PROCESS-Anweisung

- Prozesse sind nebenläufige (also parallele) Anweisungen mit einem sequentiellen Anweisungsbereich.
  - Prozesse dienen zur algorithmischen Verhaltensmodellierung von synchronen und asynchronen Schaltwerken, von Schaltnetzen und zur Beschreibung von Testumgebungen für die Simulation.

Optionale Sensitivitätsliste gibt Signale aus der Umgebung an, über die ein Prozess zur Ausführung „angestoßen“ wird. Ist die Sensitivitätsliste nicht vorhanden, so muss mindestens eine WAIT-Anweisung im Prozess-rumpf stehen. Bsp.: (rst, clk)

Prozesse können optional benannt werden. Bsp.: fsm

```
name: PROCESS Sensitivitätsliste IS
  -- Deklarationsbereich
BEGIN
  -- sequentieller Anweisungsbereich
END PROCESS;
```

Signal-/Variablenzuweisungen
Prozedur-Funktionsaufrufe
IF-/CASE-/LOOP-Anweisungen
WAIT-Anweisungen
Assertion-Anweisungen
NULL-Anweisungen
NEXT-/EXIT-/RETURN-Anweisungen

PROCESS-Anweisung

- Prozesse kommunizieren mit Hilfe von (Handshake-)Signalen miteinander => Es ist eine sorgfältige Planung von Handshake-Mechanismen erforderlich. Fehlerhaft implementierte Handshake-Mechanismen sind eine häufige Ursache für schwer zu analysierende Laufzeitfehler (sporadisch, oft schlecht reproduzierbar).
- Prozesse dürfen nur in Architekturbeschreibungen stehen und können nicht ineinander verschachtelt werden.
- Prozesse bilden eine statische Gruppe und lassen sich dynamisch nicht erzeugen oder löschen.
- Ein Prozess kann (während der Simulation) einen von zwei möglichen Zuständen annehmen:
  - entweder ist der Prozess aktiv und seine Anweisungen werden gerade abgearbeitet,
  - oder der Prozess ist suspendiert und wartet auf ein für ihn relevantes Ereignis.

IF-Anweisung

- Die IF-Anweisung ist eine Steuerflussanweisung, die bedingte Verzweigungen in sequentiellen Anweisungsbereichen ermöglicht.
  - Mit einer IF-Anweisung kann das Verhalten einer bedingten Signal-zuweisung innerhalb eines sequentiellen Bereiches nachgebildet werden.
  - IF-Anweisungen können ineinander verschachtelt werden.

```
IF Bedingung_1 THEN
  -- sequentielle Anweisungen
ELSIF Bedingung_2 THEN
  -- sequentielle Anweisungen
ELSIF Bedingung_3 THEN
  -- sequentielle Anweisungen
ELSE
  -- sequentielle Anweisungen
END IF;
```

Optionale ELSIF-Kette

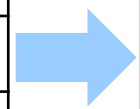
Optionaler ELSE-Block

Steht am Ende der ELSIF-Kette ein abschließender ELSE-Block, so werden die darin enthaltenen Anweisungen nur dann ausgeführt, wenn weder die Bedingung der (ersten) IF-Anweisung noch aller folgenden ELSIF-Anweisungen erfüllt waren.

IF-Anweisung

- Verhalten einer Prioritätskette
  - Aus der Reihenfolge in der IF-ELSIF-ELSE-Struktur ergibt eine Prioritätskette mit fester Priorisierungsfolge.
    - Jeder IF- bzw. ELSIF-Block beginnt mit einer Bedingung. Nur wenn diese erfüllt ist, werden die dazugehörigen Anweisungen auch ausgeführt.
    - Ansonsten wird die Bedingung des nächsten ELSIF-Blocks ausgewertet.

sel(0)	sel(1)	sel(2)	q
1	-	-	a
0	1	-	b
0	0	1	c
0	0	0	d

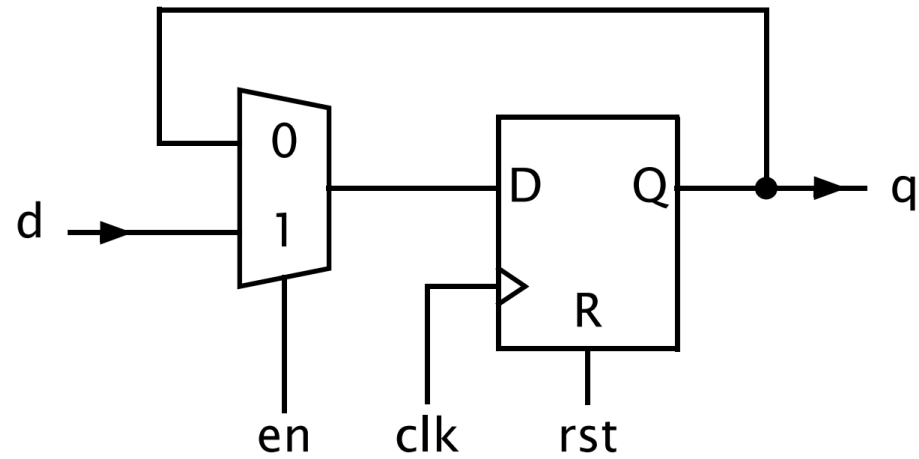


```
decoder: PROCESS (s, a, b, c, d) IS
BEGIN
  IF s(0)='1' THEN
    q <= a;
  ELSIF s(1)='1' THEN
    q <= b;
  ELSIF s(2)='1' THEN
    q <= c;
  ELSE
    q <= d;
  END IF;
END PROCESS;
```

PROCESS- mit IF-Anweisung

- Das Verhalten eines flankengesteuerten D-Flipflops mit einem asynchronen Rücksetzsignal (rst) und einem Enable-Signal (en)

rst	clk	en	d	q^{t+1}
1	–	–	–	0
0	↑	0	–	q^t
0	↑	1	0	0
0	↑	1	1	1



PROCESS- mit IF-Anweisung

- Das Verhalten eines flankengesteuerten D-Flipflops mit einem asynchronen Rücksetzsignal und einem Enable-Signal wird in VHDL mit Hilfe von PROCESS- und IF-Anweisungen modelliert.

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

ENTITY flipflop IS

  GENERIC(RSTDEF: std_logic := '1');

  PORT(rst:  IN   std_logic;
       clk:  IN   std_logic;
       en:   IN   std_logic;
       d:    IN   std_logic;
       q:    OUT  std_logic);

END flipflop;
```

```
ARCHITECTURE verhalten OF flipflop IS

  SIGNAL dff: std_logic; -- lokales Signal

BEGIN

  q <= dff;

  p1: PROCESS (rst, clk) IS
  BEGIN
    IF rst=RSTDEF THEN
      dff <= '0'; -- asynchrones Rücksetzen
    ELSIF rising_edge(clk) THEN
      IF en='1' THEN
        dff <= d; -- synchrone Datenübernahme
      END IF;
    END IF;
  END PROCESS;

END verhalten;
```

PROCESS- mit IF-Anweisung

- Syntheseregel:
  - ein Speicherelement wird u.a. immer dann erzeugt, wenn ein Signal innerhalb einer PROCESS-Anweisung nur in einigen, aber nicht in allen Zweigen einer IF- oder einer CASE-Anweisung einen Wert zugewiesen bekommt.
  - Es hängt vom Beschreibungsstil ab, ob ein flankengesteuertes Flipflop oder ein Latch generiert wird.

```
ENTITY element IS
  PORT(a, b: IN  std_logic;
       c:    OUT std_logic);
END element;
```

```
-- Tristate-Treiber
PROCESS (a, b) IS
BEGIN
  IF a='1' THEN
    c <= b;
  ELSE
    c <= 'Z';
  END IF;
END PROCESS;
```

```
-- Latch
PROCESS (a, b) IS
BEGIN
  IF a='1' THEN
    c <= b;
  ELSE
    c <= 'Z';
  END IF;
END PROCESS;
```

```
-- Flipflop
PROCESS (a, b) IS
BEGIN
  IF a='1' THEN
    c <= b;
  ELSE
    c <= 'Z';
  END IF;
END PROCESS;
```

- Eine synthesegerechte Schnittstellenbeschreibung eines parametrisierbaren Registers

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

ENTITY std_register IS

   GENERIC(RSTDEF: std_logic := '1'; -- aktiver Pegel des Reset-Signals
           LENDEF: natural := 8);    -- Anzahl der Bitstellen im Register

   PORT(rst:   IN  std_logic;   -- reset, RSTDEF active
        clk:   IN  std_logic;   -- clock, rising edge active
        swrst: IN  std_logic;   -- software reset, RSTDEF active
        en:    IN  std_logic;   -- enable, high active

        -- ggf. weitere Steuersignale: inkrementieren, schieben, usw.

        din:   IN  std_logic_vector(LENDEF-1 DOWNTO 0);  -- data input
        q:     OUT std_logic_vector(LENDEF-1 DOWNTO 0)); -- data output

END std_register;
```

- Eine synthesegerechte Architekturbeschreibung eines parametrisierbaren Registers

```
ARCHITECTURE verhalten OF std_register IS
   SIGNAL reg: std_logic_vector(LENDEF-1 DOWNTO 0); -- das eigentliche Register
BEGIN
   q <= reg;
   p1: PROCESS (rst, clk) IS
   BEGIN
      IF rst=RSTDEF THEN
         reg <= (OTHERS => '0');
      ELSIF rising_edge(clk) THEN
         IF en='1' THEN
            -- ggf. mit Abfrage weiterer Steuersignale
            reg <= din;
         END IF;
         IF swrst=RSTDEF THEN
            reg <= (OTHERS => '0');
         END IF;
      END IF;
   END PROCESS;
END verhalten;
```

- synchroner Modulo-N-Zähler mit einem Enable

rst	clk	en	cnt^t	cnt^{t+1}
1	-	-	-	0
0	↑	0	-	cnt^t
0	↑	1	=N-1	0
0	↑	1	<N-1	cnt^t+1

```
USE ieee.std_logic_unsigned.ALL;

CONSTANT N: natural := 12;
SIGNAL cnt: std_logic_vector(0 TO 3);
  . . .
p1: PROCESS (rst, clk) IS
BEGIN
  IF rst='1' THEN
    cnt <= (OTHERS => '0');
  ELSIF rising_edge(clk) THEN
    IF en='1' THEN
      IF cnt=N-1 THEN
        cnt <= (OTHERS => '0');
      ELSE
        cnt <= cnt + 1;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

```
CONSTANT N: natural := 12;
SIGNAL cnt: integer RANGE 0 TO N-1;
  . . .
p1: PROCESS (rst, clk) IS
BEGIN
  IF rst='1' THEN
    cnt <= 0;
  ELSIF rising_edge(clk) THEN
    IF en='1' THEN
      IF cnt=N-1 THEN
        cnt <= 0;
      ELSE
        cnt <= cnt + 1;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

- synchroner Frequenzteiler (Strobe-Generator) mit einem Enable

rst	clk	en	cnt^t	cnt^{t+1}	strb
1	-	-	-	0	0
0	↑	0	-	cnt^t	0
0	↑	1	=N-1	0	1
0	↑	1	<N-1	cnt^t+1	0

```
CONSTANT N: natural := 4;
SIGNAL cnt: integer RANGE 0 TO N-1;
SIGNAL strb: std_logic;
  . . .
```

```
p1: PROCESS (rst, clk) IS
BEGIN
  IF rst='1' THEN
    cnt <= 0;
  ELSIF rising_edge(clk) THEN
    IF en='1' THEN
      IF cnt=N-1 THEN
        cnt <= 0;
      ELSE
        cnt <= cnt + 1;
      END IF;
    END IF;
  END IF;
END PROCESS;
```