Dally, Harting and Aamodt: Digital Design Using VHDL

Kapitel 7

7.1 Vi får VHDL-koden

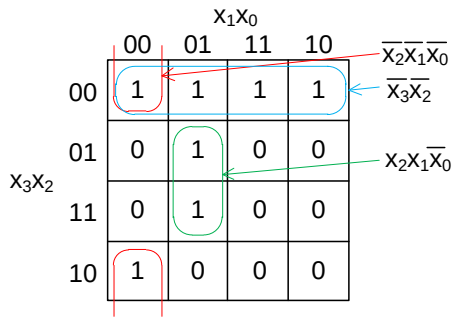
```
----------------
-- ex7_1.vhdl --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_1 IS
   PORT (x:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         y:OUT STD_LOGIC);
END ex7_1;

ARCHITECTURE arch_ex7_1 OF ex7_1 IS
BEGIN
   fibonacci_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "0000" => y<='1';
         WHEN "0001" => y<='1';
         WHEN "0010" => y<='1';
         WHEN "0011" => y<='1';
         WHEN "0101" => y<='1';
         WHEN "1000" => y<='1';
         WHEN "1101" => y<='1';
         WHEN OTHERS => y<='0';
      END CASE;
   END PROCESS fibonacci_proc;
END arch_ex7_1;
```

Som vi simulerar med do-filen

7.1 forts.

```
--------------
-- ex7_1.do --
--------------

restart -f -nowave
view signals wave
add wave x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
run 1700ns
```

7.2 Vi har sanningstabellen *Figur 7.2a* som ger Karnaughdiagrammet i *Figur 7.2b*.



Figur7.2b Karnaughdiagram

x_3	x_2	x_1	x_0	y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Figur 7.2a Sanningatabell

Vi får

$$y = \overline{x_2} \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot x_1 \cdot \overline{x_0} + \overline{x_3} \cdot \overline{x_2}$$

Vi skriver VHDL-kod

```
----------------
-- ex7_2.vhdl --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

7.2 forts.

```
ENTITY ex7_2 IS
   PORT (x:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         y:OUT STD_LOGIC);
END ex7_2;

ARCHITECTURE arch_ex7_2 OF ex7_2 IS
BEGIN
   y<=(NOT(x(2)) AND NOT(x(1)) AND NOT(x(0))) OR
      (x(2) AND x(1) AND NOT(x(0))) OR
      (NOT(x(3)) AND NOT(x(2)));
END arch_ex7_2;
```

Vi kan använda samma do-fil som i *Exempel 7.1*.

7.3 Vi skriver direkt VHDL-kod för sanningstabellen i *Figur 7.2a*

```
----------------
-- ex7_3.vhdl --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_3 IS
   PORT (x:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         y:OUT STD_LOGIC);
END ex7_3;

ARCHITECTURE arch_ex7_3 OF ex7_3 IS
BEGIN
   y<=(NOT(x(3)) AND NOT(x(2)) AND NOT(x(1)) AND
                                       NOT(x(0))) OR
      (NOT(x(3)) AND NOT(x(2)) AND NOT(x(1)) AND x(0)) OR
      (NOT(x(3)) AND NOT(x(2)) AND x(1) AND NOT(x(0))) OR
      (NOT(x(3)) AND NOT(x(2)) AND x(1) AND x(0)) OR
      (NOT(x(3)) AND x(2) AND NOT(x(1)) AND x(0)) OR
      (x(3) AND NOT(x(2)) AND NOT(x(1)) AND NOT(x(0))) OR
      (x(3) AND x(2) AND NOT(x(1)) AND x(0));
END arch_ex7_3;
```

Vi kan återigen använda samma do-fil som i *Exempel 7.1*.

7.4 Vi skriver en testbänk till *Exempel 7.1* men då den har samma entitet som *Exempel 7.2* och *Exempel 7.3* så kan en användas i de fallen också.

```
--------------------
-- ex7_4_tb3.vhdl --
--------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_4_tb3 IS

END ex7_4_tb3;

ARCHITECTURE arch_ex7_4_tb3 OF ex7_4_tb3 IS

COMPONENT ex7_1 IS
   PORT (x:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         y:OUT STD_LOGIC);
END COMPONENT ex7_1;

   SIGNAL x_tb_signal:STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL y_tb_signal:STD_LOGIC;
BEGIN
   ex7_1_comp:
   COMPONENT ex7_1
      PORT MAP(x=>x_tb_signal,
               y=>y_tb_signal);

   x_tb_signal(0)<='0',
                   '1' AFTER 100 ns,
                   '0' AFTER 200 ns,
                   '1' AFTER 300 ns,
                   '0' AFTER 400 ns,
                   '1' AFTER 500 ns,
                   '0' AFTER 600 ns,
                   '1' AFTER 700 ns,
                   '0' AFTER 800 ns,
                   '1' AFTER 900 ns,
                   '0' AFTER 1000 ns,
                   '1' AFTER 1100 ns,
                   '0' AFTER 1200 ns,
                   '1' AFTER 1300 ns,
                   '0' AFTER 1400 ns,
                   '1' AFTER 1500 ns,
```

7.4 forts.

```
                    '0' AFTER 1600 ns,
                      '1' AFTER 1700 ns,
                      '0' AFTER 1800 ns,
                      '1' AFTER 1900 ns,
                      '0' AFTER 2000 ns,
                      '1' AFTER 2100 ns,
                      '0' AFTER 2200 ns,
                      '1' AFTER 2300 ns,
                      '0' AFTER 2400 ns,
                      '1' AFTER 2500 ns,
                      '0' AFTER 2600 ns,
                      '1' AFTER 2700 ns,
                      '0' AFTER 2800 ns,
                      '1' AFTER 2900 ns,
                      '0' AFTER 3000 ns,
                      '1' AFTER 3100 ns;

    x_tb_signal(1)<='0',
                      '1' AFTER 200 ns,
                      '0' AFTER 400 ns,
                      '1' AFTER 600 ns,
                      '0' AFTER 800 ns,
                      '1' AFTER 1000 ns,
                      '0' AFTER 1200 ns,
                      '1' AFTER 1400 ns,
                      '0' AFTER 1600 ns,
                      '1' AFTER 1800 ns,
                      '0' AFTER 2000 ns,
                      '1' AFTER 2200 ns,
                      '0' AFTER 2400 ns,
                      '1' AFTER 2600 ns,
                      '0' AFTER 2800 ns,
                      '1' AFTER 3000 ns;

    x_tb_signal(2)<='0',
                      '1' AFTER 400 ns,
                      '0' AFTER 800 ns,
                      '1' AFTER 1200 ns,
                      '0' AFTER 1600 ns,
                      '1' AFTER 2000 ns,
                      '0' AFTER 2400 ns,
                      '1' AFTER 2800 ns;
```

7.4 forts.

```
      x_tb_signal(3)<='0',
                      '1' AFTER 800 ns,
                      '0' AFTER 1600 ns,
                      '1' AFTER 2400 ns;

      test_proc:
      PROCESS
      BEGIN
         WAIT FOR 50 ns; -- 50 ns x=0 -> y=0
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 150 ns x=1 -> y=0
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=1 150ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 250 ns x=2 -> y=0
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=2 250ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 350 ns x=3 -> y=1
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=3 350ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 450 ns x=4 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=4 450ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 550 ns x=5 -> y=0
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=5 550ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 650 ns x=6 -> y=1
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=6 650ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 750 ns x=7 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=7 750ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 850 ns x=8 -> y=0
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=8 850ns"
```

7.4 forts.

```
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 950 ns x=9 -> y=1
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=9 950ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1050 ns x=10 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=10 1050ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1150 ns x=11 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=11 1150ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1250 ns x=12 -> y=1
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=12 1250ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1350 ns x=13 -> y=0
        ASSERT (y_tb_signal='1')
        REPORT "Error for x=13 1350ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1450 ns x=14-> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=14 1450ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1550 ns x=15 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=15 1550ns"
        SEVERITY ERROR;
    END PROCESS test_proc;

END arch_ex7_4_tb3;
```

Vi skriver också en do-fil

7.4 forts.

```
------------------
-- ex7_4_tb3.do --
------------------

restart -f -nowave
view signals wave
add wave x_tb_signal y_tb_signal

run 1650ns
```

7.5 Syntes lämnas till läsaren

7.6 Vi gör två lösningar. Först en lösning som använder en CASE-sats och sedan en lösning som använder ett litet minne, en LUT (Look Up Table).
Vi börjar med CASE-satsen.

```
ENTITY ex7_6_case IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_6_case;

ARCHITECTURE arch_ex7_6_case OF ex7_6_case IS
BEGIN
   case_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "00001" | "00011" | "00101" | "00111" |
              "01011" | "01101" | "10001" | "10111" |
              "11101" | "11111" =>
            y<='1';
         WHEN OTHERS =>
            y<='0';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_6_case;
```

Och tar sedan LUT-lösningen där vi lägger alla utvärden i en datavektor och använder invektorn som adress till positioner i data vektorn.

7.6 forts.

```
---------------
-- ex7_6_LUT --
---------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_6_LUT IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_6_LUT;

ARCHITECTURE arch_ex7_6_LUT OF ex7_6_LUT IS
   CONSTANT value:STD_LOGIC_VECTOR(0 TO
31):="00110101000101000101000100000101";
BEGIN
   y<=value(TO_INTEGER(UNSIGNED(x)));
END arch_ex7_6_LUT;
```

Lägg märke till att vi inte direkt kan använda invektorn som adress till vektorn utan måste typomvandla den binära vektorn till ett heltal via kommandot `TO_INTEGER`. Detta kommando kräver i sin tur att vi anger om den binära vektorn skall tolkas som ett tal med eller utan tecken. Då adressen är ett positivt tal så tolkar vi vektorn som ett tal utan tecken via typningen `UNSIGNED`. Notera att detta inte är en typomvandling utan bara anger hur talet skall tolkas. För att kunna dessa omvandlinar så måste vi inkuudera IEEE-biblioteket `numeric_std`.
Då de två lösningarna har samma entitet så kan vi använda samma `do`-fil för båda simuleringarna. Vi skriver en `do`-fil där vi räknar igenom alla 32 möjliga invärden. Notera att vi genom att definiera bitarna som upprepande signaler enkelt kan gå igenom all möjliga invärden.

```
--------------
-- ex7_6.do --
--------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
```

7.6 forts.

```
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
run 3250ns
```

Notera att vi använder `-radix` för att visa värden på lite mer läsbar form

7.7 Vi använder samma metoder som i *Exempel 7.6* men nu för en fyra bitars invektor.
Först CASE-fallet

```
----------------
-- ex7_7_case --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_7_case IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_7_case;

ARCHITECTURE arch_ex7_7_case OF ex7_7_case IS
BEGIN
   case_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "0011" | "0110" | "1001" |
              "1100" | "1111" =>
            y<='1';
         WHEN OTHERS =>
            y<='0';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_7_case;
```

Och sedan LUT-fallet

```
---------------
-- ex7_7_LUT --
---------------
```

7.7 forts.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_7_LUT IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_7_LUT;

ARCHITECTURE arch_ex7_7_LUT OF ex7_7_LUT IS
   CONSTANT value:STD_LOGIC_VECTOR(0 TO
15):="0001001001001001";
BEGIN
   y<=value(TO_INTEGER(UNSIGNED(x)));
END arch_ex7_7_LUT;
```

Vi simulerar på samma sätt som i *Exempel 7.6*

```
--------------
-- ex7_7.do --
--------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
run 1650ns
```

7.8 Vi har en do-fil redan i *Exempel 7.7* men vi skriver också en testbänk där vi använder LUT-varianten av Exempel 7.7 men vi hade lika gärna kunnat använda CASE-formen.

```
--------------------
-- ex7_7_tb3.vhdl --
--------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

7.8 forts.

```
ENTITY ex7_7_tb3 IS

END ex7_7_tb3;

ARCHITECTURE arch_ex7_7_tb3 OF ex7_7_tb3 IS

COMPONENT ex7_7_LUT IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC);
END COMPONENT ex7_7_LUT;

   SIGNAL x_tb_signal:STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL y_tb_signal:STD_LOGIC;
BEGIN
   ex7_7_comp:
   COMPONENT ex7_7_LUT
      PORT MAP(x=>x_tb_signal,
               y=>y_tb_signal);

   x_tb_signal(0)<='0',
                   '1' AFTER 100 ns,
                   '0' AFTER 200 ns,
                   '1' AFTER 300 ns,
                   '0' AFTER 400 ns,
                   '1' AFTER 500 ns,
                   '0' AFTER 600 ns,
                   '1' AFTER 700 ns,
                   '0' AFTER 800 ns,
                   '1' AFTER 900 ns,
                   '0' AFTER 1000 ns,
                   '1' AFTER 1100 ns,
                   '0' AFTER 1200 ns,
                   '1' AFTER 1300 ns,
                   '0' AFTER 1400 ns,
                   '1' AFTER 1500 ns;

   x_tb_signal(1)<='0',
                   '1' AFTER 200 ns,
                   '0' AFTER 400 ns,
                   '1' AFTER 600 ns,
                   '0' AFTER 800 ns,
                   '1' AFTER 1000 ns,
                   '0' AFTER 1200 ns,
```

7.8 forts.

```
                    '1' AFTER 1400 ns;
      x_tb_signal(2)<='0',
                    '1' AFTER 400 ns,
                    '0' AFTER 800 ns,
                    '1' AFTER 1200 ns;
      x_tb_signal(3)<='0',
                    '1' AFTER 800 ns;

      test_proc:
      PROCESS
      BEGIN
         WAIT FOR 50 ns; -- 50 ns x=0 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 150 ns x=1 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 250 ns x=2 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 350 ns x=3 -> y=1
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 450 ns x=4 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 550 ns x=5 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 650 ns x=6 -> y=1
         ASSERT (y_tb_signal='1')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
         WAIT FOR 100 ns; -- 750 ns x=7 -> y=0
         ASSERT (y_tb_signal='0')
         REPORT "Error for x=0 50ns"
         SEVERITY ERROR;
```

7.8 forts.

```
    WAIT FOR 100 ns; -- 850 ns x=8 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 950 ns x=9 -> y=1
        ASSERT (y_tb_signal='1')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1050 ns x=10 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1150 ns x=11 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1250 ns x=12 -> y=1
        ASSERT (y_tb_signal='1')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1250 ns x=13 -> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1350 ns x=14-> y=0
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1450 ns x=15 -> y=0
        ASSERT (y_tb_signal='1')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
        WAIT FOR 100 ns; -- 1550 ns x=0 -> y=1
        ASSERT (y_tb_signal='0')
        REPORT "Error for x=0 50ns"
        SEVERITY ERROR;
     END PROCESS test_proc;

END arch_ex7_7_tb3;
```

med tillhörande do-fil

7.8 forts.

```
------------------
-- ex7_7_tb3.do --
------------------

restart -f -nowave
view signals wave
add wave -radix unsigned x_tb_signal y_tb_signal

run 1625ns
```

7.9 Vi skriver även här en CASE- och en LUT-lösning. Notera att utsignalen är satt till don´t care för invärdena 10-15.
Först CASE-fallet

```
----------------
-- ex7_9_case --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_9_case IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_9_case;

ARCHITECTURE arch_ex7_9_case OF ex7_9_case IS
BEGIN
   case_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "0000" | "0001" | "0010" |
              "0011" | "0101" | "1000" =>
            y<='1';
         WHEN "0100" | "0110" | "0111" | "1001" =>
            y<='0';
         WHEN OTHERS =>
            y<='-';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_9_case;
```

7.9 forts.

Och sedan LUT-lösningen

```
---------------
-- ex7_9_LUT --
---------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_9_LUT IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_9_LUT;

ARCHITECTURE arch_ex7_9_LUT OF ex7_9_LUT IS
   CONSTANT value:STD_LOGIC_VECTOR(0 TO 15):="1111010010--
----";
BEGIN
   y<=value(TO_INTEGER(UNSIGNED(x)));
END arch_ex7_9_LUT;
```

Vi kan åter använda samma do-fil i de två fallen

```
--------------
-- ex7_9.do --
--------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
run 1650ns
```

7.10 Vi löser det på samma sätt som i tidigare uppgifter och gör en CASE- och en LUT-lösning. Först CASE-fallet

```
-----------------
-- ex7_10_case --
-----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_10_case IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_10_case;

ARCHITECTURE arch_ex7_10_case OF ex7_10_case IS
BEGIN
   case_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "00000" | "00101" | "01010" | "01111" |
              "10100" | "11001" | "11110" =>
            y<='1';
         WHEN OTHERS =>
            y<='0';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_10_case;
```

och sedan LUT-lösningen

```
----------------
-- ex7_10_LUT --
----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_10_LUT IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_10_LUT;
```

7.10 forts.

```
ARCHITECTURE arch_ex7_10_LUT OF ex7_10_LUT IS
   CONSTANT value:STD_LOGIC_VECTOR(0 TO 31)
                :="10000100001000010000100001000010";
BEGIN
   y<=value(TO_INTEGER(UNSIGNED(x)));
END arch_ex7_10_LUT;
```

Och vi använder samma do-fil för båda lösningarna

```
---------------
-- ex7_10.do --
---------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
run 3250ns
```

7.11 Vi löser det på samma sätt som i tidigare uppgifter men då det blir lite jobbigt att hålla rätt på en LUT med 256 värden vilket är vad vi får från en 8 bitars invektor så nöjer vi oss med en CASE-lösning.
För att göra koden mer lättläst så typomvandlar vi invektorn till ett positivt heltal innan vi använder den i CASE-satsen.

```
------------
-- ex7_11 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_11 IS
   PORT(x:STD_LOGIC_VECTOR(7 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_11;
```

7.11 forts.

```
ARCHITECTURE arch_ex7_11 OF ex7_11 IS
   SIGNAL x_int_signal:INTEGER RANGE 0 TO 255;
BEGIN
   x_int_signal<=TO_INTEGER(UNSIGNED(x));
   case_proc:
   PROCESS(x_int_signal)
   BEGIN
      CASE x_int_signal IS
         WHEN 1 | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 |
              100 | 121 | 144 | 169 | 196 | 225 =>
            y<='1';
         WHEN OTHERS =>
            y<='0';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_11;
```

Och vi skriver en do-fil

```
---------------
-- ex7_11.do --
---------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
force x(5) 0 0ns, 1 3200ns -repeat 64000ns
force x(6) 0 0ns, 1 6400ns -repeat 12800ns
force x(7) 0 0ns, 1 12800ns -repeat 25600ns
run 25650ns
```

7.12 Lösningen blir av samma form som i *Exempel 7.11*

```
------------
-- ex7_12 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ex7_12 IS
   PORT(x:STD_LOGIC_VECTOR(7 DOWNTO 0);
        y:OUT STD_LOGIC);
END ex7_12;

ARCHITECTURE arch_ex7_12 OF ex7_12 IS
   SIGNAL x_int_signal:INTEGER RANGE 0 TO 255;
BEGIN
   x_int_signal<=TO_INTEGER(UNSIGNED(x));
   case_proc:
   PROCESS(x_int_signal)
   BEGIN
      CASE x_int_signal IS
         WHEN 1 | 8 | 27 | 64 | 125 | 216 =>
            y<='1';
         WHEN OTHERS =>
            y<='0';
      END CASE;
   END PROCESS case_proc;
END arch_ex7_12;
```

Som vi simulerar med en do-fil

```
---------------
-- ex7_12.do --
---------------

restart -f -nowave
view signals wave
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
```

7.12 forts.

```
force x(5) 0 0ns, 1 3200ns -repeat 64000ns
force x(6) 0 0ns, 1 6400ns -repeat 12800ns
force x(7) 0 0ns, 1 12800ns -repeat 25600ns
run 25650ns
```

7.13 Vi gör först en CASE-lösning. Lösningen är inte så bra då den saknar all flexibilitet

```
------------
-- ex7_13 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_13 IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END ex7_13;

ARCHITECTURE arch_ex7_13 OF ex7_13 IS
BEGIN
   case_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "00000" => --0
            y<="00000";
         WHEN "00001" => --1
            y<="10000";
         WHEN "00010" => --2
            y<="01000";
         WHEN "00011" => --3
            y<="11000";
         WHEN "00100" => --4
            y<="00100";
         WHEN "00101" => --5
            y<="10100";
         WHEN "00110" => --6
            y<="01100";
         WHEN "00111" => --7
            y<="11100";
         WHEN "01000" => --8
            y<="00010";
```

7.13 forts.

```
WHEN "01001" => --9
   y<="10010";
WHEN "01010" => --10
   y<="01010";
WHEN "01011" => --11
   y<="11010";
WHEN "01100" => --12
   y<="00110";
WHEN "01101" => --13
   y<="10110";
WHEN "01110" => --14
   y<="01110";
WHEN "01111" => --15
   y<="11110";
WHEN "10000" => --16
   y<="00001";
WHEN "10001" => --17
   y<="10001";
WHEN "10010" => --18
   y<="01001";
WHEN "10011" => --19
   y<="11001";
WHEN "10100" => --20
   y<="00101";
WHEN "10101" => --21
   y<="10101";
WHEN "10110" => --22
   y<="01101";
WHEN "10111" => --23
   y<="11101";
WHEN "11000" => --24
   y<="00011";
WHEN "11001" => --25
   y<="10011";
WHEN "11010" => --26
   y<="01011";
WHEN "11011" => --27
   y<="11011";
WHEN "11100" => --28
   y<="00111";
WHEN "11101" => --29
   y<="10111";
```

7.13 forts.

```
            WHEN "11110" => --30
               y<="01111";
            WHEN "11111" => --31
               y<="11111";
            WHEN OTHERS =>
               y<=(OTHERS=>'0');
         END CASE;
      END PROCESS case_proc;
   END arch_ex7_13;
```

Vi simulerar med en do-fil

```
---------------
-- ex7_13.do --
---------------

restart -f -nowave
view signals wave
add wave -radix binary x y
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
run 3250ns
```

7.14 I uppgiften begärs att vi skal lösa med hjälp av konkatenering men vi ger också en lösning som använder en loop.
Först konkateneringen

```
-----------------
-- ex7_14_conc --
-----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_14_conc IS
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END ex7_14_conc;
```

7.14 forts.

```
ARCHITECTURE arch_ex7_14_conc OF ex7_14_conc IS
BEGIN
   y<=x(0) & x(1) & x(2) & x(3) & x(4);
END arch_ex7_14_conc;
```

Sedan looplösningen som är mer flexibel då den via GENERICS lätt kan anpassas till vektorer av olika längd. Detta kan till och med ske när vi använder koden så allt som behöver ändras är värdet på GENERIC.

```
-----------------
-- ex7_14_loop --
-----------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_14_loop IS
   GENERIC(WIDTH:NATURAL:=5);
   PORT(x:STD_LOGIC_VECTOR(4 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END ex7_14_loop;

ARCHITECTURE arch_ex7_14_loop OF ex7_14_loop IS
BEGIN
   loop_proc:
   PROCESS(x)
   BEGIN
      FOR i IN 0 TO WIDTH-1 LOOP
         y(i) <= x(WIDTH-1-i);
      END LOOP;
   END PROCESS loop_proc;
END arch_ex7_14_loop;
```

Vi kan använda samma do-fil i båda fallen och det är samma do-fil som i *Exempel 7.13*

```
---------------
-- ex7_14.do --
---------------

restart -f -nowave
view signals wave
add wave -radix binary x y
add wave -radix unsigned x y
```

7.14 forts.

```
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
run 3250ns
```

7.15 Uppgiften är lite oklar. Vi kan skriva en oflexibel lösning som bara använder de angivna värdena eller också kan vi göra en mer flexibel lösning där vi låter koden räkna ut tillräckligt många Fibonaccivärden. Vi visar här den statiska lösningen. Lägg märke till att du uppgiften säger att vi bara kan ha Fibonaccital som insignal så utsignalen satt till don´t care i övriga fall

```
------------
-- ex7_15 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_15 IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END ex7_15;

ARCHITECTURE arch_ex7_15 OF ex7_15 IS
BEGIN
   fibo_proc:
   PROCESS(x)
   BEGIN
      CASE x IS
         WHEN "0001" =>
            y<="00011";
         WHEN "0011" =>
            y<="00101";
         WHEN "0101" =>
            y<="01000";
         WHEN "1000" =>
            y<="01101";
         WHEN "1101" =>
            y<="10101";
```

7.15 forts.

```
          WHEN OTHERS =>
             y<="-----";
      END CASE;
   END PROCESS fibo_proc;
END arch_ex7_15;
```

Och vi simulerar med en do-fil

```
---------------
-- ex7_15.do --
---------------

restart -f -nowave
view signals wave
add wave -radix binary x y
add wave -radix unsigned x y
force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
run 1650ns
```

7.16 Vi uppdaterar koden från *Exempel 7.15* med en valid-signal. Lägg märke till att i processen har valid-signalen fått ett defaultvärde i början av processen so vi bara ändrar i det fall när signalen behöver ha ett annat värde.

```
------------
-- ex7_16 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_16 IS
   PORT(x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
        valid:OUT STD_LOGIC);
END ex7_16;
```

7.16 forts.

```
ARCHITECTURE arch_ex7_16 OF ex7_16 IS
BEGIN
   fibo_proc:
   PROCESS(x)
   BEGIN
      valid<='1';
      CASE x IS
         WHEN "0001" =>
            y<="00011";
         WHEN "0011" =>
            y<="00101";
         WHEN "0101" =>
            y<="01000";
         WHEN "1000" =>
            y<="01101";
         WHEN "1101" =>
            y<="10101";
         WHEN OTHERS =>
            y<="-----";
            valid<='0';
     END CASE;
   END PROCESS fibo_proc;
END arch_ex7_16;
```

do-filen blir likadan som i *Exempel 7.15* bortsett från att vi också vill visa `valid`-signalen

7.17 Vi fortsätter modifiera Fibonacci-koden

```
------------
-- ex7_17 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_17 IS
   PORT(rst:IN STD_LOGIC;
        ivalid:IN STD_LOGIC;
        x:STD_LOGIC_VECTOR(3 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(4 DOWNTO 0);
        valid:OUT STD_LOGIC);
END ex7_17;
```

7.17 forts.

```
ARCHITECTURE arch_ex7_17 OF ex7_17 IS
   SIGNAL y_signal:STD_LOGIC_VECTOR(4 DOWNTO 0);
   SIGNAL valid_signal:STD_LOGIC;
BEGIN
   fibo_proc:
   PROCESS(x)
   BEGIN
      valid_signal<='1';
      CASE x IS
         WHEN "0001" =>
            y_signal<="00011";
         WHEN "0011" =>
            y_signal<="00101";
         WHEN "0101" =>
            y_signal<="01000";
         WHEN "1000" =>
            y_signal<="01101";
         WHEN "1101" =>
            y_signal<="10101";
         WHEN OTHERS =>
            y_signal<="-----";
            valid_signal<='0';
     END CASE;
   END PROCESS fibo_proc;
   y<=(OTHERS=>'0') WHEN rst='1' ELSE
      y_signal;
   valid<=(ivalid AND NOT(rst)) AND valid_signal;
END arch_ex7_17;
```

Som vi simulerar med en do-fil där vi kör samma simulering som tidigare men vi kör den fyra gånger så att vi kan testa alla fyra möjliga kombinationer av de nya insignalerna `rst` och `ivalid`

7.18 Vi behandlar inte implementeringen

7.19 Vi får komplettera segmentkonstanterna från *Figur 7.23* sidan 149 med koder för A-F. Vi får

```
constant SS_A : sseg_type := 7b”0001000”;
constant SS_B : sseg_type := 7b”0001011”;
constant SS_C : sseg_type := 7b”1000110”;
constant SS_D : sseg_type := 7b”0100001”;
constant SS_E : sseg_type := 7b”0000110”;
```

7.19 forts.

```
constant SS_F : sseg_type := 7b”0001110”;
```

Vi får också komplettera `case`- satsen i *Figur 7.24* sidan 150

```
When x”A” _> segs <= SS_A;
When x”B” _> segs <= SS_B;
When x”C” _> segs <= SS_C;
When x”D” _> segs <= SS_D;
When x”E” _> segs <= SS_E;
When x”F” _> segs <= SS_F;
```

7.20 Vi behöver komplettera med kod för den alternativa nian

```
constant SS_9_alt : sseg_type := 7b”0011000”;
```

och sedan komplettera `case`-satsen i *Figur 7.25* sidan 151 med

```
when SS_9_alt => valid >= ’1’; bin <= x”9”;
```

7.21 Vi modifierar processen i *Figur 7.18*, sidan 144

```
PROCESS
BEGIN
   isprime_signal <= ’1’
   FOR i IN 0 TO 15 LOOP
      Input <= STD_LOGIC_VECTOR(TO_UNSIGNED(i,4));
      WAIT FOR 10 ns;
      IF isprim = ‘0’ THEN
         isprime_signal <= ‘0’;
   END LOOP;
```

Notera att `isprime_signal` har defaultvärdet `1` som indikerar att det är ett primtal. Har vi i något loopvarv inte ett primtal så kommer signalen att bli `0` och då den inte återställs inne i loopen så kommer den då att ligga kvar låg ända till slutet. Då alla tal 0-15 inte är primtal så kommer denna process alltid att indikera fel men vi kommer via signalen `isprime_signal` att se vid vilket värde på i det första felet dyker upp.

7.22 Det här är en klumpig mutiplikationslösning men det är ju vad som begärs. Vi konkatinerar de två intalen, `a` och `b`, till envektor som vi kan använda som selektionselement i `case`-satsen

7.22 forts.

```
------------
-- ex7_22 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_22 IS
   PORT(a:STD_LOGIC_VECTOR(1 DOWNTO 0);
        b:STD_LOGIC_VECTOR(1 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex7_22;

ARCHITECTURE arch_ex7_22 OF ex7_22 IS
   SIGNAL in_signal:STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
   in_signal<=a&b;
   mult_proc:
   PROCESS(in_signal)
   BEGIN
      CASE in_signal IS
         WHEN "0000" =>
            y<="0000";
         WHEN "0001" =>
            y<="0000";
         WHEN "0010" =>
            y<="0000";
         WHEN "0011" =>
            y<="0000";
         WHEN "0100" =>
            y<="0000";
         WHEN "0101" =>
            y<="0001";
         WHEN "0110" =>
            y<="0010";
         WHEN "0111" =>
            y<="0011";
         WHEN "1000" =>
            y<="0000";
         WHEN "1001" =>
            y<="0010";
         WHEN "1010" =>
            y<="0100";
```

7.22 forts.

```
          WHEN "1011" =>
              y<="0110";
           WHEN "1100" =>
              y<="0000";
           WHEN "1101" =>
              y<="0011";
           WHEN "1110" =>
              y<="0110";
           WHEN "1111" =>
              y<="1001";
           WHEN OTHERS =>
              y<="----";
      END CASE;
   END PROCESS mult_proc;
END arch_ex7_22;
```

Vi simulerar genom att gå igenom alla möjliga insignaler

```
---------------
-- ex7_22.do --
---------------

restart -f -nowave
view signals wave
add wave a b y
add wave -radix unsigned a b y

force a(0) 0 0ns, 1 100ns -repeat 200ns
force a(1) 0 0ns, 1 200ns -repeat 400ns
force b(0) 0 0ns, 1 400ns -repeat 800ns
force b(1) 0 0ns, 1 800ns -repeat 1600ns
run 1650 ns
```

7.23 För att kunna använda don´t care-värden i insignalerna så måste vi använda den alternativa `case`-funktionen `case?`. Denna stöds tyvärr inte i 2002-års VHDL vilket är vad som är deafult i QuestaSim utan det krävs version 2008. Vi kan dock gå förbi detta.
Vi har koden

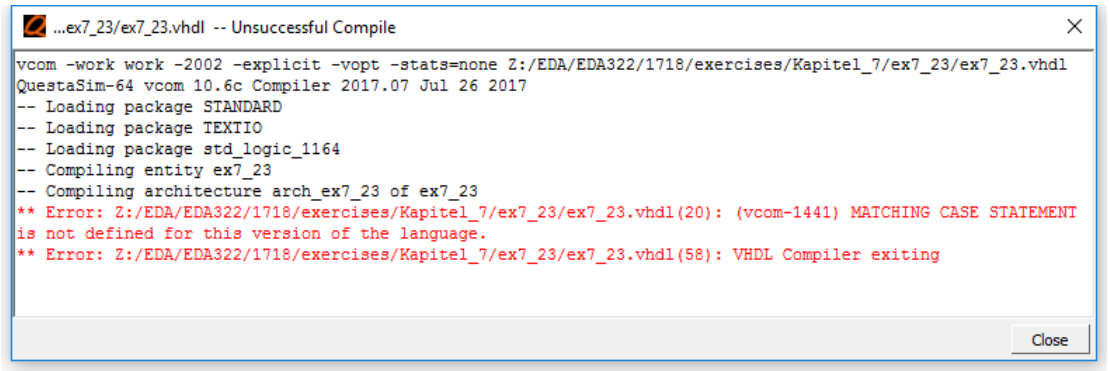
7.23 forts.

```
------------
-- ex7_23 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex7_23 IS
   PORT(x:STD_LOGIC_VECTOR(15 DOWNTO 0);
        y:OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        no_ones:OUT STD_LOGIC);
END ex7_23;

ARCHITECTURE arch_ex7_23 OF ex7_23 IS
BEGIN
   one_proc:
   PROCESS(x)
   BEGIN
      no_ones<='0';
      CASE? x IS
         WHEN "1---------------" =>
            y<="1111";
         WHEN "01--------------" =>
            y<="1110";
         WHEN "001-------------" =>
            y<="1101";
         WHEN "0001------------" =>
            y<="1100";
         WHEN "00001-----------" =>
            y<="1011";
         WHEN "000001----------" =>
            y<="1010";
         WHEN "0000001---------" =>
            y<="1001";
         WHEN "00000001--------" =>
            y<="1000";
         WHEN "000000001-------" =>
            y<="0111";
         WHEN "0000000001------" =>
            y<="0110";
         WHEN "00000000001-----" =>
            y<="0101";
```

7.23 forts.

```
            y<="0100";
         WHEN "0000000000001---" =>
            y<="0011";
         WHEN "00000000000001--" =>
            y<="0010";
         WHEN "000000000000001-" =>
            y<="0001";

       WHEN "0000000000000001" =>
            y<="0000";
         WHEN OTHERS =>
            y<=(OTHERS=>'0');
            no_ones<='1';
     END CASE?;
   END PROCESS one_proc;
END arch_ex7_23;
```

Försöker vi kompilera detta i QuestaSim så får vi felet i *Figur ex7.23*



Figur ex7.23

På den första raden i rutan ser vi det kommando som körs. Här ser vi bland annat kommandot -2002 som anger vilken version av VHDL som ska användas. Vi kan kopiera denna rad till QuestaSims kommandorad i Transcript-fönstret och ändra -2002 till -2008

```
vcom -work work -2008 -explicit -vopt -stats=none
Z:/EDA/EDA322/1718/exercises/Kapitel_7/ex7_23/ex7_23.vhdl
```

Detta kommer att kompilera koden till version 2008 och vi kan simulera den på vanligt sätt med en do-fil

7.23 forts.

```
---------------
-- ex7_23.do --
---------------

restart -f -nowave
view signals wave
add wave -radix binary x y
add wave -radix unsigned y
add wave no_ones


force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns
force x(4) 0 0ns, 1 1600ns -repeat 3200ns
force x(5) 0 0ns, 1 3200ns -repeat 6400ns
force x(6) 0 0ns, 1 6400ns -repeat 12800ns
force x(7) 0 0ns, 1 12800ns -repeat 25600ns
force x(8) 0 0ns, 1 25600ns -repeat 51200ns
force x(9) 0 0ns, 1 51200ns -repeat 102400ns
force x(10) 0 0ns, 1 102400ns -repeat 204800ns
force x(11) 0 0ns, 1 204800ns -repeat 409600ns
force x(12) 0 0ns, 1 409600ns -repeat 819200ns
force x(13) 0 0ns, 1 819200ns -repeat 1638400ns
force x(14) 0 0ns, 1 1638400ns -repeat 3276800ns
force x(15) 0 0ns, 1 3276800ns -repeat 6553600ns

run 6553650 ns
```

7.24 Vi skriver koden. Åter igen lägger vi i default-värden på utsignalerna i början av `case`-satsen

```
------------
-- ex7_24 --
------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

7.24 forts.

```
ENTITY ex7_24 IS
   PORT(x:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        two:OUT STD_LOGIC;
        three:OUT STD_LOGIC;
        five:OUT STD_LOGIC;
        seven:OUT STD_LOGIC;
        eleven:OUT STD_LOGIC;
        thirteen:OUT STD_LOGIC);
END ex7_24;

ARCHITECTURE arch_ex7_24 OF ex7_24 IS
BEGIN
   one_proc:
   PROCESS(x)
   BEGIN
      two<='0';
      three<='0';
      five<='0';
      seven<='0';
      eleven<='0';
      thirteen<='0';
      CASE x IS
         WHEN "0000" =>

         WHEN "0001" =>

         WHEN "0010" =>
            two<='1';
         WHEN "0011" =>
            three<='1';
         WHEN "0100" =>
            two<='1';
         WHEN "0101" =>
            five<='1';
         WHEN "0110" =>
            two<='1';
            three<='1';
         WHEN "0111" =>
            seven<='1';
         WHEN "1000" =>
            two<='1';
         WHEN "1001" =>
            three<='1';
```

7.24 forts.

```
            WHEN "1010" =>
               two<='1';
               five<='1';
            WHEN "1011" =>
               eleven<='1';
            WHEN "1100" =>
               two<='1';
               three<='1';
            WHEN "1101" =>
               thirteen<='1';

            WHEN "1110" =>
               two<='1';
               seven<='1';
            WHEN "1111" =>
               three<='1';
               five<='1';
            WHEN OTHERS =>
         END CASE;
      END PROCESS one_proc;
   END arch_ex7_24;
```

Som vi simulerar med en do-fil

```
---------------
-- ex7_24.do --
---------------

restart -f -nowave
view signals wave
add wave -radix binary x
add wave -radix unsigned x
add wave two three five seven eleven thirteen

force x(0) 0 0ns, 1 100ns -repeat 200ns
force x(1) 0 0ns, 1 200ns -repeat 400ns
force x(2) 0 0ns, 1 400ns -repeat 800ns
force x(3) 0 0ns, 1 800ns -repeat 1600ns

run 1650 ns
```