CHALMERS

Department of Computer Science and Engineering

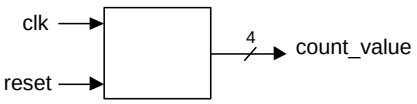
2017-01-11

EDA322 Digital konstruktion
Lösningar till uppgifter i boken
Kapitel 16

16.5 Vi skall konstruera en räknare i VHDL. Specifikationen är lite lös så vi preciserar den till ett antal punkter

a. Gör en 4 bitars uppräknare med reset. Räknaren skall bottna då den når maxvärde
b. Ändra så räknaren kan räkna både upp och ner. Räknaren skall bottna både vid uppräkning till maxvärde och vid nedräkning till noll
c. Lägg in styrsignal för att aktivera räkning
d. Lägg till en kodbestämd övre och undre räknegräns
e. Ge möjlighet att sätta styrbara övre och undre räknegränser för räknaren
f. Ge möjlighet att ladda in startvärde
g. Gör designern generisk
h. Heltal som räknarvariabler

a) Låt oss rita ett blockschema, *Figur 16.5a*.
Vi skriver VHDL-kod där vi antar att resetsignalen är aktivt låg.



Figur 16.5a Räknarversion a

```
-- ex16_5a.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5a IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5a;

ARCHITECTURE arch_ex16_5a OF ex16_5a IS
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
```

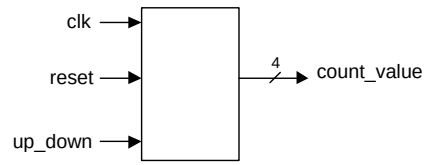
```
         count_signal<=(OTHERS=>'0');
      ELSIF rising_edge(clk) THEN
         IF (count_signal<"1111") THEN
            count_signal<=count_signal+1;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5a;
```

Då vi skall addera till räknevärdet så måste det gå att läsa. Därför inför vi en signal som används vid uppräkningen varefter värdet överförs till utgången. Det finns inga funktioner för att räkna upp STD_LOGIC_VECTOR så internt använder vi i stället undertypen UNSIGNED som finns definierad i biblioteket NUMERIC_STD. Vi har valt en asynkron reset och räkneprocessen får då triggas av `reset` eller `clk`.
Vi simulerar med en do-fil.

```
-- ex16_5a.do
restart -f -nowave
view signals wave
add wave reset clk count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
run 125ns
force reset 0
run 100ns
force reset 1
run 2000ns
```

b) Låt oss rita ett nytt blockschema, *Figur 16.5b*.
Vi kompletterar VHDL-koden med en styrsignal som ger uppräkning då den är etta och nedräkning då den är nolla.



Figur 16.5b Räknarversion b

```
-- ex16_5b.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5b IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5b;

ARCHITECTURE arch_ex16_5b OF ex16_5b IS
```

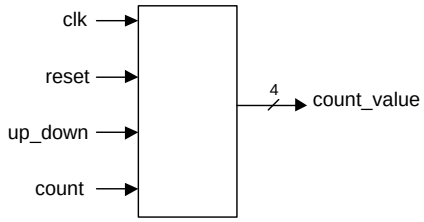
```
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=(OTHERS=>'0');
      ELSIF rising_edge(clk) THEN
         IF (up_down='1') THEN
            IF (count_signal<"1111") THEN
               count_signal<=count_signal+1;
            END IF;
         ELSE
            IF (count_signal>"0000") THEN
               count_signal<=count_signal-1;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5b;
```

Vi kompletterar do-filen

```
-- ex16_5b.do
restart -f -nowave
view signals wave
add wave reset clk up_down
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force up_down 1
run 125ns
force reset 0
run 100ns
force reset 1
run 1800ns
force up_down 0
run 1800ns
force up_down 1
run 500ns
force up_down 0
run 400ns
```

c) Vi ritar ett nytt blockschema, *Figur 16.5c*.
Vi kompletterar VHDL-koden med en styrsignal som aktiverar räkning. Vi gör signalen aktivt hög.



Figur 16.5c Räknarversion c

```
-- ex16_5c.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5c IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5c;

ARCHITECTURE arch_ex16_5c OF ex16_5c IS
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=(OTHERS=>'0');
      ELSIF rising_edge(clk) THEN
         IF (count='1') THEN
            IF (up_down='1') THEN
               IF (count_signal<"1111") THEN
                  count_signal<=count_signal+1;
               END IF;
            ELSE
               IF (count_signal>"0000") THEN
                  count_signal<=count_signal-1;
               END IF;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5c;
```

Och vi kompletterar do-filen

```
-- ex16_5c.do
restart -f -nowave
view signals wave
add wave reset clk count up_down
```

```
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force count 0
force up_down 1
run 125ns
force reset 0
run 100ns
force reset 1
run 400ns
force count 1
run 900ns
force count 0
run 300ns
force count 1
run 900ns
force up_down 0
run 1800ns
force up_down 1
run 500ns
force up_down 0
run 400ns
```

d) Här gör vi bara en intern ändring så det blir inget nytt blockschema.
Vi kompletterar VHDL-koden med ett maximalt räknar värde som kan vara något annat än 1111. För att göra det enkelt att ändra gränsen så lägger vi in den som en konstant. Samtidigt gör vi samma sak för den undre gränsen. Här får vi sätta räknevärdet till `min_const` vid reset för att inte komma utanför räkneområdet

```
-- ex16_5d.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5d IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5d;

ARCHITECTURE arch_ex16_5d OF ex16_5d IS
   CONSTANT min_const:UNSIGNED(3 DOWNTO 0):="0010";
   CONSTANT max_const:UNSIGNED(3 DOWNTO 0):="1011";
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
```

```
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=min_const;
      ELSIF rising_edge(clk) THEN
         IF (count='1') THEN
            IF (up_down='1') THEN
               IF (count_signal<max_const) THEN
                  count_signal<=count_signal+1;
               END IF;
            ELSE
               IF (count_signal>min_const) THEN
                  count_signal<=count_signal-1;
               END IF;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5d;
```

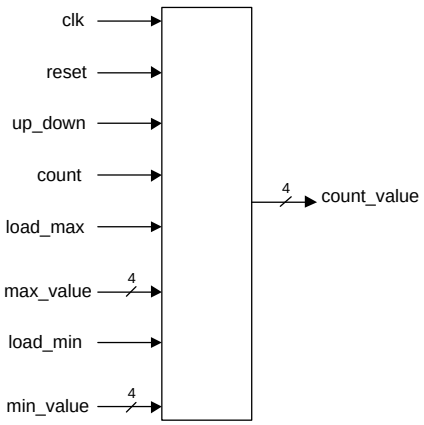
Och vi kompletterar do-filen

```
-- ex16_5d.do
restart -f -nowave
view signals wave
add wave reset clk count up_down
add wave max_const min_const
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force count 0
force up_down 1
run 125ns
force reset 0
run 100ns
force reset 1
run 400ns
force count 1
run 900ns
force count 0
run 300ns
force count 1
run 900ns
force up_down 0
run 1800ns
```

```
force up_down 1
run 500ns
force up_down 0
run 400ns
force up_down 1
run 1000ns
```

e) Vi ritar ett nytt blockschema, *Figur 16.5e*.
Vi kompletterar VHDL-koden med ingångar för max- och minvärde samt styrsignaler för att ladda in dessa. Vi gör styrsignalerna aktivt höga.
Lägg märke till konstruktionen där vi antingen laddar in nya gränser eller räknar. Lägg också märke till att vi samtidigt kan ladda in både max- och minnivå.
Lägg till sist märke till att upp- och nedräkningarna har ELSE-villkor för att säkerställa att en sänkt maxnivå eller höjd minnivå gör att vi hamnar utanför räkneintervallet.



Figur 16.5e Räknarversion e

```
-- ex16_5e.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5e IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        load_max:IN STD_LOGIC;
        max_value:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        load_min:IN STD_LOGIC;
        min_value:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5e;

ARCHITECTURE arch_ex16_5e OF ex16_5e IS
   SIGNAL max_signal:UNSIGNED(3 DOWNTO 0);
   SIGNAL min_signal:UNSIGNED(3 DOWNTO 0);
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=min_value;
      ELSIF rising_edge(clk) THEN
```

```
         IF ((load_max='1') OR (load_min='1')) THEN
            IF (load_max='1') THEN
            max_signal<=UNSIGNED(max_value);
         END IF;
            IF (load_min='1') THEN
               min_signal<=UNSIGNED(min_value);
            END IF;
         ELSIF (count='1') THEN
            IF (up_down='1') THEN
               IF (count_signal<max_signal) THEN
                  count_signal<=count_signal+1;
               ELSE
                  count_signal<=max_signal;
               END IF;
            ELSE
               IF (count_signal>min_signal) THEN
                  count_signal<=count_signal-1;
               ELSE
                  count_signal<=count_signal;
               END IF;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5e;
```

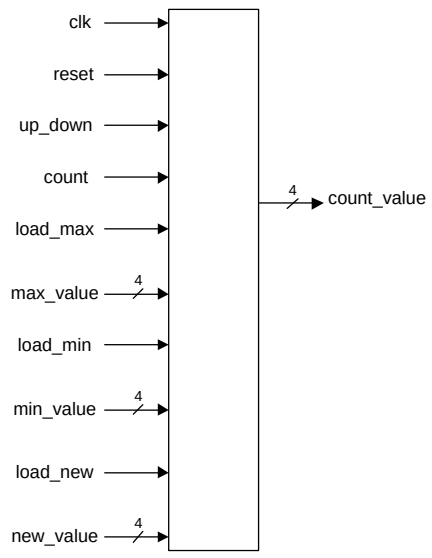
Vi uppdaterar do-filen

```
-- ex16_5e.do
restart -f -nowave
view signals wave
add wave reset clk count up_down
add wave load_max max_value max_signal
add wave load_min min_value min_signal
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force count 0
force up_down 1
force load_max 0
force max_value 4'b0101
force load_min 0
force min_value 4'b0100
run 125ns
force reset 0
run 100ns
```

```
force reset 1
run 400ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force count 1
run 900ns
force load_min 1
run 100ns
force load_min 0
run 900ns
force count 0
run 300ns
force count 1
run 900ns
force up_down 0
run 1800ns
force up_down 1
run 500ns
force up_down 0
run 400ns
force max_value 4'b1011
run 100ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force up_down 1
run 1000ns
```

f) Vi ritar ett nytt blockschema, *Figur 16.5f*.
Vi kompletterar VHDL-koden med ingång för nytt räknarvärde samt styrsignal för att ladda in detta. Vi gör styrsignalen aktivt hög.
Lägg märke till att även det nya värdet kan läggas in samtidigt med nya gränserna samt att vi bara laddar i ett nytt räknarvärde om det ligger inom tillåtet räkneområde



Figur 16.5e Räknarversion f

```
-- ex16_5_5f.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5f IS
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        load_max:IN STD_LOGIC;
        max_value:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        load_min:IN STD_LOGIC;
        min_value:IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        load_new:IN STD_LOGIC;
        new_value:IN STD_LOGIC_VECTOR(3 DOWNTO 0);

        count_value:OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ex16_5f;

ARCHITECTURE arch_ex16_5f OF ex16_5f IS
   SIGNAL max_signal:UNSIGNED(3 DOWNTO 0);
   SIGNAL min_signal:UNSIGNED(3 DOWNTO 0);
   SIGNAL count_signal:UNSIGNED(3 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=UNSIGNED(min_value);
      ELSIF rising_edge(clk) THEN
         IF ((load_max='1') OR
             (load_min='1') OR
             (load_new='1')) THEN
            IF (load_max='1') THEN
            max_signal<=UNSIGNED(max_value);
         END IF;
         END IF;
         IF (load_min='1') THEN
            min_signal<=UNSIGNED(min_value);
         END IF;
         IF ((load_new='1') AND
             (UNSIGNED(new_value)<=UNSIGNED(max_value)) AND
             (UNSIGNED(new_value)>=UNSIGNED(min_value))) THEN
            count_signal<=UNSIGNED(new_value);
         END IF;
      ELSIF (count='1') THEN
         IF (up_down='1') THEN
            IF (count_signal<max_signal) THEN
```

```
               count_signal<=count_signal+1;
            ELSE
               count_signal<=max_signal;
            END IF;
         ELSE
            IF (count_signal>min_signal) THEN
               count_signal<=count_signal-1;
            ELSE
               count_signal<=min_signal;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5f;
```

Vi uppdaterar do-filen

```
-- ex16_5f.do
restart -f -nowave
view signals wave
add wave reset clk count up_down
add wave load_max max_value max_signal
add wave load_min min_value min_signal
add wave load_new new_value
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force count 0
force up_down 1
force load_max 0
force max_value 4'b1010
force load_min 0
force min_value 4'b0100
force load_new 0
force new_value 4'b0111
run 125ns
force reset 0
run 100ns
force reset 1
run 400ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force count 1
run 900ns
```

```
force load_min 1
run 100ns
force load_min 0
run 900ns
force count 0
run 300ns
force count 1
run 900ns
force up_down 0
run 1800ns
force up_down 1
run 500ns
force up_down 0
run 400ns
force max_value 4'b1011
run 100ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force up_down 1
run 1000ns
force load_new 1
run 100ns
force load_new 0
run 1000ns
force new_value 4'b1101
run 100ns
force load_new 1
run 100ns
force load_new 0
run 1000ns
```

g) Här gör vi bara en ändring i koden som inte förändrar blockschemat. Notera att vi har två GENERIC rader och genom att välja vilken som är okomenterad så väljer vi antalet bitar. I det här fallet 4 bitar så resultatet blir det samma som i *ex16.5f*

```
-- ex16_5g_4bit.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY ex16_5g_4bit IS
   GENERIC (WIDTH:NATURAL:=4);
--   GENERIC (WIDTH:NATURAL:=8);
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
```

```
        up_down:IN STD_LOGIC;
        load_max:IN STD_LOGIC;
        max_value:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
        load_min:IN STD_LOGIC;
        min_value:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);
        load_new:IN STD_LOGIC;
        new_value:IN STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0);

        count_value:OUT STD_LOGIC_VECTOR(WIDTH-1 DOWNTO 0));
END ex16_5g_4bit;

ARCHITECTURE arch_ex16_5g_4bit OF ex16_5g_4bit IS
   SIGNAL max_signal:UNSIGNED(WIDTH-1 DOWNTO 0);
   SIGNAL min_signal:UNSIGNED(WIDTH-1 DOWNTO 0);
   SIGNAL count_signal:UNSIGNED(WIDTH-1 DOWNTO 0);
BEGIN
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=UNSIGNED(min_value);
      ELSIF rising_edge(clk) THEN
         IF ((load_max='1') OR
             (load_min='1') OR
             (load_new='1')) THEN
            IF (load_max='1') THEN
            max_signal<=UNSIGNED(max_value);
         END IF;
         IF (load_min='1') THEN
            min_signal<=UNSIGNED(min_value);
         END IF;
         IF ((load_new='1') AND
             (UNSIGNED(new_value)<=UNSIGNED(max_value)) AND
             (UNSIGNED(new_value)>=UNSIGNED(min_value))) THEN
            count_signal<=UNSIGNED(new_value);
         END IF;
         ELSIF (count='1') THEN
            IF (up_down='1') THEN
               IF (count_signal<max_signal) THEN
                  count_signal<=count_signal+1;
               ELSE
                  count_signal<=max_signal;
               END IF;
            ELSE
               IF (count_signal>min_signal) THEN
                  count_signal<=count_signal-1;
               ELSE
                  count_signal<=min_signal;
               END IF;
```

```
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;
   count_value<=STD_LOGIC_VECTOR(count_signal);
END arch_ex16_5g_4bit;
```

Vi kan använda samma do-fil som i *ex16.5f.*

För att generera kod med åtta bitars ordlängd behöver vi nu bara växla till att ha den andra GENERIC-raden okommenterad.
Vi måste uppdatera do-filen så de stimuli vi ger har längden åtta bitar

```
-- arch_ex16_5g_8bit.do
restart -f -nowave
view signals wave
add wave reset clk count up_down
add wave load_max max_value max_signal
add wave load_min min_value min_signal
add wave load_new new_value
add wave count_signal count_value

force reset 1
force clk 0 0,1 50ns -repeat 100ns
force count 0
force up_down 1
force load_max 0
force max_value 8'b10100011
force load_min 0
force min_value 8'b01000011
force load_new 0
force new_value 8'b01110011
run 125ns
force reset 0
run 100ns
force reset 1
run 400ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force count 1
run 900ns
force load_min 1
run 100ns
force load_min 0
run 900ns
force count 0
```

```
run 300ns
force count 1
run 900ns
force up_down 0
run 1800ns
force up_down 1
run 500ns
force up_down 0
run 400ns
force max_value 8'b10110011
run 100ns
force load_max 1
run 100ns
force load_max 0
run 100ns
force up_down 1
run 1000ns
force load_new 1
run 100ns
force load_new 0
run 1000ns
force new_value 8'b11010011
run 100ns
force load_new 1
run 100ns
force load_new 0
run 1000ns
```

h) Låt oss försöka göra samma sak med heltal som variabeltyp. Interfacet mot yttervärlden måste fortfarande vara `std_logic` så entiteten påverkas inte

```
-- ex16_5h.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
USE ieee.math_real.all;

ENTITY ex16_5h IS
   GENERIC (MAXIMAL_INT:NATURAL:=4);
--   CONSTANT
WIDTH:NATURAL:=INTEGER(CEIL(LOG2(REAL(MAX_INT))));
   PORT(reset:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        count:IN STD_LOGIC;
        up_down:IN STD_LOGIC;
        load_max:IN STD_LOGIC;
        max_value:IN STD_LOGIC_VECTOR(INTEGER(CEIL
                      (LOG2(REAL(MAXIMAL_INT))))-1 DOWNTO 0);
```

```
        load_min:IN STD_LOGIC;
        min_value:IN STD_LOGIC_VECTOR(INTEGER(CEIL
                        (LOG2(REAL(MAXIMAL_INT))))-1 DOWNTO 0);
        load_new:IN STD_LOGIC;
        new_value:IN STD_LOGIC_VECTOR(INTEGER(CEIL
                        (LOG2(REAL(MAXIMAL_INT))))-1 DOWNTO 0);

        count_value:OUT STD_LOGIC_VECTOR(INTEGER
                  (CEIL(LOG2(REAL(MAXIMAL_INT))))-1 DOWNTO 0));
END ex16_5h;

ARCHITECTURE arch_ex16_5h OF ex16_5h IS
   CONSTANT
WIDTH:NATURAL:=INTEGER(CEIL(LOG2(REAL(MAXIMAL_INT))));
   SIGNAL max_signal:NATURAL RANGE 0 TO MAXIMAL_INT;
   SIGNAL min_signal:NATURAL RANGE 0 TO MAXIMAL_INT;
   SIGNAL min_int:NATURAL RANGE 0 TO MAXIMAL_INT;
   SIGNAL max_int:NATURAL RANGE 0 TO MAXIMAL_INT;
   SIGNAL new_int:NATURAL RANGE 0 TO MAXIMAL_INT;
   SIGNAL count_signal:NATURAL RANGE 0 TO MAXIMAL_INT;
BEGIN
   min_int<=TO_INTEGER(UNSIGNED(min_value));
   max_int<=TO_INTEGER(UNSIGNED(max_value));
   new_int<=TO_INTEGER(UNSIGNED(new_value));
   count_process:
   PROCESS(reset,clk)
   BEGIN
      IF (reset='0') THEN
         count_signal<=0;
      ELSIF rising_edge(clk) THEN
         IF ((load_max='1') OR
             (load_min='1') OR
             (load_new='1')) THEN
            IF (load_max='1') THEN
            max_signal<=max_int;
         END IF;
         IF (load_min='1') THEN
            min_signal<=min_int;
         END IF;
         IF ((load_new='1') AND
             (new_int<=max_int) AND
             (new_int>=min_int)) THEN
            count_signal<=TO_INTEGER(UNSIGNED(new_value));
         END IF;
         ELSIF (count='1') THEN
            IF (up_down='1') THEN
               IF (count_signal<max_signal) THEN
                  count_signal<=count_signal+1;
               ELSE
```

```
                  count_signal<=max_signal;
               END IF;
            ELSE
               IF (count_signal>min_signal) THEN
                  count_signal<=count_signal-1;
               ELSE
                  count_signal<=count_signal;
               END IF;
            END IF;
         END IF;
      END IF;
   END PROCESS count_process;

count_value<=STD_LOGIC_VECTOR(TO_UNSIGNED(count_signal,WIDTH));
END arch_ex16_5h;
```