



# HØGSKOLEN i Buskerud

**Avdeling for ingeniørutdanning  
Institutt for teknologi**

<b>Oppgavetittel:</b> Obligatorisk prosjektoppgave 1
<b>Fag(nr./navn):</b> Maskinvareutvikling DMVA-2060
<b>Gruppemedlemmer:</b> T. Alexander Lystad
<b>Faglærer:</b> Zoran Dokic
<b>Dato:</b> 15. oktober 2009
Jeg/vi bekrefter at den innleverte besvarelsen helt og fullt er mitt/vårt verk.  .....

## Innholdsfortegnelse

Innledning .....	3
Hvordan representere tallet som skal vises .....	3
Hvordan manipulere tallet som skal vises.....	3
Hvordan vise tre siffer når bare ett siffer kan være aktivt .....	3
Konklusjon .....	3
entity DEC_COUNTER.....	5
entity CLOCK_DIVIDER.....	8
eq2_s3.ucf .....	9

.bit-fil er vedlagt istedenfor simuleringer.

## Innledning

Oppgaven går ut på å programmere Avnet Spartan 3A utvidet med 7-segment LED slik at den teller kontinuerlig fra 0 til 999. Når en knapp holdes nede, skal det telles nedover. Jeg har også valgt å lage en pauseknapp som gjør at tellingen står stille så lenge man holder knappen inne.

Oppgaven byr på disse hovedproblemstillingene:

1. Hvordan representere tallet som skal vises
2. Hvordan manipulere tallet som skal vises
3. Hvordan vise tre siffer når bare ett siffer kan være aktivt

## Hvordan representere tallet som skal vises

Fra en tidligere oppgave har jeg kode som tar utgangspunkt i et siffer og sender tilsvarende bitmønster til LED-displayet. Det er derfor naturlig å representere tallet som skal vises i displayet i tre heltallsvariabler:  $p1$ ,  $p2$  og  $p3$ . Dersom tallet 124 skal vises på displayet, er  $p1$  lik 1,  $p2$  lik 2 og  $p3$  lik 4.

## Hvordan manipulere tallet som skal vises

Dersom ingen knapper er trykket inn inkrementerer jeg  $p3$  for hver klokkesyklus. Dersom  $p3$  er 9 settes  $p3$  lik 0. Det betyr at  $p3$  hele tiden går fra 0 til 9. Jeg gjør det samme med  $p2$ , bortsett fra at jeg bare inkrementerer  $p2$  når  $p3$  er 9.  $p1$  inkrementeres bare når både  $p2$  og  $p3$  er 9. Når  $p1$  eller  $p2$  er 9, settes de lik 0 isteden for å inkrementeres. På denne måten kan  $p1$ ,  $p2$  og  $p3$  telle fra 000 til 999.

Når displayet skal telle nedover (dvs når push button A er trykt), dekrementerer jeg  $p1$ ,  $p2$  og  $p3$  isteden for å inkrementere dem. Når sifferne er 0 settes de til 9, isteden for å dekrementeres. På denne måten kan  $p1$ ,  $p2$  og  $p3$  telle fra 999 til 000.

Inkrementering og dekrementering skjer bare når push button D ikke er trykt. På den måten fungerer push button D som pauseknapp.

## Hvordan vise tre siffer når bare ett siffer kan være aktivt

Den interne klokken til Avnet Spartan 3A er på 16 MHz. Ved å bruke en clock divider, har jeg delt denne på 16, slik at jeg har en klokke på 1 MHz. Jeg bruker denne klokken til å inkrementere en intern teller, med variabelnavn *counter*, fra 0 til 8999.

Inkrementasjon og dekerementasjon av tallet skjer hver gang *counter* er lik 0, altså hvert 0,009 sekund. Perioden på 0,009 sekunder har jeg delt inn i tre, og hver tredjedel av tiden brukes på hvert siffer. Altså vises første siffer i 0,003 sekunder, andre siffer i 0,003 sekunder og tredje siffer i 0,003 sekunder. Dette er tilstrekkelig for at det skal se bra ut.

Disse tallene betyr forøvrig at det tar 9 sekunder å telle fra 0 til 999.

## Konklusjon

Programmet er testet på Avnet Spartan 3A og fungerer slik det skal. Det finnes imidlertid alternative løsninger og forbedringer jeg kunne ha gjort.

Istedenfor å representere tallet i tre siffer, kunne jeg kanskje ha representert tallet i en heltallsvariabel, inkrementert denne, og heller hentet ut siffer 1, 2 og 3 fra dette tallet. Dersom dette er mulig, kunne jeg spart meg for ~40 linjer kode.

Istedenfor å dedikere 0,003 sekunder til siffer 1, så 0,003 sekunder til siffer 2, og så 0,003 sekunder til siffer 3, kunne jeg byttet på å tegne siffer 1, 2 og 3 hver eneste klokkesyklus. Dette hadde vært nødvendig dersom jeg skulle telle saktere enn det jeg gjør nå.

Jeg har samme kode for å konvertere et siffer til riktig LCD-segment-bitmønster 3 steder. En måte å eliminere duplikasjonen på kunne være å skille ut denne funksjonaliteten i en egen entity. Dette kunne spart meg for mange linjer kode.

# entity DEC\_COUNTER

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

entity DEC_COUNTER is
    port(
        CLK_IN:    in std_logic;           -- 16 MHz klokka
        IN_A:      in std_logic;          -- Push button A
        IN_RESET:  in std_logic;          -- Push button RESET
        D2, D3, D4: out std_logic;         -- Valg av siffer
        display:   out std_logic_vector(0 to 6) -- Displayet
    );
end DEC_COUNTER;

architecture BEHAVIORAL of DEC_COUNTER is
    signal CLK:        std_logic;
    signal p1:         integer range 0 to 9 :=0;
    signal p2:         integer range 0 to 9 :=0;
    signal p3:         integer range 0 to 9 :=0;
    signal counter:    integer range 0 to 8999 :=0;
begin

    uut: entity work.CLOCK_DIVIDER(BEHAVIORAL)
        port map(CLK_IN, CLK);

    process(CLK)
    begin
        if (CLK'event and CLK = '1') then
            counter <= counter + 1;
            if (counter = 0 and IN_RESET = '0') then
                if (IN_A = '0') then
                    -- Increment 'one' digit
                    if (p3 = 9) then
                        p3 <= 0;
                    else
                        p3 <= p3 + 1;
                    end if;
                    -- Increment 'ten' digit
                    if (p3 = 9) then
                        if (p2 = 9) then
                            p2 <= 0;
                        else
                            p2 <= p2 + 1;
                        end if;
                    end if;
                    -- Increment 'hundered' digit
                    if (p2 = 9 and p3 = 9) then
                        if (p1 = 9) then
                            p1 <= 0;
                        else
                            p1 <= p1 + 1;
                        end if;
                    end if;
                else
                    -- Decrement 'one' digit
                    if (p3 = 0) then
                        p3 <= 9;
                    else
                        p3 <= p3 - 1;
                    end if;
                    -- Decrement 'ten' digit
                    if (p3 = 0) then
                        if (p2 = 0) then
                            p2 <= 9;
                        else
                            p2 <= p2 - 1;
                        end if;
                    end if;
                    -- Decrement 'hundered' digit
                    if (p2 = 0 and p3 = 0) then
                        if (p1 = 0) then
                            p1 <= 9;
                        else
                            p1 <= p1 - 1;
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end process;
end;
```

```

                end if;
            end if;
        end if;
    end process;

process (counter)
begin
if (counter < 3000) then
    D2 <= '0';
    D3 <= '1';
    D4 <= '1';
    case p1 is
        when 0 =>
            display(0 to 6) <= "0111111";
        when 1 =>
            display(0 to 6) <= "0000110";
        when 2 =>
            display(0 to 6) <= "1011011";
        when 3 =>
            display(0 to 6) <= "1001111";
        when 4 =>
            display(0 to 6) <= "1100110";
        when 5 =>
            display(0 to 6) <= "1101101";
        when 6 =>
            display(0 to 6) <= "1111101";
        when 7 =>
            display(0 to 6) <= "0000111";
        when 8 =>
            display(0 to 6) <= "1111111";
        when 9 =>
            display(0 to 6) <= "1101111";
        when others =>
            display(0 to 6) <= "1000000";
    end case;
elseif (counter >= 2999 and counter < 5999) then
    D2 <= '1';
    D3 <= '0';
    D4 <= '1';
    case p2 is
        when 0 =>
            display(0 to 6) <= "0111111";
        when 1 =>
            display(0 to 6) <= "0000110";
        when 2 =>
            display(0 to 6) <= "1011011";
        when 3 =>
            display(0 to 6) <= "1001111";
        when 4 =>
            display(0 to 6) <= "1100110";
        when 5 =>
            display(0 to 6) <= "1101101";
        when 6 =>
            display(0 to 6) <= "1111101";
        when 7 =>
            display(0 to 6) <= "0000111";
        when 8 =>
            display(0 to 6) <= "1111111";
        when 9 =>
            display(0 to 6) <= "1101111";
        when others =>
            display(0 to 6) <= "1000000";
    end case;
else
    D2 <= '1';
    D3 <= '1';
    D4 <= '0';
    case p3 is
        when 0 =>
            display(0 to 6) <= "0111111";
        when 1 =>
            display(0 to 6) <= "0000110";
        when 2 =>
            display(0 to 6) <= "1011011";
        when 3 =>
            display(0 to 6) <= "1001111";
        when 4 =>
            display(0 to 6) <= "1100110";
        when 5 =>

```

```
        display(0 to 6) <= "1101101";
when 6 =>
    display(0 to 6) <= "1111101";
when 7 =>
    display(0 to 6) <= "0000111";
when 8 =>
    display(0 to 6) <= "1111111";
when 9 =>
    display(0 to 6) <= "1101111";
when others =>
    display(0 to 6) <= "1000000";
    end case;
end if;
end process;
```

```
end BEHAVIORAL;
```

## entity CLOCK\_DIVIDER

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

-- CLOCK_DIVIDER
entity CLOCK_DIVIDER is
  port(
    CLKin:    in std_logic;
    CLKout:   out std_logic
  );
end CLOCK_DIVIDER;

architecture BEHAVIORAL of CLOCK_DIVIDER is
  constant DivideBy: Integer := 16;
begin
  process(CLKin)
    variable count: Natural range 0 to DivideBy-1;
  begin
    if CLKin'event and (CLKin = '1') then
      if (count = DivideBy-1) then
        count := 0;
        CLKout <= '0';
      else
        count := count + 1;
      end if;
      if count >= DivideBy/2 then
        CLKout <= '0';
      else
        CLKout <= '1';
      end if;
    end if;
  end process;
end BEHAVIORAL;
```

## eq2\_s3.ucf

```
NET CLK_IN          LOC = C10 | IOSTANDARD = LVCMOS33 ; # CLK_16MHZ
NET IN_RESET        LOC = H4   | IOSTANDARD = LVCMOS33 ; # FPGA_RESET
NET IN_A            LOC = K3   | IOSTANDARD = LVCMOS33 ; # FPGA_PUSH_A
NET "display<1>"    LOC = A14  | IOSTANDARD = LVCMOS33 ; # BANK0_IO1
NET "display<6>"    LOC = C4   | IOSTANDARD = LVCMOS33 ; # BANK0_IO2
NET "D3"            LOC = A13  | IOSTANDARD = LVCMOS33 ; # BANK0_IO3
NET "D2"            LOC = B14  | IOSTANDARD = LVCMOS33 ; # BANK0_IO4
NET "display<5>"    LOC = D13  | IOSTANDARD = LVCMOS33 ; # BANK0_IO6
NET "D4"            LOC = A12  | IOSTANDARD = LVCMOS33 ; # BANK0_IO7
NET "display<4>"    LOC = B12  | IOSTANDARD = LVCMOS33 ; # BANK0_IO9
NET "display<0>"    LOC = D11  | IOSTANDARD = LVCMOS33 ; # BANK0_IO10
NET "display<3>"    LOC = A11  | IOSTANDARD = LVCMOS33 ; # BANK0_IO11
NET "display<2>"    LOC = D10  | IOSTANDARD = LVCMOS33 ; # BANK0_IO14
```