

Lab 3: Display-multiplexing og quadrature-dekoder

E2DEL – Team 2.1 (endelig)



Team 2.1

Casper Lund Bruun
201809254

Daniel Korsgaard Vinther
Wolf
201701971

Janus Bo Andersen
JA67494

Review: Team 1.1

Morten Jacobsen Dahl
19842173

Tommy Kjær Jensen
201806167

Verner Søndergaard
201809352

Indholdsfortegnelse

1	Indledning	4
2	Review	4
3	Forkortelser og definitioner	5
4	Præ-lab og teori (display-multiplexer)	6
5	VHDL-design (display-multiplexer)	7
6	Testbench og simulering (display-multiplexer)	13
7	Implementering på FPGA (display-multiplexer)	19
8	Præ-lab og teori (quadrature-dekoder)	20
9	VHDL-design (quadrature-dekoder)	22
10	Test-bench og simulering (quadrature-dekoder)	25
11	Implementering på FPGA (quadrature-dekoder)	27
12	Fejlkilder og måleusikkerheder	28
13	Diskussion og konklusion	28
14	Referencer	29
15	Bilag	30
15.1	Bilag: Brug af 7-segmentdisplays	30
15.2	Bilag: Hardwaredesign og komponentbegrænsninger	31
15.3	Bilag: 7-segment-dekoder	33

Indholdsfortegnelse over vedlagte filer (zip)

Display-multiplexer og 3x7-tæller:

Filnavn	Beskrivelse
top_lab3.vhd	Top-level mapping til modulær brug af designs; instantierer de tre nedenstående moduler.
mux_hex_decoder.vhd	Tidsmultiplexer 3 displays og udfører hex-til-7-segment-dekodning
mod_m_counter.vhd	Mod-m-tæller til pre-scaling fra 100 MHz til 10 Hz (listing 4.10 fra [1])
univ_bin_counter.vhd	Universal binær-tæller til 12-bit 10 Hz tæller (listing 4.9 fra [1])
tb_mux_lab3.vhd	Testbench til integrationstest af ovenstående (på top-niveau)
arty_master.xcd	Design-constraints til forbindelse af top-level signaler med fysiske komponenter på Arty A7-boardet.

Quadrature-dekoder og 3x7-tæller:

Filnavn	Beskrivelse
top_lab3_extra.vhd	Top-level mapping til modulær brug af designs; instantierer de fire nedenstående moduler.
quadrature_decoder.vhd	Dekoder signaler fra roterende enkoder.
mux_hex_decoder.vhd	Tidsmultiplexer 3 displays og udfører hex-til-7-segment-dekodning
univ_bin_counter.vhd	Universal binær-tæller 12-bit system (listing 4.9 fra [1])
debounce.vhd	Debouncing af signaler fra roterende enkoder (listing 6.1 fra [1])
tb_lab3_extra.vhd	Testbench til integrationstest af ovenstående (på top-niveau)
arty_master.xcd	Design-constraints til forbindelse af top-level signaler med fysiske komponenter på Arty A7-boardet.

1 Indledning

Formålet med denne labøvelse er at implementere display-multiplexing på 3 eksterne 7-seg.-displays samt at implementere en quadrature-dekoder (behandling af forskudte firkantsignaler fra en roterende enkoder).

Display-multiplexing (obligatorisk):

Den obligatoriske opgave ligger i direkte forlængelse af ekstraopgaven fra lab 2, hvor værdien fra en prescalet 2 Hz 4-bit-tæller blev vist på et enkelt eksternt display.

Nu skal der benyttes 3 displays fra 4x7-boardet, hvilket kræver multiplexing med 33% duty cycle per display og optælling på en 12-bit-tæller. Tælleren bør inkrementeres hurtigere end 2 Hz.

Quadrature-dekoder (ekstra):

Som ekstraopgave ønskes en ekstern roterende enkoder (rotary encoder) tilkoblet. Hardware skal implementeres i FPGA'en, således at signalerne fra enkoderen (kvadraturen) kan afkodes og benyttes til at inkrementere/dekrementere en tæller. Den eksterne enkoder og den interne tæller skal altså være synkroniseret ift. både rotationsmængde og rotationsretning. Tællerens værdi skal vises på de 3 displays.

Den obligatoriske opgave findes i rapportens første del (display-multiplexer). Ekstraopgaven er indsat i forlængelse af den obligatoriske del. Konklusion og fejlkilder for hele labøvelsen findes til sidst i rapporten.

Alle overvejelser omkring tilkobling og benyttelse af 7-segment-displays, komponentbegrænsninger osv. blev behandlet i lab 2, og findes derfor udelukkende i bilag i denne rapport.

2 Review

Reviewkommentarer fra team 1.1 var:

- Der blev ikke fundet nogen fejl.
- Der blev dog kun leveret den obligatoriske opgave.
- De havde ønsket at modtage alle kodefilerne.

Reviewet har ikke udløst nogen ændringer i rapporten. I mellemtiden mellem review og aflevering er ekstraopgaven blevet udarbejdet.

3 Forkortelser og definitioner

Term	Definition
'0' eller falsk	Det digitale signal '0' el. LAV defineres og forstås medmindre andet eksplicit er nævnt som værende 0 V (eller FALSK) eller OFF.
'1' eller sand	Det digitale signal '1' el. HØJ defineres og forstås medmindre andet eksplicit er nævnt som værende 3,3 V (ELLER SAND) eller ON.
DFF	D Flip-Flop. Clocket logic (hukommelse).
Register	Samling af DFF'er, kan have vilkårlig bitbredde.
EN	Enable-signal i digital logik, især DFF/register, som aktiverer logikken til at kunne sample et inputsignal.
D	Data-input på en DFF/register. Samples typisk på stigende clock-flanke og når EN er høj.
Q	Output på en DFF/register.
R	Reset.
Tcl	Scripting-sprog, der benyttes i Vivado til at udføre forskellige kommandoer.
Waveforms	Simuleringsbilledet i Vivado, der viser hvorledes et kredsløb reagerer.
Hex	Hexadecimal (16-talssystem, base-16).
$(..)_{16}$, $(..)_{10}$, $(..)_2$	Subscript angiver basen (radix) for et tal. Fx $(A)_{16} = (10)_{10} = (1010)_2$
"0000" vs. '0'	Værdier angivet alene i dobbelt anførselstegn er altid binærværdien i en std_logic_vector. Værdier i enkelt anførselstegn er altid binærværdien i en std_logic.
sig1 vs. sig2	Vektorer er formatteret med fed skrifttype i modsætning til skalarer.
V_{IL} , V_{IH}	Input-low og input-high: Spændingsintervaller for inputpins på Arty A7, indenfor hvilke input-signaler vil blive opfattet som hhv. lave og høje.

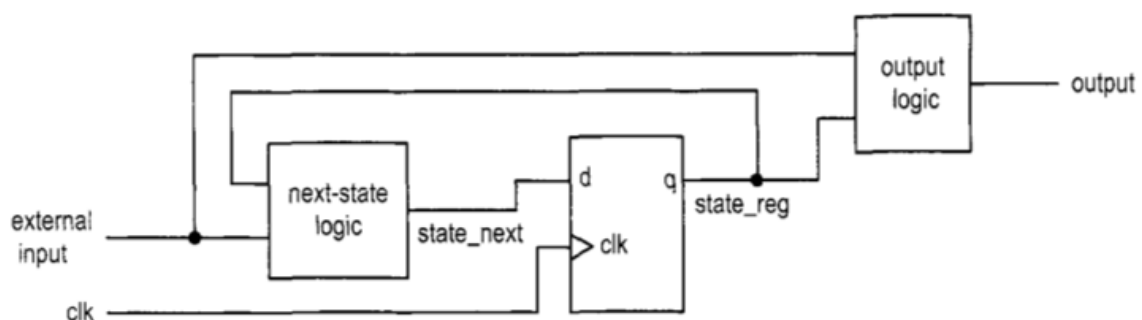
4 Præ-lab og teori (display-multiplexer)

Til det ønskede 3x7-tællersystem skal følgende logik-moduler (ingredienser) implementeres i hardware:

Tabel 1: Moduler til 3x7-tæller

Modul	Beskrivelse
Display-multiplexer og hex-til-7-segment display-dekoder	<ul style="list-style-type: none"> Multiplexer 3 displays med 33% duty cycle: Skiftevis ét display tændt, og skiftevis sendes forskellige 7-seg.-mønstre til bussen på 4x7-boardet. Mux'en har en intern 18-bits-tæller. De to MSBs herfra går i select lines til at mux'e; de øvrige 16 bits fungerer som prescaler og styrer derfor refresh rate på displays: Refresh rate med 16-bits prescaler derfor: $100 \text{ MHz} / 2^{16} \approx 1600 \text{ Hz}$ [1, s. 77-78]. Mux'en vælger skiftevis de relevante 4 bits ud af de i alt 12-bits fra tælleren, som skal sendes til 7-seg.-dekoderen og vises på det tændte display. Mux'en udvælger også kommasegmentets tilstand ud fra en 4-bits vektor. Kode til 7-seg.-dekoderen er identisk med dekoder fra Lab 2. Dekoderen konverterer "concurrent" et 4-bitsignal $(0-F)_{16}$ til et 8-bit 7-seg.-mønster.
Prescaler til tæller (mod-m, listing 4.10 i [1])	<ul style="list-style-type: none"> Skalerer Arty boardets 100 MHz clock til 10 Hz (så man stadig kan se, hvad der sker). For at ramme præcis 10 Hz benyttes en mod-M counter, der tæller til $10 \cdot 10^6$ på 100 MHz clock'en, hvorved et tick udsendes med 10 Hz. Det kræver som minimum $\log_2(10 \cdot 10^6) = 24$ bits i prescaleren.
12-bit-tæller (listing 4.9 i [1])	<ul style="list-style-type: none"> Tæller $(0-15)_{10}$ dvs. $(0-F)_{16}$ per 7-seg.-disp., så samlet værdi $(000-FFF)_{16}$ el. $(0-4095)_{10}$. Tick fra prescaler benyttes som "enable"-signal, så tælleren avancerer med 10 Hz.
Top	<ul style="list-style-type: none"> Top-modulet instantierer og forbinder de 3 nævnte moduler. Mapper hardware pins og clock (oscillator) på boardet til signaler i FPGA'en.

Gennemgående i rapporten bruges et synkront designmønster, hvilket generisk er beskrevet i figuren:

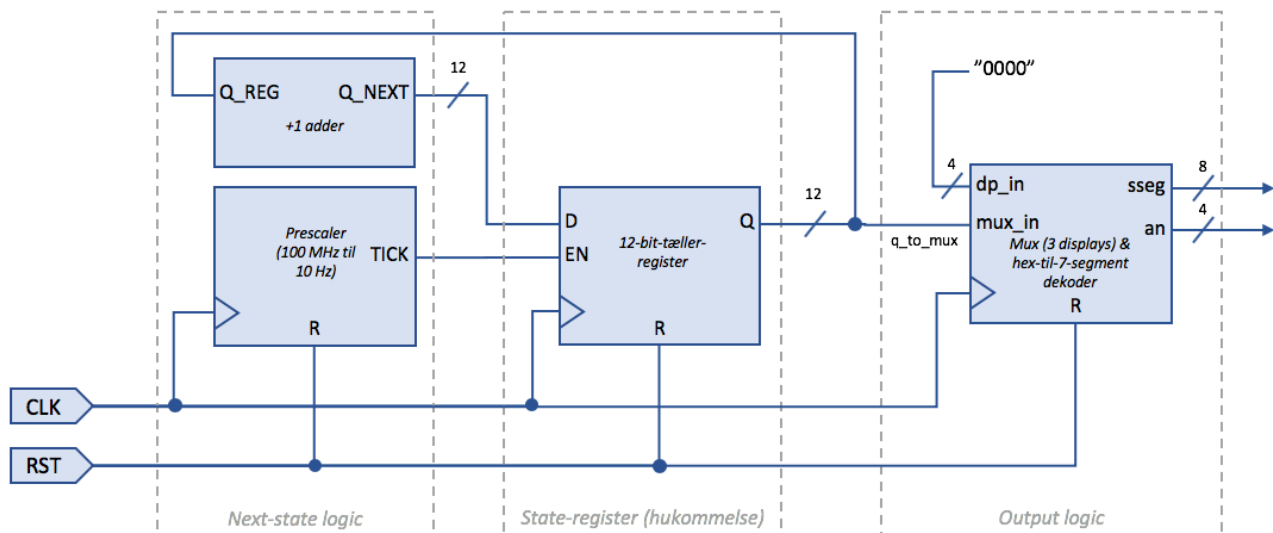


Figur 1: Synkront designmønster [1, listing 4.2]

I VHDL-afsnittets to første figurer vises systemets opbygning, som netop følger dette mønster.

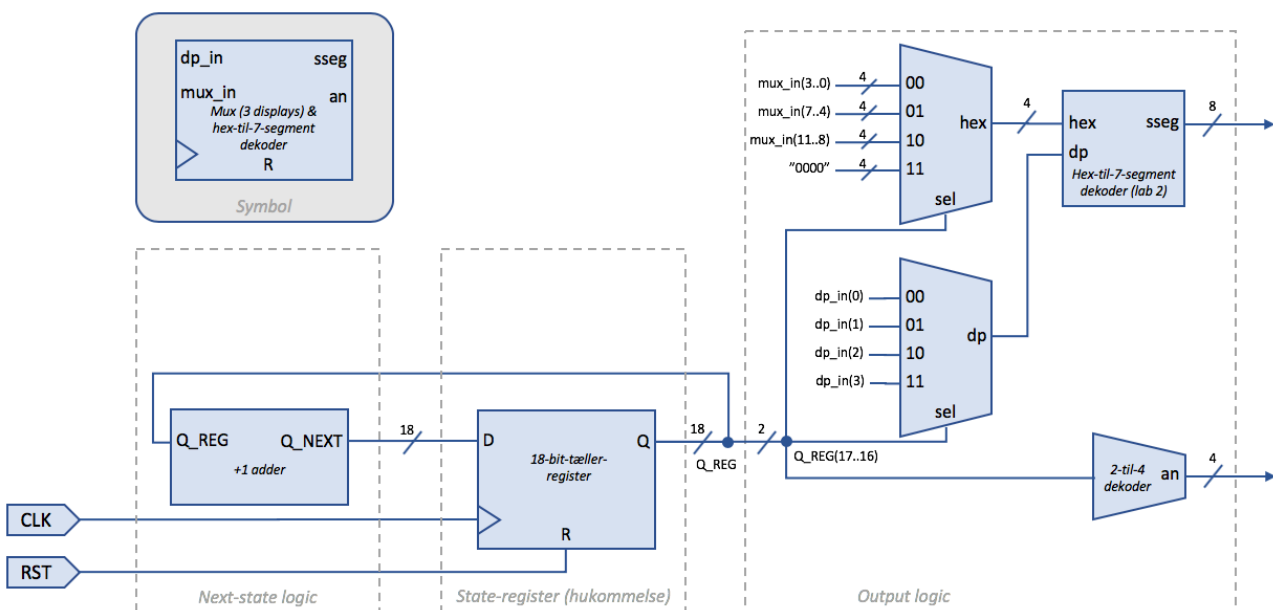
5 VHDL-design (display-multiplexer)

Systemet designes efter synkront designmønster. Det betyder, at både tællersystemet og multiplexeren er designet efter dette mønster. Først ses designet for det overordnede 3x7-tællersystem:



Figur 2: Synkront designmønster for 3x7-tælleren

Display-mux'en sidder som output logic i 3x7-systemet. Dens opbygning følger samme designmønster, og er udspecificeret i følgende figur:



Figur 3: Synkront designmønster for display-multiplexeren

Systemet implementeres via **top-modulet**. I det følgende behandles topmodulets entity og arkitektur, som i struktur stemmer overens med ovenstående blokdiagrammer.

Design-ønsker til 3x7-systemets **entity**:

- Inputs:
 - Clock-signal (clk) fra boardet, som driver clock i alle tællermoduler og mux'en.
 - Reset (rst, aktiv høj), der kan nulstille alle tællermoduler. Hvis designet skulle være en del af et større design, burde det tilsikres, at reset er synkron (clear).
- Outputs:
 - Et 8-bit-signal (**sseg**) til bussen på 4x7-boardet (går til alle 4 displays).
 - Et 4-bit-signal (**an**) til transistorerne, som tidsmultiplexes og tænder/slukker enkelte displays.
- Generic mapping:
 - Prescaler generic-mappes så tællerens frekvens kan ændres, fx ifbm. testbenches.
 - Antal bits til mux'ens interne tæller generic-mappes, så refresh rate kan styres, fx ifbm. tests.
 - Decimal-point generic-mappes, så et eller flere kommaer kan placeres.

VHDL-koden for **top entity** afspejler disse design-ønsker:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--create ports that map to hardware pins on the A7
entity top is
  generic(
    SET_PRESCALE_BITS: integer := 24;      -- bits to count to 10*10**6
    SET_PRESCALE_VAL : integer := 10*10**6; -- mod M: to divide 100 MHz to 10Hz
    DP                : std_logic_vector(3 downto 0) := "0000"; -- no decimal points
    SET_REFRESH_RATE_BITS : integer := 18    -- use only the 2 MSBs, updating at 100MHz/2^16
  );
  port (
    sseg : out std_logic_vector(7 downto 0); -- 8 bits: 7 segments and 1 dp
    an   : out std_logic_vector(3 downto 0); -- 4 sigs to enable transistors
    clk  : in std_logic;                    -- raw clock from crystal oscillator
    rst  : in std_logic;                    -- hard reset
  );
end top;
```

Figur 4: Entity til 3x7-tællersystemet

Design-ønsker til **arkitektur**:

- Der skal instantieres tre moduler til systemet:
 - Mod-m-tæller: 10 Hz prescaler (listing 4.10 fra [1], valideret i øvelsestimer)
 - Universel binær-tæller: 12-bit-tæller (listing 4.9 fra [1], valideret i øvelsestimer)
 - Mux og hex-til-7-seg.-dekoder i ét modul (mux er ny; dekoder genbrugt fra lab 2)
- Max-tick fra prescaler og enable-signal på binærtæller forbindes via et signal (tick).
- Output fra 12-bit tælleren (**Q**) forbindes til input på multiplexeren (**mux_in**) via et signal (**q_to_mux**).
- Mux'en vælger 4 af de 12 bit, og internt i mux'en forbindes 4-bit-signalet til 7-seg.-dekoderen (**hex**).

VHDL-koden for arkitekturen **top_arch** defineres som følger. Først nyttige signaler og konstanter:

```
architecture top_arch of top is
  constant CNT_BITS : integer := 12;           -- 12 bits for the counter (000-FFF)
  constant OFF_MODE : std_logic := '0';
  constant ON_MODE  : std_logic := '1';

  --Intermediate signals
  signal tick       : std_logic;               -- maps max_tick to enable
  signal q_to_mux   : std_logic_vector(CNT_BITS-1 downto 0); -- connects 12-bit counter to mux
```

Figur 5: Arkitektur til 3x7-tællersystemet (signaler og konstanter)

Dernæst instantieres og forbindes de nævnte moduler:

```
begin
  --prescaler (mod-m counter, Chu listing 4.10)
  prescaler : entity work.mod_m_counter
    generic map(
      N => SET_PRESCALE_BITS,
      M => SET_PRESCALE_VAL
    )
    port map(
      clk    => clk, -- system clock
      reset  => rst, -- active high
      max_tick => tick, -- send max_tick -> tick -> en
      q      => open -- don't need these, just discard
    );

  --binary counter (universal binary counter, Chu listing 4.9)
  counter : entity work.univ_bin_counter
    generic map(
      N => CNT_BITS -- 12-bit (3 hex digits)
    )
    port map(
      clk    => clk, -- system clock
      reset  => rst, -- active high
      syn_clr => OFF_MODE, -- active high
      load   => OFF_MODE, -- active high
      en     => tick, -- enable is on at 10 Hz
      up     => ON_MODE, -- always increment
      d      => (others => '0'),
      max_tick => open,
      min_tick => open,
      q      => q_to_mux -- get 12-bit count here
    );

  -- Mux'ed 7-segment decoder (lab 3 custom)
  -- Does not yet allow a different counter width than 12 bits
  mux_hex_decoder : entity work.mux_hex_decoder
    generic map(
      REFRESH_RATE_BITS => SET_REFRESH_RATE_BITS
    )
    port map(
      clk    => clk,
      reset  => rst,
      mux_in => q_to_mux, -- 12 bits in (3 hex digits or 3 BCDs that will be mux'ed)
      dp_in  => DP,
      sseg   => sseg, -- 7-segment patterns out (incl. dp)
      an     => an -- transistor mux'ing
    );
end top_arch;
```

Figur 6: Arkitektur til 3x7-tællersystemet (instantiering og forbindelser)

Display-mux og 7-seg.-dekoder er beskrevet i ét modul, sammensat af en ny multiplexer (baseret på ideer fra listings 4.12 og 4.14 i [1]) samt en dekoder, som blev dokumenteret i lab 2 (med ideer fra listing 3.14 i [1]).

Design-ønsker til **display-multiplexer (3 displays)** – dekoderen diskuteres kun i appendiks:

- Entity:
 - Fire input-porte:
 - 12-bit signalet fra tælleren (**mux_in**)
 - 4-bit vektor med kommasegmenternes tilstand (**dp_in**).
 - Clock og reset.
 - Outputsignaler er: 8 bits til bussen på 4x7-boardet (**sseg**) og de 4 bits til transistorer (**an**).
 - Generic-mapping af refresh-rate til brug ved fx testbenches.
- Arkitektur:
 - Synkront/sekventielt system med separat register og next-state logic.
 - Skal omfatte 3 interne mux'er: værdi (**hex**), komma (**dp**) og transistor (**an**).
 - De 3 interne mux'er kan nemmest bygges i én sekventiel proces.

Først sættes entity og signaler i arkitekturen op:

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

entity mux_hex_decoder is
  generic (
    REFRESH_RATE_BITS : integer := 18 -- use only the 2 MSBs, updating at 100MHz/2^16
  );
  port(
    clk, reset : in std_logic;
    mux_in      : in std_logic_vector(11 downto 0); -- 3x 4-bit dig. (12 bit), either hex or BCD
    dp_in       : in std_logic_vector(3 downto 0); -- decimal point
    sseg        : out std_logic_vector(7 downto 0); -- 7seg pattern signals (incl. dp)
    an          : out std_logic_vector(3 downto 0)
  );
end mux_hex_decoder;

architecture mux_arch of mux_hex_decoder is
  -- For 7-seg decoding:
  signal hex : std_logic_vector(3 downto 0); -- 4 bits to be decoded to patterns
  signal dp  : std_logic; -- single bit decimal point signal

  -- For multiplexing:
  constant N : integer := REFRESH_RATE_BITS;
  signal q_reg : unsigned(N-1 downto 0); -- 18-bit counter
  signal q_next : unsigned(N-1 downto 0); -- 18-bit next-state
  signal sel : std_logic_vector(1 downto 0); -- 2 MSBs are placed for disp sel
```

Figur 7: Display-multiplexer (entity og signaler i arkitekturen)

Følgende kode viser den synkrone logik for display-mux'en samt concurrent logik for dekoderen:

```

begin
  -- Register (18-bit)
  process(clk, reset)
  begin
    if reset = '1' then
      q_reg <= (others => '0');
    elsif rising_edge(clk) then
      q_reg <= q_next;
    end if;
  end process;

  -- Next-state logic (18-bit)
  q_next <= q_reg + 1;

  -- Selector for mux (casting 2 MSBs into sel)
  sel <= std_logic_vector(q_reg(N-1 downto N-2));

  -- Mux process - 3 h/w multiplexers in one, like in report figures
  process(sel, mux_in, dp_in)
  begin
    case sel is
      when "00" =>
        an <= "0001";
        hex <= mux_in(3 downto 0);
        dp <= dp_in(0);
      when "01" =>
        an <= "0010";
        hex <= mux_in(7 downto 4);
        dp <= dp_in(1);
      when "10" =>
        an <= "0100";
        hex <= mux_in(11 downto 8);
        dp <= dp_in(2);
      when others =>
        an <= "0000"; -- all displays off
        hex <= "0000";
        dp <= '0';
    end case;
  end process;

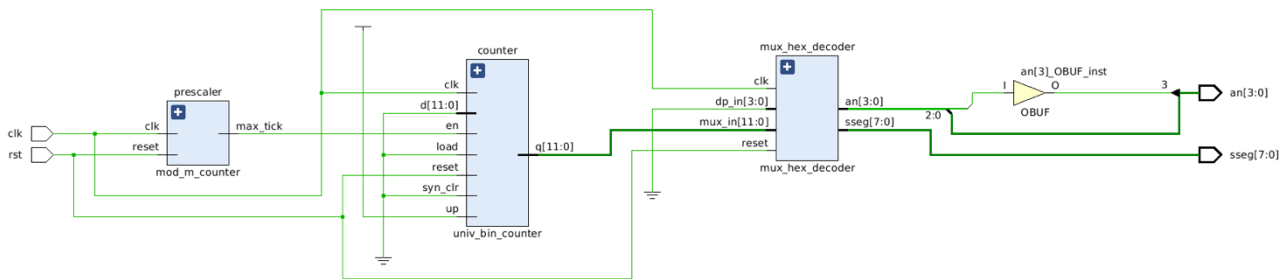
  -- Output logic (decoder routing circuit)
  -- Note: CC disp on board are active high. Avoids a latch by full assignment.
  with hex select
    sseg <=
      -- dp      gfedcba segm., binary counter value
      dp & "0111111" when "0000", -- 0
      dp & "0000110" when "0001", -- 1
      dp & "1011011" when "0010", -- 2
      dp & "1001111" when "0011", -- 3
      dp & "1100110" when "0100", -- 4
      dp & "1101101" when "0101", -- 5
      dp & "1111101" when "0110", -- 6
      dp & "0000111" when "0111", -- 7
      dp & "1111111" when "1000", -- 8
      dp & "1101111" when "1001", -- 9
      dp & "1110111" when "1010", -- A
      dp & "1111100" when "1011", -- B
      dp & "0111001" when "1100", -- C
      dp & "1011110" when "1101", -- D
      dp & "1111001" when "1110", -- E
      dp & "1110001" when others; -- F
end mux_arch;

```

Figur 8: Display-multiplexer (arkitektur inkl. dekoder)

De to tællermoduler (mod-m-tæller fra listing 4.10 og universel binærtæller fra listing 4.9) er ikke gengivet i denne rapport, da de er identiske med de nævnte listings i [1], og er valideret i øvelsestimerne i E2DEL.

Figur 9 nedenfor viser hele systemet. Figuren er taget fra Vivados "Elaborated design", og modsvarer designet i starten af afsnittet. Der optræder en buffer i outputtet på **an(3)**, da dette signal altid er '0' (stel).



Figur 9: Blokdiagram (elaborated design) af systemet

Constraints vises nedenfor. En generisk fil er benyttet som skabelon fra [8]. Den korrekte mapping er udført vha. detaljer fra appendiks, som allerede diskuteret i lab 2. Display-mux'ens transistorsignal (**an**) forbindes direkte til de 4 transistorer på 4x7-boardet. Output fra 7-seg.-dekoderen i display-mux'en forbindes direkte til bussen på 4x7-boardet.

```

=====
# Clock signal
=====
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]

=====
#Buttons
=====
set_property -dict {PACKAGE_PIN D9 IOSTANDARD LVCMOS33} [get_ports {rst}]

=====
#7-segment display stuff
=====
## 7 Segments + decimal point
##Pmod Header JB (high-speed w/o serial protection resistor)
##Pmod Header JC (high-speed w/o serial protection resistor)

set_property -dict { PACKAGE_PIN E15 IOSTANDARD LVCMOS33 } [get_ports {sseg[6]}}; #g
set_property -dict { PACKAGE_PIN E16 IOSTANDARD LVCMOS33 } [get_ports {sseg[5]}}; #f
set_property -dict { PACKAGE_PIN D15 IOSTANDARD LVCMOS33 } [get_ports {sseg[4]}}; #e
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports {sseg[3]}}; #d
set_property -dict { PACKAGE_PIN V11 IOSTANDARD LVCMOS33 } [get_ports {sseg[2]}}; #c
set_property -dict { PACKAGE_PIN U13 IOSTANDARD LVCMOS33 } [get_ports {sseg[1]}}; #b
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports {sseg[0]}}; #a
set_property -dict { PACKAGE_PIN V12 IOSTANDARD LVCMOS33 } [get_ports {sseg[7]}}; #dp

## Drivers (transistors) for time multiplexing
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports {an[0]}};
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports {an[1]}};
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports {an[2]}};
set_property -dict { PACKAGE_PIN C15 IOSTANDARD LVCMOS33 } [get_ports {an[3]}};

```

Figur 10: Design-constraints til Arty A7-board

6 Testbench og simulering (display-multiplexer)

Testbenchen for systemet skal **validere**, at:

1. Resetknappen virker til at nulstille systemet. Reset skal forhindre optælling.
2. Display-mux'en skifter mellem displays efter det ønskede antal clock flanker.
3. Display-mux'en skifter til det korrekte 7-seg.-mønster (alt efter tællerens værdi), samhörigt med skift af aktivt display.

For at få en hurtig simulering er både tæller-prescalerens frekvens og mux'ens refresh rate øget via generic-mapping. Dette er i tråd med overvejelser fra [1, s. 83]. **Testbenchindstillinger** er sat så:

- Display-mux'en skal skifte display hver 4. clock flanke, dvs. køre 25 MHz.
 - Hvert af de 3 benyttede displays er tændt i 4×10 ns, efterfulgt af 40 ns med alle displays slukket.
 - Duty cycle per display bliver 33% af samlet on-tid (men "kun" 25% af realtid).
- Tæller-prescaler skal sende tick hver 16. clock flanke (mod-16).
- Tælleren inkrementerer med 6,25 MHz (hver 16×10 ns = 160 ns). Tællerens værdier skal matche med mønstre sendt fra display-mux til 3x7-displays.
- Mux'en tager en fuld runde med samme frekvens som prescaleren. Dette letter testen betydeligt da tællerværdi så er stabil i en hel runde for mux'en. (Praktisk, men er ikke krævet i designet).

Hvis testbenchen opfylder ovenstående krav, med ovenstående indstillinger, vil den også køre korrekt i realtid med de designede normalindstillinger.

Testbenchen indeholder følgende ingredienser:

- Instantieret og generic-mappet top-modul (hele 3x7-systemet).
- Array med specs for 7-seg.-mønstre ("facitliste" ses i Tabel 2).
- Clockproces (10 ns, 100 MHz).
- Emuleret tællersystem, så en *testbruger* visuelt kan sammenligne værdier i waveforms:
 - Emuleret prescaler (mod-16), der inkrementer på clock og udsender ticks.
 - Emuleret tæller (12-bit), der *kun* inkrementerer ved stigende clock flanke OG tick='1'.

Facitlisten i nedenstående tabel benyttes til at oversætte hex-værdier (som diskuteret i lab 2 og i appendiks):

Tabel 2: "Facitliste" til oversættelse af tællerværdi til 7-seg.-mønstre

Hex-cifferværdi (4 bits)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
7-seg.-mønsterværdi (8-bits)	3F	06	5B	4F	66	6D	7D	07	7F	6F	77	7C	39	5E	79	71

Den sekventielle testproces, som systemet skal undergå, er som følger:

1. Resetknappen stimuleres i 16 clock flanker. Herefter bekræftes at alle tællere viser $(000)_{16}$.
2. Tælleren følges igennem værdierne $(000)_{16}$ til $(FFF)_{16}$, og hver gang display-mux'en skifter, bekræftes det, at den sender et korrekt mønster (**sseg**) og aktiverer den korrekte transistor (**an**).

Metoden benyttet i *hele* testbenchen er: (a) vent et antal stigende clock flanker, (b) vent i settling time, (c) observér resultat og sammenlign med forventning.

En automatisk testbench er konstrueret til at køre ovenstående igennem. Testbenchen er baseret på ideer fra listing 4.11 fra [1].

Assert benyttes til at "faile" testen, hvis specs ikke er opfyldt. Tcl-loggen skal derfor ses efter for fejl efter endt simulering. Waveforms skal naturligvis ses igennem alligevel.

Den fulde kode til testbenchen er gengivet og beskrevet i de følgende figurer.

Først sættes entity og arkitektur op. Testindstillinger sættes som konstanter i arkitekturen:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity tb_lab3 is
end tb_lab3;

architecture arch of tb_lab3 is

-- Test settings
constant clock_period      : time := 10 ns;      -- clk period for 100 MHz
constant settling_time      : time := 100 ps;    -- time for signals to register
constant PRESCALE_VAL_TEST : integer := 16;      -- prescaler tick on every 16 clocks
constant PRESCALE_BIT_TEST : integer := 4;
constant REFRESH_RATE_BITS_TEST : integer := 4; -- switch display every (X-2)^2 => 4 clocks ticks

-- Dimension constants
constant SSEG_BITS: integer := 8; --7 segs and 1 dp

-- Intermediate simulation signals
signal clk      : std_logic; -- simulated clock signal
signal rst      : std_logic; -- simulated reset
signal sseg     : std_logic_vector(SSEG_BITS-1 downto 0); -- To pick up SSEG
signal an       : std_logic_vector(3 downto 0);
signal q_reg, q_next: unsigned(11 downto 0); -- Emulate 12-bit system counter
signal p_reg, p_next: unsigned(PRESCALE_BIT_TEST-1 downto 0); -- Emulate prescaler
signal tick     : std_logic;

```

Figur 11: Testbench til 3x7-tællersystemet (entity og signaler)

7-segmentmønstre tjekkes op imod følgende "facitliste", implementeret i et array. Fx giver `sseg_spec(0)` det ønskede mønster, der svarer til et 0 på displayet (dvs. der skal sendes 3F til displayet, hvis det skal vise et 0).

```
-- Specifications
-- Array of required SSEG behaviors: output for different hex values (0-15)
type sseg_array is array (0 to 15) of std_logic_vector(SSEG_BITS-1 downto 0);
signal sseg_spec : sseg_array := (
  --dp gfedcba <- segments on the 7 seg.
  "00111111", -- 0 (3F)
  "00000110", -- 1 (06)
  "01011011", -- 2 (5B)
  "01001111", -- 3 (4F)
  "01100110", -- 4 (66)
  "01101101", -- 5 (6D)
  "01111101", -- 6 (7D)
  "00000111", -- 7 (07)
  "01111111", -- 8 (7F)
  "01101111", -- 9 (6F)
  "01110111", -- A (77)
  "01111100", -- B (7C)
  "00111001", -- C (39)
  "01011110", -- D (5E)
  "01111001", -- E (79)
  "01110001", -- F (71)
);
```

Figur 12: Specifikationer af krævet 7-seg.-mønster

Det fulde system, dvs. hele top-modulet, instantieres og generic-mappes. Desuden startes en clock-proces:

```
begin
  --Instantiate a full device. Use the whole top structure with gen. prescaler
  dut1 : entity work.top
    generic map(
      SET_PRESCALE_BITS => PRESCALE_BIT_TEST,
      SET_PRESCALE_VAL  => PRESCALE_VAL_TEST,
      SET_REFRESH_RATE_BITS => REFRESH_RATE_BITS_TEST
    )
    port map(
      sseg => sseg, --output, 7-seg. pattern
      an  => an,   --output, muxing transistors
      clk => clk,  --input, simulated clock
      rst => rst   --input, simulated reset
    );

  --Generate the clock
  clock: process
  begin
    clk <= '0';
    wait for clock_period / 2;
    clk <= '1';
    wait for clock_period / 2;
  end process clock;
```

Figur 13: Testbench (instantiering af top, start af clock)

Dernæst emuleres et tællersystem bestående af en prescaler og 12-bit-tæller. De benyttes ikke til at stimulere hardware, men med disse kan testbrugeren i waveforms følge med i forventede værdier i hardwaremodul. Sekventiel logik fra listing 4.9 og 4.10 i [1] er genbrugt, blot let modificeret.

```
--Emulate the prescaler
-- Register (4-bit)
process(clk, rst)
begin
  if (rst = '1') then
    p_reg <= (others => '0');
  elsif rising_edge(clk) then
    p_reg <= p_next;
  end if;
end process;

-- Next-state logic (4-bit) like mod-M
p_next <= (others => '0') when p_reg = (PRESCALE_VAL_TEST - 1) else p_reg + 1;

-- Output logic
tick <= '1' when p_reg = (PRESCALE_VAL_TEST - 1) else '0';

--Emulate the system counter
-- Register (12-bit)
process(clk, rst, tick)
begin
  if rst = '1' then
    q_reg <= (others => '0');
  elsif rising_edge(clk) and (tick = '1') then -- Prescaled with EN signal
    q_reg <= q_next;
  end if;
end process;

-- Next-state logic (12-bit)
q_next <= q_reg + 1;
```

Figur 14: Testbench (emulering af prescaler og tæller)

Derefter begyndes den egentlige testproces ift. specifikationen. Først testes resetknappen:

```
--Test module behavior versus specifications
test: process
begin
  --Test the reset button for a full round of the prescaler
  rst <= '1';
  for j in 1 to PRESCALE_VAL_TEST loop
    wait until rising_edge(clk);
  end loop;
  rst <= '0';

  wait for settling_time;

  assert sseg = sseg_spec(0) and an = "0001"
    report "Reset is not resetting internal counters"
    severity failure;
```

Figur 15: Testbench (test af reset)

Endelig køres den fulde test, som følger tæller og outputs igennem værdierne $(000)_{16}$ til $(FFF)_{16}$. Dette er implementeret med 3 nestede loops. Hvert loop svarer til et ciffer på et display. Således køres 12-bit-tælleren igennem alle mulige værdier, og display-mux'en dermed igennem alle mulige kombinationer.

```
-- run for all 3 digits of the 3-hex digit counter (the 3x 7-seg displays)
for digit3 in 0 to 15 loop
  for digit2 in 0 to 15 loop
    for digit1 in 0 to 15 loop

      -- check display #1 (right-most on 4x7-board)
      assert sseg = sseg_spec(digit1) and an = "0001"
      report "Failing test: Display 1. Loop #" & integer'image(digit1) & "."
      severity failure;

      -- wait for next Mux period (4 clocks + settling time)
      for c in 1 to 4 loop
        wait until rising_edge(clk);
      end loop;
      wait for settling_time;

      -- check display #2
      assert sseg = sseg_spec(digit2) and an = "0010"
      report "Failing test: Display 2. Loop #" & integer'image(digit2) & "."
      severity failure;

      -- wait for next Mux period (4 clocks + settling time)
      for c in 1 to 4 loop
        wait until rising_edge(clk);
      end loop;
      wait for settling_time;

      -- check display #3
      assert sseg = sseg_spec(digit3) and an = "0100"
      report "Failing test: Display 3. Loop #" & integer'image(digit3) & "."
      severity failure;

      -- wait for next Mux period (4 clocks + settling time)
      for c in 1 to 4 loop
        wait until rising_edge(clk);
      end loop;
      wait for settling_time;

      -- check display #4 (left-most on 4x7-board) -> must always be zero and off
      assert sseg = sseg_spec(0) and an = "0000"
      report "Failing test: Display 4. Loop #" & integer'image(0) & "."
      severity failure;

      -- wait for next Mux period (4 clocks + settling time) -> to restart loop
      for c in 1 to 4 loop
        wait until rising_edge(clk);
      end loop;
      wait for settling_time;

    end loop;
  end loop;
end loop;

--Terminate simulation OK if we reached this point
assert false
report "Simulation OK!"
severity failure;

end process test;

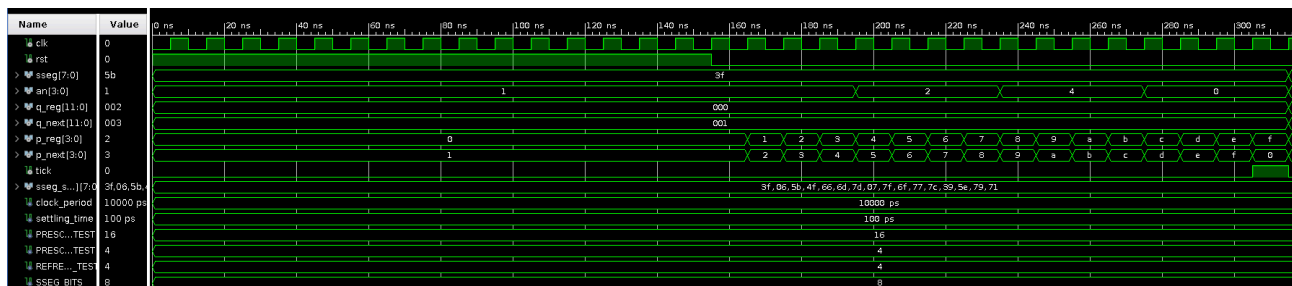
end arch;
```

Figur 16: Testbench (testproces med 3 nestede loops, til hvert ciffer i 3x7-systemet)

Efter ovenstående testbench er kørt, kigges Tcl-loggen igennem efter beskeder. Waveforms analyseres.

Designet bestod testbenchen: Ingen fejl i Tcl-loggen. Analyse af waveforms er gengivet i de følgende figurer. Resultatet er som forventet.

Første figur viser resetknappen være stimuleret indtil 16. clock flanke. Der skal zoomes ind for at se detaljer.



Figur 17: Waveforms af begyndelse af testproces

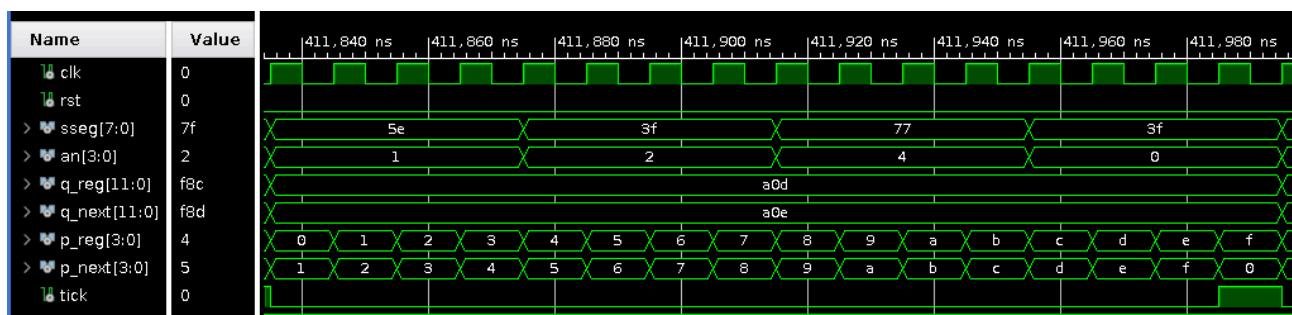
Figuren viser, at systemet forbliver i en multitilstand, indtil reset slippes, og at optælling og display-multiplexing først begynder efter reset er sluppet.

Dette ses specifikt ved at display-mux'en fastholder display 1 ($an = "0001" = 1$) som tændt indtil 4 clock flanker efter reset er sluppet, og at **sseg** giver et 0-mønster (3f svarer til symbolet 0).

Det ses, at mux'en skifter fra display 1 ($an = "0001"$) til display 2 ($an = "0010" = 2$) ved 195 ns, og at mux'en derefter "cykler" videre igennem displays på hver 4. clock flanke: Display 3 ($an = "0100" = 4$) aktiveres ved 235 ns. Alle displays slukkes ved 275 ns ($an = "0000"$). Osv.

Det er vigtigt at observere, at **sseg** her fastholdes på et 0-mønster selvom mux'en skifter aktivt display; værdien af tælleren er nemlig $(000)_{16}$ og derfor skal alle displays vise et 0-mønster.

Næste figur viser et udsnit midt i simulationen, hvor tællerens værdi (forventes) at være $(A0D)_{16} = (2573)_{10}$. Display 1 (mindst betydende ciffer) skal altså vise (D)₁₆. Ifølge facitlisten er mønster for (D)₁₆ jo 5E; (A)₁₆ har mønsteret 77, osv. Det ses altså, at de korrekte mønstre udsendes på korrekt tidspunkt ift. mux'ing. Alt OK!



Figur 18: Testbench ved tællerværdien $(A0D)_{16}$

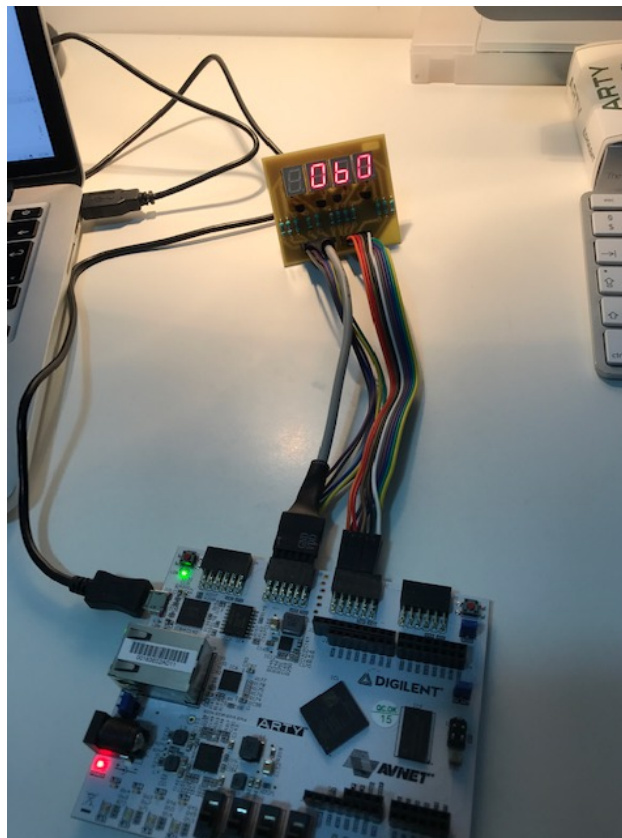
Baseret på ovenstående simulation konkluderes det, at kredsløbet består kravene til designet og fungerer efter hensigten. Det kan nu implementeres i hardware.

Der er ikke foretaget en realtidssimulering, da det tager *alt* for lang tid. Jævnfør diskussion i [1, s. 83] kan det stadig konkluderes, at systemet *også* vil fungere korrekt i realtid: Udelukkende frekvens for prescaler og refresh rate i display-mux er øget ift. realtid, og disse to er uafhængige størrelser, der ikke påvirker systemets integration – de påvirker kun systemets frekvens. Realtidsperformance analyseres i næste afsnit.

7 Implementering på FPGA (display-multiplexer)

Designet implementeres på Arty A7-boardet, og programmeringen af FPGA'en valideres ved at bekræfte, at 3 af de 4 displays optæller hex-cifrene $(000-FFF)_{16}$, med en frekvens på ca. 10 Hz (er blevet tjekket med stopur). Resetknappen testes også, med samme krav som i testbenchen.

I figuren nedenfor vises et billede fra testen.



Figur 19: Billede af testopstilling

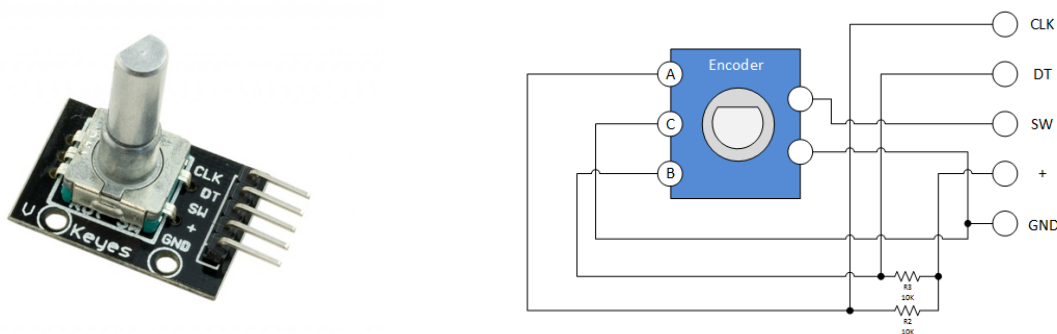
Der findes også en video, der viser en del af optællingen her: <https://youtu.be/VPvqX9AwJo0>

Det konkluderes, ligesom i testbenchen, at designet fungerer, og at FPGA'en er programmeret som ønsket.

8 Præ-lab og teori (quadrature-dekoder)

En roterende enkoder ("rotary encoder") kan være optisk eller mekanisk. I denne labøvelse er benyttet en mekanisk enkoder. Fra en mekanisk roterende enkoder genereres to firkantssignaler, det ene ude af fase med det andet. Ved at analysere rækkefølgen (faseforskydningen) af stigende og faldende flanker kan det afkodes, i hvilken omdrejningsretning enkoderen roteres.

Enkoderen benyttet i denne øvelse er en KY-040 (Keyes). Omkring denne er opbygget et kredsløb, der gør den nem at anvende. Figuren nedenfor vises den benyttede enkoder og et diagram over dens kredsløb.

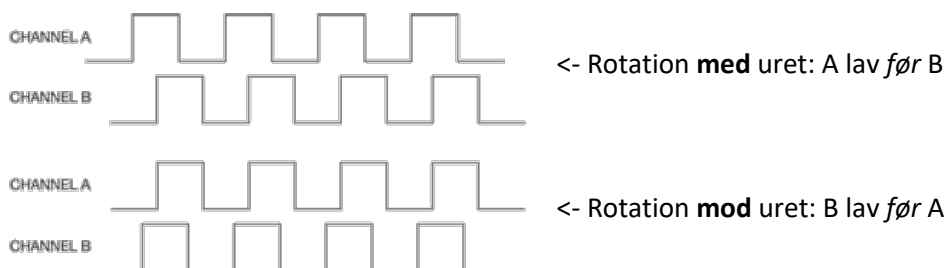


Figur 20. T.v.: Den roterende enkoder med kredsløb. T.h.: Kredsløbsdiagram for den roterende enkoder.

Der er 5 pins på enkoderen: Udover forsyning (+ og GND), ses CLK, DT og SW. CLK og DT er firkantssignalerne, der aktiveres ved at dreje på akslen. SW aktiveres ved at trykke ned på akslen. SW vil ikke blive benyttet i denne øvelse. CLK og DT vil i det følgende blive refereret som signal A og B.

Der findes to 10 k Ω pull-up-modstande på A og B, så enkoderen er **aktiv lav**. Der opstår en spændingsdeler ved de 10 k Ω og indgangsimpedansen på Arty-boardets I/O pins. Ifølge [12] er der 200 Ω modstande på de digitale ChipKit I/O-pins. Med 3,3 V forsyning over spændingsdeleren overholdes både V_{IL} og V_{IH} -grænserne for LVCMOS33 [11, tabel 8].









Den følgende figur viser konceptuelt, hvordan signalernes faseforskydning fortolkes ift. aktiv-lav.





Figur 21: Signaler for aktiv-lav enkoder. Øverst: Signal ved rotation med uret. Nederst: Signal ved rotation mod uret.

To signaler med to flanker (stigende og faldende) giver i alt 2^4 mulige signaltransitioner, der kan fortolkes og dekodes. Disse behandles i nedestående tabel.

Tabel 3: Oversigt over signaltransitioner med aktiv-lav roterende enkoder

Transition fra		Transition til		Binærværdi	Hændelse	Kommentar
A	B	A	B			
0	0	0	0	0000	-	
0	0	0	1	0001	CCW, B: 	
0	0	1	0	0010	CW, A: 	
0	0	1	1	0011	Ugyldig	
0	1	0	0	0100	CW, B: 	Benyttes i design (tæller +1)
0	1	0	1	0101	-	
0	1	1	0	0110	Ugyldig	
0	1	1	1	0111	CCW, A: 	
1	0	0	0	1000	CCW, A: 	Benyttes i design (tæller -1)
1	0	0	1	1001	Ugyldig	
1	0	1	0	1010	-	
1	0	1	1	1011	CW, B: 	
1	1	0	0	1100	Ugyldig	
1	1	0	1	1101	CW, A: 	
1	1	1	0	1110	CCW, B: 	
1	1	1	1	1111	-	Inaktiv tilstand

Note: CW er rotation med uret, CCW er rotation mod uret.  er en stigende flanke, og  faldende.

Ifølge analysen i tabellen kan både CW-rotation og CCW-rotation hver især detekteres på 4 måder. At benytte alle 4 for hver transition vil give en *meget* granulær dekoder. Hvis brugeren derimod kun skal opleve en optælling per "klik" i enkoderen, så skal der kun benyttes én af ovenstående transitioner for hver rotationsretning. De udvalgte transitioner er nævnt i tabellen sammen tællerretning ved transition.

Der kunne opstilles Karnaugh maps for hhv. CW- og CCW-rotation, men det viser sig at være nemmere blot at benytte binærværdierne direkte i VHDL, fremfor at udvikle kombinatoriske udtryk.

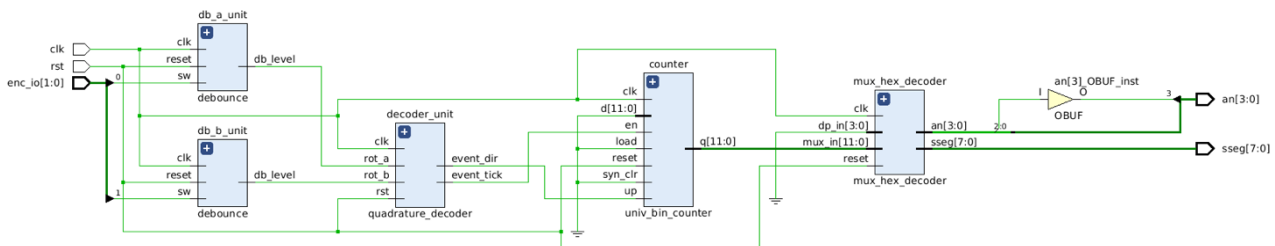
Da der må forventes at opstå prel i den roterende enkoder, som i alle andre mekaniske kontakter, vil der blive implementeret debouncing på A og B.

A og B skal derefter behandles i et dekodermodul (quadrature-dekoder), som vil udsende ticks ved rotationshændelser samt angive "retning" på rotationen.

Planen for systemet er at udvide 3x7-tælleren fra den obligatoriske opgave, således at et tick fra quadrature-dekoderen aktiverer den interne 12-bits-tæller. Systemet skal stadig være synkront på system clock, så dekoderens output benyttes udelukkende til at give tælleren et enable-signal og en "retning".

9 VHDL-design (quadrature-dekoder)

Planen for systemet, som diskuteret i analyseafsnittet, giver denne systemstruktur:



Figur 22: Blokdiagram for tællersystem med roterende enkoder

I forhold til det tidligere system er prescaleren altså erstattet med quadrature-dekoderen (og dens to debouncing-kredsløb). Derudover er en ny port sat i top-entity, som forbinder enkoderen via to hardware pins (se constraints).

Binærtælleren er stadig listing 4.9 [1], debouncing-modulet er listing 6.1. Ingen af disse moduler er ændret ift. den oprindelige VHDL, og bliver ikke vist. Display-mux'en er uændret ift. tidligere og vises heller ikke.

Det opdaterede top-modul med en ny port (og visse ubrugte signaler og konstanter fjernet):

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top_lab3_extra is
  generic(
    DP          : std_logic_vector(3 downto 0) := "0000"; -- no decimal points
    SET_REFRESH_RATE_BITS : integer := 18 -- use only the 2 MSBs, updating at 100MHz/2^16
  );
  Port (
    clk : in std_logic;
    rst : in std_logic;
    enc_io : in STD_LOGIC_VECTOR (1 downto 0); --get 2-signal quadrature
    sseg : out std_logic_vector(7 downto 0); -- 8 bits: 7 segments and 1 dp
    an : out std_logic_vector(3 downto 0) -- 4 sigs to enable transistors
  );
end top_lab3_extra;

architecture top_arch of top_lab3_extra is
  constant CNT_BITS : integer := 12; -- 12 bits for the counter (000-FFF)
  constant OFF_MODE : std_logic := '0';

  --Intermediate signals
  signal tick : std_logic; -- maps decoder tick to counter enable
  signal q_to_mux: std_logic_vector(CNT_BITS-1 downto 0); -- connects 12-bit counter to mux
  signal db_a_level, db_b_level : std_logic; --sigs for debounced rotary waveforms
  signal dir : std_logic; -- indicates direction of rotation (CW: '1'; CCW: '0')

```

Figur 23: Entity og første del af arkitektur for top-modulet

Instantiering af moduler sker i følgende kode:

```
begin

-- Debounce signal A. (Chu listing 6.1)
db_a_unit : entity work.debounce
  port map(
    clk      => clk,
    reset    => rst,
    sw       => enc_io(0), -- directly from rotary encoder, sig. A
    db_level => db_a_level, -- stable signal
    db_tick  => open
  );

-- Debounce signal B. Chu listing 6.1
db_b_unit : entity work.debounce
  port map(
    clk      => clk,
    reset    => rst,
    sw       => enc_io(1), -- directly from rotary encoder, sig. B
    db_level => db_b_level, -- stable signal
    db_tick  => open
  );

-- decode the debounced quadrature (own work)
decoder_unit : entity work.quadrature_decoder
  port map(
    rot_a    => db_a_level, -- debounced signal A
    rot_b    => db_b_level, -- debounced signal B
    clk      => clk,
    rst      => rst,
    event_tick => tick, -- goes high on rotation event
    event_dir => dir  -- direction of rotation (CW: '1'; CCW: '0')
  );

--binary counter (universal binary counter, Chu listing 4.9)
counter : entity work.univ_bin_counter
  generic map(
    N => CNT_BITS -- 12-bit (3 hex digits)
  )
  port map(
    clk      => clk,
    reset    => rst,
    syn_clr  => OFF_MODE,
    load     => OFF_MODE,
    en       => tick, -- event tick from quadrature decoder
    up       => dir, -- direction of rotation (CW '1': +1; CCW '0': -1)
    d        => (others => '0'),
    max_tick => open,
    min_tick => open,
    q        => q_to_mux -- get 12-bit count here
  );

-- Mux'ed 7-segment decoder (lab 3 custom)
mux_hex_decoder : entity work.mux_hex_decoder
  generic map(
    REFRESH_RATE_BITS => SET_REFRESH_RATE_BITS
  )
  port map(
    clk      => clk,
    reset    => rst,
    mux_in   => q_to_mux, -- 12 bits in (3 hex digits to be mux'ed)
    dp_in    => DP,
    sseg     => sseg, -- 7-segment patterns out (incl. dp)
    an       => an -- transistor mux'ing
  );

end top_arch;
```

Figur 24: Instantiering i top-modulets arkitektur

Quadrature-dekoderen er opbygget som andre synkrone systemer (A og B er debounced før dette modul):

- 4-bit register er sammensat så rækkefølge svarer til binærkoderne for transitioner i Tabel 3.
- Next-state logic skifter bits to pladser til venstre og indsætter de nyeste værdier for A og B.
- Output logic er en case statement: sætter tick og retning baseret på udvalgte transitionsværdier.

Quadrature-dekoden er derfor implementeret som følger:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity quadrature_decoder is
  Port (
    clk      : in  STD_LOGIC;      -- system clock
    rst      : in  std_logic;      -- system reset
    rot_a    : in  STD_LOGIC;      -- rotary signal A (CLK), already debounced
    rot_b    : in  STD_LOGIC;      -- rotary signal B (DT), already debounced
    event_tick : out STD_LOGIC;    -- tick for rotation event
    event_dir  : out STD_LOGIC     -- indicate rot. dir.: CW: '1'; CCW: '0'.
  );
end quadrature_decoder;

-- Method: Sample the rotary signal on each rising clock edge.
-- Register previous and current signals, usual synchronous method.
-- Total state is 4-bit sig. that relates to the report's transitions table.
-- Events can be determined fully from the 4-bit state signal.

architecture arch of quadrature_decoder is
  signal state_reg, state_next : std_logic_vector(3 downto 0); -- total state (prev & current)

begin

  -- Register
  process (clk, rst)
  begin
    if rst = '1' then
      --reset signal (active low)
      state_reg <= "1111";

    elsif rising_edge(clk) then
      --update state
      state_reg <= state_next;

    end if;
  end process;

  -- Next state logic
  state_next <= state_reg(1 downto 0) & rot_a & rot_b; -- shift in new values

  -- output logic
  process(state_reg)
  begin
    case state_reg is
      when "0100" => -- CW on falling edge for signal B
        -- Other possibilities "0010"|"1011"|"0100"|"1101"
        event_tick <= '1';
        event_dir <= '1'; -- CW rotation
      when "1000" => -- CCW on falling edge for signal A
        -- Other possibilities "0001"|"0111"|"1000"|"1110"
        event_tick <= '1';
        event_dir <= '0'; -- CCW rotation
      when others => -- Ignored states, N/C states, indeterminate states
        event_tick <= '0';
        event_dir <= '0';
    end case;
  end process;

end arch;

```

Figur 25: Quadrature-dekoder

Constraints har fået tilføjet to linjer for at forbinde enkoder-signaler (kun nye linjer vises):

```
#=====
# Arduino digital i/o
#=====
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {enc_io[0]}]
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {enc_io[1]}]
```

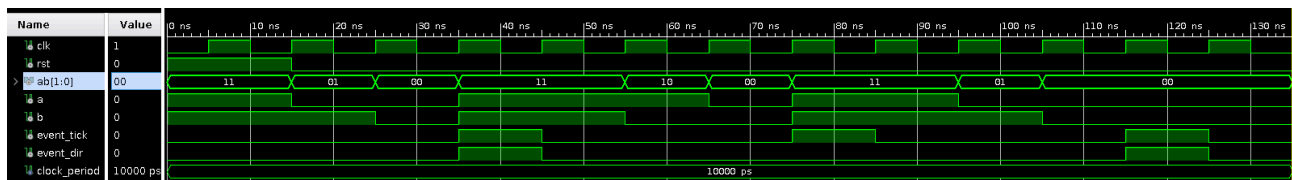
Figur 26: Nye constraints til quadrature-dekoder (i tillæg til 3x7-systemet fra obligatorisk opgave)

10 Test-bench og simulering (quadrature-dekoder)

Testbench og simulering skal validere quadrature-dekoder-modulet mht. at:



- Påvirkning af systemet svarende til de to valgte transitioner producerer de ønskede outputs.
- Systemet følger samme synkronicitet som resten af tællersystemet (tager data ind på stigende flanke, holder tilstand til næste flanke, osv.).

En simulering af dekoderen er vist i figuren (der skal zoomes ind):



Figur 27: Testbench for quadrature-dekoder-modulet

Figuren viser fem vigtige ting (transitioner refererer til Tabel 3 og til signalet ab[1:0]):

- Inaktiv tilstand 11 giver ingen event_ticks. Ok!
- Transitionen 01->00 (**cw, B:** ) producerer et event_tick fra dekoderen OG event_dir = '1'. Ok!
- event_tick og event_dir holdes kun i en clock-periode. Ok!
- Transitionen 10->00 (**ccw, A:** ) producerer et event_tick OG event_dir = '0'. Ok!
- Kun én periode event_tick selvom transition holdes længere end 1 clock-periode. Ok!

Baseret på ovenstående performer dekoder-modulet som forventet og krævet.

Testbench-koden, der frembragte ovenstående resultater findes på næste side.

```

entity tb_lab3_extra is
end tb_lab3_extra;

architecture tb_arch of tb_lab3_extra is
  constant clock_period : time := 10 ns;    -- clk period for 100 MHz
  -- Intermediate simulation signals
  signal clk : std_logic; -- simulated clock signal
  signal rst : std_logic; -- simulated reset
  -- Test signals
  signal ab : std_logic_vector(1 downto 0); --test signal to emulate AB
  signal a, b : std_logic;
  signal event_tick : std_logic;
  signal event_dir : std_logic;

begin
  -- Instantiate the decoder
  dut1: entity work.quadrature_decoder
    port map(
      clk      => clk,
      rst      => rst,
      rot_a    => a,
      rot_b    => b,
      event_tick => event_tick,
      event_dir => event_dir
    );

  a <= ab(1);
  b <= ab(0);

  -- Generate the clock
  clock: process
  begin
    clk <= '0';
    wait for clock_period / 2;
    clk <= '1';
    wait for clock_period / 2;
  end process clock;

  test : process
  begin
    -- Stimulate the decoder
    -- Active low
    ab <= "11";

    --Test the reset button for a while
    rst <= '1';
    wait until rising_edge(clk);
    wait until rising_edge(clk);
    rst <= '0';

    -- Input "0100" transition (regular rotation, direction '1')
    ab <= "01";
    wait until rising_edge(clk);
    ab <= "00";
    wait until rising_edge(clk);

    -- Go back to inactive
    ab <= "11";
    wait until rising_edge(clk);
    wait until rising_edge(clk);

    -- Input "1000" transition (regular rotation, direction '0')
    ab <= "10";
    wait until rising_edge(clk);
    ab <= "00";
    wait until rising_edge(clk);

    -- Go back to inactive
    ab <= "11";
    wait until rising_edge(clk);
    wait until rising_edge(clk);

    -- Input "0100" transition (prolonged rotation, direction '1')
    ab <= "01";
    wait until rising_edge(clk);
    ab <= "00";
    wait until rising_edge(clk);
    wait until rising_edge(clk);
    wait until rising_edge(clk);

    -- End the simulation
    assert false
      report "Simulation OK!"
      severity failure;
  end process test;
end tb_arch;

```

Figur 28: Testbench for quadrature-dekoder-modul

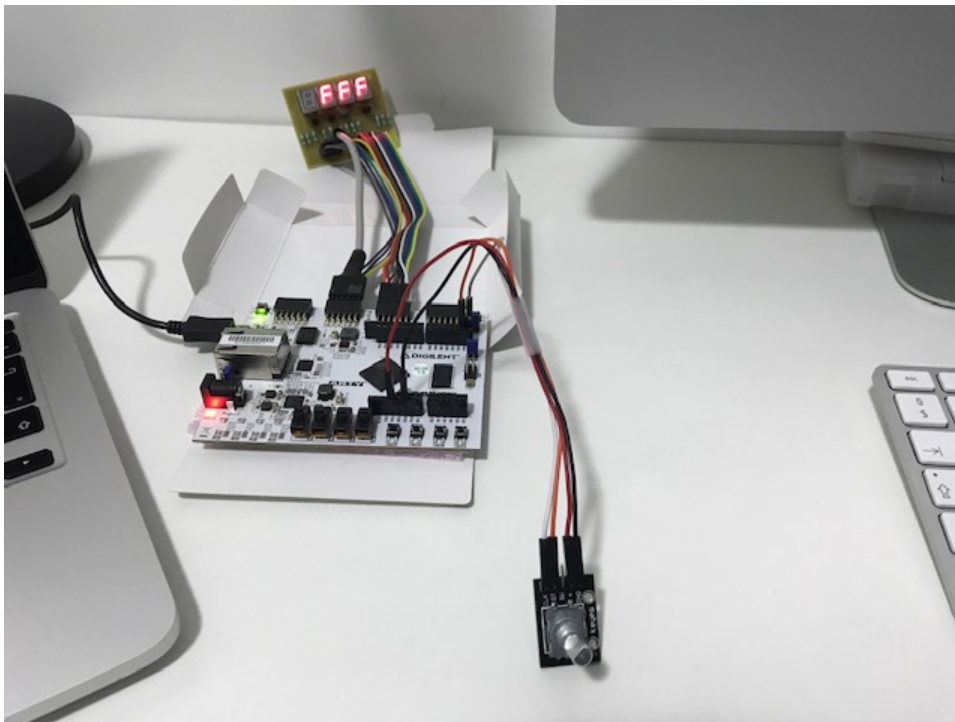
11 Implementering på FPGA (quadrature-dekoder)

Designet er implementeret på FPGA'en. Testproceduren er at:

- Rotere med og mod uret og bekræfte at ét klik på enkoderen svarer til inkrementering eller dekrementering med netop $(1)_{16}$ på displayet.
- Rotere enkoderen, så tælleren dekrementerer ned forbi $(000)_{16}$ til $(FFF)_{16}$. Inkrementere op forbi $(FFF)_{16}$ igen.
- Rotere "hurtigt" og "langsomt" og se, hvordan tælleren ændrer værdi. Der er intet krav til vinkelfrekvenser – men det observeres, at debouncing forhindrer optælling ved meget hurtig rotation (se evt. video).
- Teste resetknappen.

Testprocessen forløb som forventet, og systemet fungerer som ønsket (ud over at enkoderen er af dårlig kvalitet, og skal tilføres et let tryk for at de indre kontakter skaber god forbindelse). Det vurderes, at hardwaredesignet fungerer som krævet.

Et billede fra testprocessen er vist her:



Figur 29: Billede fra test af roterende enkoder og quadrature-dekoder

En video fra testprocessen kan ses her: <https://youtu.be/AdiPnoWK768>.

12 Fejlkilder og måleusikkerheder

Designet af både den obligatoriske og ekstraopgaven er verificeret ved inspektion af waveforms og Tcl-loggen. Implementeringen er kontrollet fysisk ved at aflæse korrekt optælling på 4x7-boardet og at afprøve den roterende enkoder med forskellige scenarier.

Menneskelige fejl kan spille ind, og derfor blev en automatisk testbench benyttet til den obligatorisk opgave. I forlængelse heraf blev prescalerens frekvens og display-mux'ens refresh rate øget for at reducere antal clock flanker for en fuld cyklus. Dette vurderes fuldt acceptabelt ift. at bedømme designets integration og virkemåde ifbm. testbench, jævnfør [1, s. 83].

For den obligatoriske opgave vurderes det alt i alt, at den udførte testbench sammen med manuel inspektion af waveforms og kontrol af korrekt optælling på 4x7-boardet er tilstrækkelig validering af kredsløbet.

For ekstraopgaven vurderes det, at en testbench på quadrature-dekoder-modulet er tilstrækkeligt (de andre moduler er allerede testet) i forening med en test af den faktiske hardwareimplementering.

Det vurderes, at der ikke var væsentlige fejlkilder i denne labøvelse.

13 Diskussion og konklusion

Der er i denne labøvelse vist, hvorledes det synkrone designmønster kan anvendes til at implementere display-multiplexing og en quadrature-dekoder.

Derved er der i hardware designet et system, hvor 3 x 7-segmentdisplays viser værdien fra et 10 Hz 12-bit tællekredsløb, samt et system hvor den automatiske tæller er erstattet af brugerinput fra en roterende dekoder.

Systemerne omfatter begge en synkron display-multiplexer og en hex-til-7-segment-dekoder. Det automatiske tællersystem indeholder desuden to generiske tællere. De to generiske tæller bruges som hhv. prescaler (100 MHz til 10 Hz) og 12-bit-tæller $(000-FFF)_{16}$. Systemet, som dekoder en roterende kontakt, indeholder et dekoder-modul opbygget på baggrund af analyser udført i denne rapport samt et generisk debouncing-modul fra [1].

Der er desuden i den obligatoriske opgave vist, hvorledes en automatisk testbench til det synkrone design kan struktureres, og hvordan testproceduren automatisk kan gennemløbe en fuld cyklus for systemet. Det er desuden beskrevet, hvorledes Waveforms i Vivado kan analyseres for at bekræfte systemets ønskede funktion. Der er ligeledes implementeret og udført en testbench til quadrature-dekoder-modulet.

Systemerne er begge implementeret i en FPGA på et Arty A7-board, og hardwaren er ligeledes testet til korrekt at repræsentere det ønskede synkrone system.

En væsentlig observation i løbet af denne lab-rapport er, at grundig test og validering af et system kræver i hvert fald 2-3 gange så meget tid og indsats som selve opbygningen af systemet.

14 Referencer

- [1]: FPGA Prototyping by VHDL Examples, Xilinx Microblaze MCS SoC (2nd Edition); CHU, P. P.; Wiley, 2017
- [2]: Digital Fundamentals (11th Edition, Global Edition); FLOYD, T. L.; Pearson, 2015
- [3]: VHDL-Håndbogen; Mølgaard, C; Comfact, 2003
- [4]: Arty A7 Reference Manual, Digilent. Tilgængelig på internettet (sidst hentet 15/2/2019): <https://reference.digilentinc.com/reference/programmable-logic/arty-a7/reference-manual>
- [5]: DS180: Xilinx 7 Series FPGAs Data Sheet: Overview. Tilgængelig på internettet (sidst hentet 15/2/2019): https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
- [6]: VHDL Guide – FPGA designs. Tilgængelig på internettet (sidst hentet 16/2/2019): <https://vhdlguide.readthedocs.io/en/latest/vhdl/testbench.html#testbench-with-look-up-table>
- [7]: Generisk constraints-fil til Arty A7-boardet (Rev. D). Tilgængelig på Github (sidst hentet 16/2/2019): <https://github.com/Digilent/digilent-xdc/blob/master/Arty-A7-35-Master.xdc>
- [8]: P. CHU constraints-fil til Arty A7-boardet (Rev. B). Tilgængelig på internettet (sidst hentet 3/3/2019): https://academic.csuohio.edu/chu_p/rtl/fpga_mcs_vhdl.html
- [9]: Jacob Bechmanns 7-segment print. PDF-fil tilgængelig på Blackboard, E2DEL 2019. Sidst hentet 3/3/2019.
- [10]: LTS-4301WC 7 segment display data sheet. LITEON. Tilgængelig på internettet (sidst hentet 3/3/2019): https://www.elpro.org/gb/index.php?controller=attachment&id_attachment=5728
- [11]: DS181: Artix-7 FPGAs Data Sheet: DC and AC Switching Characteristics. Tilgængelig online (sidst hentet 15/3/2019): https://www.xilinx.com/support/documentation/data_sheets/ds181_Artix_7_Data_Sheet.pdf
- [12]: Arty A7 schematics (Rev. E.0). Tilgængelig på internettet (sidst hentet 15/3/2019): https://reference.digilentinc.com/media/reference/programmable-logic/arty-a7/arty_a7_sch.pdf

15 Bilag

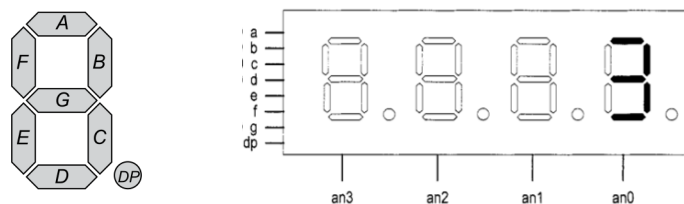
I lab 2 blev følgende analyser foretaget forud for tilkobling af 4x7-boardet. Disse analyser bekræfter designet af 4x7-boardet ift. elektriske begrænsninger for både 4x7-boardet og Arty A7-boardet. Analyserne viser desuden, hvordan 7-segmentdisplays benyttes i hardware-design.

15.1 Bilag: Brug af 7-segmentdisplays

Til systemet benyttes et "common cathode" 7-segmentdisplay, monteret på et eksternt print. Hvert segment i displayet er aktiv højt, og alle segmenterne i displayet deler katode. Katoden får stelforbindelse gennem en transistor. Dette er modsat kodeeksemplerne i [1], som benytter et "common anode"-display.

Hvert segment refereres med et standardiseret bogstav, som ses i nedestående figur (t.v.). Det konstruerede 4x7-board fra værkstedskursus har 4 displays, som hver især kan aktiveres ved at stimulere transistoren ved displayets katode (se figur t.h.).

Signaler til transistorernes basis-ben kaldes AN0-AN3, og er også er aktiv høje (burde nok hedde CA eller KA).



Figur 30: Segmentbogstaver i et display og adressering af 4x7-boardets displays. Visning af tallet 3 som i afbildet eksempel kræver at segmenterne e,f = '0'. Visning på højre display af de fire kræver an0 = '1'.

Kombinationer af signaler til at aktivere visning af de 15 forskellige hex-symboler (0-F) ses i Tabel 4:

Tabel 4: Signalkombinationer i dekoderen til frembringelse af hexværdierne 0-F (routing fra hex til dekoder)

Input (decimal)	Symbol på display	Output-dekodersignal								Dekodersignal i hex. (bruges til validering)
		dp	g	f	e	d	c	b	a	
0	0	0	0	1	1	1	1	1	1	3F
1	1	0	0	0	0	0	1	1	0	06
2	2	0	1	0	1	1	0	1	1	5B
3	3	0	1	0	0	1	1	1	1	4F
4	4	0	1	1	0	0	1	1	0	66
5	5	0	1	1	0	1	1	0	1	6D
6	6	0	1	1	1	1	1	0	1	7D
7	7	0	0	0	0	0	1	1	1	07
8	8	0	1	1	1	1	1	1	1	7F
9	9	0	1	1	0	1	1	1	1	6F
10	A	0	1	1	1	0	1	1	1	77
11	B / b	0	1	1	1	1	1	0	0	7C
12	C	0	0	1	1	1	0	0	1	39
13	D / d	0	1	0	1	1	1	1	0	5E
14	E	0	1	1	1	1	0	0	1	79
15	F	0	1	1	1	0	0	0	1	71

Note: DP sættes altid til 0 i denne lab-rapport.

15.2 Bilag: Hardwaredesign og komponentbegrænsninger

Ratings for alle komponenter, både på Arty-boardet og på 4x7-boardet, skal overholdes. Tabel 5 opsummerer de væsentligste begrænsninger og designimplikationer.

Tabel 5: Væsentlige komponentbegrænsninger og designimplikationer

Komponentbegrænsninger	Værdi [kilde]	Implikation
Arty A7: I/O og PMOD-forsyning	3,3 V (LVCMOS33) [4: tabel 3.1]	Der er 3,3 V på et høj signal fra Arty-boardet til 4x7-boardet.
Arty A7: Maksimal strømstyrke for en High-range I/O port	16 mA [11: tabel 8 note 4, s. 8-9]	Kan maks. trække 16 mA på en port og stadig overholde V_{OH} og V_{OL} -niveauer.
Arty A7: Maksimal samlet strømstyrke fra I/O og PMOD-forsyning	2,0 A [4: tabel 3.1]	Der forbindes 12 signaler; worst-case med alle sig. høje, maksimalt træk på 2,0 A / 12 = 160 mA. Langt mere end 7-seg. rating og portbegrænsning.
Arty A7: PMOD JB og JC er uden beskyttende modstande (high-speed)	shunted ned til 0 Ω [12: s.1]	JB og JC benyttes til at forbinde 4x7-boardet; for at undgå 200 Ω -modstandene på JA og JD.
Arty A7: Udgangsimpedans	25 Ω (400 mV/16 mA) [11: tabel 8, s. 8]	Udgangens/bufferens "seriemodstand"; Trækker udgangsspænding ned ved øget strømtræk.
7-seg: Maksimal effekt afsat i hvert segment på 7-seg.	75 mW [10: s. 4]	Ved 3,3 V forsyning: maks. kontinuert strømstyrke per segment altså 75 mW / 3,3 V \approx 20 mA.
7-seg: Forward voltage per segment	1,8 V ved 20 mA [10: s. 4]	Modstandsværdi per seg. på 4x7 skal som minimum være $(3,3 \text{ V} - 1,8 \text{ V}) / 16 \text{ mA} - 0 \Omega - 25 \Omega = 69 \Omega$.

Hvert segment skal have en formodstand på mindst 69 Ω for at begrænse strømstyrken. Dette er implementeret med 82 Ω formodstande. Der er også 1 k Ω modstande ved hver transistors basis-ben, så med op til 3,3 mA på basis-ben og en $h_{FE} > 100$ har transistoren ikke en strømbegrænsende virkning når tændt.

PMOD-pins på 4x7-boardet kan ses i Figur 31 (gengivet fra [9])¹. Figuren viser et kig forfra på 4x7-modulet².



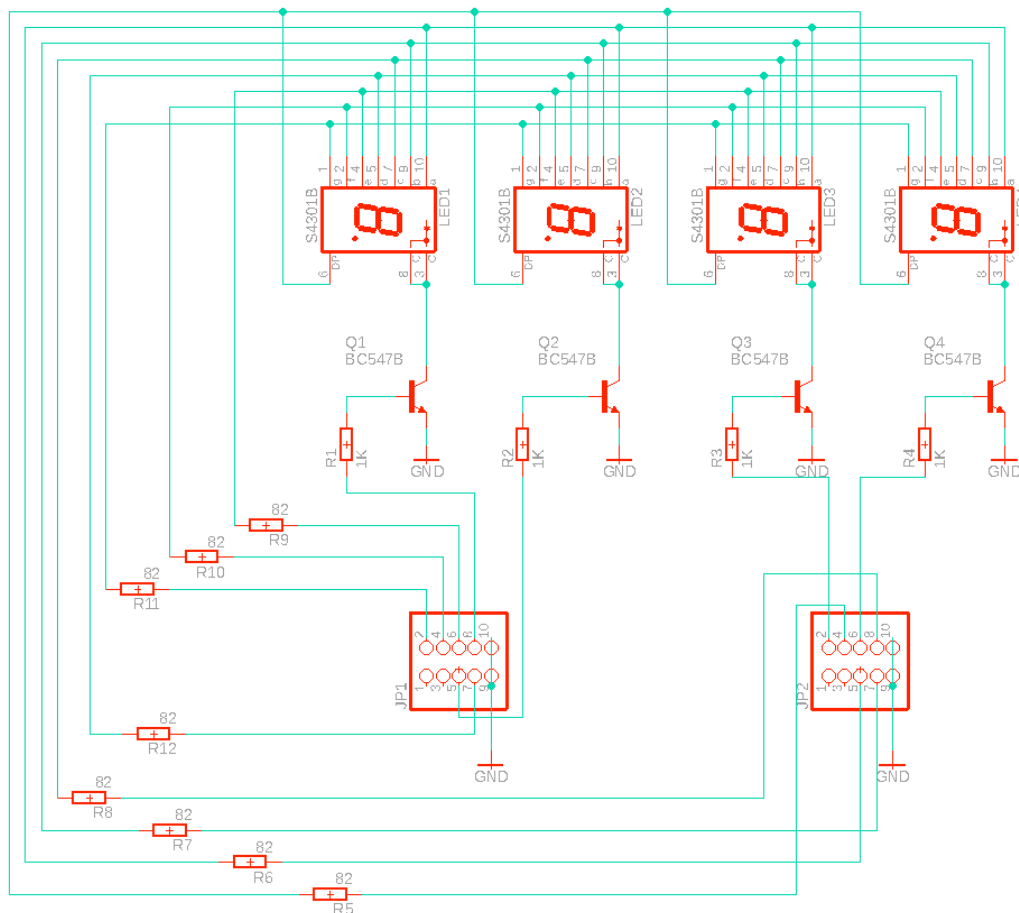
Figur 31: Mapping mellem 4x7-boardet og Arty-boardets PMOD-connectors. V_{CC} pins er ikke medtaget i 4x7-board-design.

¹ 4x7-boardet har kun 5 han-pins i hver række pinheaders. Dette er forskelligt fra PMOD-stikket og det opdaterede E2DEL-4x7-design med 6 pins per række. 4x7-boardet forbindes stadig fint til Arty-boardet med PMOD-kabler. De "uforbundne" PMOD pins er alle V_{CC} , som ikke skal benyttes, da al forsyning til LED'erne kommer fra de øvrige benyttede PMOD-pins.

² Venstre PMOD-connector er JB, højre er JC. Bogstaverne A-G påtrykt figuren refererer til LED-segmenter. Q0-Q3 refererer til AN0-AN3. NC er "not connected". Pinnavne til hardwaremapping er angivet oven på hver pin. Fx benyttes PMOD pin E15 til at stimulere segment G; segment G er pin 1 på 7-segmentet på boardet (jf. [10])

Da 4x7-boardet er valideret til ikke at have uønskede kortslutninger, og dimensionering er i orden ift. ratings, kan 4x7-boardet benyttes som allerede implementeret.

Kredsløbsdesignet for 4x7-boardet ses i Figur 32:



Figur 32: Kredsløbsdesign for 4x7-board. Designet fra 1. semester er desværre ikke helt aligned med E2DEL. Bemærk: Transistor Q4 får signal AN0, Q1 får AN3, osv..

15.3 Bilag: 7-segment-dekoder

Dekoderen, der er instantieret i top-modulet, blev udviklet og dokumenteret i E2DEL Lab 2; men alligevel dokumenteres koden her samt diskuteres nogle design-ønsker. Designet er delvis baseret på skabelon fra [1].

Design-ønsker til en **hex-til-7-segment-dekoder**:

- Hex-dekoderen er en signalvælger (routing circuit): skal vælge mellem signaler med samme prioritet. Skal vælge fra en udtømmende liste, hvor signalerne er gensidigt udelukkende (concurrent multiplexer). Den skal implementere relationen, som er vist i Tabel 4.
- Entity:
 - To input-porte:
 - Én til 4-bit signalet fra multiplexeren (hex)
 - Én til at sætte kommasegmentet (dp).
 - Outputsignalet er de 8 bits til 4x7-boardet (sseg): omfatter både 1 bit til kommasegmentet og 7 bits til de 7 segmenter.
- Arkitektur:
 - Skal være en signalvælger
 - Hele 8-bit-signalet assignes i ét step for at undgå en uønsket latch (hukommelselement), som ellers dukker op, hvis dp assignes separat.
- Aktiv-højt display: Dekoderen i denne labøvelse er til et display med segmenter, der er aktiv høje (common cathode som på 4x7-boardet). Dette er i modsætning til eksemplerne i [1], som er baseret på et aktiv lavt signal (common anode).

Den fulde kode til dekoderen, på baggrund af ovenstående overvejelser, ses herunder:

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

entity hex_decoder is
  port(
    hex  : in  std_logic_vector(3 downto 0); --4 bits HEX
    dp   : in  std_logic; --decimal point (if wanted)
    sseg : out std_logic_vector(7 downto 0) --7seg signals
  );
end hex_decoder;

--this architecture ONLY decodes, no memory register or time multiplexing
architecture arch of hex_decoder is
begin
  -- Routing circuit.
  -- Note: CC disp on board are active high. Avoids a latch by full assignment.
  with hex select
    sseg <=
      -- dp    gfedcba segm., binary counter value
      dp & "0111111" when "0000", -- 0
      dp & "0000110" when "0001", -- 1
      dp & "1011011" when "0010", -- 2
      dp & "1001111" when "0011", -- 3
      dp & "1100110" when "0100", -- 4
      dp & "1101101" when "0101", -- 5
      dp & "1111101" when "0110", -- 6
      dp & "0000111" when "0111", -- 7
      dp & "1111111" when "1000", -- 8
      dp & "1101111" when "1001", -- 9
      dp & "1110111" when "1010", -- A
      dp & "1111100" when "1011", -- B
      dp & "0111001" when "1100", -- C
      dp & "1011110" when "1101", -- D
      dp & "1111001" when "1110", -- E
      dp & "1110001" when others; -- F
end arch;
```

Figur 33: Hex-til-7-segment-dekoder