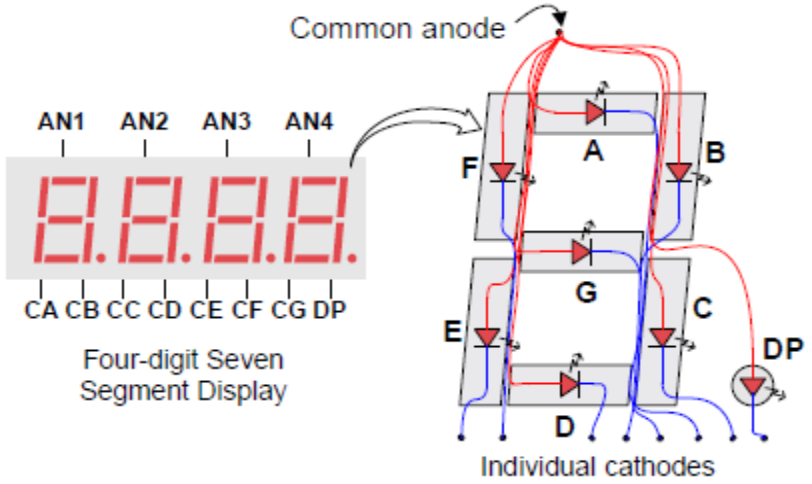
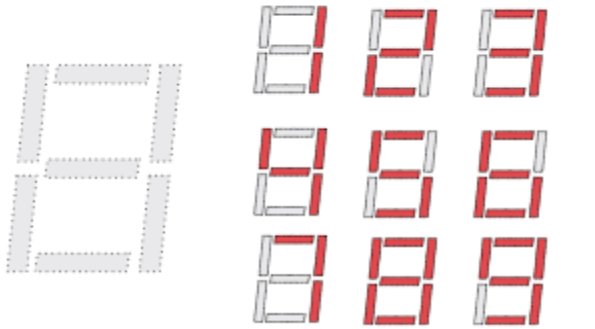
Multiplexed Display Start Help





An un-illuminated seven-segment display, and nine illumination patterns corresponding to decimal digits

Figure 7. Seven-segment display

For each of the four digits to appear bright and continuously illuminated, all four digits should be driven once every 1 to 16ms (for a refresh frequency of 1KHz to 60Hz). For example, in a 60Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for ¼ of the refresh cycle, or 4ms. The controller must assure that the correct cathode pattern is present when the corresponding anode signal is driven.

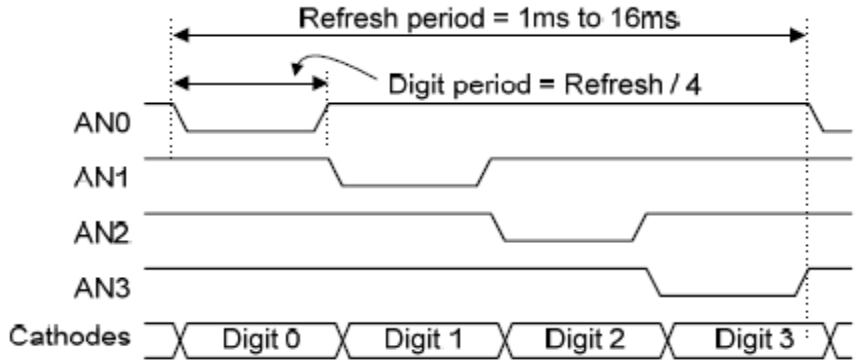
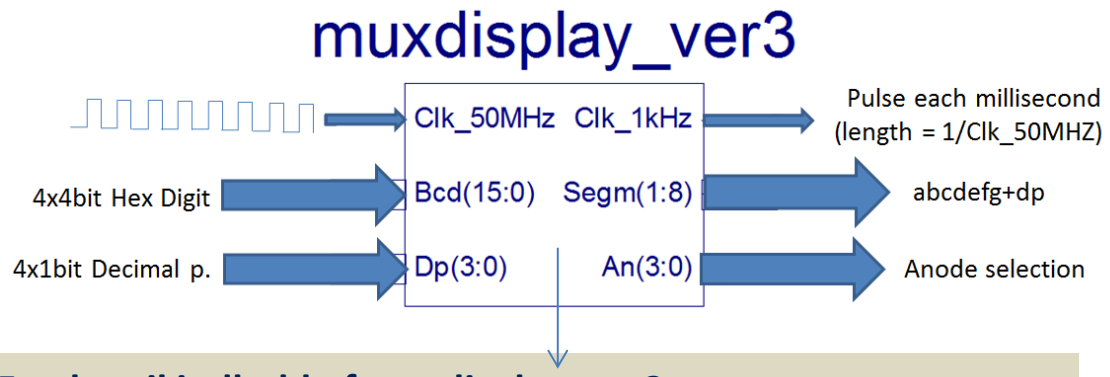
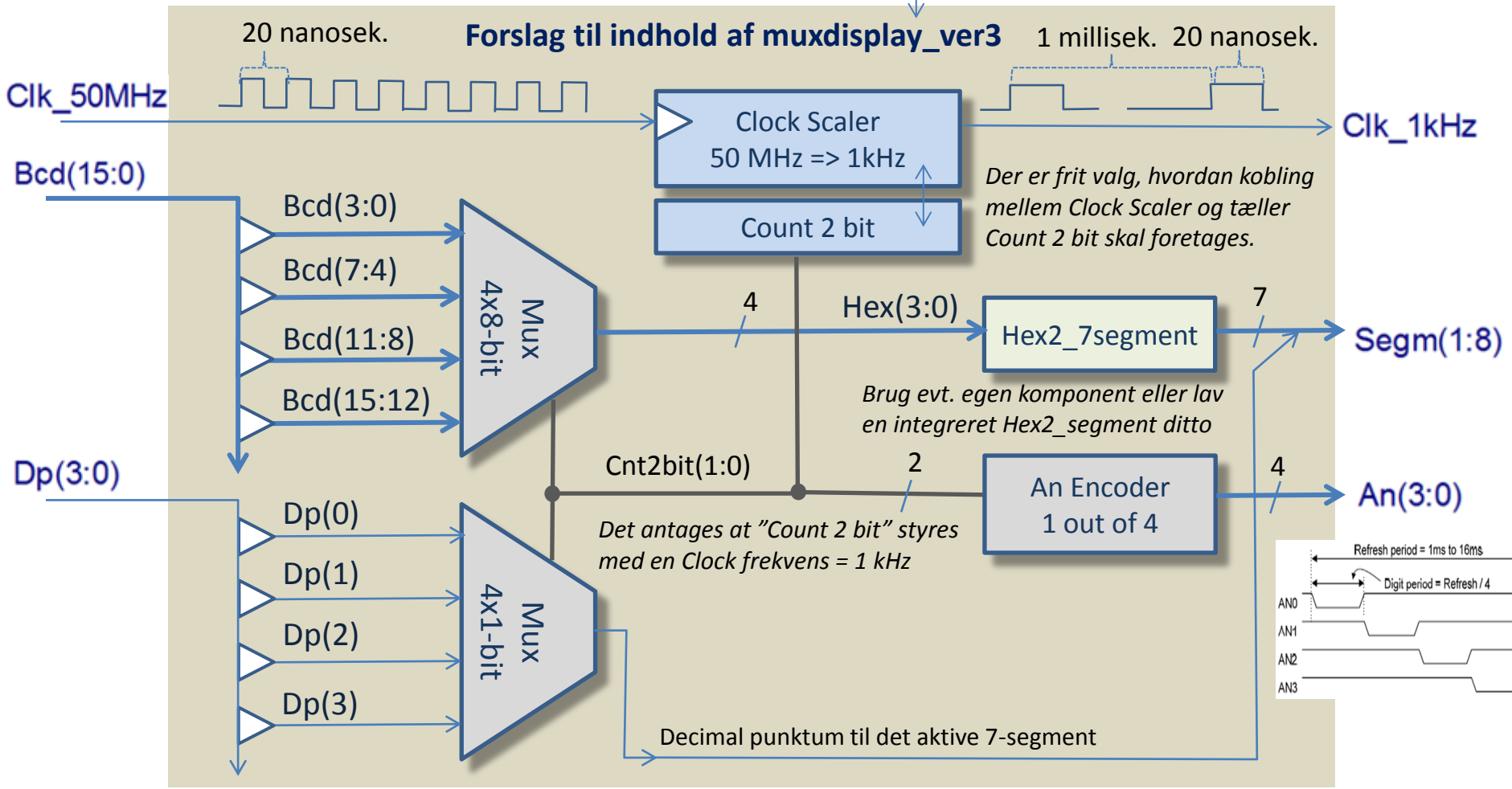


Figure 8. Multiplexed 7seg display timing

To illustrate the process, if AN1 is asserted while CB and CC are asserted, then a "1" will be displayed in digit position 1. Then, if AN2 is asserted while CA, CB and CC are asserted, then a "7" will be displayed in digit position 2. If A1 and CB, CC are driven for 4ms, and then A2 and CA, CB, CC are driven for 4ms in an endless succession, the display will show "17" in the first two digits. Figure 8 shows an example timing diagram for a four-digit seven-segment controller.

Delopgave: Der skal udvikles en komponent ”muxdisplay” som kan styre det multipleksede display på BASYS/NEXYS – bliv inspireret af det følgende.





VHDL Code of the MuxDisplay StartHelp

Download alle kildetekster fra Bb

```
1  ----------------------------------------------------------------------------
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity MuxDisplay_StartHelp is
8      Generic( n: integer := 20); -- Values from 0 to 20
9
10     Port ( Clk :  in  STD_LOGIC;
11            Sw :   in  STD_LOGIC_VECTOR (1 downto 0);
12            Segm : out STD_LOGIC_VECTOR (1 to 8);
13            An :   out STD_LOGIC_VECTOR (3 downto 0));
14 end MuxDisplay_StartHelp;
15
16 architecture Behavioral of MuxDisplay_StartHelp is
17    signal Counter: STD_LOGIC_VECTOR (35 downto 0) := X"123456789";
18    signal Cnt2bit: STD_LOGIC_VECTOR ( 1 downto 0) := "00";
19    signal Hex:     STD_LOGIC_VECTOR ( 3 downto 0);
20    signal abcdefg: STD_LOGIC_VECTOR (1 to 7); -- & Decimal Point
21    signal Scalecnt: integer;
22
23    COMPONENT Hex2_7segment_v1
24    PORT( Hex :     IN  STD_LOGIC_VECTOR(3 downto 0);
25          abcdefg : OUT STD_LOGIC_VECTOR(1 to 7));
26    END COMPONENT;
27 begin
28    ------------ Please use your decoder here ----------------------
29    U1: Hex2_7segment_v1 PORT MAP( Hex => Hex, abcdefg => abcdefg);
30
31    Segm <= abcdefg & '1';  -- & Decimal Point
32
33 -- TestCounter: process( Clk)  -- Remove the comments to use
34 -- begin
35 --    if rising_edge(Clk) then
36 --       Counter <= Counter+1;
37 --    end if;
38 -- end process;
39
```

```
40    ScaleProcess: PROCESS( Clk, Sw)
41       variable Scale_count: integer range 0 to 50000000 := 0;
42       variable xScale:      integer range 0 to 50000000 := 0;
43       ---------------------
44       constant   x3Hz:     integer  := 50000000/3;
45       constant  x75Hz:     integer  := 50000000/75;
46       constant x200Hz:     integer  := 50000000/200;
47       constant x10MHz:     integer  := 50000000/10000000; -- For simulation
48    begin
49       -------------- Select a xScale for clk scaling ----------------------
50       case Sw is
51          when "00" =>    xScale := x10MHz;
52          when "01" =>    xScale := x3Hz;
53          when "10" =>    xScale := x75Hz;
54          when others =>  xScale := x200Hz;
55       end case;
56       ------------------------------- Wait for a 50 MHz _+-- edge
57       if rising_edge( Clk) then
58          Scale_count := Scale_count+1;
59
60          if Scale_count > xScale then
61             Scale_count := 1;
62             Cnt2bit     <= Cnt2bit+1;  -- The Two bit Counter
63          end if;
64          ScaleCnt <= Scale_count;
65       end if;
66    end process;
67
68    -- Multiplexer for the Hex selection ----------------------------
69    with Cnt2bit select
70    Hex <=     Counter( n+15 downto n+12) when "11",
71               Counter( n+11 downto n+8)  when "10",
72               Counter( n+7  downto n+4)  when "01",
73               Counter( n+3  downto   n)  when others;
74
75    -- Encoder (1 of 4) for the An selection ------------------------
76    with Cnt2bit select
77    An <= "1110" when "00",
78          "1101" when "01",
79          "1011" when "10",
80          "0111" when others;
81
82 end Behavioral;
```

Block diagram of the principles behind a multiplexed display

```
entity MuxDisplay_StartHelp is
    Generic( n: integer := 20); -- Values from 0 to 20

    Port ( Clk :  in  STD_LOGIC;
           Sw :   in  STD_LOGIC_VECTOR (1 downto 0);
           Segm : out STD_LOGIC_VECTOR (1 to 8);
           An :   out STD_LOGIC_VECTOR (3 downto 0));
end MuxDisplay_StartHelp;
```

Bemærk! Denne process er kommenteret ud i oplægget

```
Counter := X"123456789"
```

```
TestCounter: process( Clk)
begin
   if rising_edge(Clk) then
      Counter <= Counter+1;
   end if;
end process;
```

```
-- Multiplexer for the Hex selection ------------
with Cnt2bit select
Hex <=     Counter( n+15 downto n+12) when "11",
           Counter( n+11 downto n+8)  when "10",
           Counter( n+7  downto n+4)  when "01",
           Counter( n+3  downto   n)  when others;
```

Mindst Betydende Ciffer

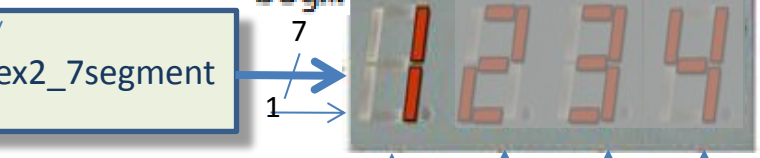
n angiver hvilke cifre som skal vises

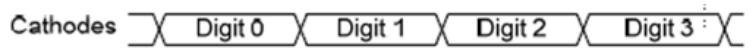
4 4 4 4

4

Hex

```
COMPONENT Hex2_7segment_v1
PORT( Hex :      IN  STD_LOGIC_VECTOR(3 downto 0);
      abcdefg : OUT STD_LOGIC_VECTOR(1 to 7));
END COMPONENT;
```



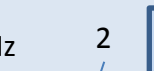


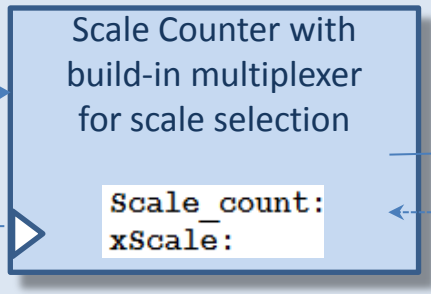
An

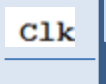
Cnt2bit

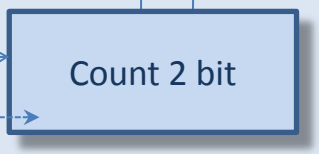

```
with Cnt2bit select
An <= "1110" when "00",
      "1101" when "01",
      "1011" when "10",
      "0111" when others;
```

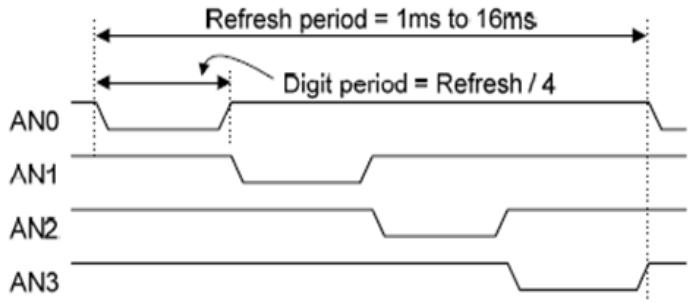
Sw
00 => 10 MHz
01 => 3 Hz
10 => 75 Hz
11 => 200 Hz





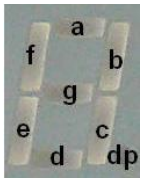


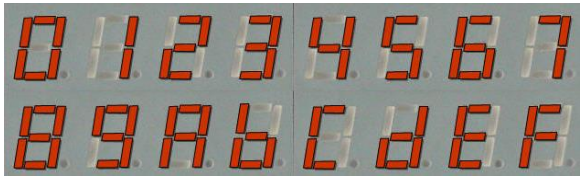




Two versions of Hex to 7 segment component

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

entity Hex2_7segment_v1 is
    Port ( Hex :     in  STD_LOGIC_VECTOR (3 downto 0);
           abcdefg : out STD_LOGIC_VECTOR (1 to 7));
end Hex2_7segment_v1;

architecture Behavioral of Hex2_7segment_v1 is
   signal a,b,c,d,e,f,g: STD_LOGIC;
   signal Hexi: integer range 0 to 15;
begin
   Hexi    <= conv_integer( Hex);           -- Convert to integer
   abcdefg <= a & b & c & d & e & f & g;    -- Concentate to vector

   a <= '1' when Hexi=1 or Hexi=4 or Hexi=11 or Hexi=13 else '0';

   with Hexi select
   b <= '1' when 5|6|11|12|14|15,
        '0' when others;

   c <= (not Hex(3) and not Hex(2) and     Hex(1) and not Hex(0)) or -- "0010"
        (    Hex(3) and     Hex(2) and not Hex(1) and not Hex(0)) or -- "1100"
        (    Hex(3) and     Hex(2) and     Hex(1) and not Hex(0)) or -- "1110"
        (    Hex(3) and     Hex(2) and     Hex(1) and     Hex(0));   -- "1111"

   process( Hexi)
   begin
      d <= '0'; -- It's only in a process that default values allowed
      if (Hexi=1) or (Hexi=4) or (Hexi=7) or (Hexi=10) or (Hexi=15) then
         d <= '1';
      end if;
   end process;

   process( Hexi)
   begin
      if (Hexi=1)or(Hexi=3)or(Hexi=4) or (Hexi=5) or (Hexi=7) or Hexi=9 then
         e <= '1';
      else
         e <= '0';
      end if;
   end process;
```



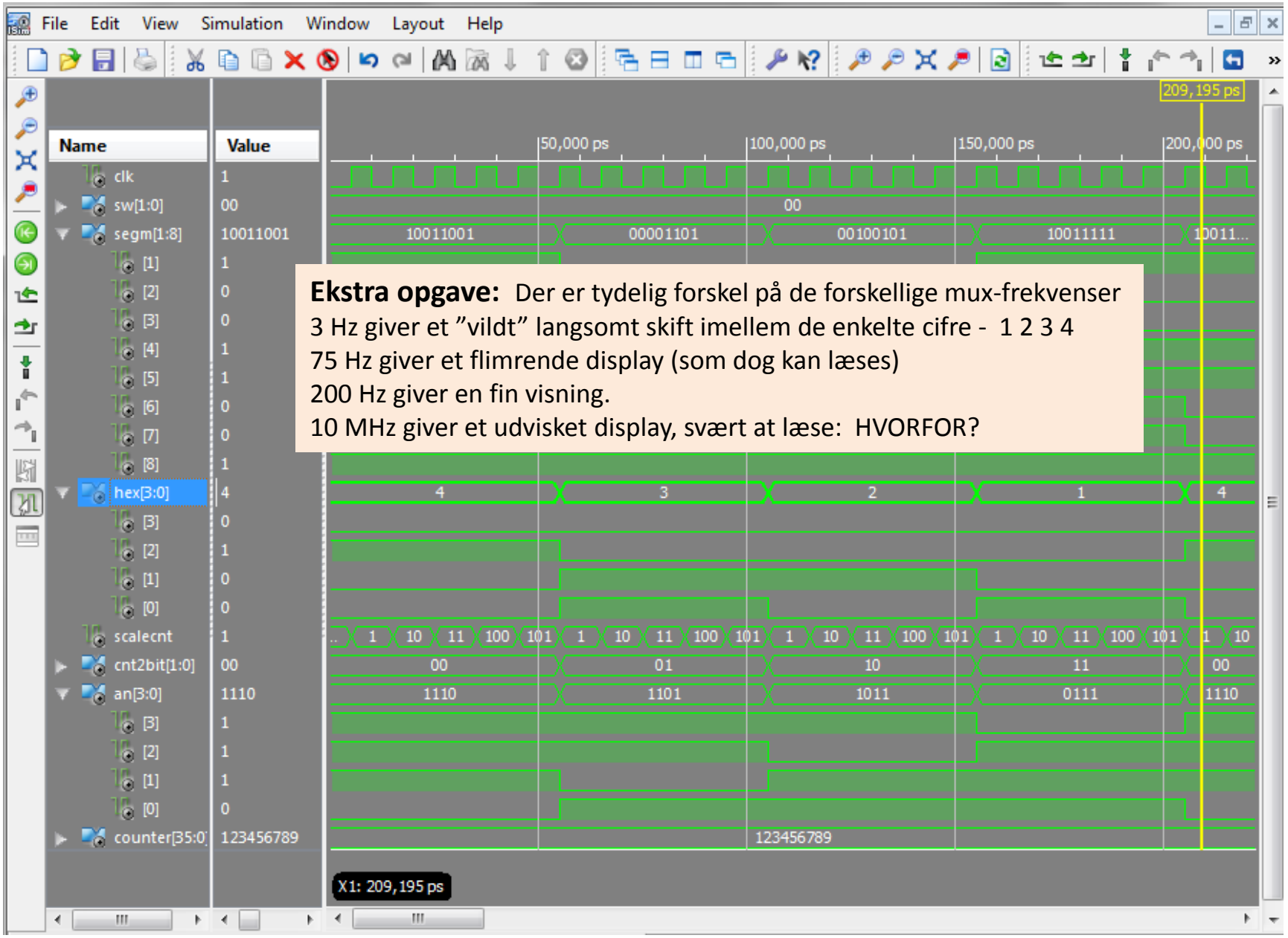



```vhdl
   process( Hexi)
   begin
      case Hexi is
         when 1|2|3|7|13 => f <= '1';
         when others     => f <= '0';
      end case;
   end process;

   process( Hexi)
      constant Segm_g: STD_LOGIC_VECTOR( 0 to 15) := "1100000100001000";
   begin
      g <= Segm_g( Hexi);
   end process;

end Behavioral;
```

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

entity Hex2_7segment_v2 is
    Port ( Hex :     IN  STD_LOGIC_VECTOR (3 downto 0);
           abcdefg: OUT STD_LOGIC_VECTOR (1 to 7));
end Hex2_7segment_v2;

architecture Behavioral of Hex2_7segment_v2 is
begin
   WITH Hex SELECT
   abcdefg <= "0000001" WHEN "0000", --OUTPUT WHEN INPUT
              "1001111" WHEN "0001",
              "0010010" WHEN "0010",
              "0000110" WHEN "0011",
              "1001100" WHEN "0100",
              "0100100" WHEN "0101",
              "0100000" WHEN "0110",
              "0001111" WHEN "0111",
              "0000000" WHEN "1000",
              "0000100" WHEN "1001",
              "0001000" WHEN "1010",
              "1100000" WHEN "1011",
              "0110001" WHEN "1100",
              "1000010" WHEN "1101",
              "0110000" WHEN "1110",
              "0111000" WHEN "1111",
              "0000000" WHEN OTHERS;
end Behavioral;
```

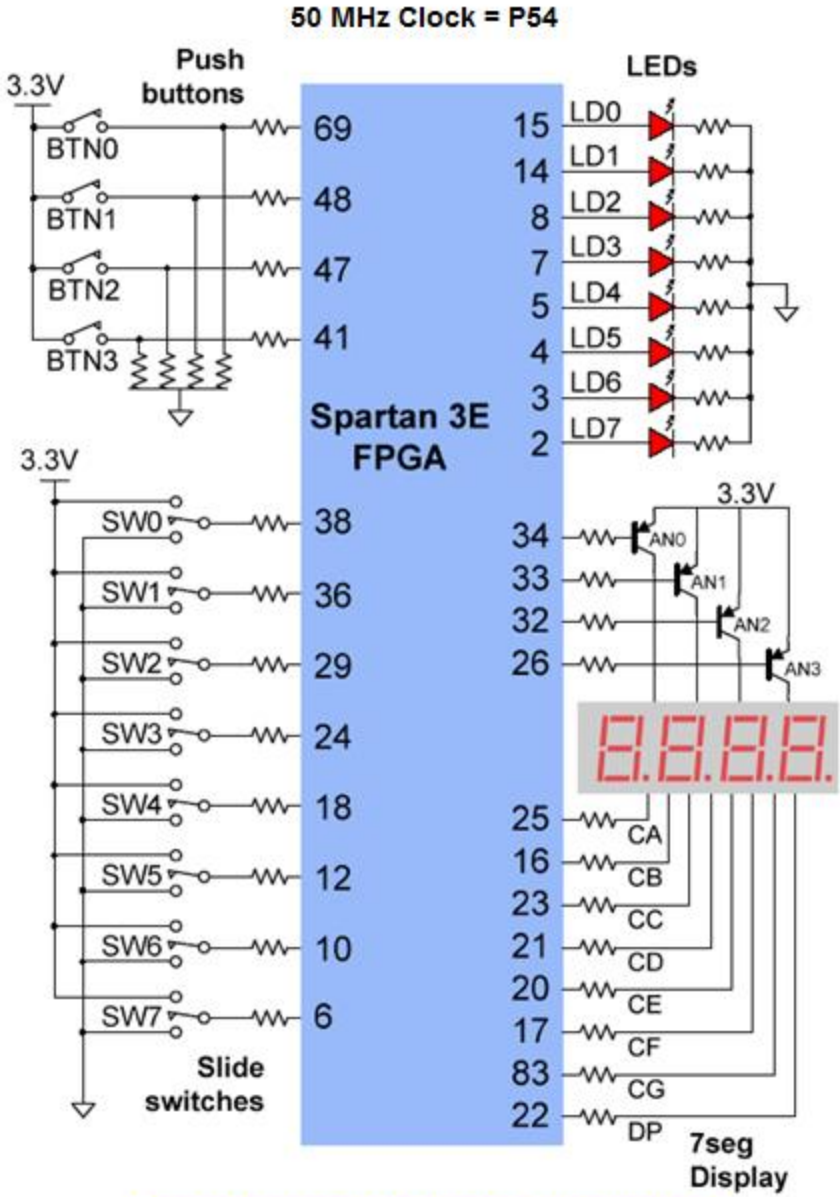
Simulation of the MuxDisplay StartHelp

Bemærk! Denne simulation kræver blot at man opretter en TestBench – den vil kunne bruges uden der skal skives en eneste linje VHDL kode



Ekstra opgave: Der er tydelig forskel på de forskellige mux-frekvenser
3 Hz giver et ”vildt” langsomt skift imellem de enkelte cifre - 1 2 3 4
75 Hz giver et flimrende display (som dog kan læses)
200 Hz giver en fin visning.
10 MHz giver et udvisket display, svært at læse: HVORFOR?

BASYS - pins



Remember to add "P" to the pin numbers

NEXYS2 - pins

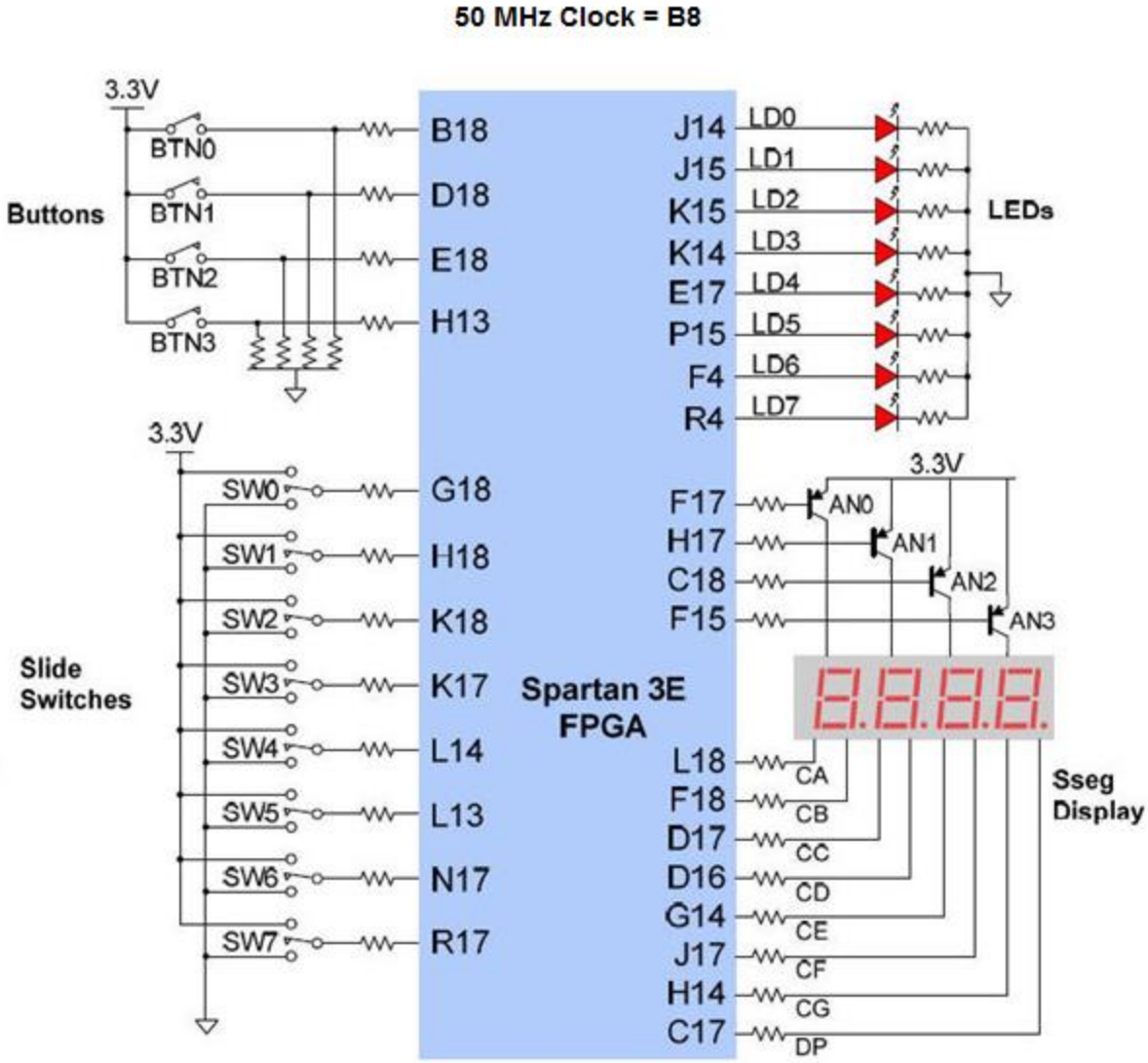


Figure 8: Nexys2 I/O devices and circuits

Content of the UCF (User Constrain File) – Depend on the kit used

```
### B A S Y S - kit ########
#--------------------- 7 Segment display
NET "An<0>"     LOC = "p34"  ;
NET "An<1>"     LOC = "p33"  ;
NET "An<2>"     LOC = "p32"  ;
NET "An<3>"     LOC = "p26"  ;
NET "Segm<1>"   LOC = "p25"  ;  #a segment
NET "Segm<2>"   LOC = "p16"  ;  #b
NET "Segm<3>"   LOC = "p23"  ;  #c
NET "Segm<4>"   LOC = "p21"  ;  #d
NET "Segm<5>"   LOC = "p20"  ;  #e
NET "Segm<6>"   LOC = "p17"  ;  #f
NET "Segm<7>"   LOC = "p83"  ;  #g
NET "Segm<8>"   LOC = "p22"  ;  #Dp

#NET "Btn<0>"      LOC = "p69"  ;
#NET "Btn<1>"      LOC = "p48"  ;
#NET "Btn<2>"      LOC = "p47"  ;
#NET "Btn<3>"      LOC = "p41"  ;
#
#NET "Ld<0>"    LOC = "p15"  ;
#NET "Ld<1>"    LOC = "p14"  ;
#NET "Ld<2>"    LOC = "p8"   ;
#NET "Ld<3>"    LOC = "p7"   ;
#NET "Ld<4>"    LOC = "p5"   ;
#NET "Ld<5>"    LOC = "p4"   ;
#NET "Ld<6>"    LOC = "p3"   ;
#NET "Ld<7>"    LOC = "p2"   ;

NET "Sw<0>"     LOC = "p38"  ;
NET "Sw<1>"     LOC = "p36"  ;
#NET "Sw<2>"    LOC = "p29"  ;
#NET "Sw<3>"    LOC = "p24"  ;
#NET "Sw<4>"    LOC = "p18"  ;
#NET "Sw<5>"    LOC = "p12"  ;
#NET "Sw<6>"    LOC = "p10"  ;
#NET "Sw<7>"    LOC = "p6"   ;

NET "Clk" LOC = "p54" ; #Clk 50MHz
```

```
### N E X Y S - kit ########
##--------------------- 7 Segment display
NET "An<0>"     LOC = "F17"  ;
NET "An<1>"     LOC = "H17"  ;
NET "An<2>"     LOC = "C18"  ;
NET "An<3>"     LOC = "F15"  ;
NET "Segm<1>"   LOC = "L18"  ;  #a - Segment
NET "Segm<2>"   LOC = "F18"  ;  #b
NET "Segm<3>"   LOC = "D17"  ;  #c
NET "Segm<4>"   LOC = "D16"  ;  #d
NET "Segm<5>"   LOC = "G14"  ;  #e
NET "Segm<6>"   LOC = "J17"  ;  #f
NET "Segm<7>"   LOC = "H14"  ;  #g
NET "Segm<8>"   LOC = "C17"  ;  #dp
#
#NET "Btn<0>"      LOC = "B18"  ;
#NET "Btn<1>"      LOC = "D18"  ;
#NET "Btn<2>"      LOC = "E18"  ;
#NET "Btn<3>"      LOC = "H13"  ;
#
#NET "Ld<0>"    LOC = "J14"  ;
#NET "Ld<1>"    LOC = "J15"  ;
#NET "Ld<2>"    LOC = "K15"  ;
#NET "Ld<3>"    LOC = "K14"  ;
#NET "Ld<4>"    LOC = "E17"  ;
#NET "Ld<5>"    LOC = "P15"  ;
#NET "Ld<6>"    LOC = "F4"   ;
#NET "Ld<7>"    LOC = "R4"   ;
#
NET "Sw<0>"     LOC = "G18"  ;
NET "Sw<1>"     LOC = "H18"  ;
#NET "Sw<2>"    LOC = "K18"  ;
#NET "Sw<3>"    LOC = "K17"  ;
#NET "Sw<4>"    LOC = "L14"  ;
#NET "Sw<5>"    LOC = "L13"  ;
#NET "Sw<6>"    LOC = "N17"  ;
#NET "Sw<7>"    LOC = "R17"  ;

NET "Clk"       LOC = "B8"   ; # Clk 50 MHz
```