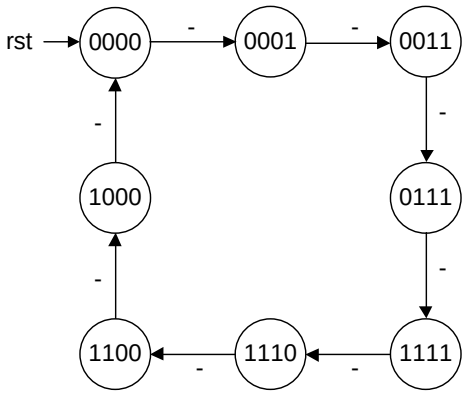
Dally, Harting and Aamodt: Digital Design Using VHDL

Kapitel 16

16.1 Vi har uttrycket

```
nxt <= ”0000” WHEN rst = ’1’ ELSE
       state(2 DOWNTO 0) & NOT(state(3));
```

som leder till sekvensen i *Figur 16.1*, dvs vi går upprepande genom en sekvens av åtta olika 4-bits värden.
I uppgiften står det fel och reset-värdet anges med bara tre bitar



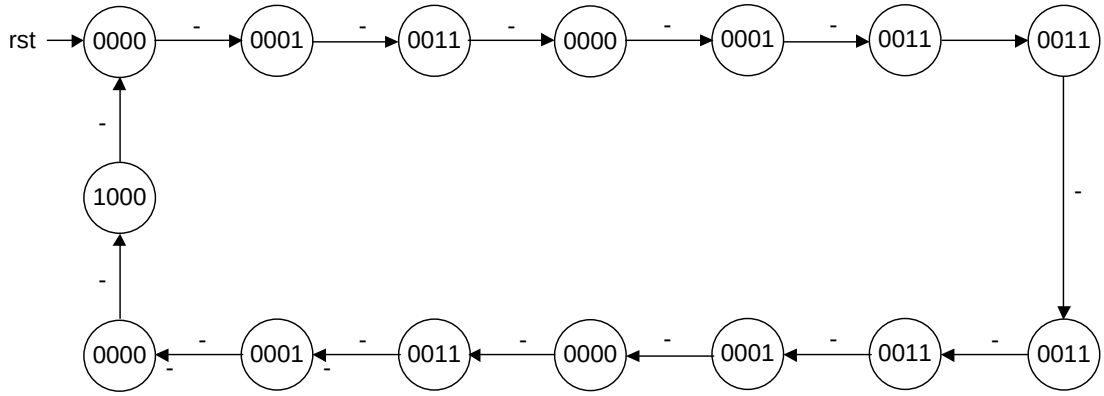
Figur 16.1

16.2 Vi har uttrycket

```
nxt <= ”0001” WHEN rst = ’1’ ELSE
       state(2 DOWNTO 0) & ((state(3) XOR state(2));
```

som leder till sekvensen i *Figur 16.2*, dvs vi går upprepande genom en sekvens av 15 olika 4-bits värden, dvs alla möjliga värden utom `0000`.

16.2 forts.

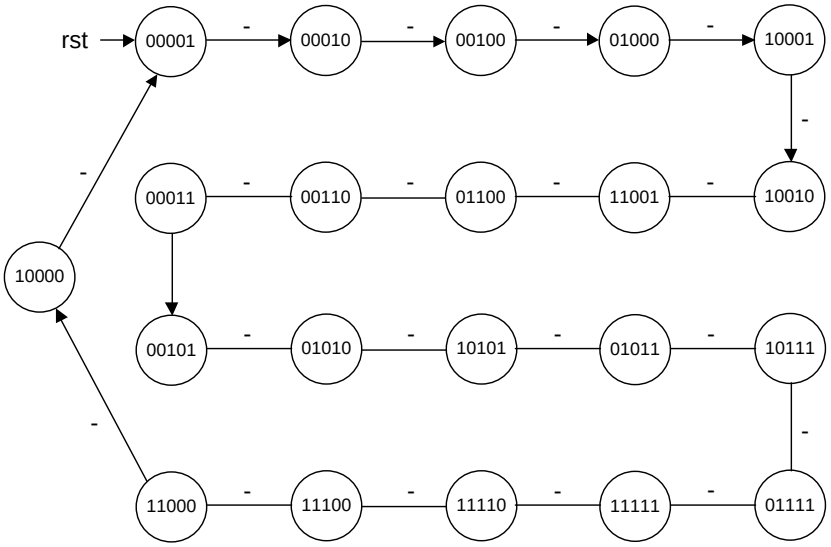


Figur 16.2

16.3 Vi har uttrycket

```
nxt <= ”00001” WHEN rst = ’1’ ELSE
       state(3 DOWNTO 0) & ((state(3) XOR state(2));
```

som leder till sekvensen i *Figur 16.3*, dvs vi går upprepande genom en sekvens av 21 olika 4-bits värden.



Figur 16.3

16.4 I bokens uppgifter gör man på flera ställen så att man först ritar blockschema och sedan skriver VHDL-kod för att koppla ihop blocken till ett system. Om blocken är små och bara beskriver grundfunktioner och inte specialiserade delsystem så är det inte befogat att göra så utan då är det bättre att direkt skriva VHDL-kod och det är vad jag gör i dessa lösningar.

16.5Vi skriver VHDL-kod. Koden är generisk vilket betyder at den kan konfigureras om genom att ändra en GENERIC-parameter i entiteten. Här ändrar vi räknarens storlek genom att via en GENERIC sätta antalet bitar i räknevärdet.
I arkitekturen används en decimal räknarvariabel i ställe för ett bitvärde bara för att det känns naturligare att jobba med decimaltal. Den resulterande koden kommer att bli den samma i båda fallen

```
-----------------
-- ex16_5.vhdl --
-----------------

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

ENTITY ex16_5 IS
   GENERIC(WIDTH:NATURAL:=6);
   PORT (clk:IN STD_LOGIC;
         reset_n:IN STD_LOGIC;
         up_down:IN STD_LOGIC;
         load:IN STD_LOGIC;
         count_max:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
         count:OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0));
END ex16_5;

ARCHITECTURE arch_ex16_5 OF ex16_5 IS
   SIGNAL count_signal:NATURAL RANGE 0 TO 2**WIDTH-1;
BEGIN
   count_process:
   PROCESS(reset_n,clk)
   BEGIN
      IF (reset_n='0') THEN
         count_signal<=0;
```

16.5forts.

```
         ELSIF RISING_EDGE(clk) THEN
            IF (load = '1') THEN
               count_signal <=
                           TO_INTEGER(UNSIGNED(count_max));
            ELSIF (up_down='1') THEN
               IF (count_signal <
                        TO_INTEGER(UNSIGNED(count_max))) THEN
                  count_signal <= count_signal + 1;
               END IF;
            ELSE
               IF (count_signal > 0) THEN
                  count_signal <= count_signal - 1;
               END IF;
            END IF;
         END IF;
      END PROCESS count_process;

   count<=STD_LOGIC_VECTOR(TO_UNSIGNED(count_signal,WIDTH));
   END arch_ex16_5;
```

Vi simulerar med en do-fil

```
---------------
-- ex16_5.do --
---------------

restart -f -nowave
view signals wave
add wave reset_n clk load up_down
add wave count count_max count
force clk 1 0ns, 0 50ns -repeat 100ns
force count_max 6'b001100
force up_down 1
force load 0
force reset_n 1
run 125ns
force reset_n 0
run 200ns
force reset_n 1
run 1700ns
force up_down 0
run 2000ns
```

16.5 forts.

```
force up_down 1
run 900ns
force count_max 6'b000011
run 400ns
```

16.6 Vi utökar koden från *Exempel 16.5*. Tyvärr tillåter inte VHDL att man deklarerar en signaltyp vars värdeområde styrs av en GENERIC. Vi har två alternativ, att låta området vara fixt eller tillåta alla värden på räknevärdet. Jag har valt det senare även om det då leder till 32 bitar breda signaler

```
-----------------
-- ex16_6.vhdl --
-----------------

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

ENTITY ex16_6 IS
   GENERIC(WIDTH:NATURAL:=6);
   PORT (clk:IN STD_LOGIC;
         reset_n:IN STD_LOGIC;
         up_down:IN STD_LOGIC;
         load:IN STD_LOGIC;
         reg:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
         count_max:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
         count:OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0));
END ex16_6;

ARCHITECTURE arch_ex16_6 OF ex16_6 IS
   TYPE count_array IS ARRAY (0 TO 3) OF NATURAL;
   SIGNAL count_signal:count_array;
   SIGNAL reg_int:NATURAL RANGE 0 TO 3;
BEGIN
   reg_int <= TO_INTEGER(UNSIGNED(reg));
   count_process:
   PROCESS(reset_n,clk)
   BEGIN
      IF (reset_n='0') THEN
         FOR index IN 0 TO 3 LOOP
            count_signal(index) <= 0;
         END LOOP;
```

16.6 forts.

```
    ELSIF RISING_EDGE(clk) THEN
          IF (load = '1') THEN
             count_signal(reg_int) <=
                      TO_INTEGER(UNSIGNED(count_max));
          ELSIF (up_down='1') THEN
             IF (count_signal(reg_int) <
                  TO_INTEGER(UNSIGNED(count_max))) THEN
                count_signal(reg_int) <=
                            count_signal(reg_int) + 1;
             END IF;
          ELSE
             IF (count_signal(reg_int) > 0) THEN
                count_signal(reg_int) <=
                            count_signal(reg_int) - 1;
             END IF;
          END IF;
       END IF;
    END PROCESS count_process;
    count<=STD_LOGIC_VECTOR(
                TO_UNSIGNED(count_signal(reg_int),WIDTH));
END arch_ex16_6;
```

Återigen använder vi en do-fil

```
---------------
-- ex16_6.do --
---------------

restart -f -nowave
view signals wave
add wave reset_n clk load up_down reg
add wave count count_max count_signal count
force clk 1 0ns, 0 50ns -repeat 100ns
force count_max 6'b001100
force up_down 1
force reg 2'b00
force load 0
force reset_n 1
run 125ns
force reset_n 0
run 200ns
force reset_n 1
run 1700ns
```

16.6 forts.

```
force up_down 0
run 2000ns
force up_down 1
run 900ns
force reg 2'b01
run 400ns
```

16.7 Det är inte fullt klart hur det är tänkt att applikationen skall fungera så lösningen är en tolkning som inte är speciellt funktionell

```
-----------------
-- ex16_7.vhdl --
-----------------

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

ENTITY ex16_7 IS
   GENERIC(WIDTH:NATURAL:=6);
   PORT (clk:IN STD_LOGIC;
         reset_n:IN STD_LOGIC;
         up_down:IN STD_LOGIC;
         load:IN STD_LOGIC;
         rs:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
         rd:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
         count_max:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
         count:OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0));
END ex16_7;

ARCHITECTURE arch_ex16_7 OF ex16_7 IS
   TYPE count_array IS ARRAY (0 TO 3) OF NATURAL;
   SIGNAL count_signal:count_array;
   SIGNAL rs_int:NATURAL RANGE 0 TO 3;
   SIGNAL rd_int:NATURAL RANGE 0 TO 3;
BEGIN
   rs_int <= TO_INTEGER(UNSIGNED(rs));
   rd_int <= TO_INTEGER(UNSIGNED(rd));
```

16.7 forts.

```
   count_process:
   PROCESS(reset_n,clk)
   BEGIN
      IF (reset_n='0') THEN
         FOR index IN 0 TO 3 LOOP
            count_signal(index) <= 0;
         END LOOP;
      ELSIF RISING_EDGE(clk) THEN
         IF (load = '1') THEN
            count_signal(rs_int) <=
                        TO_INTEGER(UNSIGNED(count_max));
         ELSIF (up_down='1') THEN
            IF (count_signal(rd_int) <
                  TO_INTEGER(UNSIGNED(count_max))) THEN
               count_signal(rd_int) <=
                           count_signal(rs_int) + 1;
            END IF;
         ELSE
            IF (count_signal(rs_int) > 0) THEN
               count_signal(rd_int) <=
                           count_signal(rs_int) - 1;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count<=STD_LOGIC_VECTOR(
               TO_UNSIGNED(count_signal(rd_int),WIDTH));
END arch_ex16_7;
```

och vi simulerar

```
---------------
-- ex16_7.do --
---------------

restart -f -nowave
view signals wave
add wave reset_n clk load up_down rs rd
add wave count count_max count_signal count
force clk 1 0ns, 0 50ns -repeat 100ns
force count_max 6'b001100
force up_down 1
force rs 2'b00
```

16.7 forts.

```
force rd 2'b10
force load 0
force reset_n 1
run 125ns
force reset_n 0
run 200ns
force reset_n 1
run 1700ns
force up_down 0
run 2000ns
force up_down 1
run 900ns
force rs 2'b01
run 400ns
```

16.8 Vi utelämnar detta och går direkt på VHDL i *Exempel 16.9*

16.9 Vi skriver VHDL-kod

```
-----------------
-- ex16_9.vhdl --
-----------------

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

ENTITY ex16_9 IS
   PORT(reset_n:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        fibonacci:OUT STD_LOGIC_VECTOR(16 DOWNTO 0);
        to_high:OUT STD_LOGIC);
END ex16_9;

ARCHITECTURE arch_ex16_9 OF ex16_9 IS
   SIGNAL old_fibonacci_signal:NATURAL RANGE 0 TO 2**17-1;
   SIGNAL fibonacci_signal:NATURAL RANGE 0 TO 2**17-1;
BEGIN
```

16.9 forts.

```
   fibonacci_process:
   PROCESS(reset_n,clk)
   BEGIN
      IF (reset_n='0') THEN
         old_fibonacci_signal <= 0;
         fibonacci_signal <= 1;
         to_high <= '0';
      ELSIF RISING_EDGE(clk) THEN
         IF (fibonacci_signal < 2**16) THEN
            old_fibonacci_signal <= fibonacci_signal;
            fibonacci_signal <=
                old_fibonacci_signal + fibonacci_signal;
         ELSE
            to_high <= '1';
         END IF;
      END IF;
   END PROCESS fibonacci_process;
   fibonacci <=
STD_LOGIC_VECTOR(TO_UNSIGNED(fibonacci_signal,17));
END arch_ex16_9;
```

Som vi simulerar

```
---------------
-- ex16_9.do --
---------------

restart -f -nowave
view signals wave
add wave reset_n clk to_high
add wave -radix unsigned old_fibonacci_signal
add wave -radix unsigned fibonacci_signal
add wave fibonacci
add wave -radix unsigned fibonacci
force clk 1 0ns, 0 50ns -repeat 100ns
force reset_n 1
run 125ns
force reset_n 0
run 200ns
force reset_n 1
run 3500ns
```

16.10 Den lösning vi gav i *Exempel 14.20* uppfyller redan detta villkor