
E2DSD — Journal #3
“Sequential-style Combinatorial Designs”
og “Sequential Designs”

Gruppe #37
Adam Ryager Høj — 201803767
Rasmus Kahr — 201803491
Hoang Phuoc Pho Tran — 201507142

2020-04-21

Indhold

ØVELSE 1: Sequential-style Combinatorial Designs	5
Opgave 1. Binary to 7-Segment Decoder Using “CASE”.....	5
Opgave 2. Guess Game	7
Opgave 3. Two Player Guess Game.....	15
Opgave 4. 8-input NAND Gate	19
Opgave 5. Count Ones.....	22
ØVELSE 2: Sequential Designs in VHDL (The Watch Exercise)	25
Opgave 1. Counter - One Digit.....	25
Opgave 2. Clock - One Digit	28
Opgave 3. Counter - Six Digit	31
Opgave 4. Alarm Watch.....	39
Opgave 5. Peer Feedback: Journal 2	46

Listings

1.1	Binary to 7-segment entity og arkitektur	5
1.2	Test bench: Binary to 7-segment	7
1.3	Guess Game Entity	8
1.4	Guess Game arkitektur og interne signaler	8
1.5	Guess Game arkitekturen del 1	9
1.6	Guess Game arkitekturen del 2	10
1.7	Latch entity og arkitektur	10
1.8	Compare entity og arkitektur	11
1.9	Fælles mux entity	11
1.10	Mux2 arkitektur	12
1.11	Mux4 arkitektur	12
1.12	Guess Game testbench	13
1.13	Udsnit af output report	14
1.14	Two Player Guess Game Entity	15
1.15	Two Player Guess Game arkitektur og interne signaler	16
1.16	Two Player Guess Game arkitektur	16
1.17	Two Player Guess Game arkitektur	17
1.18	Two Player Guess Game multiplexers	18
1.19	Test mapping til 2-player Guess Game	18
1.20	Udsnit af output report	19
1.21	Generic input NAND-gate entity og arkitektur	20
1.22	Testbench til 8-input NAND-gate	20
1.23	Interface til count_ones entity	22
1.24	Arkitektur til optælling af 1'ere vha. FOR-LOOP	23
2.1	Count one digit entity	25
2.2	Count one digit arkitektur	26
2.3	Test bench: Multi Counter	27
2.4	Clock generatoren entity og start på arkitekturen (1 af 2)	29
2.5	Clock generatoren speed select og reset (2 af 2)	30
2.6	Entity til håndtering af sammenhørende 1'er og 10'er counters	32
2.7	Reset logik til 24-timers uret	33
2.8	Interface til watch entity	34
2.9	Begyndelse på arkitektur og definerede signaler	34
2.10	Arkitekturen for watch bestående af delblokke (1 af 2)	35
2.11	Arkitekturen for watch bestående af delblokke (2 af 2)	36
2.12	Testbench for entity watch	37
2.13	Alarm watch entity	39
2.14	Alarm watch arkitektur	39
2.15	Alarm watch arkitektur	40
2.16	Alarm watch arkitektur	41
2.17	Quad HEX display	42
2.18	Alarm watch arkitektur: Compare	43
2.19	Alarm watch arkitektur: Input limiting	44
2.20	Alarm watch test bench	45

Figurer

1.1	RTL view: Binary to 7-segment	6
1.2	Postmapping view (udsnit): Binary to 7-segment	6
1.3	RTL view: Guess game	13
1.4	Postmapping view: Guess game	13
1.5	RTL view: Guess Game elementer	14
1.6	HEX view: Guess Game	15
1.7	Antallet af <i>inferred latches</i> efter syntetisering	19
1.8	RTL view: 8 bit NAND	21
1.9	RTL view: 32 bit NAND	21
1.10	Funktionel simulation af 8-input NAND	21

1.11	RTL view af count_ones	23
1.12	count_ones test cases	24
2.1	Funktionel simulation 1: almindelig optælling	27
2.2	Funktionel simulation 2: <code>reset</code> sættes midt i optælling	27
2.3	Funktionel simulation 3: skift af <code>mode</code>	28
2.4	Netlist views af clockgeneratoren	28
2.5	RTL view for den samlede <code>watch entity</code>	38
2.6	Fotos der viser de forskellige tidspunkter.	38
2.7	<i>Rollover</i> implementationen, der viser overgang m. [23:59:59]->[00:00:00].	38
2.8	RTL view: Alarm watch	41
2.9	Fotos der viser de forskellige tilstade for alarmen	45

ØVELSE 1: SEQUENTIAL-STYLE COMBINATORIAL DESIGNS

Indhold

Opgave 1. Binary to 7-Segment Decoder Using “CASE”.....	5
Opgave 2. Guess Game	7
Opgave 3. Two Player Guess Game.....	15
Opgave 4. 8-input NAND Gate	19
Opgave 5. Count Ones.....	22

Opgave 1: Binary to 7-Segment Decoder Using “CASE”

Introduktion

I denne opgave vil vi igen skrive en Binary to 7-segment decoder. Vi har tidligere lavet en med **WITH-SELECT** syntaks, denne gang laves den i en **process** med en **CASE** statement.

Funktionaliteten bør være den sammen som vores forrige kombinatoriske implementation med **WITH-SELECT** statements.

Design og implementering

Decoderen blev designet på samme måde som i Øvelse 4, og med de samme betingelsesværdier, dog med en **CASE** i stedet. De to syntakser minder om hinanden, men der er dog nogle forskelle. Implementationen ses i source code table 1.1

```
1 ENTITY bin2hex IS
2     PORT (
3         bin : IN std_logic_vector(3 DOWNTO 0); -- Binary input
4         seg : OUT std_logic_vector(6 DOWNTO 0) -- 7 segment output
5     );
6 END bin2hex;
7
8 ARCHITECTURE behavioral OF bin2hex IS
9 BEGIN
10    binProcess : PROCESS (bin)
11    BEGIN
12        CASE(bin) IS
13            WHEN "0000" => seg <= "1000000"; -- 0
14            WHEN "0001" => seg <= "1111001"; -- 1
15            WHEN "0010" => seg <= "0100100"; -- 2
16            WHEN "0011" => seg <= "0110000"; -- 3
17            WHEN "0100" => seg <= "0011001"; -- 4
18            WHEN "0101" => seg <= "0010010"; -- 5
19            WHEN "0110" => seg <= "0000010"; -- 6
20            WHEN "0111" => seg <= "1111000"; -- 7
21            WHEN "1000" => seg <= "0000000"; -- 8
22            WHEN "1001" => seg <= "0011000"; -- 9
23            WHEN "1010" => seg <= "0001000"; -- A
24            WHEN "1011" => seg <= "0000011"; -- B
25            WHEN "1100" => seg <= "0100111"; -- C
26            WHEN "1101" => seg <= "0100001"; -- D
27            WHEN "1110" => seg <= "0000110"; -- E
28            WHEN "1111" => seg <= "0001110"; -- F
29            WHEN OTHERS => seg <= "1001001"; -- //Error display//
30        END CASE;
31    END PROCESS; -- binProcess
32 END behavioral;
```

Listing 1.1: Binary to 7-segment entity og arkitektur

Resultater

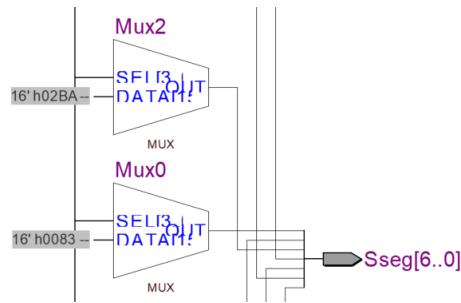


Fig. 1.1: RTL view: Binary to 7-segment

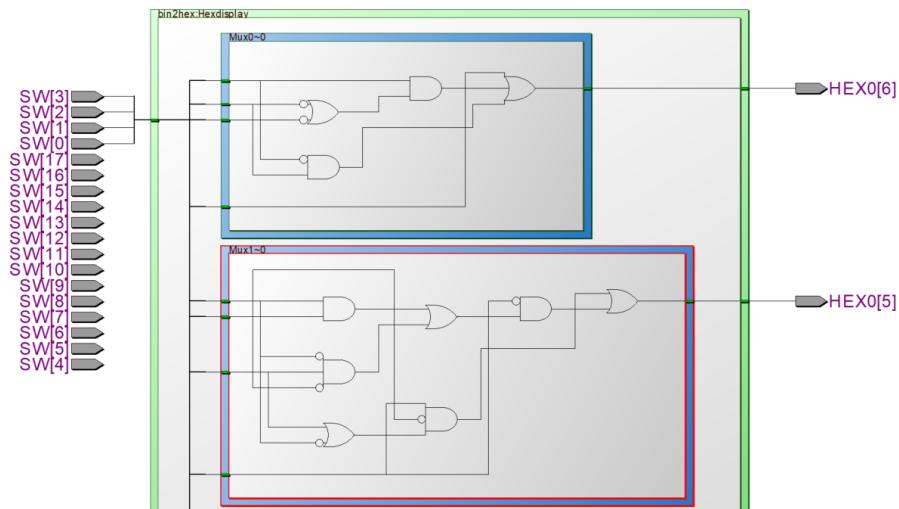


Fig. 1.2: Postmapping view (udsunit): Binary to 7-segment

For at teste funktionaliteten blev der også lavet en testcase, denne ses i tab. 1, og dens testbench kode ses i source code table 1.2

Input	Forventet display	Reelt display
0101	5	5
0111	7	7
1111	F	F

Tab. 1: Test case for binary to 7-segment

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 USE work.ALL;
5
6 ENTITY bin2hex_tester IS
7 PORT (
8     SW : IN std_logic_vector(3 DOWNTO 0);
9     HEX0 : OUT std_logic_vector(6 DOWNTO 0)
10 );
11 END;
12
13 ARCHITECTURE testbench OF bin2hex_tester IS
14 BEGIN
15     UUT : ENTITY bin2hex(behavioral)
16     PORT MAP
17     (
18         -- INPUTS
19         bin => SW,
20         -- OUTPUTS
21         seg => HEX0
22     );
23 END ARCHITECTURE testbench;

```

Listing 1.2: Test bench setup, hvori Binary to 7-segment afprøves

Diskussion

Det ses på fig. 1.1 at den ligesom i det kombinatoriske design består af en række multiplexers. Der er altså ikke den store forskel på **WITH-SELECT** og **CASE** i dette tilfælde. Post-mapping viewet i fig. 1.2 viser heller ikke nogen forskel, i forhold til sidste øvelse.

Konklusion

Der er ingen reel forskel på, om man implementere en binary to 7-segment vha. **CASE** eller **WITH-SELECT** statements—begge er lige velfungerende. Det bør dog undersøges om der er en forskel i ren praksis.

Opgave 2: Guess Game

Introduktion

Målet med denne opgave var at skabe et gættespil, hvor en spiller forsøger at gætte en talværdi bestemt af en anden person. Spillet skal alt imens kunne give et visuelt feedback, dette baseret på spillernes input. Dette lægger grund for brugen af et register der går systemet sekventielt, i stedet for kombinatorisk som de andre vi tidligere har arbejdet med.

Design og implementering

For at spillet kan muligøres, skal der laves en række elementer som udgøre selve spillets kerne. Alle entities indeholder de almindeligt brugte **library** og **use** statements som er udeladt for at være lidt mere kortfattet. De entities der skal bruges er som følger:

Guess Game

Delblokkene blev samlet i en **guess_game** entity der blev opbygget strukturelt og udnyttede muligheden for at lave interne signaler.

Da koden blev meget lang, er den delt op i flere stykker.

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE work.ALL;
4 ENTITY guess_game IS
5 PORT (
6     inputs : IN std_logic_vector(7 DOWNTO 0);
7     set    : IN std_logic;                                -- set predef. vals.
8     show   : IN std_logic;                                -- show predef. vals.
9     try    : IN std_logic;                                -- eval. guess
10    hex1  : OUT std_logic_vector(6 DOWNTO 0); -- 7-seg ones
11    hex10 : OUT std_logic_vector(6 DOWNTO 0) -- 7-seg tens
12 );
13 END guess_game;

```

Listing 1.3: Interface til guess_game entity.

```

1 ARCHITECTURE guessgame OF guess_game IS
2
3     SIGNAL secret_value : std_logic_vector(7 DOWNTO 0);
4     SIGNAL muxTwoOut    : std_logic_vector(7 DOWNTO 0);
5     SIGNAL bin2hexOut   : std_logic_vector(13 DOWNTO 0);
6     SIGNAL muxFourSel   : std_logic_vector(1 DOWNTO 0);

```

Listing 1.4: Her ses begyndelsen på Guess Game arkitekturen, samt de signaler der oprettes

```

1   displayOnes : ENTITY bin2hex(Behavioral)
2     PORT MAP
3   (
4     -- INPUTS
5     bin(3 DOWNTO 0) => muxTwoOut(3 DOWNTO 0),
6     -- OUTPUTS
7     seg(6 DOWNTO 0) => bin2hexOut(6 DOWNTO 0)
8   );
9
10  displayTens : ENTITY bin2hex(blank_zero)
11    PORT MAP
12  (
13    -- INPUTS
14    bin(3 DOWNTO 0) => muxTwoOut(7 DOWNTO 4),
15    -- OUTPUTS
16    seg(6 DOWNTO 0) => bin2hexOut(13 DOWNTO 7)
17  );
18
19  Latcher : ENTITY generic_latch
20    GENERIC MAP(
21      bits => 8
22    )
23    PORT MAP
24  (
25    -- INPUTS
26    en => set,
27    d  => inputs(7 DOWNTO 0),
28    -- OUTPUTS
29    q  => secret_value(7 DOWNTO 0)
30  );
31  Compare : ENTITY CompareLogic(compare)
32    PORT MAP
33  (
34    -- INPUTS
35    en => try,
36    a  => inputs(7 DOWNTO 0),
37    b  => secret_value(7 DOWNTO 0),
38    -- OUTPUTS
39    comp => muxFourSel(1 DOWNTO 0)
40  );

```

Listing 1.5: Her ses de interne forbindelser mellem de oprettede elementer.

```

1  MultiplexFourOne : ENTITY mux(muxFour)
2      PORT MAP
3      (
4          -- INPUTS
5          inSelect => muxFourSel(1 DOWNTO 0),
6          inA      => bin2hexOut(13 DOWNTO 0),
7          inB      => (OTHERS => '0'),
8          inC      => (OTHERS => '0'),
9          inD      => (OTHERS => '0'),
10         -- OUTPUTS
11         o(6 DOWNTO 0) => hex1(6 DOWNTO 0),
12         o(13 DOWNTO 7) => hex10(6 DOWNTO 0)
13     );
14  MultiplexTwoOne : ENTITY mux(muxTwo)
15      PORT MAP
16      (
17          -- INPUTS
18          inSelect(0)      => show,
19          inA(7 DOWNTO 0)  => inputs,
20          inB(7 DOWNTO 0)  => secret_value,
21          inC => (OTHERS => '0'),
22          inD => (OTHERS => '0'),
23          -- OUTPUTS
24          o(7 DOWNTO 0) => muxTwoOut
25      );

```

Listing 1.6: Slutningen af koden, hvor de sidste elementer forbindes, og evt. ubrugte porte bliver der taget hånd om.

Latch

Et register der kan fastholde en værdi. Vores implementation ses på source code table 1.7 er generisk og kan derfor gemme værdier af forskellig størrelse. Denne har til formål at gemme det korrekte tal, når der trykkes på `set`.

```

1 ENTITY generic_latch IS
2     GENERIC (
3         bits : POSITIVE := 8
4     );
5     PORT (
6         d : IN std_logic_vector(bits - 1 DOWNTO 0); -- Input
7         en : IN std_logic;                          -- Enable switch
8         q : OUT std_logic_vector(bits - 1 DOWNTO 0) -- Output
9     );
10 END generic_latch;
11
12 ARCHITECTURE latch OF generic_latch IS
13 BEGIN
14     PROCESS (en, d)
15     BEGIN
16         IF en = '0' THEN -- Hvis en = TRUE
17             q <= d;           -- q gets what d is
18         END IF;
19     END PROCESS;
20 END latch;

```

Listing 1.7: Latch entity og arkitektur

Compare

Compare logikken der er vist i source code table 1.8 er sat op vha. en **IF** og **WHEN-ELSE** statement, der sammenligninger de inputs der er tilstede—men kun når **en** er aktiv, hvilket **process()** søger for.

```
1 ENTITY CompareLogic IS
2     PORT (
3         a      : IN std_logic_vector(7 DOWNTO 0);
4         b      : IN std_logic_vector(7 DOWNTO 0);
5         en    : IN std_logic;
6         comp  : OUT std_logic_vector(1 DOWNTO 0)
7     );
8 END CompareLogic;
9
10 ARCHITECTURE compare OF CompareLogic IS
11 BEGIN
12     PROCESS (en, a, b)
13     BEGIN
14         comp <= "10";
15
16         IF en = '0' THEN
17             IF a > b THEN
18                 comp <= "11";
19             ELSIF a < b THEN
20                 comp <= "00";
21             ELSIF a = b THEN
22                 comp <= "01";
23             ELSE
24                 comp <= "10";
25             END IF;
26         END IF;
27     END PROCESS;
28 END compare;
```

Listing 1.8: Compare entity og arkitektur

Multiplexers

```
1 ENTITY mux IS
2     PORT
3     (
4         inSelect          : IN std_logic_vector(1 DOWNTO 0); --
5         Selector          : Selector;
6         inA, inB, inC, inD       : IN std_logic_vector(13 DOWNTO 0); --
7         input vectors
8         o, o1, o2, o3, o4, o5, o6, o7 : OUT std_logic_vector(13 DOWNTO 0) --
9         Output vectors
10    );
11 END mux;
```

Listing 1.9: Fælles mux entity

Det var tydeligt at binary to 7-segement decoderen bestod af multiplexers - hvilket gør at følgende multiplexers kan skrives på samme måde.

```

1 ARCHITECTURE muxTwo OF mux IS
2
3 BEGIN
4     muxProcess : PROCESS (inSelect(0), inA, inB)
5     BEGIN
6         CASE(inSelect(0)) IS
7             WHEN '1'      => o(13 DOWNTO 0)      <= inA(13 DOWNTO 0);
8             WHEN '0'      => o(13 DOWNTO 0)      <= inB(13 DOWNTO 0);
9             WHEN OTHERS => o(13 DOWNTO 0)      <= inA(13 DOWNTO 0);
10        END CASE;
11    END PROCESS; -- muxProcess
12 END muxTwo; -- muxTwo

```

Listing 1.10: Mux2 arkitektur

Derfor kunne den let skrives på samme måde, hvilket ses i source code table 1.10

```

1 ARCHITECTURE muxFour OF mux IS
2
3 BEGIN
4     muxProcess : PROCESS (inSelect, inA)
5     BEGIN
6         CASE(inSelect) IS
7             WHEN "00"      => o      <= "10001110100011"; -- "Lo"
8             WHEN "01"      => o      <= "01111110111111"; --
9             WHEN "11"      => o      <= "00010011101111"; -- "Hi"
10            WHEN "10"     => o      <= inA; -- Numbers from input
11            WHEN OTHERS => o      <= inA;
12        END CASE;
13    END PROCESS; -- muxProcess
14 END muxFour; -- muxFour

```

Listing 1.11: Mux4 arkitektur

Ved at udvide multiplexeren, kunne der ligeledes laves en multiplexer, source code table 1.11, der håndterede visning af hvad der vises på 10'ere og 1'ere på de 2 hexdisplays. Dette vil enten være det numeriske input eller en af de *hardcoded* tre outputs ind :*Hi*, *Lo* og *-*

Resultater

Guess Game blev testet i en testbench, som ses på source code table 1.12.

```

1 LIBRARY ieee;
2 USE ieee.std_logic_1164.ALL;
3 USE work.ALL;
4
5 ENTITY guess_game_tester IS
6     PORT (
7         SW    : IN std_logic_vector(7 DOWNTO 0);
8         KEY   : IN std_logic_vector(2 DOWNTO 0);
9         HEX0  : OUT std_logic_vector(6 DOWNTO 0);
10        HEX1  : OUT std_logic_vector(6 DOWNTO 0)
11    );
12 END guess_game_tester;
13
14 ARCHITECTURE testbench OF guess_game_tester IS
15 BEGIN
16     uut : ENTITY guess_game
17         PORT MAP
18     (
19         inputs => SW,
20         set      => KEY(0),
21         try      => KEY(1),
22         show     => KEY(2),
23         hex1    => HEX0,
24         hex10   => HEX1
25     );
26 END testbench;

```

Listing 1.12: Guess Game testbench

I både fig. 1.3 og fig. 1.4, ses implementationen af de forskellige elementer.

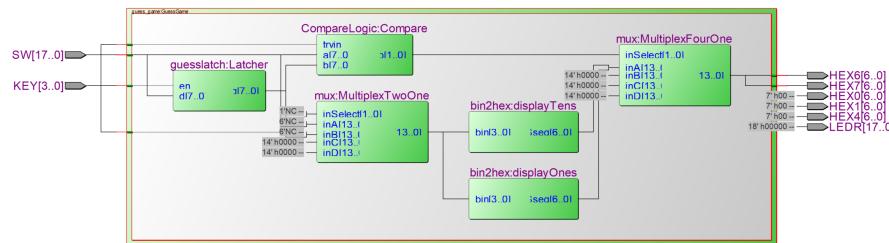


Fig. 1.3: RTL view: Guess game

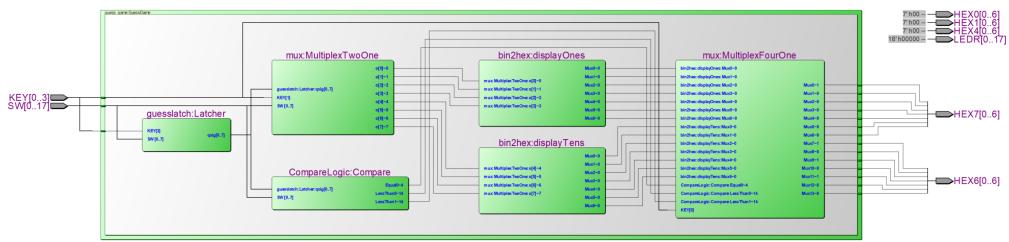
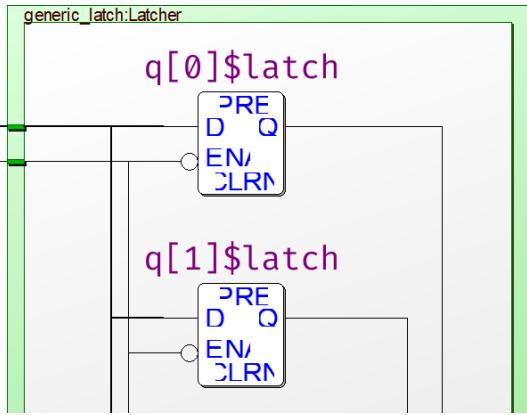
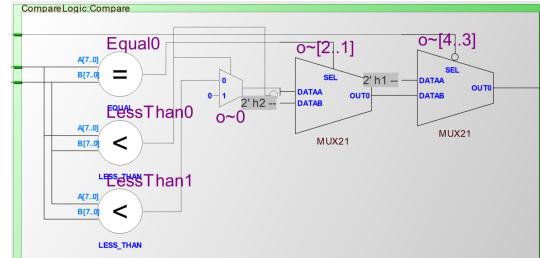


Fig. 1.4: Postmapping view: Guess game

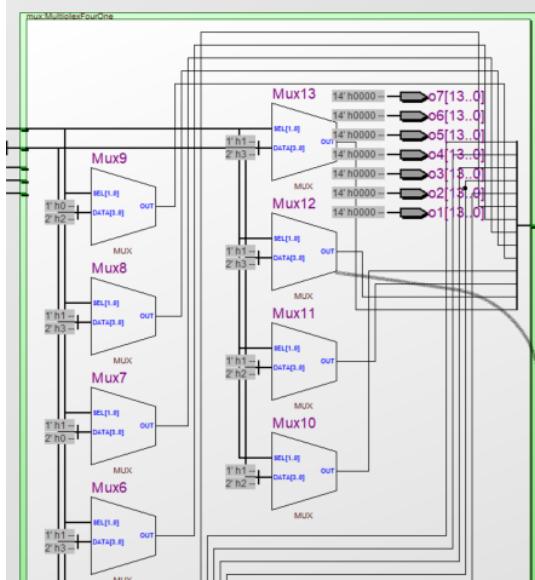
I fig. 1.5 ses udsnit af de forskellige elementers RTL views.



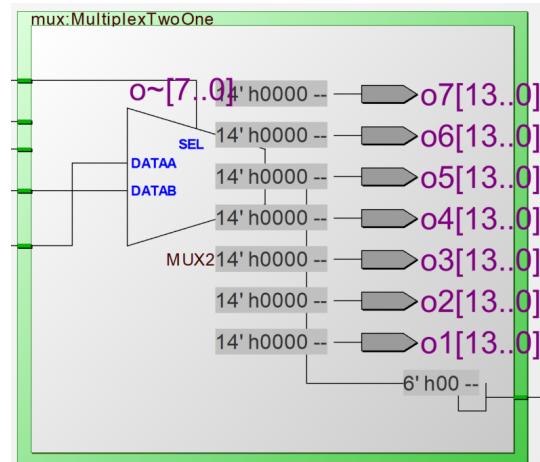
(a) Latch entity



(b) Compare entity



(c) Muxfour entity



(d) Muxtwo entity

Fig. 1.5: Udsnit af RTL views fra de forskellige elementer i Guess Game

Latches

Vi ser ud fra compilation report i Listing 1.13 at der bliver lavet 8 inferred latches.

```

1 Warning (10631): VHDL Process Statement warning at latch.vhd(18): inferring
    latch(es) for signal or variable "q", which holds its previous value in
    one or more paths through the process
2 Info (10041): Inferred latch for "q[0]" at latch.vhd(18)
3 Info (10041): Inferred latch for "q[1]" at latch.vhd(18)
4 Info (10041): Inferred latch for "q[2]" at latch.vhd(18)
5 Info (10041): Inferred latch for "q[3]" at latch.vhd(18)
6 Info (10041): Inferred latch for "q[4]" at latch.vhd(18)
7 Info (10041): Inferred latch for "q[5]" at latch.vhd(18)
8 Info (10041): Inferred latch for "q[6]" at latch.vhd(18)
9 Info (10041): Inferred latch for "q[7]" at latch.vhd(18)

```

Listing 1.13: Udsnit af output report

Desuden har vi testet spillet på vores DE2 board og observeret input-output relationerne som set i fig. 1.6.

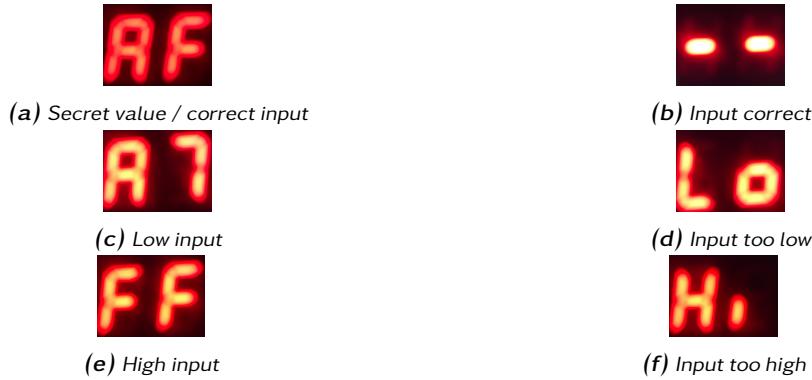


Fig. 1.6: De forskellige HEX outputs vist.

Diskussion

Som vi så i compilation report i Listing 1.13 blev der lavet 8 inferred latches, hvilket stemmer overens med vores forventning. Hele ideen med vores latch komponent er jo at den skal gemme en værdi som et register på n bits, hvor vi i dette tilfælde har sat $n = 8$ og derfor forventede 8 latches.

Konklusion

Vi har haft success med at bruge latches til at gemme værdi, hvilket er en meget grundlæggende ting at kunne for at lave mere komplekse sekventielle systemer.

Opgave 3: Two Player Guess Game

Introduktion

I denne opgave udvides vores Guess Game således at det kan håndtere to spil simultant, dvs. at begge spillere kan indtaste et hemmeligt tal som den anden spiller skal forsøge at gætte.

Design og implementering

Den overordnede implementering er primært strukturel da vi allerede har lavet en indkapslet `guess_game` entity. En udgave med to spillere kræver blot 2 instanser af vores spil og multiplexing som bestemmer hvilket spil inputs blev sendt til og outputs bliver læst fra. Desuden tilføjes et ekstra 7-segment display for at vise hvilken spiller der er valgt på nuværende tidspunkt.

```

1 ENTITY two_player_guess_game IS
2   PORT (
3     inputs      : IN std_logic_vector(7 DOWNTO 0);
4     playerSel  : IN std_logic;                      -- set predef. vals.
5     set         : IN std_logic;                      -- set predef. vals.
6     show        : IN std_logic;                      -- show predef. vals.
7     try         : IN std_logic;                      -- eval. guess
8     hexOnes    : OUT std_logic_vector(6 DOWNTO 0);  -- 7-seg ones
9     hexTens    : OUT std_logic_vector(6 DOWNTO 0);  -- 7-seg tens
10    hexPlayer  : OUT std_logic_vector(6 DOWNTO 0)  -- 7-seg player
11  );
12 END two_player_guess_game;

```

Listing 1.14: Two Player Guess Game Entity

```

1 ARCHITECTURE TwoPlayerGame OF two_player_guess_game IS
2
3     -- Player one signals
4     SIGNAL p1Show    : std_logic;
5     SIGNAL p1Set     : std_logic;
6     SIGNAL p1Input   : std_logic_vector(7 DOWNTO 0);
7     SIGNAL p1Try     : std_logic;
8     SIGNAL p1Output  : std_logic_vector(13 DOWNTO 0);
9     -- Player two signals
10    SIGNAL p2Show    : std_logic;
11    SIGNAL p2Set     : std_logic;
12    SIGNAL p2Input   : std_logic_vector(7 DOWNTO 0);
13    SIGNAL p2Try     : std_logic;
14    SIGNAL p2Output  : std_logic_vector(13 DOWNTO 0);

```

Listing 1.15: Her ses begyndelsen på Two player Guess Game arkitekturen, samt de signaler der oprettes

```

1 displayPlayer : ENTITY bin2hex(Behavioral)
2     PORT MAP
3     (
4         -- INPUTS
5         bin(0) => playerSel,
6         -- OUTPUTS
7         seg => hexPlayer
8     );
9 playerOneGame : ENTITY guess_game
10    PORT MAP
11    (
12        inputs(7 DOWNTO 0) => p1Input, -- => (OTHERS => '0'),
13        set                  => p1Set,
14        show                 => p1Show,
15        try                  => p1Try,
16        -- OUTPUTS
17        hex1    => p1Output(6 DOWNTO 0),
18        hex10   => p1Output(13 DOWNTO 7)
19    );
20
21 playerTwoGame : ENTITY guess_game
22     PORT MAP
23     (
24        inputs(7 DOWNTO 0) => p2Input, -- => (OTHERS => '0'),
25        set                  => p2Set,
26        show                 => p2Show,
27        try                  => p2Try,
28        -- OUTPUTS
29        hex1    => p2Output(6 DOWNTO 0),
30        hex10   => p2Output(13 DOWNTO 7)
31    );

```

Listing 1.16: Her ses spillerdisplayer, og de to Guess Games

```

1  MultiplexPlayerSelect : ENTITY mux(mux4In)
2    PORT MAP
3    (
4      -- INPUTS
5      inSelect(0)      => playerSel,
6      inA(0)           => show,
7      inB(0)           => set,
8      inC(7 DOWNTO 0)  => inputs,
9      inD(0)           => try,
10     -- OUTPUTS
11     ---- P1
12     o(0)             => p1Show,
13     o1(0)            => p1Set,
14     o2(7 DOWNTO 0)   => p1Input,
15     o3(0)            => p1Try,
16     ---- P2
17     o4(0)            => p2Show,
18     o5(0)            => p2Set,
19     o6(7 DOWNTO 0)   => p2Input,
20     o7(0)            => p2Try
21   );
22
23 MultiplexShowPlayer : ENTITY mux(muxTwo)
24   PORT MAP
25   (
26     -- INPUTS
27     inSelect(0)      => playerSel,
28     inA(13 DOWNTO 0) => p1Output(13 DOWNTO 0),
29     inB(13 DOWNTO 0) => p2Output(13 DOWNTO 0),
30     inC => (OTHERS => '0'),
31     inD => (OTHERS => '0'),
32     -- OUTPUTS
33     o(6 DOWNTO 0)   => hexOnes(6 DOWNTO 0),
34     o(13 DOWNTO 7)  => hexTens(6 DOWNTO 0),
35     o1               => OPEN,  -- Unused
36     o2               => OPEN,  -- Unused
37     o3               => OPEN,  -- Unused
38     o4               => OPEN,  -- Unused
39     o5               => OPEN,  -- Unused
40     o6               => OPEN,  -- Unused
41     o7               => OPEN  -- Unused
42   );
43 END TwoPlayerGame; -- TwoPlayerGame

```

Listing 1.17: Her ses de to multiplexers der holder styr på spillerens valg.

De to multiplexers der ses i source code table 1.17 blev implementeret vha. CASE statements, den ene er vist i source code table 1.18, og den anden er vist tidligere i source code table ???. Vi vælger CASE statements da der basalt set er en mux, med et select signal og forskellige outputs baseret på dette signal.

```

1 ARCHITECTURE mux4In OF mux IS
2 BEGIN
3     muxProcesser : PROCESS (inSelect, inA, inB, inC, inD)
4     BEGIN
5         CASE(inSelect(0)) IS
6             WHEN '1' =>
7                 o(0)          <= inA(0); --
8                 o1(0)         <= inB(0); --
9                 o2(7 DOWNTO 0) <= inC(7 DOWNTO 0); --
10                o3(0)         <= inD(0); --
11            WHEN '0' =>
12                o4(0)          <= inA(0);
13                o5(0)          <= inB(0);
14                o6(7 DOWNTO 0) <= inC(7 DOWNTO 0);
15                o7(0)          <= inD(0);
16        END CASE;
17    END PROCESS;
18 END mux4In; -- mux4In

```

Listing 1.18: Her ses de den ene multiplexer der holder styr på spillerens valg.

Resultater

2-player spillet testes med en testbench hvor mapping til DE2 boardet kan ses på source code table 1.19.

```

1 TwoPlayerGuessGame : ENTITY two_player_guess_game
2     PORT MAP
3     (
4         inputs      => SW(7 DOWNTO 0),      -- inputs
5         playerSel  => SW(17),           -- Player Select
6         set         => key(3),           -- set predef. vals.
7         show        => key(1),           -- show predef. vals.
8         try         => key(0),           -- eval. guess
9         hexPlayer   => HEX6(6 DOWNTO 0), -- 7-seg ones
10        hexOnes    => HEX0(6 DOWNTO 0), -- 7-seg ones
11        hexTens    => HEX1(6 DOWNTO 0)  -- 7-seg tens
12    );

```

Listing 1.19: Test mapping til 2-player Guess Game

Latches

I vores compilation report ser vi de samme *inferred latches* som i Listing 1.13 og derudover også dem set i Listing 1.20. Dykker vi ind i compilation reporten ser vi som på fig. 1.7 at der er lavet 38 latches.

```

1 Info (10041): Inferred latch for "o7[0]" at mux.vhd(55)
2 Info (10041): Inferred latch for "o6[0]" at mux.vhd(55)
3 Info (10041): Inferred latch for "o6[1]" at mux.vhd(55)
4 Info (10041): Inferred latch for "o6[2]" at mux.vhd(55)
5 Info (10041): Inferred latch for "o6[3]" at mux.vhd(55)
6 Info (10041): Inferred latch for "o6[4]" at mux.vhd(55)
7 Info (10041): Inferred latch for "o6[5]" at mux.vhd(55)
8 Info (10041): Inferred latch for "o6[6]" at mux.vhd(55)
9 Info (10041): Inferred latch for "o6[7]" at mux.vhd(55)
10 Info (10041): Inferred latch for "o5[0]" at mux.vhd(55)
11 Info (10041): Inferred latch for "o4[0]" at mux.vhd(55)
12 Info (10041): Inferred latch for "o3[0]" at mux.vhd(55)
13 Info (10041): Inferred latch for "o2[0]" at mux.vhd(55)
14 Info (10041): Inferred latch for "o2[1]" at mux.vhd(55)
15 Info (10041): Inferred latch for "o2[2]" at mux.vhd(55)
16 Info (10041): Inferred latch for "o2[3]" at mux.vhd(55)
17 Info (10041): Inferred latch for "o2[4]" at mux.vhd(55)
18 Info (10041): Inferred latch for "o2[5]" at mux.vhd(55)
19 Info (10041): Inferred latch for "o2[6]" at mux.vhd(55)
20 Info (10041): Inferred latch for "o2[7]" at mux.vhd(55)
21 Info (10041): Inferred latch for "o1[0]" at mux.vhd(55)
22 Info (10041): Inferred latch for "o[0]" at mux.vhd(55)

```

Listing 1.20: Udsnit af output report

User-Specified and Inferred Latches	
	Latch Name
1	Number of user-specified and inferred latches = 38

Fig. 1.7: Antallet af inferred latches efter syntetisering

Diskussion

Da der er to spil må der nødvendigvis være minimum 16 *inferred latches* da hver spil gemmer en 8-bit værdi. De resterende 22 latches forklares med vores `mux4In`: samlet set er inputtet til hvert spil 11 bit. Da muxen skifter mellem de to spil gemmes input værdierne for det spil der ikke er aktivt, således at det inaktive spil ikke ændrer sig før det bliver aktivt igen. Her kunne man måske have valgt at lave et specifikt register komponent til det inaktive spil, således at der kun blev oprettet 11 ekstra latches, i stedet for 22.

Ved at bruge `CASE` statements, kunne vi implementere den multiplexer der holder styr på hvilken spiller der i brug, direkte i den multiplexer entity der allerede var lavet. Desuden kunne en tidlige multiplexer genbruges.

Konklusion

Vi fik det til at spille, men selvom det er nemt at sammensætte små isolerede dele, så kan det stadig hurtigt blive et komplekst design, som måske kunne forbedres med lidt ekstra arbejde.

Opgave 4: 8-input NAND Gate

Introduktion

I denne opgave skal der implementeres en 8-input NAND-gate. Dette giver anledning til at afprøve VHDLs `FOR-LOOP`. Desuden er det også en anledning til at bruge `generics` for at implementere en n -input NAND-gate.

Design og implementering

I vores design er vi sprunget direkte til en generisk implementation. Dette ses på source code table 1.21 hvor det på linje 2 ses at vi har brugt `GENERIC` til at gøre denne entity skalerbar. Den generiske

værdi har en default på 8, således at NAND-gaten er 8-input medmindre andet specificeres. Værdi `bits` bruges til at sætte input signal størrelsen på linje 4 og til at definere vores loop range på linje 16.

```

1 ENTITY nand_8 IS
2   GENERIC (bits : POSITIVE := 8);
3   PORT (
4     a : IN std_logic_vector(bits - 1 DOWNTO 0);
5     y : OUT std_logic
6   );
7 END ENTITY nand_8;
8
9 ARCHITECTURE genericType OF nand_8 IS
10 BEGIN
11   PROCESS (a)
12     VARIABLE nandOut : std_logic;
13   BEGIN
14     nandOut := a(0);
15
16     FOR I IN 1 TO (bits - 1) LOOP
17       nandOut := nandOut AND a(I);
18     END LOOP;
19
20     y <= NOT(nandOut);
21   END PROCESS;
22 END genericType;

```

Listing 1.21: Generic input NAND-gate entity og arkitektur

Resultater

NAND-gaten kobles til switches og en LED i en testbench som set i source code table 1.22.

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 USE work.ALL;
5
6 ENTITY nand_8_tester IS
7   PORT (
8     SW : IN std_logic_vector(7 DOWNTO 0);
9     LEDR : OUT std_logic_vector(0 DOWNTO 0)
10   );
11 END ENTITY nand_8_tester;
12
13 ARCHITECTURE testbench OF nand_8_tester IS
14 BEGIN
15
16   UUT : ENTITY nand_8
17     PORT MAP(
18       a => SW,
19       y => LEDR(0)
20     );
21
22 END ARCHITECTURE testbench;

```

Listing 1.22: Testbench til 8-input NAND-gate

RTL-view for både en 8-input og 32-input NAND-gate ses på henholdsvis fig. 1.8 og fig. 1.9.

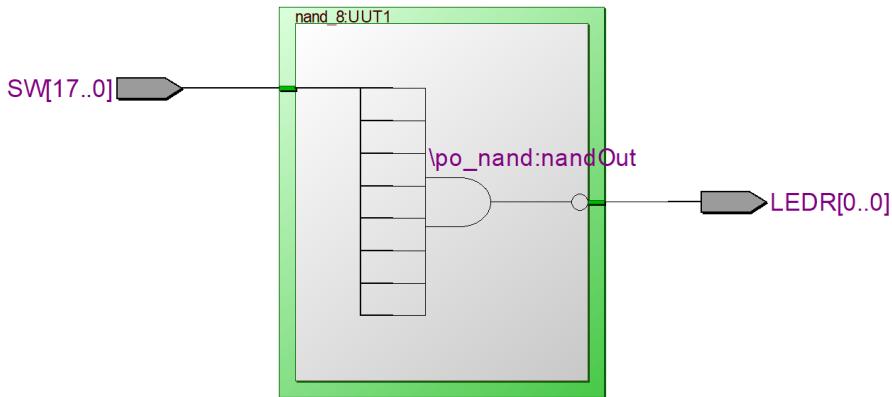


Fig. 1.8: RTL view: 8 bit NAND

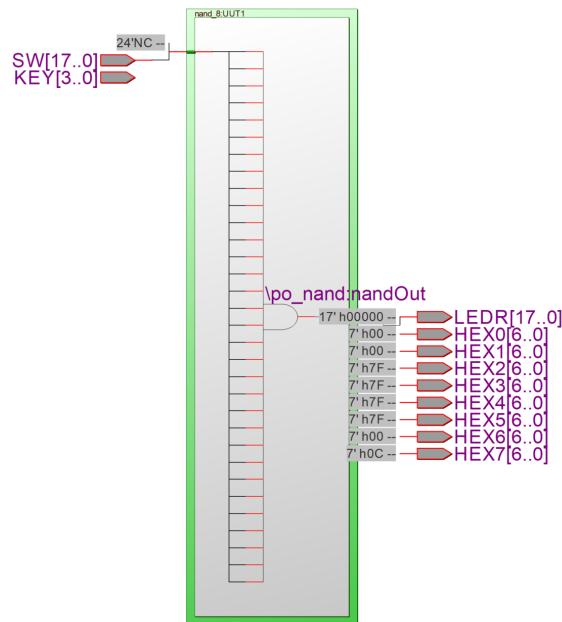


Fig. 1.9: RTL view: 32 bit NAND

Desuden har vi lavet en funktionel simulation for at teste at komponenten virker korrekt. Simulationen blev lavet med grænseværdierne `00000000` og `11111111`, og nogle tilfælde værdier ind imellem. Den funktionelle test ses på fig. 1.10.

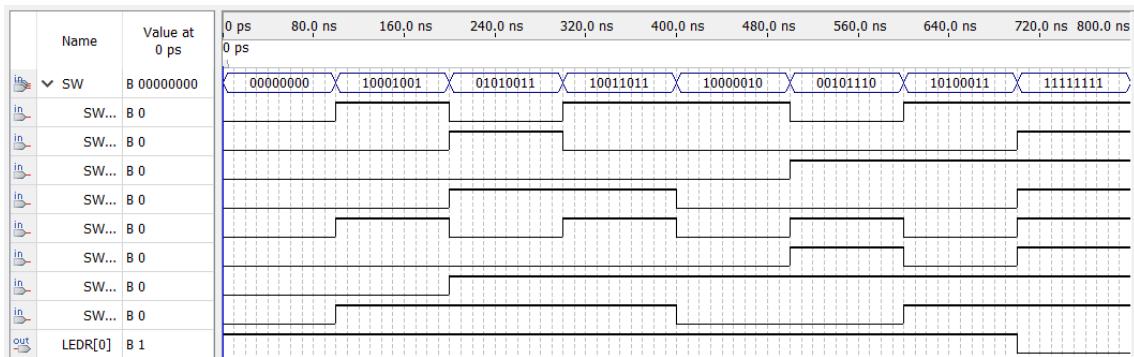


Fig. 1.10: Funktionel simulation af 8-input NAND

Diskussion

Ud fra den funktionelle test ser vi at komponenten virker som den skal, altså den holder 1 på outputtet i alle situation undtagen den hvor alle inputs er 1. Dette stemmer overens med sandhedstabellen for NAND-gates.

Vi ser at det vha. FOR-LOOP er nemt at bygge delblokke der gentager det samme logik mange gange, vi ser netop i RTL-viewene at vores loops bliver mappet direkte over på n -input AND-gates og en inverter, som det forventes af en NAND-gate. Et loop spiller desuden godt sammen med generiske entities, da de generiske værdier kan bruges til at definere ranges. Vi ser af den grund at det er nemt med et loop at implementere n -input gates af mange forskellige slags!

Konklusion

Vi fik løst opgaven om at implementere en 8-input NAND-gate og desuden gjorde vi dette ved at lave den generisk, så denne NAND-gate kan skaleres til n -input, hvor mange man nu lige skal bruge.

Opgave 5: Count Ones

Introduktion

I denne del af øvelsen vil vi forsøge at tælle antallet af 1'ere i en array af data vha. et FOR-LOOP.

Design og implementering

Vores implementation baseret på opgaven ses i source code table 1.23 og 1.24. Processen har SW i sin sensitivity liste, hvilket vil sige den kører hver gang SW ændrer sig. FOR-LOOP'et gennemgår alle bits i input signalet og tæller en intern process variabel op hvis der er et 1-tal. Talte 1'ere assignes til et signal i arkitekturen som bruges til at vise tallet på et 7-segment display.

```
1 ENTITY count_ones IS
2     PORT (
3         SW    : IN std_logic_vector(7 DOWNTO 0);
4         HEX0 : OUT std_logic_vector(6 DOWNTO 0)
5     );
6 END ENTITY count_ones;
```

Listing 1.23: Interface til count_ones entity

```

1 ARCHITECTURE behavioural OF count_ones IS
2     SIGNAL ones_counted : std_logic_vector(3 DOWNTO 0);
3 BEGIN
4
5     PROCESS (SW)
6         VARIABLE count : unsigned(3 DOWNTO 0);
7     BEGIN
8         count := (OTHERS => '0');
9
10        FOR i IN 7 DOWNTO 0 LOOP
11            IF SW(i) = '1' THEN
12                count := count + 1;
13            END IF;
14        END LOOP;
15
16        ones_counted <= std_logic_vector(count);
17    END PROCESS;
18
19    b2h : ENTITY bin2hex
20        PORT MAP
21        (
22            bin => ones_counted,
23            seg => HEX0
24        );
25 END ARCHITECTURE;

```

Listing 1.24: Arkitektur til optælling af 1'ere vha. FOR-LOOP

Resultater

På RTL-viewet for vores `count_ones` på fig. 1.11 ses det at komponent laves som en kæde af additoner og multiplexere, der bruger vores input som deres selects.

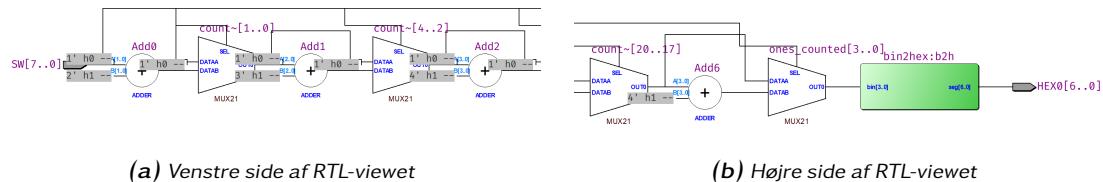


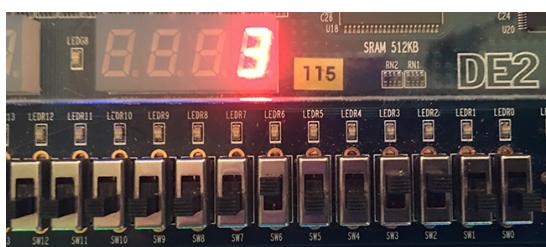
Fig. 1.11: Venstre og højre side af count_ones RTL-view

Vi har testet vores 1-tæller på DE2 boardet og har fået resultaterne som set i tab. 2.

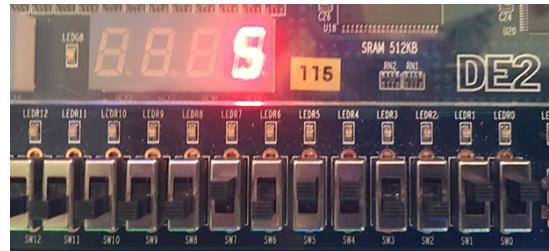
Input	Forventet på display	Reelt på display
01001100	3	3
10110011	5	5

Tab. 2: Test case for 1-tællereren

På fig. 1.12 ses vores test på DE2 svarende til test casene ovenfor.



(a) Test case 1



(b) Test case 2

Fig. 1.12: Test på DE2 board

Diskussion

Vi ser at count_ones virker som forventet og tæller hvor mange 1'ere der er på input signalet (altså hvor mange switches er slået til).

FOR-LOOP giver god mening når man vil gentage ikke-trivial kode et bestemt antal gange. Som set tidligere gør sådanne loops det også nemt at lave generiske komponenter. I denne opgave kunne man nemt introducere en GENERIC til at skalere input signalet og loopen.

Konklusion

Vi har set at man i loops også sagtens kan bruge behavioural statements så som IF og dermed kan bruge loops til at skrive kompleks kode der skal gentages en bestemt antal gange.

ØVELSE 2: SEQUENTIAL DESIGNS IN VHDL (THE WATCH EXERCISE)

Indhold

Opgave 1. Counter - One Digit.....	25
Opgave 2. Clock - One Digit	28
Opgave 3. Counter - Six Digit	31
Opgave 4. Alarm Watch.....	39
Opgave 5. Peer Feedback: Journal 2	46

Opgave 1: Counter - One Digit

Introduktion

Denne opgave handler om at skabe en clock tæller som baseret på `mode` kan tælle enten `[0:2]`, `[0:5]` eller `[0:9]`.

Design og implementering

Implementationen af tæller ses i source code table 2.1 og 2.2. For at reagere på et clock signal skriver vi arkitekturen som behavioural style, altså en process med signalet `clk` i sensitivity listen. For at sikre at der kun tælles på en rising edge vil vi bruge funktionen `rising_edge()` fra `STD_LOGIC_1164` biblioteket.

På en *rising edge* opdateres den maksimale tæller-værdi baseret på nuværende `mode`. Tælleren nulstilles hvis den ville overskride maksimal værdien, ellers tæller den op.

```
1 ENTITY multi_counter IS
2   PORT (
3     --- Inputs
4     clk    : IN std_logic;
5     reset : IN std_logic;
6     mode   : IN std_logic_vector(1 DOWNTO 0);
7     --- Outputs
8     count  : OUT std_logic_vector(3 DOWNTO 0);
9     cout    : OUT std_logic
10    );
11 END ENTITY multi_counter;
```

Listing 2.1: Interface for multi_counter entitien

```

1 ARCHITECTURE ThreeMode OF multi_counter IS
2     SIGNAL internal_count : unsigned(3 DOWNTO 0) := "0000";
3 BEGIN
4     PROCESS (clk, reset)
5         VARIABLE maxValue : unsigned(3 DOWNTO 0);
6     BEGIN
7         IF reset = '0' THEN
8             -- reset the counter
9             internal_count <= "0000";
10            cout <= '0';
11        ELSIF rising_edge(clk) THEN
12            -- set the maximum counter value
13            CASE mode IS
14                WHEN "00" => maxValue := to_unsigned(9, maxValue'length);
15                WHEN "01" => maxValue := to_unsigned(5, maxValue'length);
16                WHEN OTHERS => maxValue := to_unsigned(2, maxValue'length);
17            END CASE;
18
19            -- update counter
20            IF (internal_count + 1) > maxValue THEN
21                internal_count <= "0000";
22                cout <= '1';
23            ELSE
24                internal_count <= internal_count + 1;
25                cout <= '0';
26            END IF;
27        END IF;
28    END PROCESS;
29
30    count <= std_logic_vector(internal_count);
31 END;

```

Listing 2.2: Her ses multi_counter digit arkitekturen, hvor bla. mode og reset implementers

Resultater

Vores `multi_counter` testes med testbenchen som set i source code table 2.3.

```

1 ENTITY multi_counter_tester IS
2 PORT (
3     SW : IN std_logic_vector(17 DOWNTO 16);
4     KEY : IN std_logic_vector(3 DOWNTO 0);
5     HEX0 : OUT std_logic_vector(6 DOWNTO 0);
6     LEDR : OUT std_logic_vector(0 DOWNTO 0)
7 );
8 END ENTITY multi_counter_tester;
9
10 ARCHITECTURE testbench OF multi_counter_tester IS
11     SIGNAL countOut : std_logic_vector(3 DOWNTO 0);
12 BEGIN
13     uut0 : ENTITY multi_counter
14         PORT MAP
15     (
16         clk      => KEY(0),
17         mode    => SW,
18         reset   => KEY(3),
19         count   => countOut,
20         cout    => LEDR(0)
21     );
22     uut1 : ENTITY bin2hex
23         PORT MAP
24     (
25         bin      => countOut,
26         seg     => HEX0
27     );
28 END ARCHITECTURE;

```

Listing 2.3: Testbench til multi_counter

Funktionelle simulationer blev lavet for at verificere at implementeringen virker som forventet. Fig. 2.1 viser først simulation med almindelig optælling og rollover til 0 som sætter cout høj. Fig. 2.2 viser anden simulation som sætter den asynkrone reset midt i optælling. Fig. 2.3 viser trejde simulation som skifter mode til et mode med en lavere maksimum værdi.

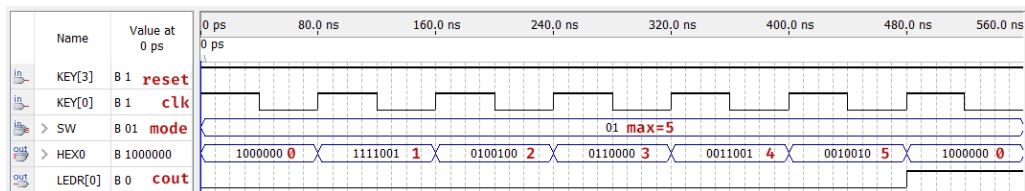


Fig. 2.1: Funktionel simulation 1: almindelig optælling

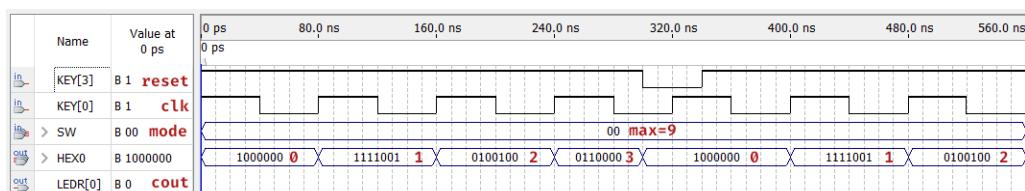


Fig. 2.2: Funktionel simulation 2: reset sættes midt i optælling

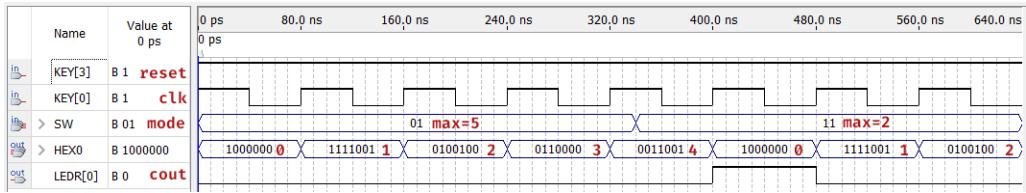


Fig. 2.3: Funktionel simulation 3: skift af mode

Diskussion

Vi ser ud fra vores funktionelle simulationer at `multi_counter` komponenten virker efter hensigten. Funktionen `rising_edge` detekterer korrekt vores clock signal som er koblet til en knap.

Konklusion

VHDL gør det nemt at lave programmer der reagerer på clock signaler. Desuden gør IF statements i en clock'd process det simpelt at implementere synkrone og/eller asynkrone inputs, da der er en indbygget prioritering.

Opgave 2: Clock - One Digit

Introduktion

Denne opgave handler om at skabe en *synchronous clock generator* med *asynchronous reset* som baseret på `speed` og `CLOCK_50` kan tælle i med en konstant hastighed hhv. 1s og 500ms.

Design og implementering

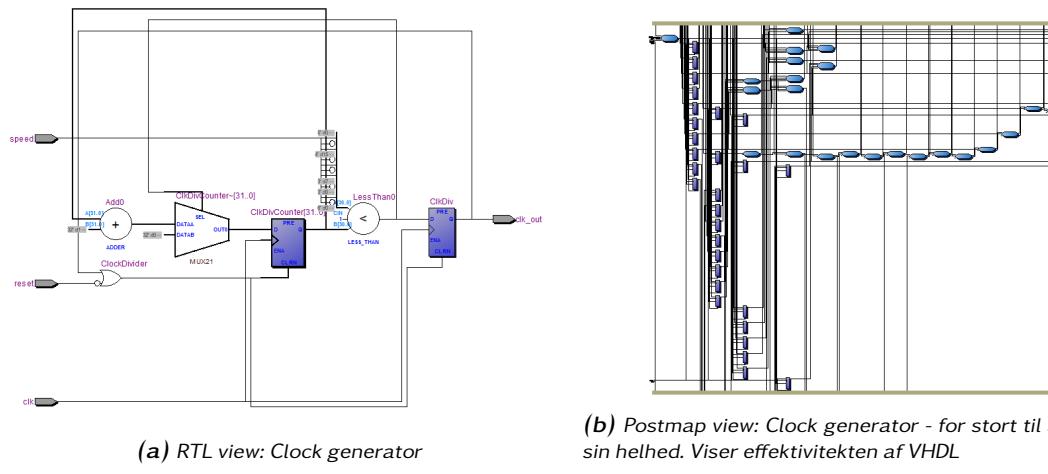


Fig. 2.4: Netlist views af clockgeneratoren

Implementationen af *clock generatoren* ses i source code table 2.4 og 2.5. For at reagere på et clock signal skriver vi igen arkitekturen som behavioural style. For at sikre at der kun tælles på en rising edge vil vi bruge funktionen `rising_edge()` fra `STD_LOGIC_1164` biblioteket. På en rising edge opdateres den maksimale tæller-værdi baseret på nuværende `speed`. *clock generatoren* nulstilles overskriden den valgte værdi, og ellers tæller den op. For at udregne maksimalværdien, gjorde vi flg:

$$\text{speedSelector} = \text{frequency} \cdot \text{duration} \quad (1)$$

$$50 \cdot 10^6 = \frac{1}{50\text{MHz}} \cdot 1\text{s} \quad (2)$$

$$25 \cdot 10^6 = \frac{1}{50\text{MHz}} \cdot 0.5\text{s} \quad (3)$$

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 ENTITY clock_gen IS
5     PORT
6     (
7         -- inputs
8         clk      : IN std_logic;
9         reset    : IN std_logic;
10        speed   : IN std_logic;
11        -- outputs
12        clk_out : OUT std_logic
13    );
14 END ENTITY clock_gen;
15 ARCHITECTURE ClockDivider OF clock_gen IS
16
17    -- Signals to be used:
18    SIGNAL ClkDivCounter : std_logic_vector(31 DOWNTO 0) := (OTHERS => '0');
19    -- For clock divisons.
20    SIGNAL ClkDiv          : std_logic                      := '0';
21    -- Positive vs negative period
22
23 BEGIN
24     ClockDivider : PROCESS (clk, reset)
25         VARIABLE speedSelector : INTEGER;
26     BEGIN
27         -- speed selection
28         CASE (speed) IS
29             WHEN '0'      => speedSelector      := 25e6;  -- 0.5 sec clock
30             WHEN '1'      => speedSelector      := 50e6;  -- 1.0 sec clock
31             WHEN OTHERS => speedSelector      := 100e6; -- 2.0 sec clock for
32             errors
33         END CASE;

```

Listing 2.4: Clock generatoren entity og start på arkitekturen (1 af 2)

Hastighed på clock

Ved at lave `speedSelector` som en variable integer, har vi mulighed for at skifte hastighed på `clock'en`. Denne er lavt en `casestatemetent`, hvilket åbner for en mulighed for udvidelser af selektoren — der kan let laves flere muligheder, ved at tilføje flere *bits* til `speed`inputtet. Under senere test af de forskellige moduler blev '`0`' valget sat til: `10e3` hvilket gav en clockperiode på $200\mu\text{s}$.

```

1      -- reset on speed limit
2  IF rising_edge(clk) THEN
3      --50MHz clock counting to sel. speed.
4      IF (to_integer(unsigned(ClkDivCounter)) >= speedSelector) THEN
5          --Reset clock when limit is reached.
6          ClkDivCounter <= (OTHERS => '0');
7          ClkDiv         <= '1'; --Set output hi
8      ELSE
9          --Increment - else keep counting
10         ClkDivCounter <= std_logic_vector(unsigned(ClkDivCounter) + 1);
11         -- with a low output
12         ClkDiv         <= '0';
13     END IF;
14
15 END IF;
16 --hard reset / reset is activated.
17 IF (reset = '0' OR ClkDiv = '1') THEN
18     -- reset counter to 0..
19     ClkDivCounter <= (OTHERS => '0');
20     -- with a low output
21     ClkDiv         <= '0';
22 END IF;
23 END PROCESS;
24 -- assign the generated clock as main clock.
25 clk_out <= ClkDiv;
26 END ARCHITECTURE ClockDivider;

```

Listing 2.5: Clock generatoren speed select og reset (2 af 2)

Clock logikken

For hver *rising_edge* bliver der tjekket om *ClkDivCounteren* har nået *speedSelector*værdien. Hvis den er større end eller lig *speedSelector*værdien, bliver den nulstillet og *clockoutputtet* sættes højt.

Efter nulstilling vil *ClkDivCounteren* givet være lavere end *speedSelector*værdien, og den inkrementeres med 1, og *outputtet* holdes lavt.

Ovenstående process gentages ved en frekvens af 50MHz og sker hele tiden, meget hurtigt og er en synkronproces da det styres af en *rising_edge* drevet proces.

Reset logikken

For at kunne nulstille Counteren UDEN for *enrising_edge()* implementeres en hard reset. Denne undersøger om *reset* knappen er aktiveret. Hvis dette er tilfældet nulstilles counteren og *clockoutputtet* sættes lavt.

Resultater

Fig. 2.4a viser noget af det interessante ved VHDL synthesizing — Clock generatoren består réelt set af 6 elementer: 2 logiske elementer, *Add* og *Less than*, 1 enkelt multiplexer, en *OR-gate* med et inveret input, og så to *flipflop*-kredse, hvor af den ene fungere som *clockoutput*. Fig. 2.4b viser effektivitekten af VHDL's synthesizer og konceptet bag *FPGA*boards hardwaremuligheder.

Diskussion

Clockgeneratoren virker efter hensigten, og tæller uden problemer op.

Konklusion

Vha. VHDL's biblioteker *STD_LOGIC_1164* og *NUMERIC_STD* kan en clock generator implementeres på et *FPGA* board. Denne kan enkelt sættes til en specifik frekvens.

Opgave 3: Counter - Six Digit

Introduktion

I denne opgave vil vi kombinere vores `multi_counter` fra opgave 1 og `clock_gen` fra opgave 2 til at lave et ur der viser timer, minutter og sekunder på seks 7-segment displays.

Design og implementering

I designet af uret kaskadekobles seks counters, to til henholdsvis sekunder, minutter og timer. Clock generatoren bruges som clock signal til den 1'er sekund counteren. 1'er sekund counterens carry bruges som clock til 10'er sekund counteren osv. op til timernes 10'er counter.

Indkapslet håndtering af 1'er og 10'er

Da parene for 1'er og 10'er counters fungerer ens for sekunder, minutter og timer, har vi for nemheds skyld indkapslet dem i en entity til at håndtere begge samtidig, denne ses på source code table 2.6.

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 USE WORK.ALL;
5 ENTITY TwoCounters IS
6   PORT (
7     clkIn      : IN std_logic;                      --! Clockinput
8     modeOnes  : IN std_logic_vector(1 DOWNTO 0);    --! Mode select
9     modeTens  : IN std_logic_vector(1 DOWNTO 0);    --! Mode select
10    resetIn   : IN std_logic;                       --! Reset in // Active
11    low
12    segOnesOut : OUT std_logic_vector(6 DOWNTO 0);  --! Display output
13    segTensOut : OUT std_logic_vector(6 DOWNTO 0);  --! Display output
14    countOnes : OUT std_logic_vector(3 DOWNTO 0);  --! Ones binary output
15    countTens : OUT std_logic_vector(3 DOWNTO 0);  --! Tens binary output
16    coutOut   : OUT std_logic
17  );
18 END ENTITY TwoCounters;
19
20 ARCHITECTURE rtl OF TwoCounters IS
21   SIGNAL clock_int      : std_logic;
22   SIGNAL displayOnesCount : std_logic_vector(3 DOWNTO 0);
23   SIGNAL displayTensCount : std_logic_vector(3 DOWNTO 0);
24 BEGIN
25   --- count ones
26   MultiCounterOnes : ENTITY multi_counter(ThreeMode)
27     PORT MAP
28   (
29     clk      => clkIn,
30     reset   => resetIn,
31     mode    => modeOnes,
32     count   => displayOnesCount,
33     cout    => clock_int
34   );
35   --- count tens
36   MultiCounterTens : ENTITY multi_counter(ThreeMode)
37     PORT MAP
38   (
39     clk      => clock_int,
40     reset   => resetIn,
41     mode    => modeTens,
42     count   => displayTensCount,
43     cout    => coutOut
44   );
45   -- display ones
46   HexdisplayOnes : ENTITY bin2hex
47     PORT MAP
48   (
49     bin   => displayOnesCount,
50     seg   => segOnesOut
51   );
52   -- display tens
53   HexdisplayTens : ENTITY bin2hex
54     PORT MAP
55   (
56     bin   => displayTensCount,
57     seg   => segTensOut
58   );
59   CountOnes <= displayOnesCount;
60   CountTens <= displayTensCount;
61
62 END ARCHITECTURE rtl;

```

Listing 2.6: Entity til håndtering af sammenhørende 1'er og 10'er counters

Reset logik

Desuden er det nødvendigt at skrive noget ekstra reset logik, da uret både skal kunne resettes med en knap og når det ruller over til 24:00:00. Denne reset logik ses i source code table 2.7. Logikken får timernes 1'er og 10'er og et `reset_in` som input. Hvis `reset_in` er aktiv (active-low) eller timerne samlet set er lig med 24, bliver `reset_out` aktiv (også active-low) hvilket kobles til alle reset input på de andre delblokke.

```
1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4
5 ENTITY reset_logic IS
6   PORT
7   (
8     -- inputs
9     reset_in : IN std_logic;
10    hrs_bin1 : IN std_logic_vector(3 DOWNTO 0);
11    hrs_bin10 : IN std_logic_vector(3 DOWNTO 0);
12    -- outputs
13    reset_out : OUT std_logic
14  );
15 END ENTITY reset_logic;
16 ARCHITECTURE rtl OF reset_logic IS
17 BEGIN
18   twentyFourReset : PROCESS (hrs_bin1, hrs_bin10, reset_in)
19   BEGIN
20     IF ((hrs_bin10 = "0010" AND hrs_bin1 = "0100") OR (reset_in = '0'))
21     THEN
22       reset_out <= '0';
23     ELSE
24       reset_out <= '1';
25     END IF;
26   END PROCESS twentyFourReset;
27 END ARCHITECTURE rtl;
```

Listing 2.7: Reset logik til 24-timers uret

Samlet `watch`

Alle delblokkene sat sammen i en `watch` entity ses på source code table 2.8, 2.9, 2.10 og 2.11.

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 USE work.ALL;
5
6 ENTITY watch IS
7     PORT (
8         -- INPUTS
9         clk, reset, speed : IN std_logic;
10        -- OUTPUTS
11        sec_1,
12        sec_10,
13        min_1,
14        min_10,
15        hrs_1,
16        hrs_10 : OUT std_logic_vector(6 DOWNTO 0);
17        tm      : OUT std_logic_vector(15 DOWNTO 0)
18    );
19 END ENTITY watch;

```

Listing 2.8: Interface til `watch` entity

```

1 ARCHITECTURE rtl OF watch IS
2     SIGNAL secCarry      : std_logic;
3     SIGNAL minCarry      : std_logic;
4     SIGNAL ResetSignal   : std_logic;
5     SIGNAL minBinOnes    : std_logic_vector(3 DOWNTO 0);
6     SIGNAL minBinTens    : std_logic_vector(3 DOWNTO 0);
7     SIGNAL hrsBinOnes    : std_logic_vector(3 DOWNTO 0);
8     SIGNAL hrsBinTens    : std_logic_vector(3 DOWNTO 0);
9     SIGNAL ClkOut_internal : std_logic;

```

Listing 2.9: Begyndelse på arkitektur og definerede signaler

```

1   -- Main clock generator:
2   ClockGenerator : ENTITY clock_gen
3       PORT MAP
4   (
5       clk      => clk,
6       reset    => ResetSignal ,
7       speed    => speed ,
8       clk_out  => ClkOut_internal
9   );
10
11  -- Seconds counters
12  DualSecCounter : ENTITY TwoCounters
13      PORT MAP
14  (
15      clkIn     => ClkOut_internal ,
16      resetIn   => ResetSignal ,
17      modeOnes  => "00",
18      modeTens  => "01",
19      coutOut   => secCarry ,
20      segOnesOut=> sec_1,
21      segTensOut=> sec_10,
22      countOnes=> OPEN ,
23      countTens=> OPEN
24  );
25
26  --- Minutes counter
27  DualMinCounter : ENTITY TwoCounters
28      PORT MAP
29  (
30      clkIn     => secCarry ,
31      resetIn   => ResetSignal ,
32      modeOnes  => "00",
33      modeTens  => "01",
34      coutOut   => minCarry ,
35      segOnesOut=> min_1,
36      segTensOut=> min_10,
37      countOnes=> minBinOnes ,
38      countTens=> minBinTens
39
40  );

```

Listing 2.10: Arkitekturen for watch bestående af delblokke (1 af 2)

```

1   --- Hrs counter
2   DualHrsCounter : ENTITY TwoCounters
3   PORT MAP
4   (
5       clkIn      => minCarry ,
6       resetIn    => ResetSignal ,
7       modeOnes   => "00",
8       modeTens   => "11",
9       coutOut    => OPEN,
10      segOnesOut => hrs_1 ,
11      segTensOut => hrs_10 ,
12      countOnes  => hrsBinOnes ,
13      countTens  => hrsBinTens
14  );
15
16  -- Reset logic
17  ResetLogic : ENTITY reset_logic
18  PORT MAP(
19      -- input
20      reset_in  => reset ,
21      hrs_bin1  => hrsBinOnes ,
22      hrs_bin10  => hrsBinTens ,
23      -- output
24      reset_out => ResetSignal
25  );
26
27  --- tm section
28  tm <= (hrsBinTens & hrsBinOnes & minBinTens & minBinOnes);

```

Listing 2.11: Arkitekturen for `watch` bestående af delblokke (2 af 2)

Resultater

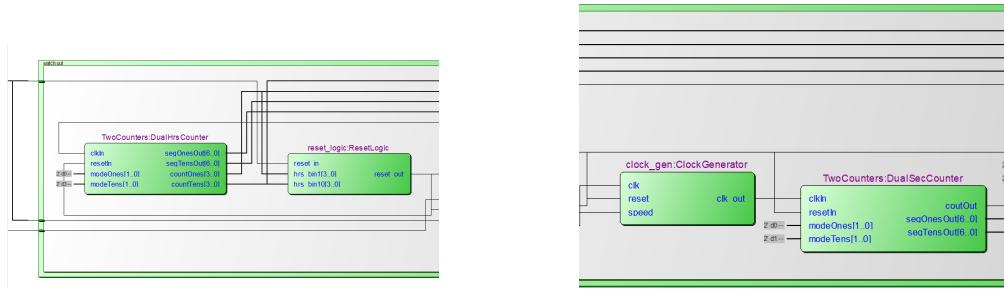
Implementationen testes vha. testbench i source code table 2.12 som producerede RTL-view som set på fig. 2.5.

```

1 LIBRARY IEEE;
2 USE IEEE.std_logic_1164.ALL;
3 USE IEEE.numeric_std.ALL;
4 USE work.ALL;
5
6 ENTITY watch_tester IS
7 PORT (
8     CLOCK_50 : IN std_logic;
9     KEY       : std_logic_vector(3 DOWNTO 0);
10    HEX2,
11    HEX3,
12    HEX4,
13    HEX5,
14    HEX6,
15    HEX7 : OUT std_logic_vector(6 DOWNTO 0)
16 );
17 END ENTITY watch_tester;
18
19 ARCHITECTURE testbench OF watch_tester IS
20 BEGIN
21
22     uut : ENTITY watch
23     PORT MAP(
24         -- INPUTS
25         clk      => CLOCK_50,
26         speed   => KEY(0),
27         reset   => KEY(3),
28         -- OUTPUTS
29         sec_1   => HEX2,
30         sec_10  => HEX3,
31         min_1   => HEX4,
32         min_10  => HEX5,
33         hrs_1   => HEX6,
34         hrs_10  => HEX7,
35         tm      => OPEN
36 );
37
38 END ARCHITECTURE;

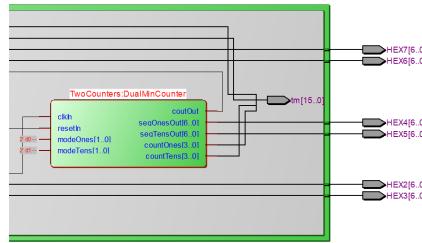
```

Listing 2.12: Testbench for entity `watch`



(a) [RTL af `watch` entity] Timetæller og reset logik

(b) [RTL af `watch` entity] Clock generator og sekundtæller



(c) [RTL af `watch` entity] Minuttælleren

Fig. 2.5: RTL view for den samlede `watch` entity

Testen blev først over på vores DE2 board, i fig. 2.6 ses eksempler på tidspunkter som er talt op.



(a) Tidseksempel: [07:13.18]



(b) Tidseksempel: [11:10.23]

Fig. 2.6: Fotos der viser de forskellige tidspunkter.

Ved at øge hastigheden på uret kunne vi teste rollover fra `23:59:59` til `00:00:00` hvilket ses på før og efter billedeerne i fig. 2.7.



(a) Tidseksempel: [23:51.58]



(b) Billede taget nogle minutter efter: [00:00.15]

Fig. 2.7: Rollover implementationen, der viser overgang m. [23:59:59]->[00:00:00].

Diskussion

Vores implementation af uret virkede upåklageligt. Undervejs havde vi dog nogle problemer med rollover, hvilket viste sig at ligge i vores `multi_counter` som satte carry når den blev reset.

Konklusion

Vi har i denne øvelse øget vores færdigheder med clock signaler. Vi har brugt carries som clock signaler for at skalere clock periode til de individuelle komponenter, hvilket fremhæver hvor meget logik man kan lave med simpel *edge detection*.

Opgave 4: Alarm Watch

Introduktion

Ved at tilføje nogle enkelte moduler kan uret fra tidligere udvides med en sammenligningsfunktionalitet — dette vil kunne bruges til at lave tidsindstillet alarm.

Design og implementering

Vi implementerede `watch` entity og en række `bin2sevenseg` entities, og tilføjede desuden 3 nye elementer: `Multiplexer`, `InputLimiter` og `Compare`.

```
1 ENTITY alarm_watch_tester IS
2   PORT
3   (
4     -- inputs
5     bin_min1 : IN std_logic_vector(3 DOWNTO 0); --!
6     bin_min10 : IN std_logic_vector(3 DOWNTO 0); --!
7     bin_hrs1 : IN std_logic_vector(3 DOWNTO 0); --!
8     bin_hrs10 : IN std_logic_vector(3 DOWNTO 0); --!
9     clk      : IN std_logic;                      --!
10    speed    : IN std_logic;                      --!
11    reset    : IN std_logic;                      --!
12    view     : IN std_logic;
13
14    --outputs
15    alarm    : OUT std_logic;
16    buzzer   : OUT std_logic;
17    HEX02    : OUT std_logic_vector(6 DOWNTO 0); --!
18    HEX03    : OUT std_logic_vector(6 DOWNTO 0); --!
19    HEX04    : OUT std_logic_vector(6 DOWNTO 0); --!
20    HEX05    : OUT std_logic_vector(6 DOWNTO 0); --!
21    HEX06    : OUT std_logic_vector(6 DOWNTO 0); --!
22    HEX07    : OUT std_logic_vector(6 DOWNTO 0)  --!
23  );
24 END ENTITY alarm_watch_tester;
```

Listing 2.13: Alarm watch entity

```
1 SIGNAL mux_b_1      : std_logic_vector(6 DOWNTO 0);
2 SIGNAL mux_b_2      : std_logic_vector(6 DOWNTO 0);
3 SIGNAL mux_b_3      : std_logic_vector(6 DOWNTO 0);
4 SIGNAL mux_b_4      : std_logic_vector(6 DOWNTO 0);
5 SIGNAL mux_b_5      : std_logic_vector(6 DOWNTO 0);
6 SIGNAL mux_b_6      : std_logic_vector(6 DOWNTO 0);
7
8  -- tm signals
9 SIGNAL tmSignal     : std_logic_vector(15 DOWNTO 0);
10 SIGNAL timeAlarmSignal : std_logic_vector(15 DOWNTO 0);
```

Listing 2.14: Signaldekleration for alarmen

```

1      mainWatch : ENTITY watch
2          PORT MAP
3          (
4              ---- Input
5              clk      => clk,
6              speed    => speed,
7              reset    => reset,
8              ---- Output
9              sec_1    => mux_b_1,
10             sec_10   => mux_b_2,
11             min_1    => mux_b_3,
12             min_10   => mux_b_4,
13             hrs_1    => mux_b_5,
14             hrs_10   => mux_b_6,
15             tm       => tmSignal
16
17      SelectorMux : ENTITY mux
18          PORT
19          MAP(
20              -- selector
21              view => view,
22              -- inputs
23              -- Select A
24              a1    => "1111111",
25              a2    => "1111111",
26              a3    => mux_a_3,
27              a4    => mux_a_4,
28              a5    => mux_a_5,
29              a6    => mux_a_6,
30              -- Select B
31              b1    => mux_b_1,
32              b2    => mux_b_2,
33              b3    => mux_b_3,
34              b4    => mux_b_4,
35              b5    => mux_b_5,
36              b6    => mux_b_6,
37              --
38              -- outputs
39              o1    => HEX02,
40              o2    => HEX03,
41              o3    => HEX04,
42              o4    => HEX05,
43              o5    => HEX06,
44              o6    => HEX07

```

Listing 2.15: Her ses hvordan selve uret (2.8) og multiplexeren implementeres

```

1      inputLimiter : ENTITY inputLimiting
2          PORT
3              MAP
4          (
5              ---- Input
6                  bin_min1    => bin_min1,
7                  bin_min10   => bin_min10,
8                  bin_hrs1     => bin_hrs1,
9                  bin_hrs10    => bin_hrs10,
10                 ---- Output
11                 time_alarm => timeAlarmSignal
12
13      compareModule : ENTITY compareTime
14          PORT
15              MAP
16          (
17              -- input
18                  tm_watch => tmSignal,
19                  tm_alarm  => timeAlarmSignal,
20              -- output
21                  alarm      => alarm
22
23      quadDisplay : ENTITY quadBin2Sevenseq
24          PORT
25              MAP
26          (
27              -- input
28                  bin       => timeAlarmSignal,
29              -- output
30                  min_1    => mux_a_3,
31                  min_10   => mux_a_4,
32                  hrs_1    => mux_a_5,
33                  hrs_10   => mux_a_6
34          );
35
36

```

Listing 2.16: Her ses input limiteren, compare module og et quaddisplay

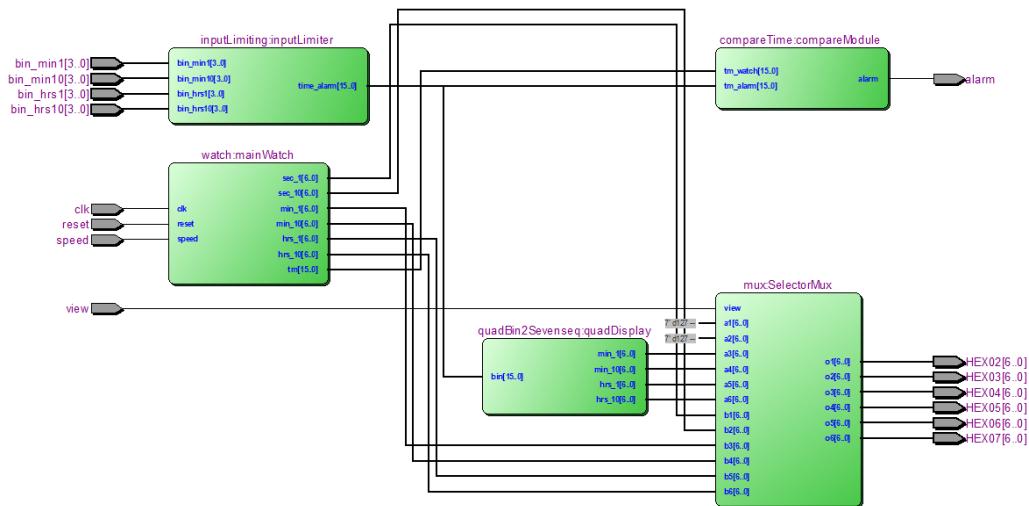


Fig. 2.8: RTL view: Alarm watch

Quad display

```
1 ENTITY quadBin2Sevenseq IS
2   PORT (
3     -- inputs
4     bin : IN std_logic_vector(15 DOWNTO 0);
5     -- outputs
6     min_1 : OUT std_logic_vector(6 DOWNTO 0);
7     min_10 : OUT std_logic_vector(6 DOWNTO 0);
8     hrs_1 : OUT std_logic_vector(6 DOWNTO 0);
9     hrs_10 : OUT std_logic_vector(6 DOWNTO 0)
10   );
11 END ENTITY quadBin2Sevenseq;
12
13 ARCHITECTURE rtl OF quadBin2Sevenseq IS
14
15 BEGIN
16   --- minutes
17   -- display ones
18   HexdisplayMinOnes : ENTITY bin2hex
19     PORT MAP
20   (
21     bin => bin(3 DOWNTO 0),
22     seg => min_1
23   );
24   -- display tens
25
26   HexdisplayMinTens : ENTITY bin2hex
27     PORT MAP
28   (
29     bin => bin(7 DOWNTO 4),
30     seg => min_10
31   );
32   --- Hours
33   -- display ones
34   HexdisplayHoursOnes : ENTITY bin2hex
35     PORT MAP
36   (
37     bin => bin(11 DOWNTO 8),
38     seg => hrs_1
39   );
40   -- display tens
41   HexdisplayHoursTens : ENTITY bin2hex
42     PORT MAP
43   (
44     bin => bin(15 DOWNTO 12),
45     seg => hrs_10
46   );
47 END ARCHITECTURE rtl;
```

Listing 2.17: Samling af fire displays

Da de fire outputdisplays består af *slice's* af et enkelt signal, valgte vi at samle dem i en entity for sig selv - dette gjorde også at RTL view'et i fig. 2.8 blev nær identisk med det IBD (fig. 6 i øvelsesvejlednigen) der var udleveret.

Compare logic

```
1 ENTITY compareTime IS
2     PORT
3     (
4         -- input
5         tm_watch : IN std_logic_vector(15 DOWNTO 0);
6         tm_alarm : IN std_logic_vector(15 DOWNTO 0);
7         -- output
8         alarm      : OUT std_logic
9     );
10 END ENTITY compareTime;
11
12 ARCHITECTURE rtl OF compareTime IS
13
14 BEGIN
15
16     Comparison : PROCESS (tm_watch, tm_alarm)
17     BEGIN
18         IF (tm_watch = tm_alarm) THEN
19             alarm <= '1';
20         ELSE
21             alarm <= '0';
22         END IF;
23     END PROCESS Comparison;
24
25 END ARCHITECTURE rtl;
```

Listing 2.18: Alarm watch arkitektur: Compare

Modulet i source code table 2.18 består blot af en simpel komparator og en IF/ELSE statement der aktivere et `alarm`output, hvis indholdet at de to inputtede STD_LOGIC_VECTOR er ens.

Input limiting

```
1 ENTITY inputLimiting IS
2   PORT
3   (
4     -- inputs
5     bin_min1 : IN std_logic_vector(3 DOWNTO 0);
6     bin_min10 : IN std_logic_vector(3 DOWNTO 0);
7     bin_hrs1 : IN std_logic_vector(3 DOWNTO 0);
8     bin_hrs10 : IN std_logic_vector(3 DOWNTO 0);
9     ---- Output
10    time_alarm : OUT std_logic_vector(15 DOWNTO 0)
11  );
12 END ENTITY inputLimiting;
13
14 ARCHITECTURE rtl OF inputLimiting IS
15   SIGNAL bin_min1signal : std_logic_vector(3 DOWNTO 0);
16   SIGNAL bin_min10signal : std_logic_vector(3 DOWNTO 0);
17
18   SIGNAL bin_hrs1signal : std_logic_vector(3 DOWNTO 0);
19   SIGNAL bin_hrs10signal : std_logic_vector(3 DOWNTO 0);
20
21 BEGIN
22
23   minutLimiting : PROCESS (bin_min1, bin_min10)
24   BEGIN
25     IF (unsigned(bin_min1) >= "1001") THEN
26       bin_min1signal <= "1001";
27     ELSE
28       bin_min1signal <= bin_min1;
29     END IF;
30
31     IF (unsigned(bin_min10) >= "0101") THEN
32       bin_min10signal <= "0101";
33     ELSE
34       bin_min10signal <= bin_min10;
35     END IF;
36   END PROCESS minutLimiting;
37   hoursLimiting : PROCESS (bin_hrs1, bin_hrs10)
38   BEGIN
39     IF (bin_hrs10 > "0010") THEN
40       bin_hrs10signal <= "0010";
41     ELSE
42       bin_hrs10signal <= bin_hrs10;
43     END IF;
44     IF (bin_hrs1 > "0011" AND bin_hrs10 > "0010") THEN
45       bin_hrs1signal <= "0011";
46     ELSE
47       bin_hrs1signal <= bin_hrs1;
48     END IF;
49     time_alarm <= (bin_hrs10signal & bin_hrs1signal & bin_min10signal &
50     bin_min1signal);
51   END PROCESS hoursLimiting;
52 END ARCHITECTURE rtl;
```

Listing 2.19: Alarm watch arkitektur: Input limiting

Tanken bag dette modul, var at begrænse inputværdierne, så den maksimale værdi var [23:59]. Der blev lavet en limiter til både minutsektionen og timesektionen, hvor begge blev lavet med **IF/ELSE** statements. Minutdelen blev begrænset til 5 for 10'erne og 9 for 1'erne på minutsektionen.

Timesektionen blev ligeledes begrænset, men med flere argumenter. Først blev 10'er sektionen

begrænset til 2. Derefter satte vi en betingelse op, der skulle blokere 1'erne i at tælle op fra 3, når 10'erne viste 2. Dette endte med en total maksimalværdi på 23:59.

Resultater

Alarm watch blev testet i source code table 2.20, hvor også pin mappings ses.

```

1  AlarmWatch : ENTITY alarm_watch_tester
2    PORT MAP
3    (
4      -- inputs
5      bin_min1  => SW(3 DOWNTO 0),
6      bin_min10 => SW(7 DOWNTO 4),
7      bin_hrs1   => SW(11 DOWNTO 8),
8      bin_hrs10  => SW(15 DOWNTO 12),
9      clk        => CLOCK_50,
10     speed      => KEY(0),
11     reset      => KEY(3),
12     view       => KEY(2),
13     --outputs
14     alarm      => LEDR(0),
15     buzzer    => GPIO_0(0),
16     HEX02     => HEX2,
17     HEX03     => HEX3,
18     HEX04     => HEX4,
19     HEX05     => HEX5,
20     HEX06     => HEX6,
21     HEX07     => HEX7
22   );

```

Listing 2.20: Alarm watch test bench

Alarmfunktionaliteten blev testet med en alarm sat til 10:54, hvorfaf resultatet ses i fig. 2.9.

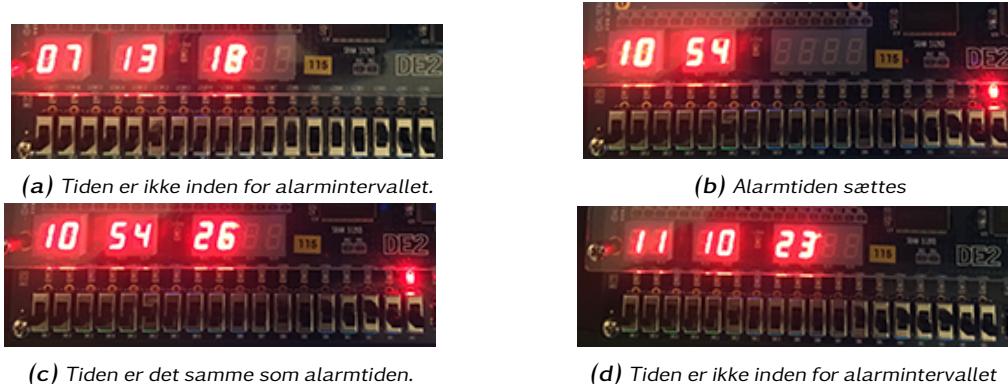


Fig. 2.9: Fotos der viser de forskellige tilstande for alarmen

Funktionaliteten i uret var meget lig den tidligere opgave - da det jo blot er source code table 2.8 med flere elementer bygget uden på.

Diskussion

Input limiteren voldte problemer i den forstand, det ikke lykkedes at få den til at virke helt korrekt. Begrænsningen fungerede uden problemer i minutsektionen, men i timesektionen kunne vi ikke få begrænsningen til at fungere hvis det *kun* var *bin_hrs1* der var aktiv. Her kunne resultater som [1F:59] fremkomme - hvilket jo tydeligvis er en fejl — som ligger i den række IF/ELSE statements der er kaldt. Vi valgte at fjerne den sektion der skabte problemer og beholdte den del der fungerede. Dette må være vores første bug.

Konklusion

Ved tænke tilbage på funktionaliteten af de moduler vi tidligere har bygget, og bruge dem i nye designs, har vi haft mulighed for at skabe større og mere komplekse projekter end hidtil — Her har vi har her lavet et 24 timers ur (med en enkelt fejl), i formattet **HH:MM:SS**, der har en alarmfunktion med visuelt feedback - denne opererer i spændet **[00:00]-[23:59]**.

Opgave 5: Peer Feedback: Journal 2

Gruppe 36

Vores review af dem

ARITHMETIC AND LOGICAL OPERATORS IN VHDL

Exe 1: Signed and Unsigned Arithmetic

- | | | |
|--|---|--|
| 1b) er der en funktionel simulation? | 4 | Uklart, udfra billedetekst, fig. 4, om det er signed eller unsigned der er vist. |
| 1c) er der en tester eller testbench kode i journalen? | 5 | Fig. 3 viser en fungerende testbench |
| 1g) er der tilføjet kode til at håndtere Cin og Cout? | 4 | Fig. 2 viser på linje 25 & 30 hvordan dette håndteres. Dog ikke beskrevet i teksten. |
| 1g) og er resize-problematikken beskrevet? | 3 | Håndteret i kode (fig 2), men ikke beskrevet i teksten. |

Exe 2: Concatenation

- | | | |
|---|---|---|
| 2a) er concatenate-koden implementeret og vist? | 5 | Dette er vist i fig.1 |
| 2a) er antallet af LE lig med 0 (som er det korrekte antal)? | 5 | Ja - vist i fig. 3 |
| 2a) er der svaret på, hvordan skifte operationerne er implementeret på FPGA'en? | 5 | Ja vist i fig. 4 med technology map view. |
| Er der en tilpas mængde test simuleringer og fotos, plus tilhørende tekst? | 5 | Ja - vist i fig. 5 og fig. 6 med LED beskrivelser |

Exe 3: Multiplication

- | | | |
|--|---|--|
| 3c) er den en resultat-tabel i journalen for forbruget af LE'er? | 4 | Vist i tabel ?? - mangler ref. nummer |
| 3c) er der lavet et LE-vs-bitsize plot for bitsizes=32,16,8,4,3,2,1? | 4 | Ja - vist på fig. ?? - mere beskrivende tekst - mangler ref. nummer |
| 3c) er der svaret på, hvordan LE skalerer mht. bitstørrelse-n (ca. n^2)? | 5 | Der er vist en potensregression |
| 3d) er der en resultat-tabel over antallet af LE'er vdr. multiplikation med en konstant? | 2 | Der er vist to resultater (fig. 11, fig. 12) der viser $a^2 \cdot 2^2$, og a^7 , men ingen tabel. |
| 3d) er der svaret på, hvorfor multiplikation med 2,4,8,16 osv. (2^k) giver 0 LE's?
Hint: gange med 2,4 eller 2^k giver blot et venstreskift 1, 2 eller k gange. | 5 | Forklaret fint I konklusionen |

Deres review af os

DATAFLOW-STYLE COMBINATORIAL DESIGNS IN VHDL

Exe 1: Binary to 7-Segment Decoder Using "WITH-SELECT"	5	Vist I fig. 13
1a) er koden til 7-segment-dekoderen vist?	0	Nej
1b) er RTL-viewet vist?	0	Nej
1b) er der evt. svaret på hvorfor RTL-vieweren ser ud som den gør?		
Exe 2: Demultiplexing Using "WHEN"		
1b) er der kode for hex-mux'en (og evt testeren) i journalen?	5	Ja, vist I fig. 16
1c) er der kommenteret på "inferred latches" i journalen, dvs. har de fundet nogle og/eller dokumenteret at de er blevet håndteret/fjernet?	5	Ja, vist I fig. 17, fig. 22 og fig. 23. Beskrevet I konklusionen.
Exe 2: Table Lookup		
1a) er opgaven "Table Lookup" implementeret som en rigtig lookup tabel? (dvs. IKKE via en with-select eller en when-sprogkonstruktion, men som en rigtig lookup tabel)	5	Ja - vist I fig. 24
1b) er der dokumentation for testen?	5	Ja - vist i fig. 26;30
Exe 3: Bidirectional Ports (OPTIONAL)		
Super bonus, hvis i har lavet denne ekstra opgave!	0	

GENERELT

Er der forside med DSD projekt gruppenr navne og id'er	5	Velinformerende forside, der beskriver opgavenavne og gruppe-medlemmer
Er der svaret på alle underopgaver a), b), c) osv?	4	Der mangler en hel del exe 4-1
Er der en introduktion til hver opgave?	5	Ja, alle opgaver har en introduktion
Er der en resultater og diskussion til hver opgave?	4	Exe 4-1 mangler en del.
Er der en konklusion til hver opgave?	5	Ja, alle opgaver har en konklusion
Er al VHDL kode formateret (indenteret) for læsbarhed?	4	Noget kode (fig 1;2) er meget småt. Ellers meget klart og læsbart.
Hvad vil du overordnet give opgaven af point?	4	Der er lidt mangler I opgaven, men ellers rigtigt fint.

ARITHMETIC AND LOGICAL OPERATORS IN VHDL

Exe 1: Signed and Unsigned Arithmetic

1b) er der en funktionel simulering?	2	Små billeder, svært at se. Det er en timing simulering i har med, ikke funktionel.
1c) er der en tester eller testbench kode i journalen?	3	Der er et billede af den, men koden er udkommenteret. Man kan ikke se hele koden.
1g) er der tilføjet kode til at håndtere Cin og Cout?	4	Cin har i forsøgt at sætte lig med 0
1g) og er resize-problematikken beskrevet?	2	Den er med i koden, men ikke beskrevet.

Exe 2: Concatenation

2a) er concatenate-koden implementeret og vist?	5	
2a) er antallet af LE lig med 0 (som er det korrekte antal)?	4	Enventuelt Compilation report
2a) er der svaret på, hvordan skifte operationerne er implementeret på FPGA'en?	5	
Er der en tilpas mængde test simuleringer og fotos, plus tilhørende tekst?	3	Nej, der mangler billeder at testen på DE2

Exe 3: Multiplication

3c) er den en resultat-tabel i journalen for forbruget af LE'er?	5	
3c) er der lavet et LE-vs-bitsize plot for bitsizes=32,16,8,4,3,2,1?	5	
3c) er der svaret på, hvordan LE skalerer mht. bitstørrelse-n (ca. n^2)?	0	mangler beskrivelse af hvordan udviklingen sker, evt. potens regression
3d) er der en resultat-tabel over antallet af LE'er vdr. multiplikation med en konstant?	0	Hele opgaven mangler
3d) er der svaret på, hvorfor multiplikation med 2,4,8,16 osv. (2^k) giver 0 LE's?	0	Hele opgaven mangler
Hint: gange med 2,4 eller 2^k giver blot et venstreskift 1, 2 eller k gange.		

DATAFLOW-STYLE COMBINATORIAL DESIGNS IN VHDL

Exe 1: Binary to 7-Segment Decoder Using "WITH-SELECT"	5	
1a) er koden til 7-segment-dekoderen vist?	5	
1b) er RTL-viewet vist?	5	
1b) er der evt. svaret på hvorför RTL-vieweren ser ud som den gør?		
Exe 2: Demultiplexing Using "WHEN"		
1b) er der kode for hex-mux'en (og evt testeren) i journalen?	4	Mangler entity for tester
1c) er der kommenteret på "inferred latches" i journalen, dvs. har de fundet nogle og/eller dokumenteret at de er blevet håndteret/fjernet?	5	
Exe 2: Table Lookup		
1a) er opgaven "Table Lookup" implementeret som en rigtig lookup tabel? (dvs. IKKE via en with-select eller en when-sprogkonstruktion, men som en rigtig lookup tabel)	2	Koden ser ud til ikke at virke, da de to bindestreger gør det til en kommentar
1b) er der dokumentation for testen?	4	Mangler entity for tester, ingen billeder af board
Exe 3: Bidirectional Ports (OPTIONAL)		
Super bonus, hvis i har lavet denne ekstra opgave!	0	Ikke lavet
<hr/>		
GENEREKT		
Er der forside med DSD projekt gruppenr navne og id'er	5	
Er der svaret på alle underopgaver a), b), c) osv?	3	Mangler opgave 3 i exercise 3
Er der en introduktion til hver opgave?	5	
Er der en resultater og diskussion til hver opgave?	5	
Er der en konklusion til hver opgave?	5	
Er al VHDL kode formateret (indenteret) for læsbarhed?	4	Nogle steder er der kommenteret noget ud
Hvad vil du overordnet give opgaven af point?	3	Der mangler generelt billeder af test på DE2 boardet

Fra Gruppe 39

Vores review af dem

Deres review af os

ARITHMETIC AND LOGICAL OPERATORS IN VHDL

Exe 1: Signed and Unsigned Arithmetic

- | | | |
|--|---|--|
| 1b) er der en funktionel simulering? | 5 | Ja - vist i fig. 8;11 med forklarende tekst. |
| 1c) er der en tester eller testbench kode i journalen? | 5 | Ja - vist I fig. 3 og fig. 7 |
| 1g) er der tilføjet kode til at håndtere Cin og Cout? | 5 | Ja - vist I fig. 4 og med forklarende tekst |
| 1g) og er resize-problematikken beskrevet? | 5 | Ja - Beskrevet under fig. 5 |

Exe 2: Concatenation

- | | | |
|---|---|--|
| 2a) er concatenate-koden implementeret og vist? | 5 | Ja - vist i fig. 12 |
| 2a) er antallet af LE lig med 0 (som er det korrekte antal)? | 5 | Ja vist og forklaret v. fig. 17 |
| 2a) er der svaret på, hvordan skifte operationerne er implementeret på FPGA'en? | 5 | Ja - vist I fig. 15 med technology map view. |
| Er der en tilpas mængde test simuleringer og fotos, plus tilhørende tekst? | 5 | Meget flot illustration I fig. 20 - med markeringer! |

Exe 3: Multiplication

- | | | |
|--|---|---|
| 3c) er den en resultat-tabel i journalen for forbruget af LE'er? | 5 | Ja - I fig 34 er dette vist |
| 3c) er der lavet et LE-vs-bitsize plot for bitsizes=32,16,8,4,3,2,1? | 5 | Ja - I fig 35 er denne illustretet. |
| 3c) er der svaret på, hvordan LE skalerer mht. bitstørrelse-n (ca. n^2)? | 5 | Ja, meget fint beskrevet! |
| 3d) er der en resultat-tabel over antallet af LE'er vdr. multiplikation med en konstant? | 5 | Ja - I fig. 38 er dette vist |
| 3d) er der svaret på, hvorfor multiplikation med 2,4,8,16 osv. (2^k) giver 0 LE's?
Hint: gange med 2,4 eller 2^k giver blot et venstreskift 1, 2 eller k gange. | 5 | Ja - en dette er flot beskrevet I afsnittet under fig. 38 |

DATAFLOW-STYLE COMBINATORIAL DESIGNS IN VHDL

Exe 1: Binary to 7-Segment Decoder Using "WITH-SELECT"

- | | | |
|---|-------------|--|
| 1a) er koden til 7-segment-dekoderen vist?
1b) er RTL-viewet vist?
1b) er der evt. svaret på hvorför RTL-vieweren ser ud som den gør? | 4
5
3 | Denne er vist I fig. 40 - dog spejlvendt.
Ja ! Vist i fig. 45
Der mangler en reel kobling m. de 7 MUX og SSEG - måskeunfold outputtet? |
|---|-------------|--|
-

Exe 2: Demultiplexing Using "WHEN"

- | | | |
|--|--------|--|
| 1b) er der kode for hex-mux'en (og evt testeren) i journalen?
1c) er der kommenteret på "inferred latches" i journalen, dvs. har de fundet nogle og/eller dokumenteret at de er blevet håndteret/fjernet? | 5
5 | Fig. 50 viser koden og fig. 52 viser testen.
Meget flot beskrevet! Fine illustrationer fra test |
|--|--------|--|
-

Exe 2: Table Lookup

- | | | |
|--|--------|---|
| 1a) er opgaven "Table Lookup" implementeret som en riktig lookup tabel? (dvs. IKKE via en with-select eller en when-sprogkonstruktion, men som en riktig lookup tabel)
1b) er der dokumentation for testen? | 5
4 | Ja ses I fig. 59
Ja - ses I fig. 66 og fig. 67 - dog ingen funk. test! |
|--|--------|---|
-

Exe 3: Bidirectional Ports (OPTIONAL)

- | | | |
|--|---|--|
| Super bonus, hvis i har lavet denne ekstra opgave! | 0 | Desværre! Vi ville gerne have læst hvad I havde fundet på! |
|--|---|--|
-

GENEREKT

- | | | |
|--|---------------------------------|--|
| Er der forside med DSD projekt gruppenr navne og id'er
Er der svaret på alle underopgaver a), b), c) osv?
Er der en introduktion til hver opgave?
Er der en resultater og diskussion til hver opgave?
Er der en konklusion til hver opgave?
Er al VHDL kode formateret (indenteret) for læsbarhed?
Hvad vil du overordnet give opgaven af point? | 4
5
5
5
5
5
5 | Opgavenavne kunne være en god ting.
Meget flotte beskrivelser
Gode introduktioner (Men i har en "with select"fejl, I sektion 4.3)
Ja - meget gode resultater og illustrationer
Ja - meget gode konklusioner der beskriver opgaven!
God læselig kode!
En fornøjelse at læse denne opgave! Godt arbejde! |
|--|---------------------------------|--|
-