VHDL
Ein Überblick

Thomas B. Preußer
preusser@ite.inf.tu-dresden.de

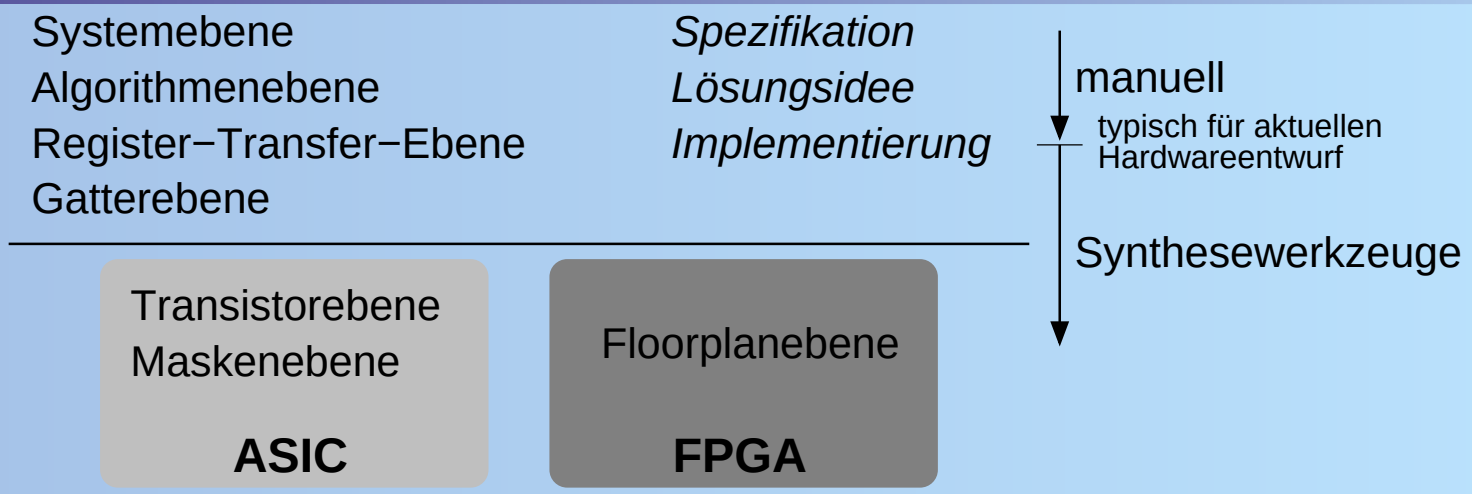
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/

Geschichte

VHDL – Very High Speed Integrated Circuit (VHSIC) Hardware Description Language

1981	Initiierung vom US-amerikanischen Verteidigungsministerium (DoD) zur Vereinfachung der Reproduktion von Hardware auf neuen Technologien: „Hardware Life Cycle Crisis“
1983	Auftrag an Intermetrics, IBM und TI zum Entwurf einer HDL
1985	Fertigstellung der Basissprache VHDL mit Version 7.2
1986	DoD überlässt IEEE alle Rechte an VHDL
1987	VHDL wird IEEE-Standard 1076-1987
1987	DoD Mil Std 454: ausschließliche Beschaffung von ASICs mit vollständigem VHDL-Modell
1988	VHDL wird ANSI-Standard
1993	Überarbeitung zum IEEE-Standard 1076-1993

Abstraktionsebenen



- VHDL erlaubt Beschreibungen von der Systemebene bis zur Gatterebene
- die automatische (technologiespezifische) Synthese ab der RT-Ebene ist üblich
- die Synthese von höheren Beschreibungsebenen ist Forschungsaufgabe

Syntaxüberblick

Vorbild: ADA (darum auch gewisse Ähnlichkeit mit Pascal)

- strikte Trennung zwischen Schnittstelle und Implementierung:

```
-- Schnittstelle
entity FA is
  port(
    a, b, c : in  bit;
    s, cout : out bit;
  );
end FA;
```

```
-- (Eine) Implementierung
architecture FA_1 of FA is
begin
  s    <= a xor b xor c;
  cout <= (a and b) or (a and c)
          or (b and c);
end FA_1;
```

- Aber: Beides in *einer* `.vhdl`-Datei.
- Kommentare beginnen mit `--`.
- Zwischen Groß- und Kleinschreibung wird *nicht* unterschieden.

Basisdatentypen

- Basisdatentypen:

bit	einzelnes Bit
integer	ganze Zahl
natural	natürliche Zahl
positive	positive Zahl

- Aufzählungstypen:

type tState **is** (S0, S1, S2, S3);

Abgeleitete Datentypen

- Ableitung von Feldtypen:

```
type byte is array(7 downto 0) of bit; -- Feste Grenzen
type bit_vector is array(natural range <>) of bit;
-- Freie Grenzen (ein vordefinierter Datentyp)
```

- Mehrdimensionale Felder:

```
type matrix is
  array(natural range <>, natural range <>) of integer;
```

- Unterdatentypen:

```
subtype z16  is natural range 0 to 15;
subtype byte is bit_vector(7 downto 0);
-- Alternative zur vorherigen Definiton von byte
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Operatoren

Nach aufsteigender Priorität:

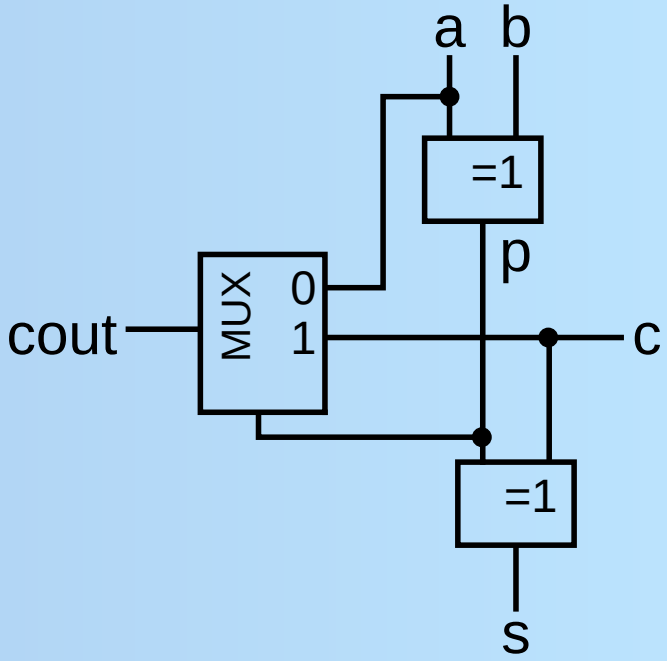
- Logisch binär: **and or xor nand nor xnor**
- Vergleich: = /= < <= > >=
- Verschieben / Rotieren: **sll srl sla sra rol ror**
- Additiv / Konkatenation: + – &
- Vorzeichen: + –
- Multiplikativ: * / **mod rem**
- Sonstige unär: **not abs** **

Binäre Operatoren sind bei gleicher Priorität linksassoziativ.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Signale

... repräsentieren getypte implementierungsspezifische Zwischenknoten:




```
architecture FA_2 of FA is
signal p : bit;  -- "Propagate"
begin
  p    <= a xor b;
  s    <= p xor c;
  cout <= c when p = '1' else a;  -- 2:1-MUX
end FA_2;
```

Nebenläufige Zuweisungen

. . . beschreiben kombinatorische Logik:

→ Sie sind vollkommen *parallel* zueinander.

→ Ihre Spezifikationsreihenfolge spielt *keine* Rolle:

```
p <= a xor b;        und        s <= p xor c;
s <= p xor c;                   p <= a xor b;
```

sind äquivalent!

Nebenläufige Zuweisungen: Arten

Unbedingte Zuweisung

```
s <= p xor c;
```

Bedingte Zuweisung

```
y <= '0' when clr = '1' else
     '1' when set = '1' else
      x;
```

- Priorisierte Auswahl des *ersten* Treffers.
- Abschließende Alternative für Kombinatorik notwendig.

Auswahlzuweisung

```
with s select
  y <= a when "00",
       b when "01" | "10",
       c when others;
```

- Balancierter Multiplexer.
- Erschöpfende Fallaufzählung ggf. mit **others** komplettieren.

... bilden komplexe nebenläufige Anweisungen.

- sind selbst sequentiell beschrieben:
  → Die Reihenfolge der Anweisungen ist von Bedeutung!
- besitzen einen Kontrollfluss, der gesteuert wird durch:

```
-- Verzweigung
if <cond1> then
  ...
elsif <cond2> then
  ...
end if;

-- Schleife
for i in 0 to N loop
  a(i) <= '0';
end loop;
```

```
-- Auswahl
case s is
  when "00" =>
    ...
  when "01" | "10" =>
    ...
  when others =>
    ...
end case;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Kombinatorische Prozesse

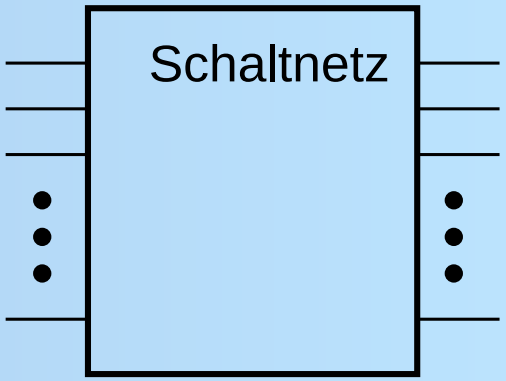
. . . beschreiben das Verhalten (komplexer) Schaltnetze

. . . vermeiden den strukturellen Feinentwurf

. . . überlassen die Optimierung auf Gatterebene dem Synthesetool

. . . sind gegenüber ausgearbeiteten Einzelgleichungen

- leichter adaptierbar
- verständlicher



Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Kombinatorische Prozesse: Beispiel

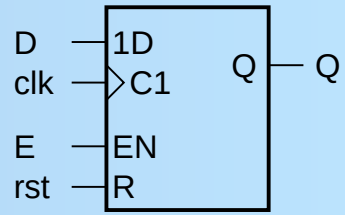
```
architecture FA_3 of FA is
begin  -- / "Sensitivitätsliste"
  process(a, b, c)     -- !!!: ALLE Eingänge hier aufzählen!
    variable p : bit; -- Lokale Zwischenknoten als Variablen
  begin
    p := a xor b;      -- !!!: Variablenzuweisung ANDERS
    s <= c;
    if p = '0' then    -- !!!: Fuer JEDEN Kontrollflusspfad
      cout <= a;       --      ALLE Ausgaenge berechnen
    else
      cout <= c;
      s    <= not c;  -- überschreibt ggf. vorherige Zuweisung
    end if;
  end process;
end FA_3;
```

Getaktete Prozesse

... beschreiben die Zustandsübergänge von Registern

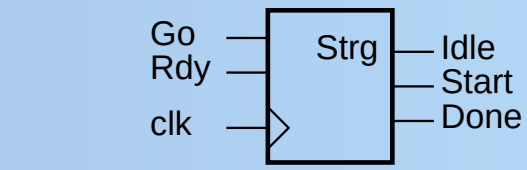
- Beispiel: D-FF mit Enable und asynchronem Rücksetzen

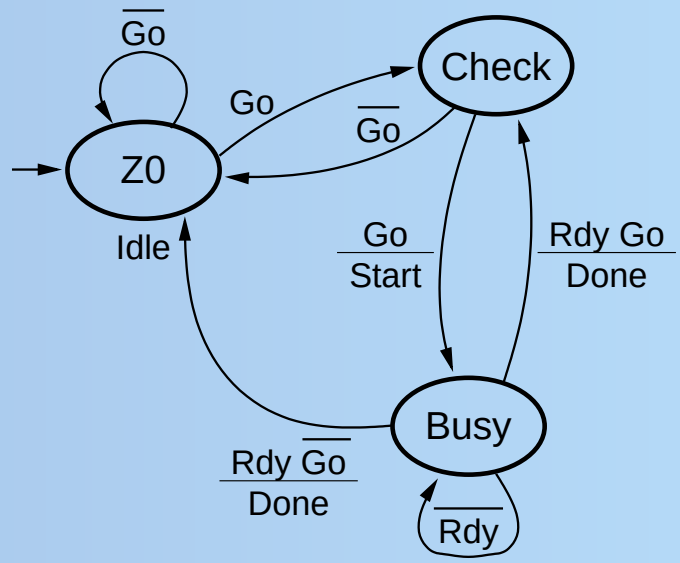
```
process(rst, clk) -- !!!: NUR die Eingänge,
begin                 --      die Zustandsübergang erwirken
  if rst = '1' then                  -- asynchrones Rücksetzen
    Q <= '0';
     -- / Taktflanke  / auf '1'
  elsif clk'event and clk = '1' then -- positive Taktflanke
    if E = '1' then                  -- Enable
      Q <= D;
    end if;
  end if;
end process;
```



Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

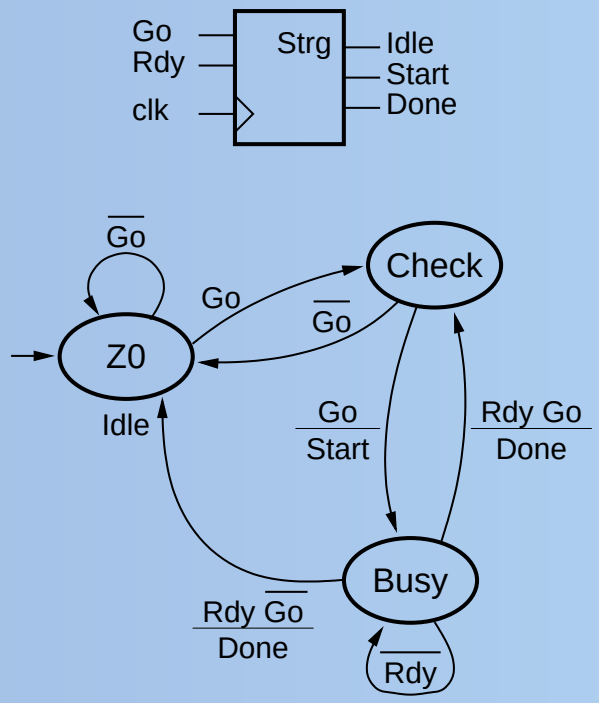
Beispiel: Implementierung eines Steuerautomaten





Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Schnittstellendefinition

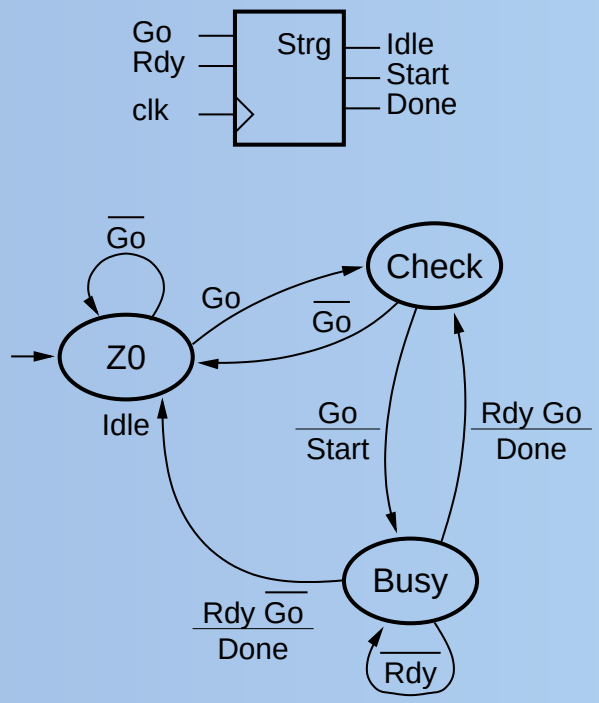


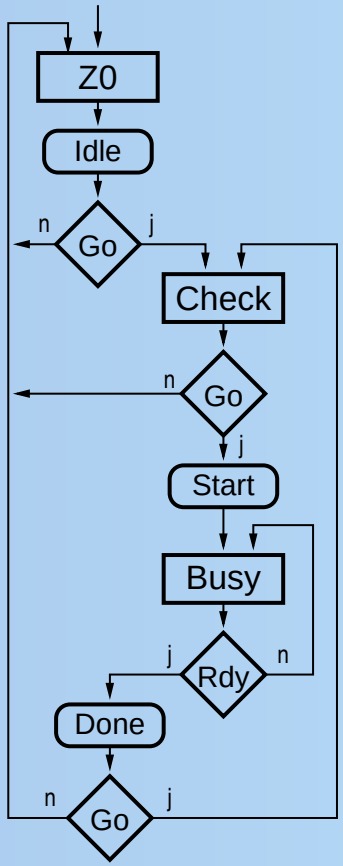

```
entity strg is
 port(
    rst, clk : in  bit; -- Reset, Takt
    Go       : in  bit; -- Starte Zyklus
    Rdy      : in  bit; -- Beende Zyklus

    Idle     : out bit; -- Leerlauf
    Start    : out bit; -- Zyklusbeginn
    Done     : out bit  -- Zyklusende
 );
end strg;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: SM-Chart





- äquivalent zum Zustandsdiagramm
- präzise Modellierung hierarchischer Entscheidungen
- leicht in HDL übertragbar

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: Typen und Signale

```
architecture rtl of strg is
  type tState is (Z0, Check, Busy);
  signal State     : tState := Z0; -- Zustandsregister
  signal NextState : tState;       -- kombinatorische Berechnung
begin                              --   des Folgezustands
  ...
end rtl;
```

Hartnäckiges Fettnäpfchen

Initialisierung von Registersignalen sollten mit der Reset-Zuweisung äquivalent sein! Sonst sind Inkonsistenzen zwischen den Systemzuständen nach der Initialisierung (z.B. FPGA-Programmierung) und einem Reset möglich.

Bei nicht vorgegebener Initialisierung versuchen Synthese-werkzeuge den gewünschten Initialisierungszustand aus der Resetbeschreibung zu inferieren – Simulatoren nicht!

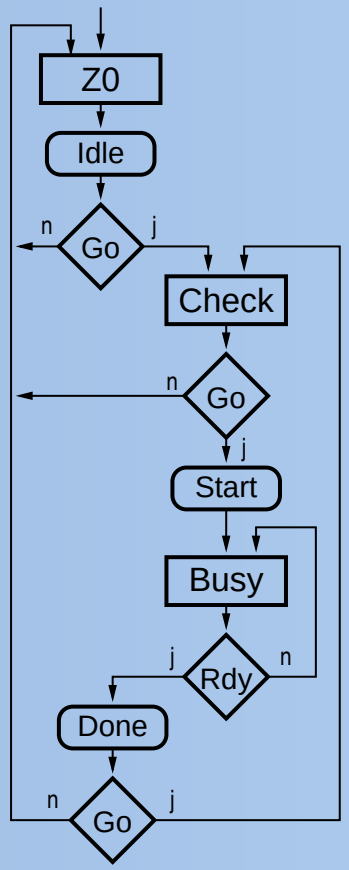
Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: Register

```
architecture rtl of strg is
  type tState is (Z0, Check, Busy);
  signal State     : tState := Z0; -- Zustandsregister
  signal NextState : tState;       -- kombinatorische Berechnung
begin                              --   des Folgezustands
  -- Getakteter Prozess
  process(clk)
  begin
    if clk'event and clk = '1' then -- steigende Taktflanke
      if rst = '1' then             -- synchrones Reset
        State <= Z0;
      else
        State <= NextState;
      end if;
    end if;
  end process;
  ...
end rtl;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: Kombinatorik




```
-- Kombinatorischer Prozess
process(State, Go, Rdy)
begin
  -- Default-Belegungen
  NextState <= State;
  Idle      <= '0';
  Start     <= '0';
  Done      <= '0';

  case State is
    when Z0 =>
      Idle <= '1';
      if Go = '1' then
        NextState <= Check;
      end if;

    when Check =>
      if Go = '0' then
        NextState <= Z0;
      else
        Start     <= '1';
        NextState <= Busy;
      end if;

    when Busy =>
      if Rdy = '1' then
        Done <= '1';
        if Go = '0' then
          NextState <= Z0;
        else
          NextState <= Check;
        end if;
      end if;

    -- hier ueberfluessiger
    --   Rueckfallzweig
    when others =>
      null; -- tut nichts
  end case;
end process;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Verzögerte Zuweisung

- Signaländerung erst nach angegebener Zeit:

```
a <= b and c after 20 ns;
```

- dient lediglich der funktionalen Modellierung.
- Einsatz in Testbenches zur Beschreibung von Stimuli:

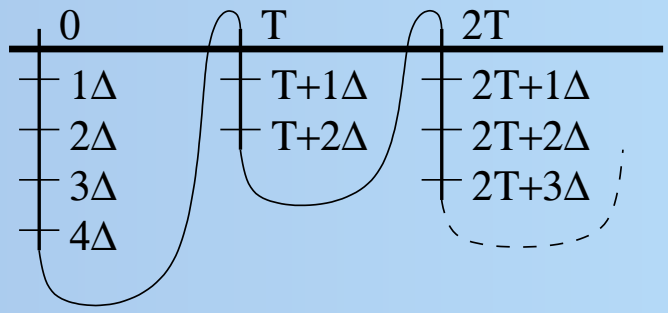
```
clk <= not clk after 50 ns; -- 10 MHz-Takt
```

- wird von Synthesewerkzeugen abgelehnt oder ignoriert.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Semantisches Modell: Simulationszeit

Simulation paralleler Hardware im sequentiellen Programm
→ zweidimensionale Simulationszeit mit Δ-Scheiben:



- Ereignisgetriebene Simulation auf Signalen.
- Jedem Ereignis ist eine Simulationszeit zugeordnet.
- Jedes Signal hat innerhalb einer Δ-Scheibe genau einen *festen* Wert.

Semantisches Modell: Ausführung

1. Initialisiere jedes Signal zu seinem Defaultwert.
2. Setze Simulationszeit $t = 0$.
3. Nimm jede nebenläufige Anweisung (einschließlich der Prozesse!) in die Menge der auszuführenden Anweisungen auf.
4. Führe jede Anweisung aus der Menge der auszuführenden Anweisungen aus und erzeuge für jeden berechneten Signalwert ein Ereignis:
   - für $t + \Delta$, falls keine explizite Zeitangabe vorhanden.
   - für $\lfloor t \rfloor + d$, für **after** d - Zuweisungen.
5. Leere die Menge der auszuführenden Anweisungen.
6. Schreibe die Simulationszeit bis zur Zeit des nächsten Ereignisses fort oder terminiere die Simulation, falls es kein weiteres Ereignis gibt.
7. Aktualisiere Signalwerte auf Basis der für die aktuelle Simulationszeit vorgesehenen Ereignisse. Bei Signaländerungen nimm jeden Prozess, der dieses Signal in der Sensitivitätsliste aufweist, und jede nebenläufige Anweisung mit diesem Signal auf der rechten Seite in die Menge der auszuführenden Anweisungen auf.
8. Gehe zu Schritt 4.

Semantisches Modell: Konsequenzen

- Fehlende Einträge in der Sensitivitätsliste kombinatorischer Prozesse:
  - verhindern die kombinatorische Neuauswertung bei der Änderung dieses Signals und
  - produzieren:
    - einen Zustand in Form von Latches, oder zumindest
    - Warnungen eines gutmeinenden Synthesewerkzeuges.
- Weitere in aller Regel unbeabsichtigte Quellen eines Zustandes durch Latches sind:
  - kombinatorische Signale ohne explizite Berechnung in jedem Kontrollflusspfad eines Prozesses und
  - bedingte nebenläufige Zuweisungen ohne abschließendes **else**.

9-wertige Logik (IEEE-1164)

’U’	Uninitialisiert
’X’	Starkes Unbekannt (Simulation: oft Fehler!)
’0’	Starke 0
’1’	Starke 1
’Z’	Hochohmig (Tristate-Buffer, Open-Collector/Open-Drain)
’W’	Schwaches Unbekannt
’L’	Schwache 0 (z.B. Pull-Down-Widerstand)
’H’	Schwache 1 (z.B. Pull-Up-Widerstand)
’–’	Don’t Care

- Erleichterte Fehlersuche in der Simulation.
- Beschreibung von Optimierungsmöglichkeiten: **else** ’–’**.**
- Beschreibung von Transmissiongates, z.B. in Bustreibern.
- Modellierung externer Pull-Up- oder Pull-Down-Widerstände.

Hinweise: Synthese

Die Synthese generiert digitale Logik:

- Interne Signale führen immer ’0’ oder ’1’.
- ’L’ und ’H’ werden auf ’0’ bzw. ’1’ abgebildet.
- ’-’ und ’X’ werden *nach Belieben* des Synthesewerkzeuges an ’0’ oder ’1’ gebunden.
- ’Z’ wird intern kombinatorisch eliminiert, in Ausgängen durch Tristate-Treiber implementiert.
- ’U’ und ’W’ erzeugen in expliziten Zuweisungen einen Fehler.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Hinweise: Simulation

In der Simulation von großer Bedeutung:

- ’U’ kennzeichnet noch nicht evaluierte Signale und deren abhängige.
- ’-’ beschreibt und *dokumentiert* Gleichgültigkeit.
  $\rightarrow$ Nutzen, wann immer möglich!
- ’X’ kennzeichnet:
  - die Evaluierung unbestimmter Werte (’X’, ’Z’, ’W’, ’-’), oder
  - das Treiben eines Buses auf verschiedene Werte $\rightarrow$ **Fehler!**

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: 7-Segment-Dekodierung

```
library IEEE;                          -- Deklariere IEEE-Bibliothek
use IEEE.std_logic_1164.all;           -- Erlaube Nutzung der 9-wertigen Logik

entity seg7dec is
  port(
    dig  : in  std_logic_vector(3 downto 0); -- dezimale BCD-Ziffer
    seg7 : out std_logic_vector(6 downto 0)  -- H-aktive 7-Segment-Ansteuerung
  );
end seg7dec;

architecture seg7dec_impl of seg7dec is
begin
  with dig select
    seg7 <= "1111110" when "0000",
            "0110000" when "0001",
            "1101101" when "0010",
            "1111001" when "0011",
            "0110011" when "0100",
            "1011011" when "0101",
            "1011111" when "0110",
            "11100-0" when "0111",
            "1111111" when "1000",
            "1111011" when "1001",
            "-------" when others;
end seg7dec_impl;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Beispiel: MUX durch Tristate-Buffer

```
entity mux4 is
  port(
    sel         : in  std_logic_vector(1 downto 0);
    a, b, c, d  : in  std_logic;
    y           : out std_logic;
  );
end mux4;

architecture mux4_impl of mux4 is
begin
  y <= a when sel = "00" else 'Z';
  y <= b when sel = "01" else 'Z';
  y <= c when sel = "10" else 'Z';
  y <= d when sel = "11" else 'Z';
end mux4_impl;
```

Auflösung mehrerer Bustreiber:

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Hardwarenahe arithmetische Datentypen

Zur Darstellung von vorzeichenlosen und vorzeichenbehafteten Zahlen im Zweierkomplement mit fester, aber *beliebiger* Bitlänge:

- Paket: **use** IEEE.numeric_std.**all**;
- Definition:
  vorzeichenlos: **type** unsigned **is array**(natural **range** <>) **of** std_logic;
  vorzeichenbehaftet: **type** signed **is array**(natural **range** <>) **of** std_logic;
- Unterscheiden sich lediglich in der Implementierung der arithmetischen Operationen wie $*$ und der Vergleichsoperationen wie < oder >=
- Addition mit integer ist erlaubt:

  Count <= Count + 1;
- Erlauben:
  - sichere Kontrolle über Datenbreiten und
  - einfache Beschreibung von Bitoperationen.

Typkonvertierungen

- dienen der sauberen Umsetzung des Typkonzepts,
- inferieren in der Regel *keine* Hardware,
- erfolgen in VHDL funktional:

```
function to_integer (arg : unsigned)       return natural;
function to_integer (arg : signed)         return integer;
function to_unsigned(arg, size : natural) return unsigned;
function to_signed  (arg  : integer;
                     size : natural) return signed;

-- Pseudofunktionen für "Related Types":
function signed  (arg : std_logic_vector) return signed;
function unsigned(arg : std_logic_vector) return unsigned;
function signed  (arg : unsigned)         return signed;
...
```

Blöcke

- strukturieren die Implementierung und
- erlauben lokale Typ-, Signal- und Funktionsdeklarationen:

```
<Marke>: block
   [lokale Deklarationen]
begin
   ...
end block [Marke];
```

Generics

- ermöglichen einen generischen, parametrierbaren Entwurf:

```
-- Bestimmt die Paritaet eines N-Bit-Arguments
entity par_n is
  generic(N : positive := 8); -- Generic mit Default
  port(
    arg : in  std_logic_vector(1 to N);
    par : out std_logic
  );
end par_n;

architecture rtl of par_n is
  signal  tmp : std_logic_vector(1 to N);
begin
  tmp(1) <= arg(1);
  xor_chain: for i in 2 to N generate
    tmp(i) <= tmp(i-1) xor arg(i);
  end generate;
  par <= tmp(N);
end rtl;
```

Das `generate`-Statement

- erlaubt die Beschreibung sich wiederholender oder bedingter Strukturelemente und
- ist besonders nützlich im generischen Entwurf.

```
<Marke>: if <Bedingung> generate
[ [lokale Deklarationen]
begin]
   ...                    -- else-Zweig ist (leider) nicht vorgesehen
end generate [Marke];

<Marke>: for <Bezeichner> in <Bereich> generate
[ [lokale Deklarationen]
begin]
   ...
end generate [Marke];
```

Feldattribute

- Erleichtern generische Beschreibungen von Feldzugriffen:
  Beispiel: **signal** arr : std_logic_vector(7 **downto** 0); −−*Feldtyp*

arr'length	Feldlänge	8
arr'**range**	Indexbereich	7 **downto** 0
arr'reverse_range	umgekehrter Indexbereich	0 **to** 7
arr'left	linke Grenze des Indexbereichs	7
arr'right	rechte Grenze des Indexbereichs	0
arr'high	größter Feldindex	7
arr'low	kleinster Feldindex	0

  - Statt Variablennamen können Typnamen verwendet werden.
  - Bei mehrdimensionalen Feldtypen ist betrachtete Dimension durch Argument auszuwählen.
    Im Beispiel sind arr'length(1) und arr'length äquivalent.

Komponenten

- realisieren den hierarchisch strukturierten Entwurf und
- ermöglichen die Wiederverwendung von fertigen Lösungen.

```
architecture rtl of xyz is
-- Komponentendeklaration (analog der Deklaration der entity)
component par_n is
  generic(N : positive := 8); -- Generic mit Default
  port(
    arg : in  std_logic_vector(1 to N);
    par : out std_logic
  );
end component;
-- Verbindungssignale
signal arg1, arg2 : std_logic_vector(5 downto 0);
signal par1, par2 : std_logic;
...
begin
  -- Instanziierung
  p1 : par_n     -- Benannte Zuordnung der Parameter / Signale
    generic map(N   => 6)
    port    map(arg => arg1, par => par1);
  p2 : par_n     -- Positionszuordnung der Parameter / Signale
    generic map(6)
    port    map(arg2, par2);
  ...
end rtl;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Komponenten II

- Offene Ausgänge werden mit dem Schlüsselwort **open** gebunden.
- Bei mehreren Architekturen zu einer Entity ist Auswahl zu treffen.

```
architecture rtl of xyz is
...
-- work ist implizite Bibliothek des aktuellen Projektes
for par1 : par_n use entity work.par_n(rtl_fast);
...
begin
  -- Instanziierung
  par1 : par_n  -- Benannte Zuordnung der Parameter / Signale
    generic map(N   => 6)
    port    map(arg => arg1, par => par1);
  ...
end rtl;
```

Pakete

- sinnvoll zur Bündelung von:
  - nicht lokalen Typen und Konstanten
  - Komponentendeklarationen
  - Funktionen
  - Prozeduren
- Teilung in Schnittstelle und Implementierung:

```
package Paket is
  subtype tALUOp is std_logic_vector(1 downto 0);
  constant ALU_OP_ADD : tALUOp := "00";
  ...

  function max(arg1 : integer; arg2 : integer) return integer;
end package Paket;

package body Paket is
  function max(arg1 : integer; arg2 : integer) return integer is
  begin
    if arg1>arg2 then return arg1; end if;
    return arg2;
  end;
end package body Paket;
```

Funktionen und Prozeduren

- sind ein mächtiges Hilfsmittel für Verhaltensmodelle, aber
- nur beschränkt von der Synthese unterstützt:

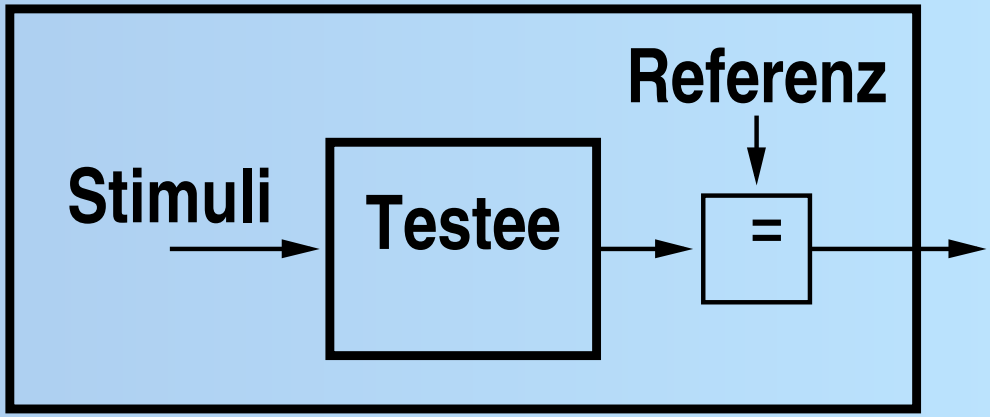
  → z.B. verbreitete Unterstützung für Berechnung konstanter oder kombinatorischer Ausdrücke.
- Parameter kommen in verschiedenen Ausprägungen:

		Aufrufparameter	
Modus	**Klasse**	**Prozedur**	**Funktion**
in[a]	**constant**[b]	Ausdruck	Ausdruck
	signal	Signal	Signal
	variable	Variable	⊘
out / inout	**signal**	Signal	⊘
	variable[b]	Variable	⊘

[a] Default
[b] Default für Modus

Testbenches

- dienen der Entwurfsvalidierung durch Simulation,
- besitzen Testobjekte als Komponente, und
- implementieren ein Verhaltensmodell zur:
  - Generierung der Stimuli (Eingangssignale) sowie
  - Überprüfung der Ausgangssignale (z.B. durch Vergleich).



Endlosprozesse

- sind ohne Sensitivitätsliste deklariert,
- werden in Endlosschleife ausgeführt und
- müssen durch **wait**-Statements unterbrochen werden:

```
wait on    <Sensitivitaetsliste>;
wait for   <Zeit>;
wait until <Bedingung>;
```

warten um Δ-Zyklus mit: **wait for** 0ns;

- nur wenige feste **wait**-Konstrukte sind synthetisierbar

  z.B.: **wait until** clk'event **and** clk = '1'; ––*nur einmal im Prozess*

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Assertions

- dienen der Überprüfung
  - statischer Annahmen z.B. über Generics, oder
  - dynamischer Annahmen während der Simulation.
- werden nicht zu Logik synthetisiert.

```
assert <Bedingung>                                  -- Behauptung
  [report   <String>]                               -- Meldung
  [severity (note|warning|error|failure)];          -- Schwere
```

- Bei verletzter Behauptung:
  - Ausgabe der Meldung, soweit angegeben.
  - Simulationsabbruch je nach Schwere und Simulatoreinstellung.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Assertions: Beispiel

```
architecture tb_impl of tb is
  signal arg : std_logic_vector(1 to 4);
  signal par : std_logic;
begin
  par : par_n
    generic map(N => 4)
    port    map(arg, par);

  process
  begin
    for i in 0 to 15 loop
      arg <= to_stdlogicvector(i, 4);
      wait for 10ns;
      assert par = (arg(1) xor arg(2) xor arg(3) xor arg(4))
        report "Parity Mismatch"
        severity error;
    end loop;
    report "Test complete";
    wait; -- forever
  end process;
end tb_impl;
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Hinweise zur Synthese

- Gute Lösungen verlangen Überblick!
  Struktur → Blockdiagramm
  Verhalten → SM-Chart
- Entwürfe auf möglichst hohem Beschreibungsniveau.
  Das Technologiemapping kann Synthesewerkzeug in aller Regel besser.
- Asynchrone Eingänge (extern, andere Taktdomaine) mit FFs puffern!
  Es sei denn, es gibt *garantiert* nur *einen* I/O-Register-Pfad.
  Es drohen Inkonsistenzen aufgrund von Laufzeitunterschieden.
- FPGA: Lieber mehr FFs als kompliziertere Logik.
  Jede Logikzelle besitzt ein FF.
  One-Hot-Zustandskodierung ist für größere Automaten oft die günstigste.

Meta-Werte

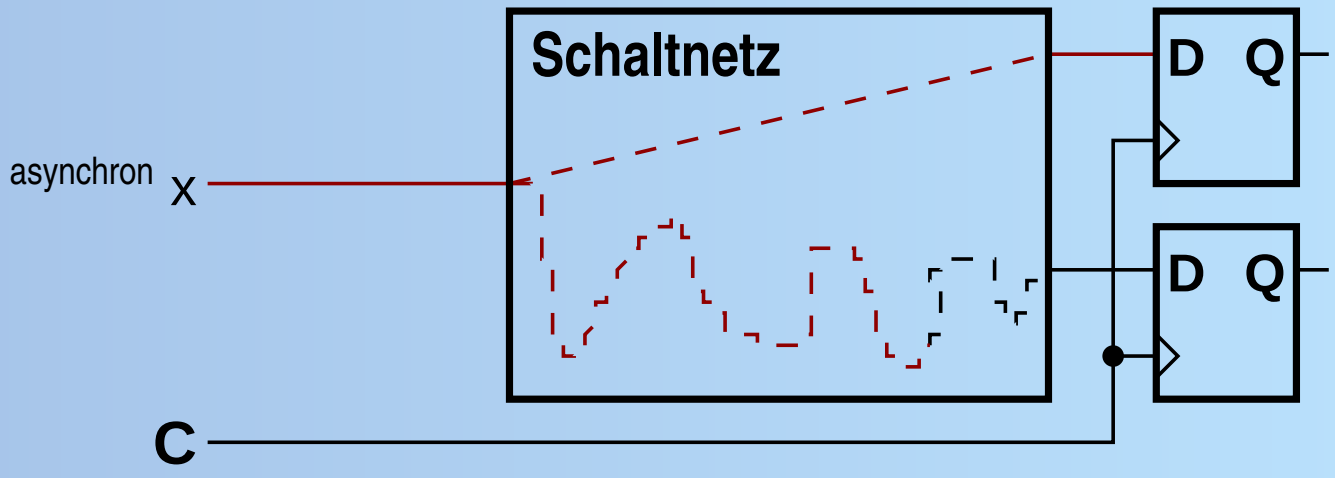
Nützliche Funktionen:

Funktion	Simulation	Synthese
Is_X(s)	s nicht klar auf ’0’ oder ’1’ abbildbar	false
Is_X(’X’)	true	false
to_X01	Abbildung auf ’0’, ’1’ oder ’X’ (’U’, ’X’, ’W’, ’-’, ’Z’)	Identität
rising_edge(clk)	clk’event **and** to_X01(clk) = ’1’ **and** to_X01(clk’last_value) = ’0’	

Don’t Cares intensiv nutzen!

- offenbart in der Simulation die Verwendung vermeintlich irrelevanter Signalwerte durch ’X’ anstatt nur falschem Ergebnis,
- dokumentiert irrelevante Signalzustände und
- eröffnet der Synthese Optimierungsmöglichkeiten.

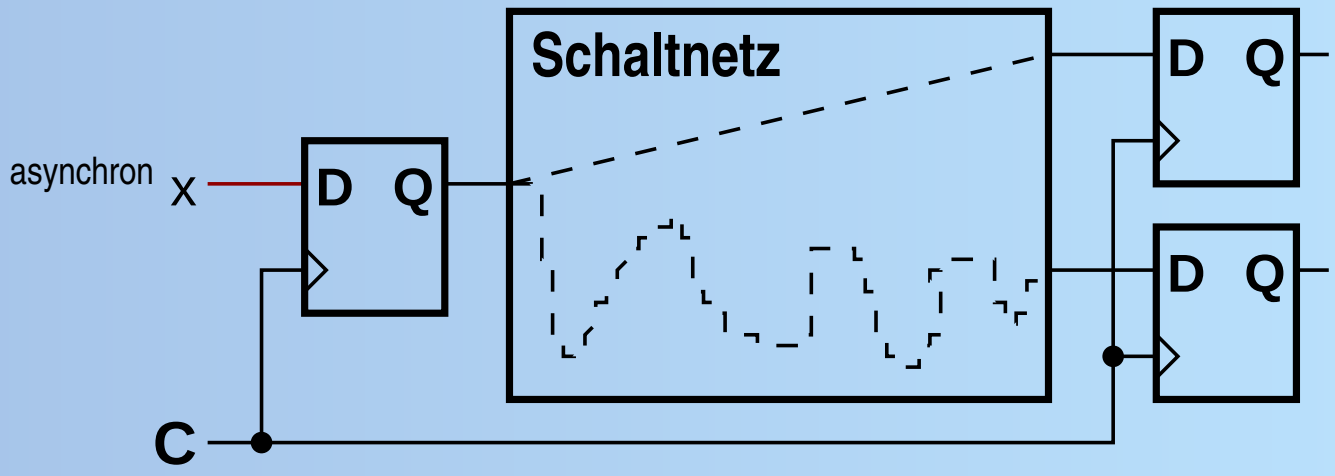
Asynchrone Eingänge



- Eingang schaltet zu *beliebigem* Zeitpunkt (relativ zum Takt).
- Propagierung durch Kombinatorik bis zur nächsten aktiven Flanke ist *nicht* garantiert.
- Inkonsistenzen im Systemzustand bei unterschiedlichen Pfadlängen.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Asynchrone Eingänge

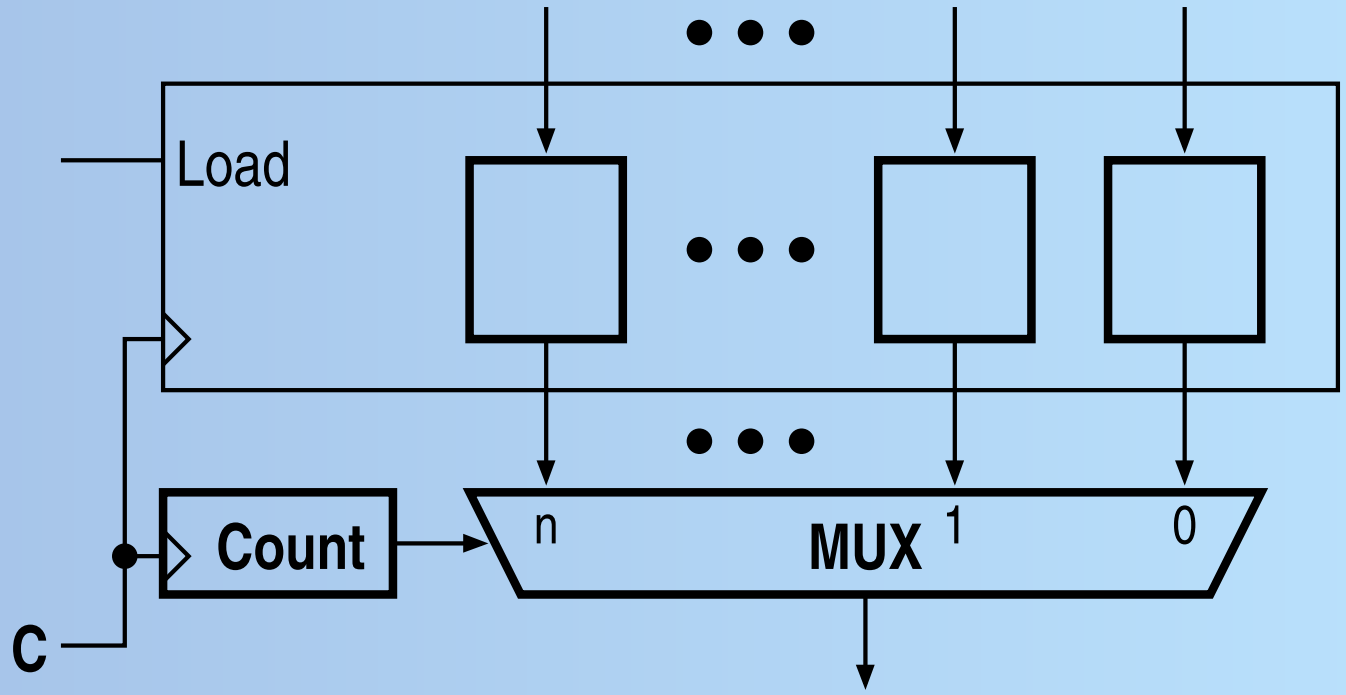


- Register-Register-Laufzeiten werden durch Synthesetool bestimmt.
  → Warnung, falls Taktvorgaben nicht erfüllbar.
- Oft auch zweite FF-Stufe zur Maskierung von Meta-Stabilitäten.
- Das Reset-Signal ist ein asynchroner Eingang.

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Iterationen über Felder

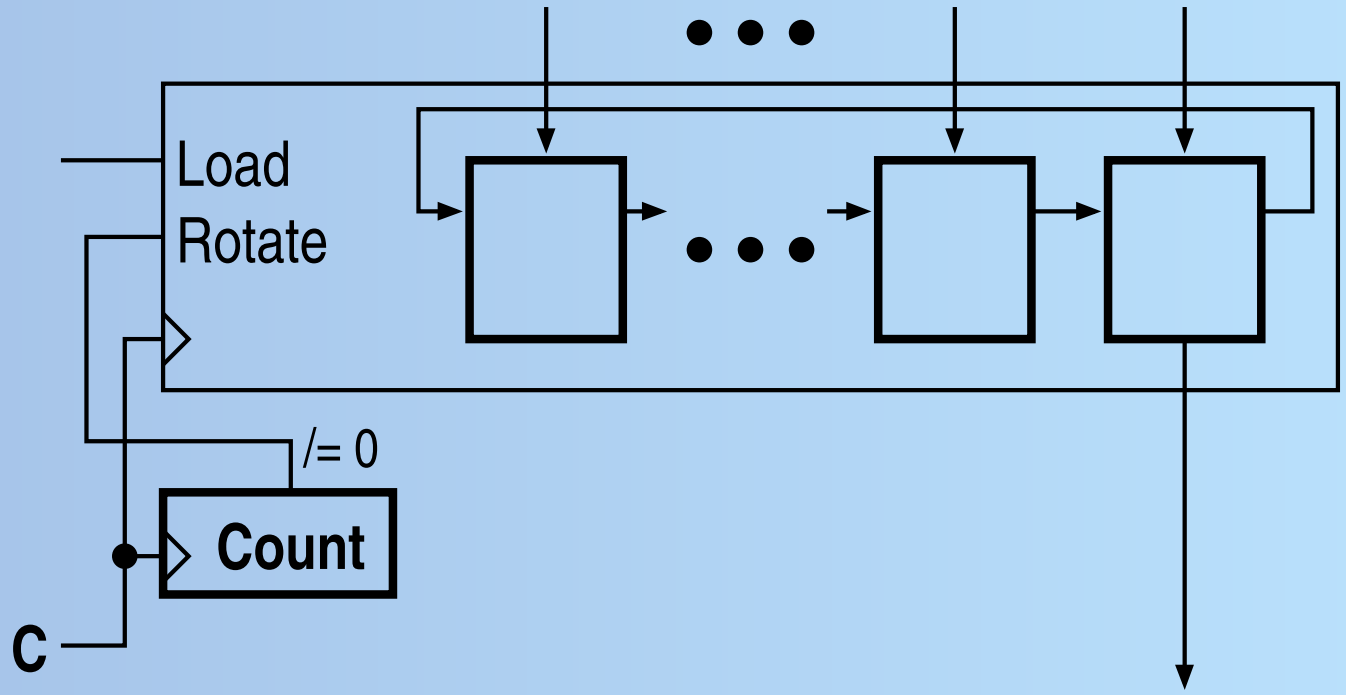
- Feldzugriffe durch Indexinkrement impliziert aufwendige Multiplexnetzwerke oder Enable-Logik.



Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Iterationen über Felder

- Oft besser:
  Einfache reguläre Strukturen durch Schieberegister.



Anwendungen:
Parallelisierung, Serialisierung, Iteration über Ziffern (Arithmetik), . . .

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Breite Gatter und Vergleicher

Durch die Verwendung von Feldattributen wartbar halten:

```
-- Breites ODER
y <= '1' when x /= (x'range => '0') else '0';

-- Breites UND
y <= '1' when x  = (x'range => '1') else '0';

-- Breites XOR
process(x)
  variable t : std_logic;
begin
  t := '0';  -- Neutrale Initialisierung
  for i in x'range loop
    t := t xor x(i);
  end loop;
  y <= t;    -- Abschließende Ergebniszuweisung (Signal)
end process;

-- Vergleicher
y <= '1' when x = to_unsigned(42, x'length) else '0';
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Klassiche Zählstandserkennung

```
-- Zählt: 0 .. CNT_CNT-1
process(clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      Cnt <= (others => '-');
    else
      if Init = '1' then
        Cnt <= (others => '0');
      elsif Step = '1' then
        Cnt <= Cnt + 1;
      end if;
    end if;
  end if;
end process;

-- AND über alle (ggf. negierten) Stellen von Cnt
Done <= '1' when Cnt = CNT_CNT-1 else '0';
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Optimierte Zählstandserkennung

```
-- Zählt: 0 .. CNT_CNT-1
process(clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      Cnt <= (others => '-');
    else
      if Init = '1' then
        Cnt <= (others => '0');
      elsif Step = '1' then
        Cnt <= Cnt + 1;
      end if;
    end if;
  end if;
end process;

-- AND über alle erwarteten '1' von Cnt (nur bei Aufwärtszähler!)
Done <= '1' when (Cnt or not to_unsigned(CNT_CNT-1, Cnt'length))
                  = (Cnt'range => '1') else '0';
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Zählstandserkennung durch Vorzeichenwechsel

```
-- Zählt: CNT_CNT-2 .. -1
--  -> Cnt muss ein Bit für das Vorzeichen bereithalten!
process(clk)
begin
  if rising_edge(clk) then
    if rst = '1' then
      Cnt <= (others => '-');
    else
      if Init = '1' then
        Cnt <= to_signed(CNT_CNT-2, Cnt'length);
      elsif Step = '1' then
        Cnt <= Cnt - 1;
      end if;
    end if;
  end if;
end process;

-- Keine Kombinatorik!
Done <= Cnt(Cnt'left);
```

Thomas B. Preußer
Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/
preusser@ite.inf.tu-dresden.de

Literatur

- Roth, Charles H., Jr.: *Digital Systems Design Using VHDL*, PWS Publishing Company, 1998.
- Smith, Douglas J.: *HDL Chip Design: A Practical Guide for Designing, Synthesizing & Simulating ASICs & FPGAs Using VHDL or Verilog*, Doone Publishing, 1996.
- *Hamburger VHDL-Archiv*, `http://tech-www.informatik.uni-hamburg.de/vhdl/vhdl.html`, insbesonder Abschnitt *Documentation*.