110" ;--F
   when others =>
      seg <= "0111110" ;--U


end case;
end process;
```

Figure 1.8: bin2hex

De enkelte dele samler vi i en guess game implementering

```
entity guess_game is
port(
    input : in std_logic_vector(7 downto 0);
    set : in std_logic; -- Set predefined value
    show : in std_logic; -- Show predefined value
    try : in std_logic; -- Evaluate guess
    hex1 : out std_logic_vector(6 downto 0); -- 7-seg ones
    hex10: out std_logic_vector(6 downto 0) -- 7-seg tens
);
end;
```

Figure 1.9: Guess game entity

vores arkitektur er som følgende: Vi bruger følgende signaler

```
architecture guess_game_impl OF guess_game is

    signal secret_value  : std_logic_vector(7 downto 0);
    signal mux2display   : std_logic_vector(13 downto 0);
    signal compareSignal : std_logic_vector(1 downto 0);
    signal mux1display   : std_logic_vector(13 downto 0);
    signal mux12bin      : std_logic_vector(13 downto 0);
    signal bin2mux       : std_logic_vector(13 downto 0);

BEGIN
```

Figure 1.10: Guess game signaler

```
BEGIN
mylatch: ENTITY work.mylatch
PORT MAP
(
   set => set,
   input => input,
   preDef => secret_value
);
compare_logic: ENTITY work.compare_logic
PORT MAP
(
   input => input,
   mylatch => secret_value,
   try => try,
   result => compareSignal
);
mux1: ENTITY work.mux1
PORT MAP
(
   bin(3 downto 0) => mux1display(3 downto 0),
   bin(7 downto 4) => mux1display(7 downto 4),
   show => show,
   input => input,
   secret_value => secret_value
);

mux2: ENTITY work.mux2
PORT MAP
(
   input => bin2mux,
   hex => mux2display,
   compare => compareSignal
);

hex10<= mux2display(13 downto 7);
hex1<= mux2display(6 downto 0);

bin2hex1: ENTITY work.bin2hex
PORT MAP
(
   bin => mux1display(3 downto 0),
   seg(6 downto 0) => bin2mux(6 downto 0)
);
```

Figure 1.11: implementering komponenter

```
bin2hex2: ENTITY work.bin2hex
PORT MAP
(
    bin => mux1display(7 downto 4),
    seg(6 downto 0) => bin2mux(13 downto 7)
);


mux2: ENTITY work.mux2
PORT MAP
(
    input => bin2mux,
    hex => mux2display,
    compare => compareSignal
);

hex10<= mux2display(13 downto 7);
hex1<= mux2display(6 downto 0);
```

Figure 1.12: implementering komponenter

Baseret på ibd'et har vi implementeret en tester

```
ENTITY guess_game_tester IS
    PORT
    (
        -- Input ports
        SW     : IN STD_LOGIC_VECTOR(7  DOWNTO 0); -- input
        KEY    : IN STD_LOGIC_VECTOR(4 DOWNTO 0); -- set

        -- Output ports
        HEX0   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);-- 1'ere
        HEX1   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0); -- 10'ere
        LEDG   : OUT STD_LOGIC_VECTOR(7 downto 0)
    );
END guess_game_tester;

ARCHITECTURE guess_game_tester_impl OF guess_game_tester IS
BEGIN
LEDG(0) <= KEY(0);
LEDG(1) <= KEY(1);
LEDG(2) <= KEY(2);
    uut : ENTITY work.guess_game
    PORT MAP
        (
            input => SW(7 downto 0),
            set   => KEY(0),
            show  => KEY(1),
            try   => KEY(2),

            hex1  => HEX0(6 downto 0),
            hex10 => HEX1(6 downto 0)

        );


END guess_game_tester_impl;
```

Figure 1.13: implementering af tester

1.2.3 Resultater

for at give et overblik over systemet har vi lavet et technology map.

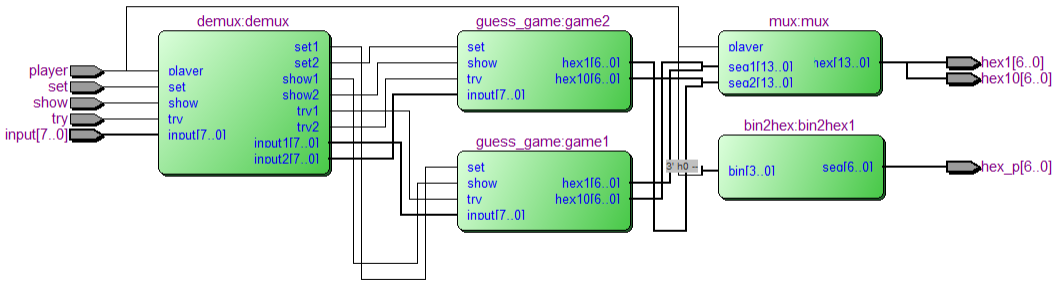

Figure 1.14: technology map

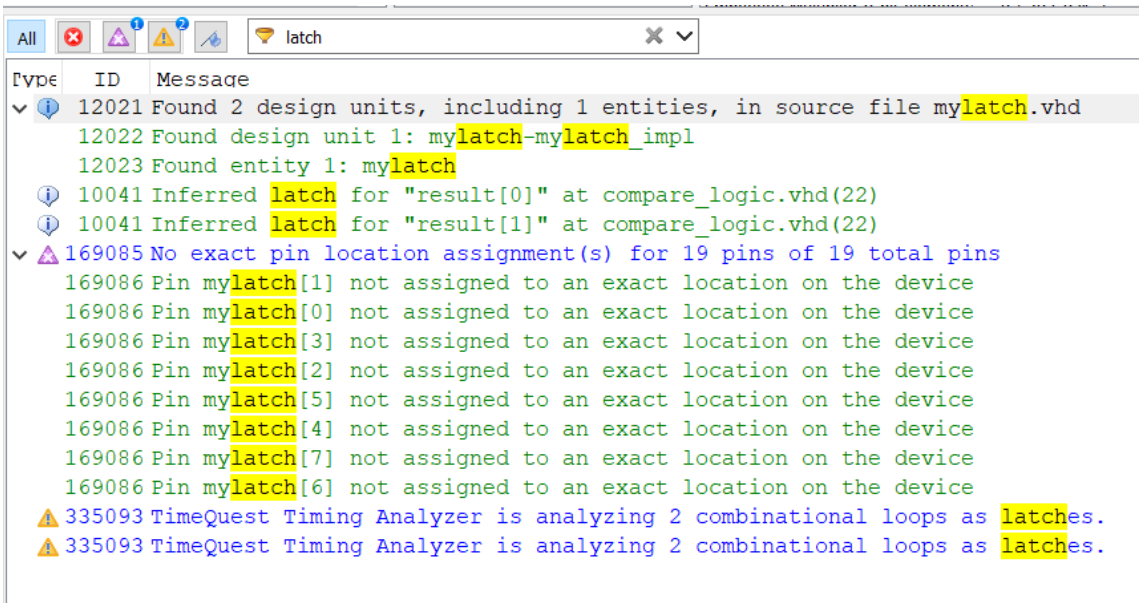

Figure 1.15: Compinerings rapport - se latches

1.2.4 Konklusion

Vi ser i vores resultater, at der opstår latches. Dette giver god mening, fordi vi netop ønsker en latch, som kan huske status på spillet.

1.3 Two Player Guess Game

1.3.1 Introduktion

Formålet med opgaven er, at er at lave et system, hvor to spillere kan spille et spil mod hinanden, hvor målet er at gætte det rette svar.

1.3.2 Design og Implementering

Vi har i henhold til IBD implementeret en demux

```
entity demux is
port (
          show : in std_logic;
          set: in std_logic;
          try: in std_logic;
          input : in std_logic_vector(7 downto 0);
          player : in std_logic;

          show1 : out std_logic;
          set1: out std_logic;
          try1: out std_logic;
          input1 : out std_logic_vector(7 downto 0);

          show2 : out std_logic;
          set2: out std_logic;
          try2: out std_logic;
          input2 : out std_logic_vector(7 downto 0)
);
end;
architecture demux_impl OF demux is
BEGIN
    demuxer : PROCESS(show, set, input,try)
    BEGIN

       -- show = 1 fwd input -- show = 0 fwd secret_value
       if player = '1' then
          show1 <= show;
          set1 <= set;
          try1 <= try;
          input1 <= input;
       else
          show2 <= show;
          set2 <= set;
          try2 <= try;
          input2 <= input;
       END if;
    END PROCESS demuxer;
END demux_impl;
```

Figure 1.16: Demux

```
entity mux is
port (
          player : in std_logic;
          seg1 : in std_logic_vector(13 downto 0);
          seg2 : in std_logic_vector(13 downto 0);
          hex: out std_logic_vector(13 downto 0)

);
end;


architecture mux_impl OF mux is
BEGIN
   muxer : PROCESS(player,seg1,seg2)
   BEGIN

if player = '1' then
   hex <= seg1;
else
   hex <= seg2;
end if;

   END PROCESS muxer;
END mux_impl;
```

Figure 1.17: mux til two player game

Vi har implementeret guess game

```
entity two_player_guess_game is
port(
   input : in std_logic_vector(7 downto 0);
   set : in std_logic; -- Set predefined value
   show : in std_logic; -- Show predefined value
   try : in std_logic; -- Evaluate guess
   player : in std_logic;

   hex1 : out std_logic_vector(6 downto 0); -- 7-seg ones
   hex10: out std_logic_vector(6 downto 0); -- 7-seg tens
   hex_p: out std_logic_vector(6 downto 0) -- player hex
);
end;

architecture two_player_guess_game_impl of two_player_guess_game is
   signal mux2display   : std_logic_vector(13 downto 0);
   signal bin2player    : std_logic_vector(6 downto 0);

   signal p1show  : std_logic;
   signal p1input : std_logic_vector(7 downto 0);
   signal p1set   : std_logic;
   signal p1try   : std_logic;
   signal p2show  : std_logic;
   signal p2input : std_logic_vector(7 downto 0);
   signal p2set   : std_logic;
   signal p2try   : std_logic;

   signal p1seg2mux  : std_logic_vector(13 downto 0);
   signal p2seg2mux  : std_logic_vector(13 downto 0);


   signal player2bin: std_logic_vector(2 downto 0):="000";
```

Figure 1.18: Guess Game

```
 demux: ENTITY work.demux
PORT MAP
(
    player => player,
    show => show,
    input => input,
    set => set,
    try => try,

    show1 => p1show,
    set1 => p1set,
    try1 => p1try,
    input1 => p1input,

    show2 => p2show,
    set2 => p2set,
    try2 => p2try,
    input2 => p2input
);

game1: ENTITY work.guess_game
PORT MAP
(
    show => p1show,
    set => p1set,
    try => p1try,
    input => p1input,
    hex1 => p1seg2mux(6 downto 0),
    hex10 => p1seg2mux(13 downto 7)
);
```

```
game2: ENTITY work.guess_game
PORT MAP
(
    show => p2show,
    set => p2set,
    try => p2try,
    input => p2input,
    hex1 => p2seg2mux(6 downto 0),
    hex10 => p2seg2mux(13 downto 7)
);

bin2hex1: ENTITY work.bin2hex
PORT MAP
(
    bin => player2bin&player,
    seg(6 downto 0) => hex_p
);

mux: ENTITY work.mux
PORT MAP
(
    seg1 => p1seg2mux,
    seg2 => p2seg2mux,
    hex => mux2display,
    player => player

);

hex10<= mux2display(13 downto 7);
hex1<= mux2display(6 downto 0);
```

1.3.3 Diskussion

Vi har valg, at bruge if/else i implementeringen af demux. Der er flere muligheder specificeret i opgave beskrivelsen, men vi har valgt denne, fordi vi gen fandt vi det lettest, at kode på denne måde. I lighed med den tidligere opgave opstod der latches. Det giver igen god mening.

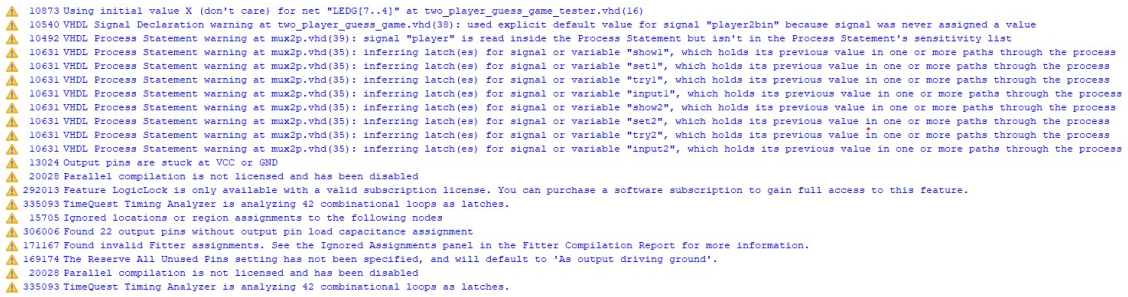

Figure 1.21: latches i two player game

1.3.4 Konklusion

Vi har lavet et system, hvor to spillere kan spille et gættespil mod hinanden. Der er flere latches i denne opgave end der er i enkelt spiller versionen, hvilket giver mening, da systemet er mere konplekst.

2 Øvelse 6

I denne øvelse vil vi lave et VHDL design med ur funktionallitet. Vi skal bruge en 50MHz Oscillator til at tælle tiden et sekundt ad gangen.

2.1 Counter - One Digit

2.1.1 Introduktion

Vi starter øvelsen med at lave et design kan outputte en værdi til gennem en bin2sevenseg ti et hex display. Ved tryk på en key forøges værdien på hex displayet. Vi skal også via en switch kunne variere hvad max væredien som vi skal kunne tælle til er. Switch input ”00” skal kunne tælle mellem 0 og 9, ”01” mellem 0 og 5, og ”10” eller ”11” skal tælle mellem 0 og 2.

2.1.2 Design og Implementering

```
ENTITY multi_counter IS
    GENERIC (
        MIN_COUNT : NATURAL := 0; -- min og max count for tÃ¦ller
        MAX_COUNT : NATURAL := 10
    );
    PORT (
        -- Input ports
        clk : IN STD_LOGIC;
        mode : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        reset : IN STD_LOGIC;
        clken : IN STD_LOGIC;
        -- Output ports
        count : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        cout : OUT STD_LOGIC;
    );
```

Figure 2.1: multi-counter entity baseret på IBD

```
ARCHITECTURE multi_counter_impl OF multi_counter IS
BEGIN

    counter_proc :
    PROCESS (clk, reset) --process reagerer bÃ¥de clk og reset
    -- bruger "variable" for Ã¸jeblikkelig opdatering af counter variable
        VARIABLE cnt : INTEGER RANGE MIN_COUNT TO MAX_COUNT;
        -- MAX_COUNT betyder IKKE at counteren af sig selv ikke tÃ¦ller hÃ¸jere end til MAX_COUNT
        VARIABLE cntMax : INTEGER;
    BEGIN
        IF reset = '0' THEN-- asynkron reset, ikke afhÃ¦ngig af clk
            -- Reset the counter to 0
            cnt := 0;
        ELSIF (rising_edge(clk)) THEN
            cout <= '0';
            IF clken = '1' THEN
                CASE mode IS
                    WHEN "00" =>
                        cntMax := 9;
                    WHEN "01" =>
                        cntMax := 5;
                    WHEN OTHERS =>
                        cntMax := 2;
                END CASE;

                IF cnt < cntMax THEN
                    cnt := cnt + 1;
                    cout <= '0';
                ELSE
                    cnt := 0;
                    cout <= '1';
                END IF;
            END IF;
        END IF;
        -- Output the current count
        count <= STD_LOGIC_VECTOR(to_unsigned(cnt, count'length));
    END PROCESS;

END multi_counter_impl;
```

Figure 2.2: multi-counter architecture baseret på IBD

```
entity bin2sevenseg is
port (
   bin : in std_logic_vector(3 downto 0);
   seg: out std_logic_vector(6 downto 0)
);
end;
```

Figure 2.3: bin2sevenseg entity baseret på IBD

```
architecture bin2sevenseg_impl OF bin2sevenseg IS
BEGIN
process(bin) is
BEGIN
case bin is
   when "0000" =>
      seg <= "1000000";--0
   when "0001"  =>
      seg <= "1111001";--1
   when "0010" =>
      seg <= "0100100";--2
   when "0011" =>
      seg <="0110000" ;--3
   when "0100" =>
      seg <= "0011001" ;--4
   when "0101" =>
      seg <= "0010010" ;--5
   when "0110" =>
      seg <= "0000010" ;--6
   when "0111" =>
      seg <= "1111000" ;--7
   when "1000" =>
      seg <= "0000000" ;--8
   when "1001" =>
      seg <= "0011000" ;--9
   when "1010" =>
      seg <= "0001000" ;--A
   when "1011" =>
      seg <= "0000011" ;--b
   when "1100" =>
      seg <= "1000110" ;--C
   when "1101" =>
      seg <= "0100001" ;--d
   when "1110" =>
      seg <= "0000110" ;--E
   when "1111" =>
      seg <= "0001110" ;--F
   when others =>
      seg <= "0111110" ;--U
end case;
end process;

END bin2sevenseg impl;
```

Figure 2.4: bin2Sevenseg architecture baseret på IBD

2.1.3 Resultater

Vi har opstillet en funktionel simulering

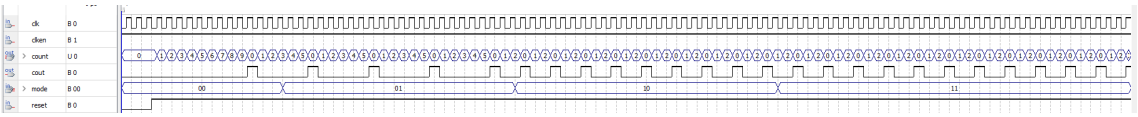

Figure 2.5: funktionel simulering af systemet

2.1.4 Test på DE2-Board

implementering af tester

```
ARCHITECTURE multi_counter_tester_impl OF multi_counter_tester IS
   signal counter2bin : std_logic_vector(3 downto 0);
BEGIN
LEDG(0) <= KEY(0);
LEDG(1) <= KEY(1);
LEDG(2) <= KEY(2);
   uut : ENTITY work.bin2sevenseg
   PORT MAP
      (
         bin => counter2bin,
         seg => HEX0(6 downto 0)

      );

   mc : ENTITY work.multi_counter
   PORT MAP
      (
         clk => KEY(0),
         mode => SW,
         reset => KEY(3),
         clken => '1',

         count=> counter2bin,
         cout =>LEDR(0)
      );


END multi_counter_tester_impl;
```

Figure 2.6: implementering af tester

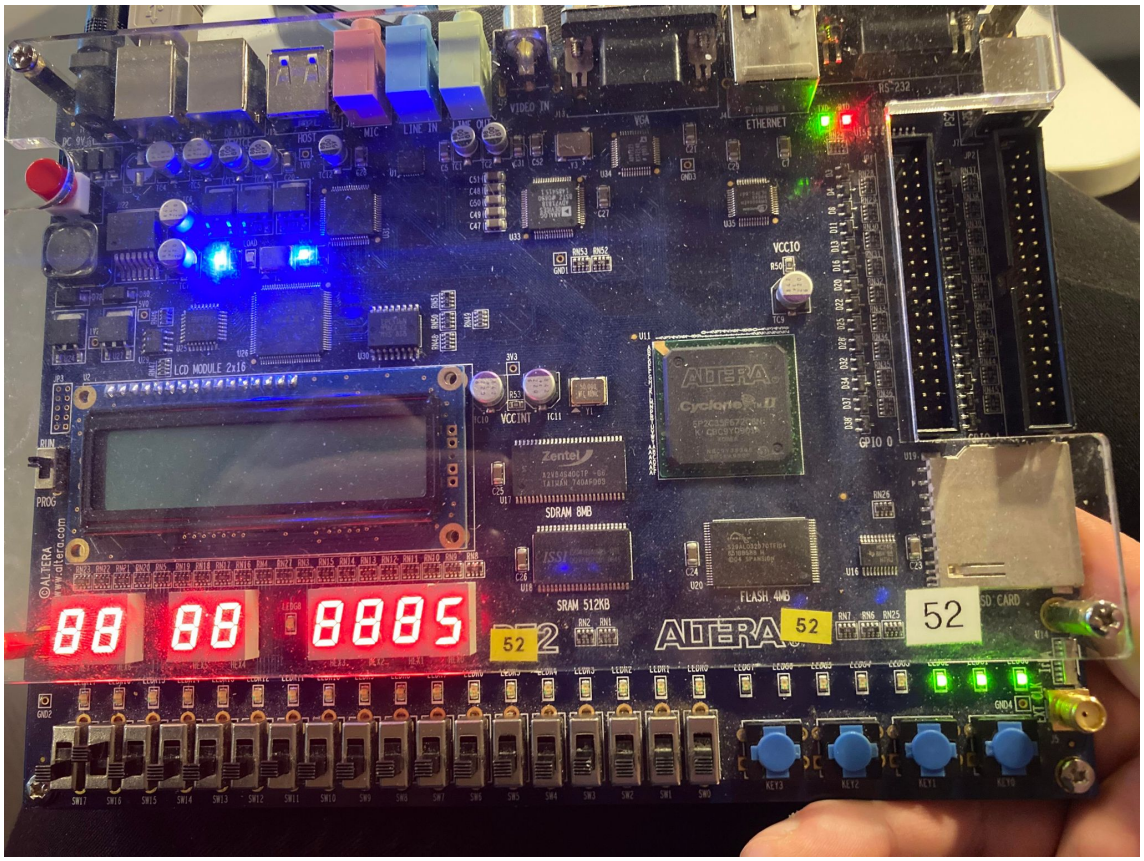

Figure 2.7: Test på DE2 Board

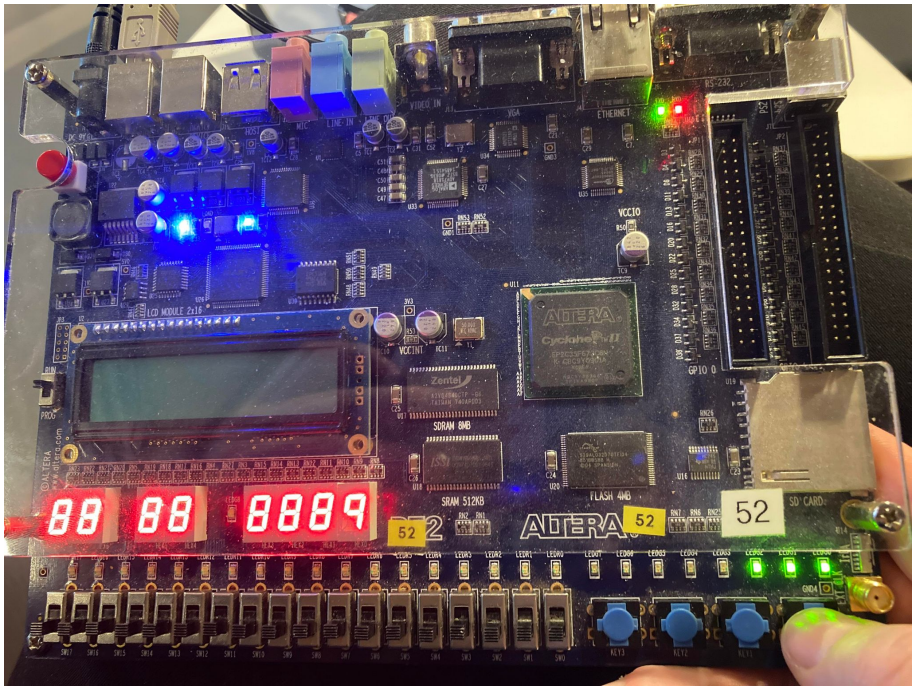

Figure 2.8: Test på DE2 Board. Der er her incrementeret

2.1.5 Konklusion

Vi har i denne opgave lavet en et segments tæller. Den fungerer som beskrevet i kravene.

2.2 Clock - One Digit

2.2.1 Introduktion

Vi vil her implementere optælning over tid via 50MHz krystal oscillator. Det vil sige at viderbygger på vores design fra Counter - One Digit med et clock_gen

2.2.2 Design og Implementering

I henhold til IBD'et har vi opstillet en række komponenter. Implementering af clock_gen

```
ENTITY multi_counter IS
   GENERIC (
      MIN_COUNT : NATURAL := 0; -- min og max count for tÃ¦ller
      MAX_COUNT : NATURAL := 10
   );
   PORT (
      -- Input ports
      clk : IN STD_LOGIC;
      mode : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      reset : IN STD_LOGIC;
      clken : IN STD_LOGIC;
      -- Output ports
      count : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
      cout : OUT STD_LOGIC;
   );
```

Figure 2.9: implementering af multi-counter

```
ARCHITECTURE multi_counter_impl OF multi_counter IS
BEGIN

    counter_proc :
    PROCESS (clk, reset) --process reagerer bÃ¥de clk og reset
    -- bruger "variable" for Ã¸jeblikkelig opdatering af counter variable
       VARIABLE cnt : INTEGER RANGE MIN_COUNT TO MAX_COUNT;
       -- MAX_COUNT betyder IKKE at counteren af sig selv ikke tÃ¦ller hÃ¸jere end til MAX_COUNT
       VARIABLE cntMax : INTEGER;
    BEGIN
       IF reset = '0' THEN-- asynkron reset, ikke afhÃ¦ngig af clk
          -- Reset the counter to 0
          cnt := 0;
       ELSIF (rising_edge(clk)) THEN
          cout <= '0';
          IF clken = '1' THEN
             CASE mode IS
                WHEN "00" =>
                   cntMax := 9;
                WHEN "01" =>
                   cntMax := 5;
                WHEN OTHERS =>
                   cntMax := 2;
             END CASE;

             IF cnt < cntMax THEN
                cnt := cnt + 1;
                cout <= '0';
             ELSE
                cnt := 0;
                cout <= '1';
             END IF;
          END IF;
       END IF;
       -- Output the current count
       count <= STD_LOGIC_VECTOR(to_unsigned(cnt, count'length));
    END PROCESS;

END multi_counter_impl;
```

Figure 2.10: implementering af multi-counter

```
ENTITY multi_counter_tester IS
   PORT (
      -- Input ports
      SW : IN STD_LOGIC_VECTOR(17 DOWNTO 16);
      KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;

      -- Output ports
      HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      LEDR : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)

   );
END multi_counter_tester;

ARCHITECTURE multi_counter_tester_impl OF multi_counter_tester IS
   SIGNAL counter2bin : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL clkSignal : STD_LOGIC;

BEGIN
   LEDG(0) <= KEY(0);
   LEDG(1) <= KEY(1);
   LEDG(2) <= KEY(2);
   LEDG(3) <= KEY(3);

      clk : ENTITY work.clock_gen
      PORT MAP
      (
         clk => CLOCK_50,
         speed => KEY(0),
         reset => KEY(3),

         clk_out => clkSignal
      );

      uut0 : ENTITY work.bin2sevenseg
      PORT MAP
      (
         bin => counter2bin,
         seg => HEX0(6 DOWNTO 0)

      );

   mc0 : ENTITY work.multi_counter
      PORT MAP
      (
         clk => CLOCK_50,
         mode => SW(17 downto 16),
         reset => KEY(3),
         clken => clkSignal,

         count => counter2bin,
         cout =>  LEDR(0)
      );

END multi_counter_tester_impl;
```

Figure 2.11: tester til mult-counter

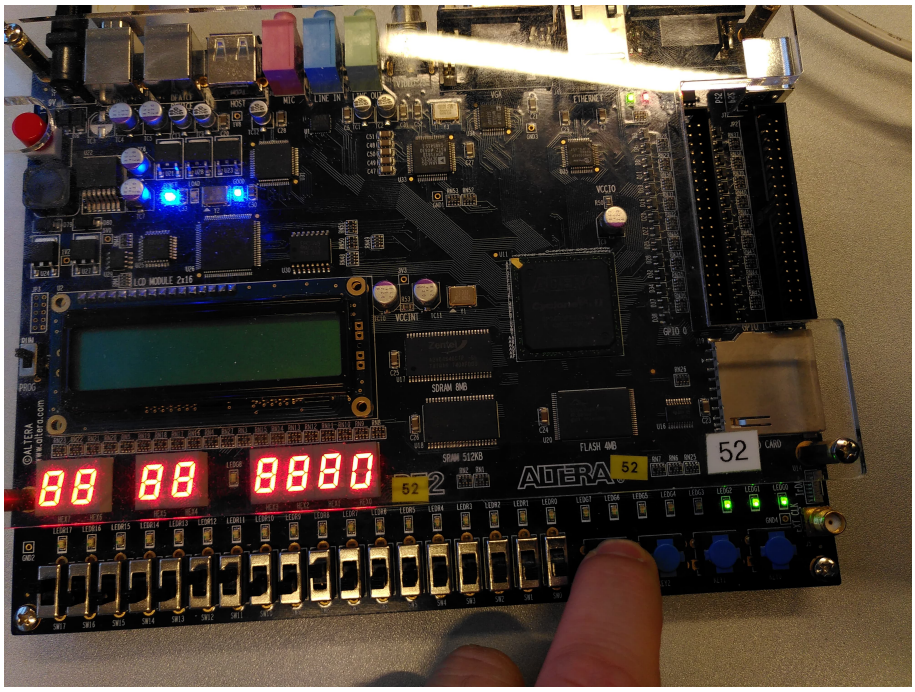

Figure 2.12: Reset holdes nede på KEY3

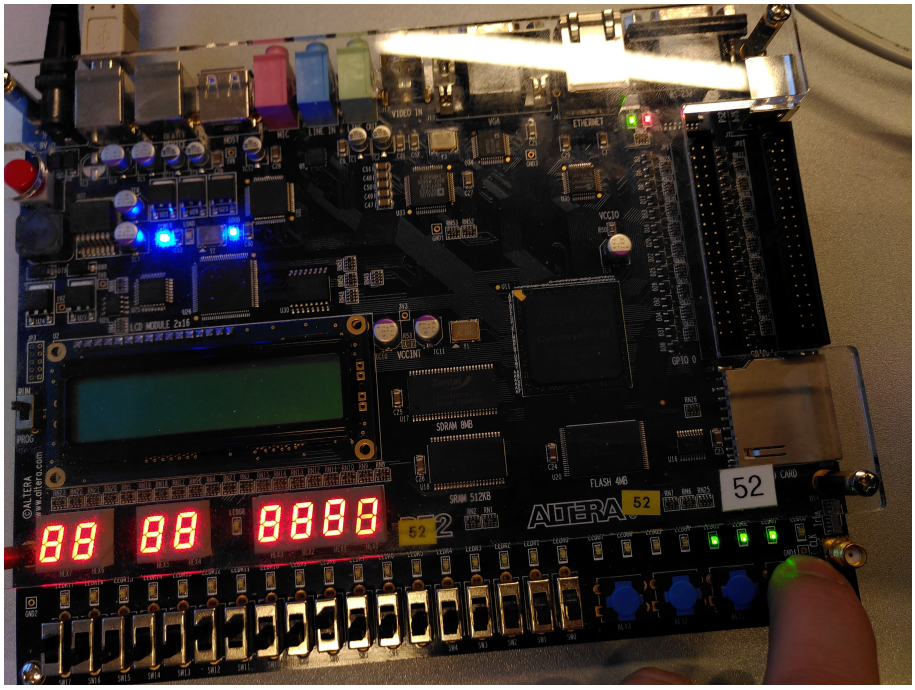

Figure 2.13: Speed holdes nede på KEY0

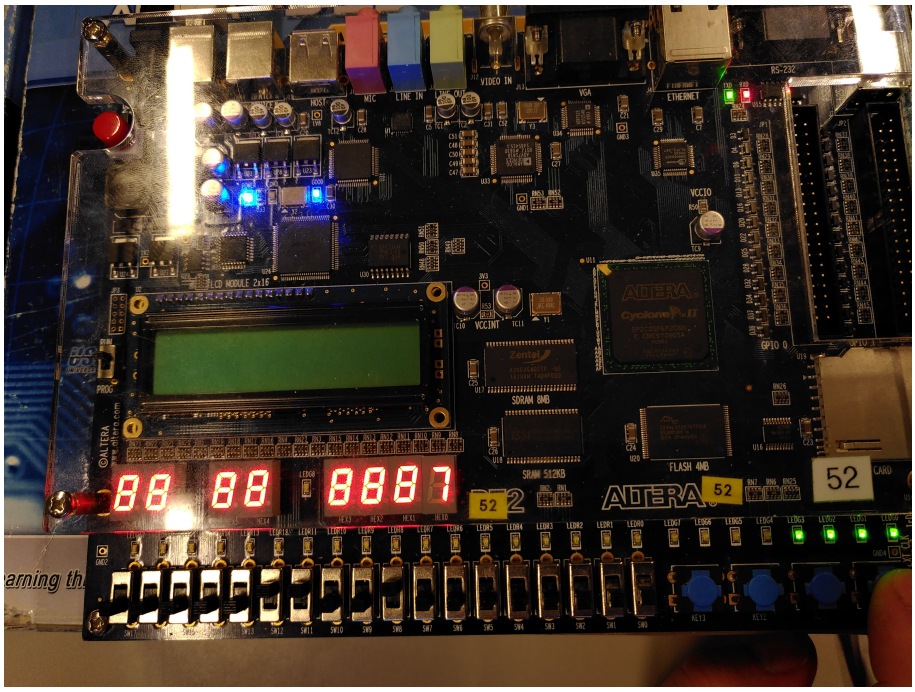

Figure 2.14: Mode ”00” sat på switch 16 og 17

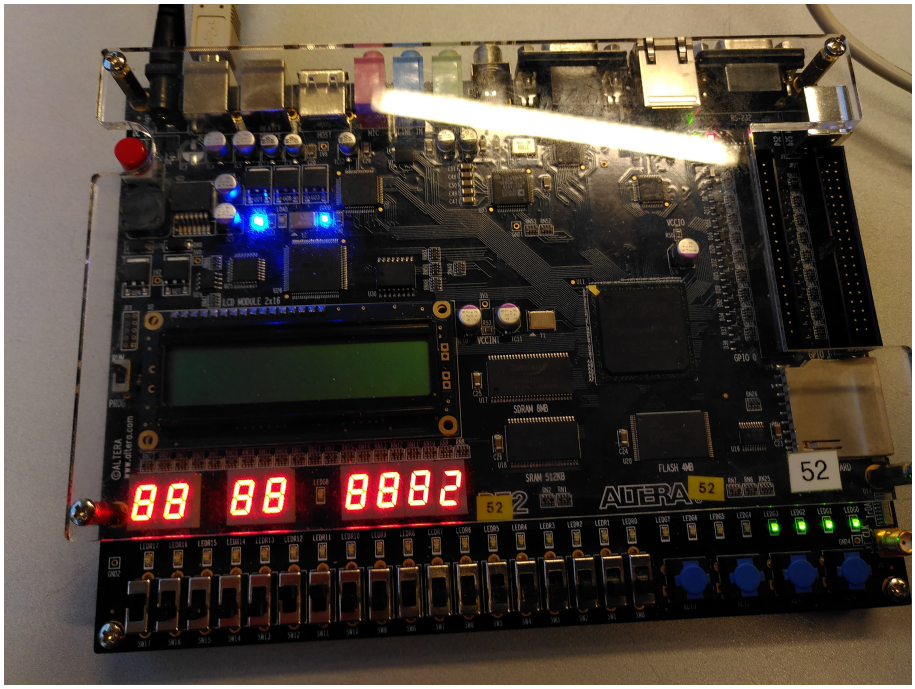

Figure 2.15: Mode ”10” sat sat på switch 16 og 17

2.2.3 Resultater

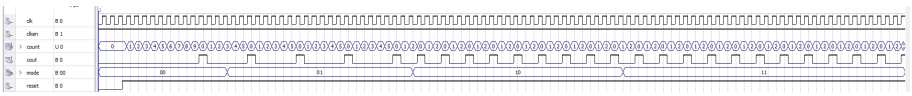

Figure 2.16: implementering af clock_gen

Vi har lavet en funktionel simulering, og observerer at den opføre sig i henhold til kravene beskrevet i opgaven.

2.2.4 Konklusion

Vi har i denne øvelse lavet en tæller i henhold til kravene i opgaven.

2.3 Clock - Six Digit

2.3.1 Introduktion

Vi vil denne opgave lave en six digit tæller i henhold til de krav, som er specificeret i opgaven.

2.3.2 Design og Implementering

Vores watch har følgende entity:

```
ENTITY watch IS
   PORT (
      -- Input ports
      speed : IN STD_LOGIC;
      reset : IN STD_LOGIC;
      clk : IN STD_LOGIC;

      -- Output ports
      sec_1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      sec_10 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      min_1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      min_10 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      hrs_1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      hrs_10 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      tm : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
   );
END watch;
```

Figure 2.17: implementering af watch entity

```
ARCHITECTURE watch_impl OF watch IS
   SIGNAL counter2bin : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL counter2bin1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL counter2bin2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL counter2bin3 : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL counter2bin4 : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL counter2bin5 : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL clkSignal : STD_LOGIC;
   SIGNAL cout2clken0 : STD_LOGIC;
   SIGNAL cout2clken1 : STD_LOGIC;
   SIGNAL cout2clken2 : STD_LOGIC;
   SIGNAL cout2clken3 : STD_LOGIC;
   SIGNAL cout2clken4 : STD_LOGIC;
   SIGNAL reset out : STD LOGIC;
```

Figure 2.18: implementering af watch signals

Instansiering af komponenter

```
rst : ENTITY work.reset_logic
   PORT MAP
   (
      clk => clkSignal,
      hrs_bin1 => counter2bin4,
      hrs_bin10 => counter2bin5,
      reset_in => reset,
      reset_out => reset_out
   );

clk1 : ENTITY work.clock_gen
   PORT MAP
   (
      clk => clk,
      speed => speed,
      reset => reset_out,

      clk_out => clkSignal
   );

uut5 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin5,
      seg => hrs_10

   );

   tm(15 downto 12) <= counter2bin5;

uut4 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin4,
      seg => hrs_1

   );

   tm(11 downto 8) <= counter2bin4;
```

Figure 2.19: Instansiering af komponenter

```
uut3 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin3,
      seg => min_10

   );

   tm(7 downto 4) <= counter2bin3;


uut2 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin2,
      seg => min_1
   );

   tm(3 downto 0) <= counter2bin2;

uut1 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin1,
      seg => sec_10

   );

uut0 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => counter2bin,
      seg => sec_1

   );

mc5 : ENTITY work.multi_counter
   PORT MAP
   (
      clk => clk,
      mode => "01",
      reset => reset_out,
      clken => cout2clken4,

      count => counter2bin5
      );
```

Figure 2.20: Instansiering af komponenter

```
mc4 : ENTITY work.multi_counter
   PORT MAP
   (
      clk => clk,
      mode => "00",
      reset => reset_out,
      clken => cout2clken3,

      count => counter2bin4,
      cout => cout2clken4
   );

mc3 : ENTITY work.multi_counter
   PORT MAP
   (
      clk => clk,
      mode => "01",
      reset => reset_out,
      clken => cout2clken2,

      count => counter2bin3,
      cout => cout2clken3
   );

mc2 : ENTITY work.multi_counter
   PORT MAP
   (
      clk => clk,
      mode => "00",
      reset => reset_out,
      clken => cout2clken1,

      count => counter2bin2,
      cout => cout2clken2
   );

mc1 : ENTITY work.multi_counter
   PORT MAP
   (
      clk => clk,
      mode => "01",
      reset => reset_out,
      clken => cout2clken0,

      count => counter2bin1,
      cout => cout2clken1
   );
```

Figure 2.21: Instansiering af komponenter

```
    mc0 : ENTITY work.multi_counter
        PORT MAP
        (
            clk => clk,
            mode => "00",
            reset => reset_out,
            clken => clkSignal,

            count => counter2bin,
            cout => cout2clken0
        );

END watch_impl;
```

Figure 2.22: Instansiering af komponenter

2.3.3 Test på DE2-Board

```
ENTITY watch_tester IS
   PORT (
      -- Input ports
      SW : IN STD_LOGIC_VECTOR(17 DOWNTO 16);
      KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;

      -- Output ports
      HEX2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX4 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX5 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX6 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX7 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      LEDR : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      tm : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)

   );
END watch_tester;

ARCHITECTURE watch_tester_impl OF watch_tester IS

BEGIN
   LEDG(0) <= KEY(0);
   LEDG(1) <= KEY(1);
   LEDG(2) <= KEY(2);
   LEDG(3) <= KEY(3);
   -- LEDG(4) <= clkSignal;

   watch : ENTITY work.watch
      PORT MAP
      (
         clk => CLOCK_50,
         speed => key(0),
         reset => key(3),
         sec_1 => HEX2,
         sec_10 => HEX3,
         min_1 => HEX4,
         min_10 => HEX5,
         hrs_1 => HEX6,
         hrs_10 => HEX7
      );

END watch_tester_impl;
```

Figure 2.23: Implementering af tester

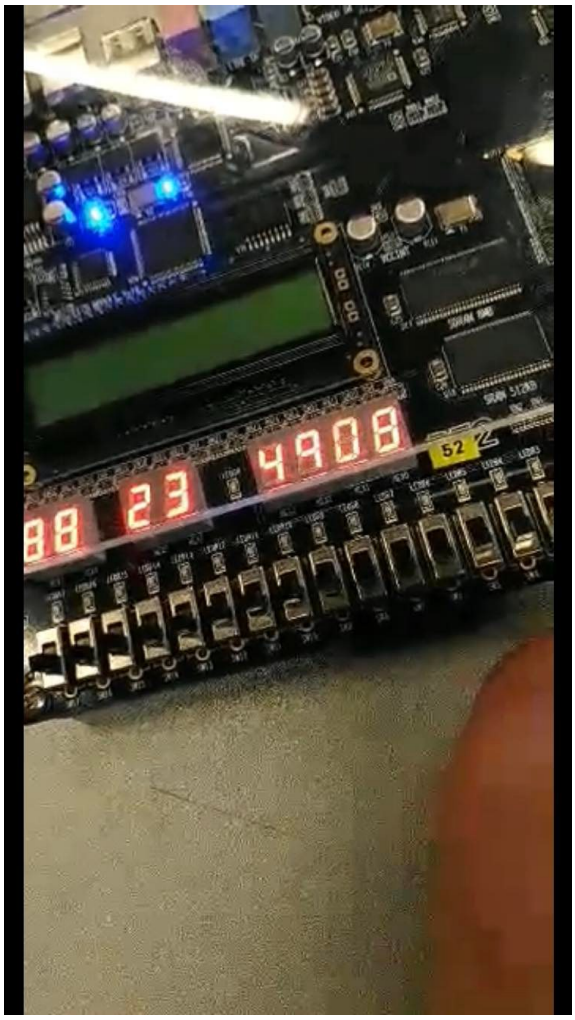

Figure 2.24: Test på boardet

2.3.4 Konklusion

Vi har lavet et ur, som kan tælle op

2.4 Alarm Watch

2.4.1 Introduktion

Formålet med denne opgave er, at lave en alarm. Vil vil implementere den i henhold til ibd'et.

2.4.2 Design og Implementering

i henhold til IBD'et har vi implementeret de påkrævede komponenter

```
ENTITY Input_Limiter IS
   PORT (
      -- Input ports
      bin_min1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      bin_min10 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      bin_hrs1 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      bin_hrs10 : IN STD_LOGIC_VECTOR(3 DOWNTO 0);

      -- Output ports
      time_alarm : OUT STD_LOGIC_vECTOR(15 downto 0)
   );
END Input_Limiter;

ARCHITECTURE Input_Limiter_impl OF Input_Limiter IS
BEGIN

   PROCESS (bin_min1, bin_min10, bin_hrs1, bin_hrs10) IS
      -- Declaration(s)
   BEGIN
      if ((bin_min1 >= "0000") and (bin_min1 <= "1001")) then
         time_alarm(3 downto 0) <= bin_min1;
      else
         time_alarm(3 downto 0)  <= "1111";
      end if;
      if ((bin_min10 >= "0000") and (bin_min10 <= "0101")) then
         time_alarm(7 downto 4) <= bin_min10;
      else
         time_alarm(7 downto 4) <= "1111";
      end if;
      if (((bin_hrs1 >= "0000") and (bin_hrs1 <= "0011")) AND (bin_hrs10 >= "0010")) then -- Tjek om enkelt time højst skal kunne sættes til 3 eller 9
         time_alarm(11 downto 8) <= bin_hrs1;
      elsif (((bin_hrs1 >= "0000") and (bin_hrs1 <= "1001")) AND (bin_hrs10 >= "0000" AND bin_hrs10 < "0010")) then -- Tjek om enkelt time højst skal kunne sættes til 3 eller 9
         time_alarm(11 downto 8) <= bin_hrs1;
      else
         time_alarm(11 downto 8) <= "1111";
      end if;
      if ((bin_hrs10 >= "0000") and (bin_hrs10 <= "0010")) then
         time_alarm(15 downto 12) <= bin_hrs10;
      else
         time_alarm(15 downto 12) <= "1111";
      end if;
   END PROCESS;
END Input_Limiter_impl;
```

Figure 2.25: implementering af input limiter

Her har vi valgt at ændre navnet til input_limiter, fordi compileren havde problemer med mellemrum.

```
ENTITY compare IS
   PORT (
      -- Input ports
      tm_watch : IN STD_LOGIC_VECTOR(15 downto 0);
      tm_alarm : IN STD_LOGIC_VECTOR(15 downto 0);
      -- Output ports
      alarm : OUT STD_LOGIC
   );
END compare;

ARCHITECTURE compare_impl OF compare IS
BEGIN

   PROCESS (tm_alarm,tm_watch) IS
      -- Declaration(s)
   BEGIN
      IF tm_alarm = tm_watch THEN
         alarm <= '1';
      else
         alarm <= '0';

      END IF;
   END PROCESS;
END compare_impl;
```

Figure 2.26: implementering af compare

Dette har vi bundet sammen i en alarm tester

```
ENTITY alarm_watch_tester IS
   PORT (
      -- Input ports
      SW : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
      KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;

      -- Output ports
      HEX2 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX3 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX4 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX5 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX6 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      HEX7 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
      LEDR : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)

   );
END alarm_watch_tester;
```

Figure 2.27: implementering af alarm$_t ester$

```
ARCHITECTURE alarm_watch_tester_impl OF alarm_watch_tester IS

   SIGNAL BinToA1 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToA2 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToA3 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToA4 : STD_LOGIC_VECTOR(6 downto 0);

   SIGNAL BinToB1 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToB2 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToB3 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToB4 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToB5 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL BinToB6 : STD_LOGIC_VECTOR(6 downto 0);
   SIGNAL tmWatchSignal : STD_LOGIC_VECTOR(15 downto 0);
   SIGNAL time_alarm_signal : STD_LOGIC_VECTOR(15 downto 0);
```

Figure 2.28: implementering af alarm$_t ester$

```
rst : ENTITY work.watch
PORT MAP
(
   clk => CloCK_50,
   speed => KEY(0),
   reset => KEY(3),
   sec_1 => BinToB1,
   sec_10 => BinToB2,
   min_1 => BinToB3,
   min_10 => BinToB4,
   hrs_1 => BinToB5,
   hrs_10 => BinToB6,
   tm => tmWatchSignal

);
rst2 : ENTITY work.Input_Limiter
PORT MAP
(
   -- Input ports
   bin_min1 => SW(3 downto 0),
   bin_min10 => SW(7 downto 4),
   bin_hrs1 => SW(11 downto 8),
   bin_hrs10 => SW(15 downto 12),

   -- Output ports
   time_alarm  => time_alarm_signal
);
```

Figure 2.29: implementering af alarm$_t$*ester*

```
mux1 : ENTITY work.Mux
   PORT MAP
   (
   A1 => BinToA1,
   A2 => BinToA2,
   A3 => BinToA3,
   A4 => BinToA4,
   B1 => BinToB1,
   B2 => BinToB2,
   B3 => BinToB3,
   B4 => BinToB4,
   B5 => BinToB5,
   B6 => BinToB6,
   VIEW => key(2),
   -- Output ports
   C1 => HEX2,
   C2 => HEX3,
   C3 => HEX4,
   C4 => HEX5,
   C5 => HEX6,
   C6 => HEX7
   );

min1 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => time_alarm_signal(3 downto 0),
      seg => BinToA1

   );

min10 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => time_alarm_signal(7 downto 4),
      seg => BinToA2

   );
```

Figure 2.30: implementering af alarm$_t$*ester*

```
hrs1 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => time_alarm_signal(11 downto 8),
      seg => BinToA3

   );

hrs10 : ENTITY work.bin2sevenseg
   PORT MAP
   (
      bin => time_alarm_signal(15 downto 12),
      seg => BinToA4

   );

compare: ENTITY work.compare
   PORT MAP
   (
      tm_watch => tmWatchSignal,
      tm_alarm => time_alarm_signal,
      alarm => LEDR(0)
   );
```

Figure 2.31: implementering af alarm$_t$*ester*

2.4.3 Test på DE2-Board

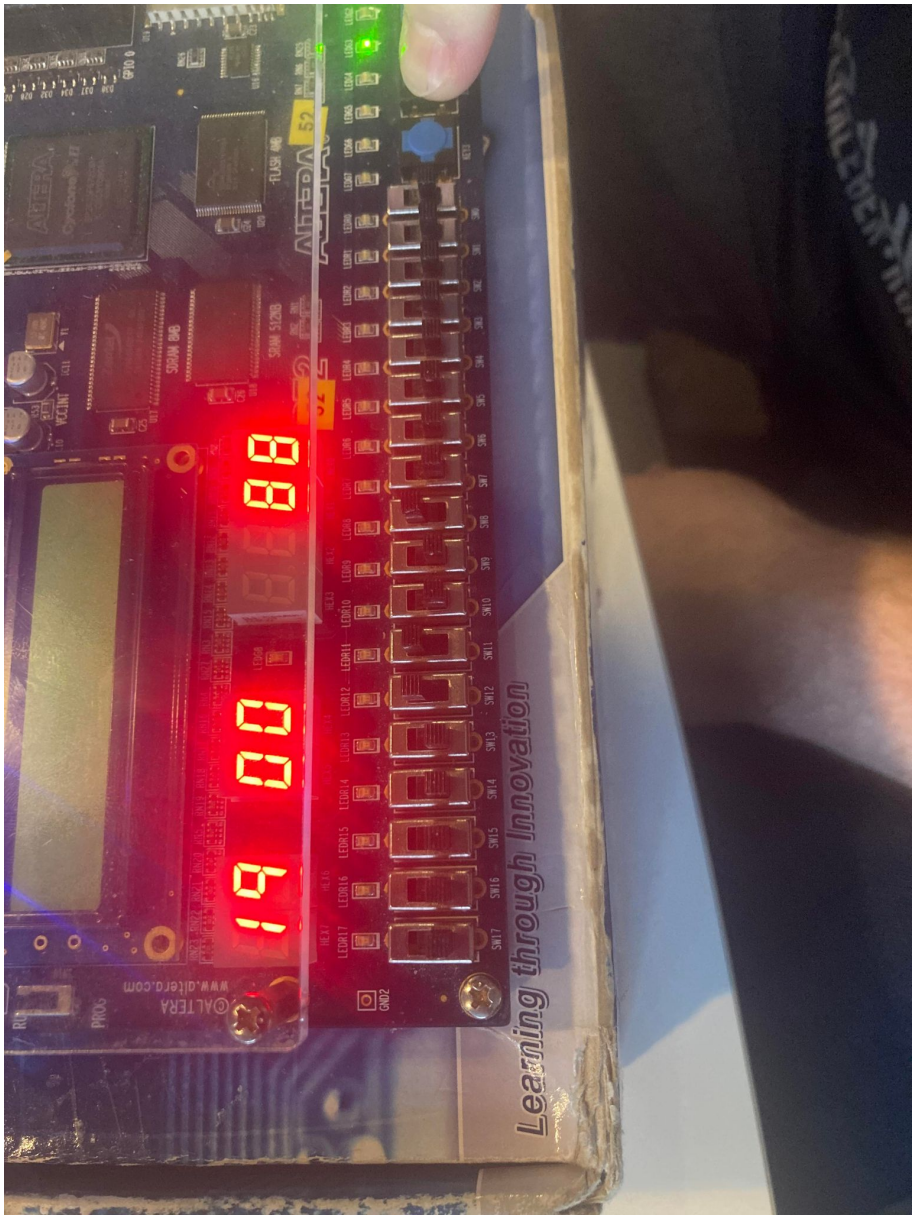

Figure 2.32: Test på DE2 Board. her vises 19 timer

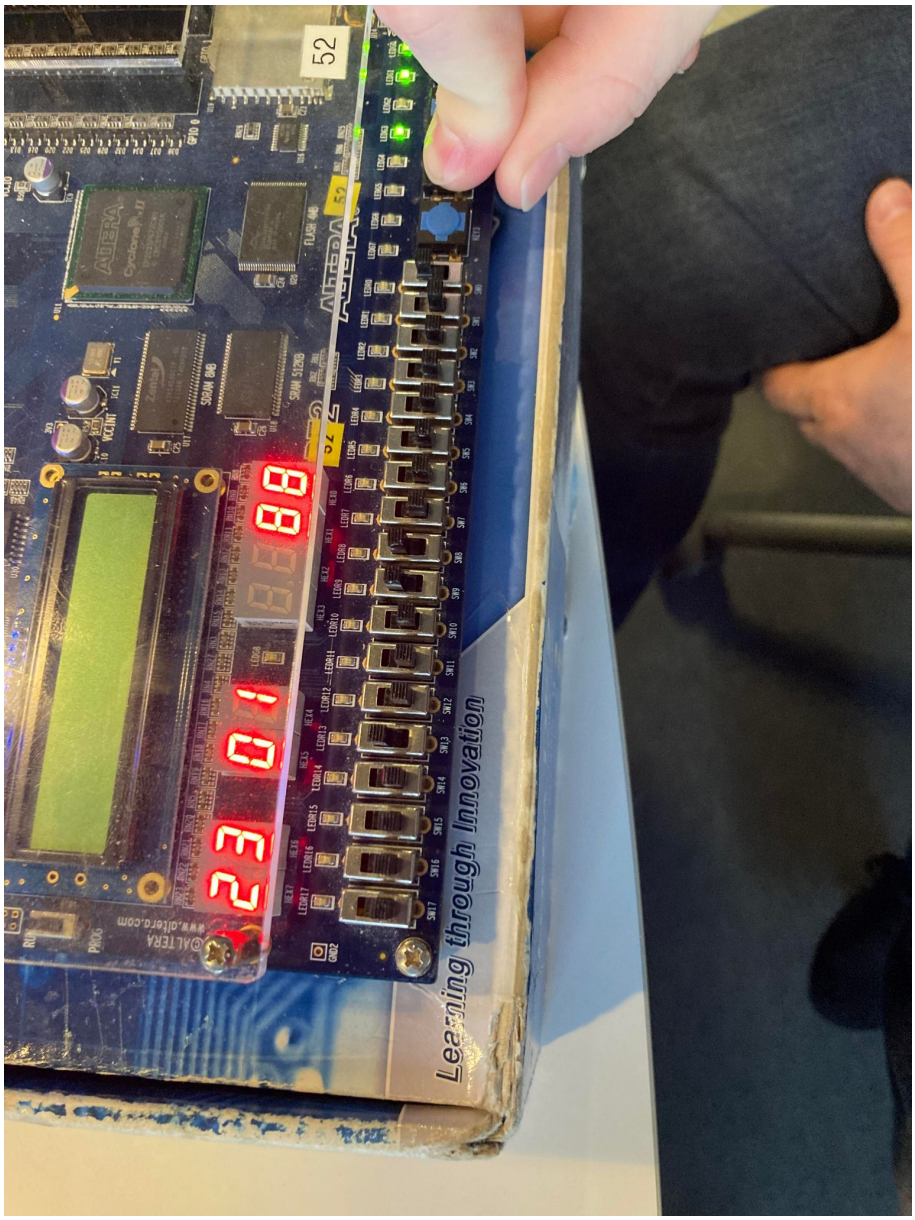

Figure 2.33: Test på DE2 Board. Her vises 23 timer

2.4.4 Konklusion

Vi har implementeret en alarm i henhold til kravene i opgaven.Vi har testet programmet på boardet, og det fungerer som forventet.