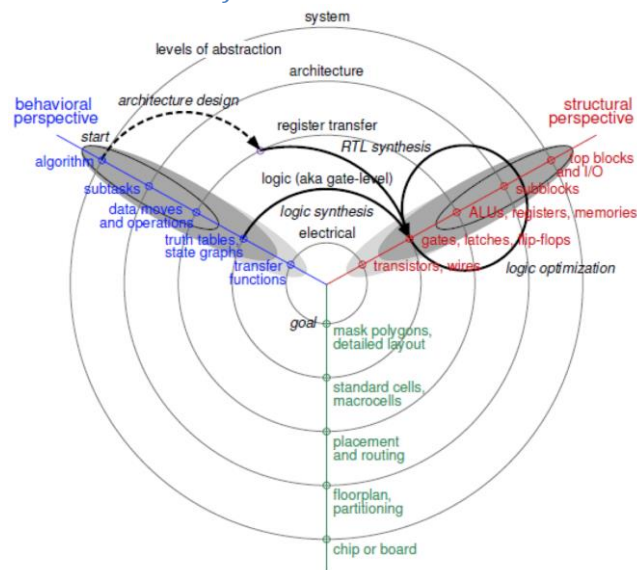


Entwurfsprozess

Y-Model von Gajski



Verhaltenssicht

Wie muss sich das System Verhalten?

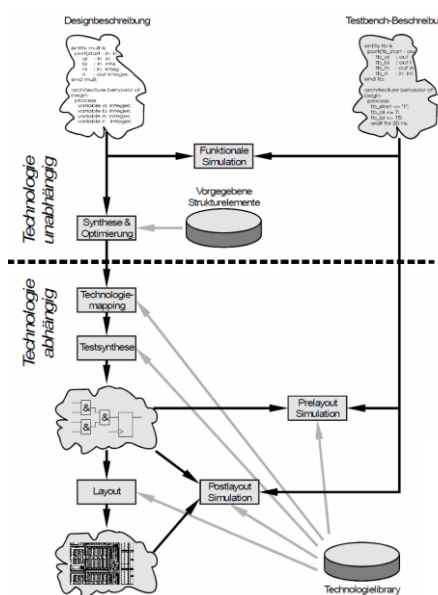
Strukturelle Sicht

Welche elektronische Schaltungen/Bausteine

Physikalische Sicht

Wo und wie platziere ich meine Blöcke?

2. Sobald der Hardwareblock bekannt ist kann man mit dem digitalen Entwurfsprozess starten
3. Verhalten werden Verfeinert, Algorithmen zerlegt und in Strukturelle Ebenen übersetzt



1. Designbeschreibung
2. Synthese und Optimierung
3. Technology Mapping
4. Place & Route
5. Simulation

→ in VHDL

→ Übersetzung in generische Netzliste (noch kein timing)

→ auf real existierende Bausteine anpassen

→ Hardware wird "programmiert" (real timing)

→ mit Hilfe einer Test-Bench (meistens mit einem Zeitdiagramm)

Eigenschaften VHDL

- Very Highspeed Description Language
- Nicht Case-Sensitiv
- Pinbelegungen sind nicht in vhdl -> ucf file ist dafür verantwortlich

UCF File

Signal einem Pin zuweisen

NET "Led(7)" LOC = "N13"; # DP

Bit 7 vom Signal Led wird auf Pin N13 gemapt.
ist ein Zeilenkommentar

Libraries

Std_logic- / ulogic- / -vector

Unsigned / signed

Nutzer sollte wissen was er tut:

Rechnen mit std_logic / std_ulogic

Rechnen mit std_logic / std_ulogic

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Mit denen kann gerechnet werden

```
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
```

Mit std_logic kann als signed gerechnet werden
Mit std_logic kann als unsigned gerechnet werden

VHDL

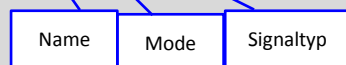
Aufbau Library

```
-- <- Kommentar
-- Bsp: And2
-----
-- Bekannt machen der Bibliothek (deklaration)
library ieee;
-- ganze Bibliothek nutzen
use ieee.all;
-- oder nur Teile aus der Bibliothek
use ieee.std_logic_1164.all; -- kann noch
--weiter verfeinert werden (hier nicht notwend.)
```

- bereits kompiliert
- kann Komponenten und/oder Packages enthalten
- work Arbeitsbibliothek immer vorhanden (wird automatisch generiert)
- std Standardfunktionen im IEEE 1076 spezifiziert (BIT, INTEGER, ...)
- ieee Praktische Packages z.B. std_logic
- PLD, ... Hersteller Lib. Beschreibung spezifischer Elemente

Entity

```
entity and2 is -- and2 ist der Name des "Bauteils"
  port( -- alle ein und Ausgänge des "Bauteils"
    a, b : in bit; -- Eingänge -> in
    y : out bit); -- Ausgänge -> out
end;
```



- Port alle digitale Signale, die von aussen sichtbar sind
- Mode in oder out - Ein- Ausgangssignale buffer - zurückführende Ausgangssignale inout - bidirektionale Signale z.B. Busse

Architecture

```
architecture behavioral of and2 is
-- Signaldeklarationen hier
begin
-- ab hier kommt die Logik
y <= a and b;
-- bis hier
end behavioral; -- name behavioral ist optional
```

- Typische architecture Namen:
 - Behavioral Verhaltensbeschreibung (ähnlich wie Prog.sprachen)
 - Structural Strukturbeschreibung (Schaltschema beschrieben -> nur Bauteile zusammensetzen)
 - TB Test-Bench

Es können mehrere Architectures in einem File vorkommen. Bsp. Eine für die (erste) Simulation, eine für die Hardware

Einbinden einer Komponente (=Bauteil)

Bauteil wird in der Architecture eingebunden

```
-- ...
architecture structural of half_adder is
-- Signale ... top_a, top_b, top_out
-- Komponenten deklarieren:
component and2
  port(a, b : in bit;
        y : out bit);
end component;
-- evtl. weitere Komponente
for all : and2 use entity work.and2 (behavioral);
begin
ic1 : and2
-- explizit:
  port map(
    a => top_a,
    b => top_b,
    y => top_out);
-- oder implizit: (Reihenfolge muss stimmen)
--   port map(top_a, top_b, top_out);
end structural;
```

Deklaration von and2
and2 ist der Componentenname

Benutze für alle and2 die Architektur behavioral, ohne das wird die Letzte verwendet

ic1 ist der Instanzname

a, b, y sind die Signal Namen des "Bauteils"
Top_a, top_b, top_out
Sind die Namen der angeschlossenen Leitungen/Signale (der TB oder der Toplevelfiles)

Signaldefinition

```
signal signal1, signal2 : bit_vector(7 downto 0); --8Bit Bus
```

↑ ↑ ↑ ↑
Signalname(n) Sig. Typ nichts oder 'downto' (oder 'to')

```
signal signal3 : signed(7 downto 0);
signal sig4 : integer range 0 to 99;
```

(integer mit welchem bis 99 gezählt werden kann)

Einzelnes Bit aus Vektor
Teile aus Vektor
Vektor zusammensetzen

```
signal1(0)
signal2(3 downto 0)
signal1(3 downto 0) & signal2 (3 downto 0)
```

Bit 0 von signal1
Bits 3 bis 0 werden genommen (=4Bit Vektor)
Mit & können Vektoren aneinander gereiht werden (auch einzelne Bits oder fixe Werte)
Passender (Init)Vektor mit nullen
Passender (Init)Vektor mit ,H'

Passende Initialisierung

```
(others => 0);
(others => 'H');
```

Zuweisungen ausserhalb eines Prozesses – Nebenläufige Zuweisungen

Normale Signalzuweisung

Selektive Signalzuweisung

```
y <= a and b;
with sel select oup <=
    inp0 when "00",
    inp1 when "01",
    inp2 when "10",
    "00" when others;
```

Geht immer

Nur ausserhalb Prozess

Bedingte Signalzuweisung

```
oup <= inp0 when sel = "00" else
    inp1 when sel = "01" else
    inp2 when sel = "10" else
    inp0;
```

Nur ausserhalb Prozess

Zuweisungen innerhalb eines Prozesses – Nebenläufige Prozesse

Normale Signalzuweisung

```
y <= a and b;
```

Geht immer

Switch Case

```
case sel is
    when "00" => oup <= inp0;
-- Weitere Anweisungen in diesem Case
    when "01" => oup <= inp1;
    when "10" => oup <= inp2;
    when others => oup <= inp0;
end case;
```

Nur im Prozess

If - then - else

```
if sel = "00" then
    oup <= inp0;
elsif sel = "01" then
    oup <= inp1;
elsif sel = "10" then
    oup <= inp2;
else
    oup <= inp0;
end if;
```

Nur im Prozess

Prozess

Gerüst

```
Prozessname : process(sel, inp0, inp1, inp2)
-- Prozessvariablen, Signale
Begin
-- Zuweisungen
end process;
```

Name des Prozesses ist optional

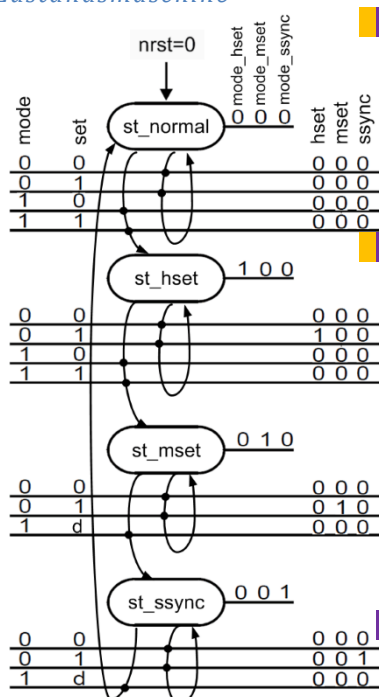
In der Klammer ist die Sensitivitätsliste. Es gehören alle Signale die rechts von einer Zuweisung stehen hinein.

Variablen

```
-- vor „Begin“
variable VarName : std_logic := 0;
```

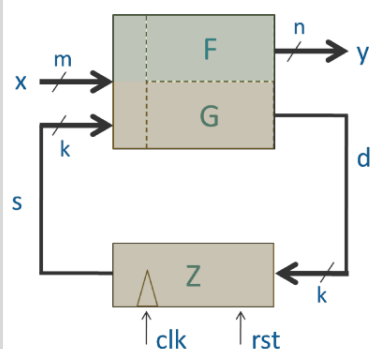
Normale Signale werden erst am Ende eines Durchlaufs durch den Prozess aktualisiert. Variablen sofort.

Zustandsmaschine



```
Registers_Z : process(rst, clk)
begin
    if (rst = '1') then
        present_state_s <= start_state;
    elsif (clk = '1') and clk'event then
        present_state_s <= next_state_d;
    end if;
end process;
Next_st_G: process(present_state_s, inp_x)
begin
    -- Default Werte
    next_state_d <= present_state_s;
    case present_state_s is
        when state_foo =>
            next_state_d <= state_blah;
            -- Do something / if something ...
        when state_blah =>
            -- Do something / if something ...
            next_state_d <= state_foo;
        when others => null; -- or default
    end case;
end process;
Oup_st_F: process(present_state_s, inp_x)
begin
    case present_state_s is
        when state_foo =>
            -- Do something / if something ...
        when others => null; -- or default
    end case;
end process;
```

Für std_logic

→ **rising_edge**(clk)Oder **falling_edge**(clk)

inp_x nur bei Mealy

Mealy: F, G, Z

Moore: G, Z (y nur abhängig von s)

Medwedjew: G, Z, y <= s;

Datentypen

std_logic

- solved logic
- erlaubt mehrere Treiber an einem Signal. Siehe Tabelle
→ Kurzschluss Gefahr!
- Subklasse von std_ulogic
- erlaubt Modellierung von Schwachen Signalen (L und H)
- erlaubt DONT'CARES (Signalwert '-')
- erlaubt bidirektionale Busse
- grosser Simulationsaufwand
- Vektorform: std_logic_vector

Treiber	'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'

std_ulogic

- unsolved logic
- erlaubt nur einen Treiber
→ Kein Kurzschluss möglich
- erkennt in der Sim. nicht initialisierte Signale bezeichnet sie als „U“
- erlaubt Modellierung von Schwachen Signalen (L und H)
- erlaubt DONT'CARES (Signalwert '-')
- nur einfache Busse erlaubt
- kleinerer Simulationsaufwand als std_logic
- Vektorform: std_ulogic_vector

Falls möglich immer (in den Übungen) std_ulogic verwenden, obwohl in der Praxis std_logic verbreitet ist. std_logic ist eine Unterklasse von std_ulogic und damit kompatibel, was aber nicht andersrum gilt! std_logic_vector ist trotz Unterklasse erst seit 2008 mit std_ulogic_vector kompatibel (dieser neuer STD ist noch nicht weit verbreitet!).

Eigene Datentypen

Enum

```
-- Enumdeklaration1 -----
type state_type is
    (st_normal, st_hset, st_mset, st_ssync);
signal present_state, next_state : state_type;
-- Enumdeklaration1 -----
type state_type2 is
    (st_normal, st_hset, st_mset, st_ssync);
-- explizite Codierung mit enum_encoding
attribute enum_encoding : string;
attribute enum_encoding of state_type2 :
type is "0001 0010 0100 1000"; -- one hot
codierung
signal present_state2, next_state2 :
state_type2;
-- Subtyp mit Konstanten -----
subtype MeinSubTyp is bit_vector(2 downto 0);
constant Constant1 : MeinSubTyp := "111";
constant Constant2 : MeinSubTyp := "110";
constant Constant3 : MeinSubTyp := "101";
signal present_state : MeinSubTyp;
signal next_state : MeinSubTyp;
```

Wird von Synthese „effizient“ oder nach Voreinstellungen synthetisiert

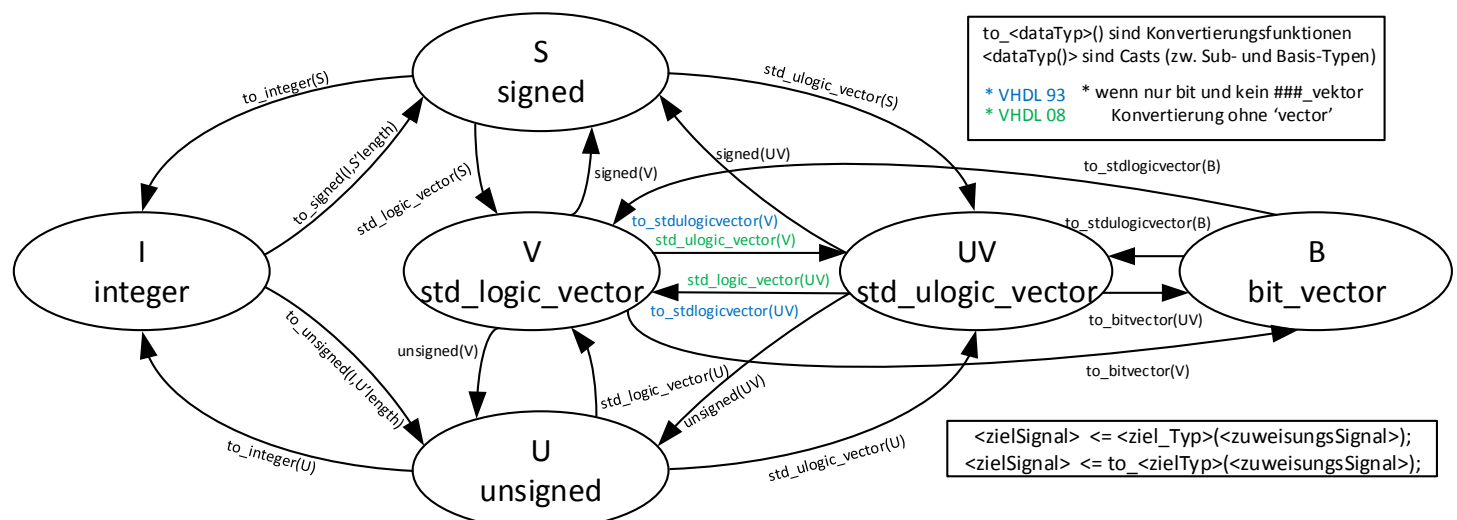
Enum mit Wertzuweisung

(Geeignet für Medwedjew:)
Falls der Typ als Bit benötigt wird, muss er noch konvertiert werden → und dafür ist ein hässlicher hack notwendig. (macht man nicht)
Man muss in der Nextstatelogic also zusätzlich auf Ausgang schreiben (vgl. Moore)

Für Medwedjew

Da ein String nur als hack in einen vernünftigen Datentyp gewandelt werden kann → Subtyp konfigurieren und Konstanten verwenden

Typen Konvertierungen und Typen casts



Arithmetik

*bit / bit_vectoren, std_logic /
std_logic_vector*

signed / unsigned

Addition
Subtraktion
Absolutwertbildung
Multiplikation
Division
Potenzbildung
Rest der Division A/B, Vorzeichen des
Ergebnisses gleich dem von B
Rest der Division A/B, Vorzeichen des
Ergebnisses gleich dem von A

and, nand, or, nor, xor, xnor, not
=, /=, <, >, >=, <=

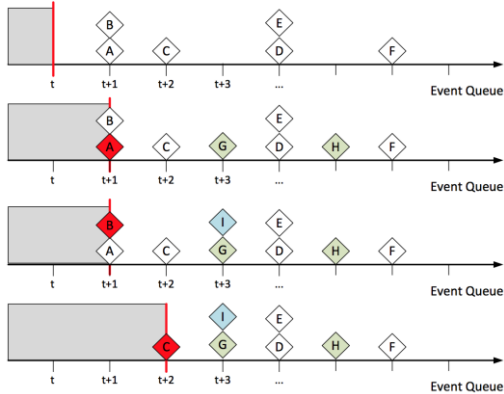
+	Y <= A + B
-	Y <= A - B
abs	Y <= abs(A)
*	Y <= A * B
/	Y <= A / B
**	Y <= 2 ** B
mod	Y <= A mod B
rem	Y <= A rem B

use ieee.std_logic_1164.all
für std_logic / std_logic_vector

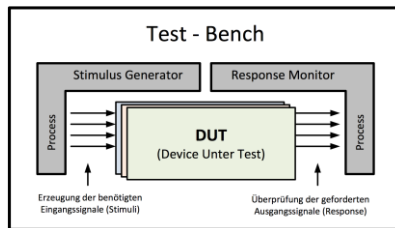
Normale Gatterlogik
Vergleichsoperatoren (/= entspricht ungleich)
use IEEE.numeric_std.all

synthesefähig
synthesefähig
synthesefähig
synthesefähig
Meist nicht synthesefähig
Nur Potenzen von 2 erlaubt (= Shift Left)
Synthesefähig falls B Zweierpotenz
Synthesefähig falls B Zweierpotenz

Testbench



- Die Simulation / Funktionstest wird mit Hilfe einer Test-Bench gemacht
- Die Test-Bench sollte Wiederverwendbar sein
- Test-Bench ist unabhängig davon wie das DUT implementiert wurde
- Je früher ein Fehler bemerkt wird desto „billiger“ ist er
- VHDL-Simulatoren arbeiten mit einer Event-Queue (siehe Bild)
- (SPICE-Simulatoren arbeiten mit DGL → brauchen viel mehr Rechenleistung)
- Delta-Zyklus: (auf Bild t's)
- 1. Update Request: wartet auf Trigger
- 2. Prozessausführung: führt alle Prozesse aus bis zum nächsten „wait“
- 3. Signalzuweisung: neue Signale werden zugewiesen
- Transport-Delay: rechnet die Delays der Logikgatter ein
- Internal-Delay: simuliert die Trägheit der Elektronik



- In Test-Benchs werden oft Schleifen eingesetzt
- Sie muss nicht Synthesefähig sein
- Automatische Überprüfung mit ASSERTs
- Eine Test-Bench sollte eine leere entity haben → keine Schnittstelle nach aussen
- „Stimuli“ Eingang des DUT's, „Response“ erwartete Antwort
- DUT „1 zu 1“ aus work - library

For loop

```
process (A)
begin
  Z <= "0000";
  for I in 0 to 3 loop
    if (A(I) = 'H') then
      Z(I) <= '1';
    end if;
  end loop;
end process;
```

For loop der im 4 bit std_logic_array, falls A = H ist, in Z '1' an selber Stelle schreibt

Clock-Prozess

```
-- Bsp fuer RST und CLK
rst <= '1', '0' after
150ms;
CLOCK : process
begin
  clk <= '0';
  wait for (PERIOD / 2);
  clk <= '1';
  wait for (PERIOD / 2);
end process CLOCK;
```

- **wait for** nur in TB verwenden → ist nicht synthetisierbar
- bei **wait** wird der Prozess sofort ausgewertet

Assert

```
assert (dut_signal = signal_expected) report "Error occured" severity error;
```

Falls das erwartete Signal nicht auftritt wird „Error occured“ ausgegeben. Die Standard-Priorität ist Error (NOTE, WARNING, ERROR, FAILURE)

Note → Meldung erscheint
Warning → Meldung + Warnungscouter
Error → Simulation bricht ab
Failure → Simulationsprogramm wird beendet
Alle Signale werden 10ns verspätet Y zugewiesen
Wie transport, Signal muss jedoch min 10ns dauern
Inertial ist optional (engl. = träge, macht also das selbe)

transport after

```
Y <= A transport 10ns;
Y <= A after 10ns;
Y <= inertial A after 10ns;
```

Generic Entity

```
entity counter_generic is
  generic(
    maxcnt : integer := 100;
    cnt : integer := 4
  );
  port(
    clk      : in  std_ulogic;
    nrst     : in  std_ulogic;
    count_out : out integer range 0 to maxcnt - 1;
    oup2     : out std_logic_vector(cnt-1 downto 0)
  );
end counter_generic;
```

Zusätzlich generic vorhanden
Grösse bei synthetisierung festgelegt
:= 100 → default Wert

Generic-Werte dürfen bereits verwendet werden

Architecture

```
architecture behavioral of counter_generic is
  signal cnt : integer range 0 to maxcnt - 1;
  signal next_cnt : integer range 0 to maxcnt - 1;
begin
  -- do something
end behavioral;
```

Generic-Werte dürfen auch hier verwendet werden

Bauteil einbinden

```
-- TopLevelFile or TB or ...-----
architecture structural of counter is
-- Signals ...
  component counter_generic
    generic(
      maxcnt : integer := 100;
      cnt : integer := 4
    );
    port(clk, nrst : in  std_ulogic;
          count_out : out integer range 0 to maxcnt - 1;
          oup2 : out std_logic_vector(cnt-1 downto 0)
    );
  end component;
-- evtl. weitere Komponenten
  for all : and2 use entity work.and2(behavioral);
begin
  ic2 : counter
  -- explizit:
  generic map(
    maxcnt => 50,
    cnt => 8
  )
  port map(
    -- normal, siehe Seite 2 -
  );
  -- do something
end structural;
```

Wie normale Komponente einbinden,
zusätzlich einfach die generic map

Beispiele

∅ bedeutet nächste Zeile gehört auf diese Zeile

<pre>-- Edge detection in 3 Prozessen library ieee; use ieee.std_logic_1164.all; entity edgedetneg is port(clk : in std_ulogic; nrst : in std_ulogic; inp : in std_ulogic; oup : out std_ulogic); end; architecture behavioral of edgedetneg is type state_type is (resetstate, ∅ wait_for_0, pulse, wait_for_1); signal present_state : state_type; signal next_state : state_type; begin Output_logic : process(present_state) begin case present_state is when pulse => oup <= '1'; when others => oup <= '0'; end case; end process;</pre>	<pre>registers : ∅ process(nrst, clk) begin if (nrst = '0') then present_state ∅ <= resetstate; elsif rising_edge(clk) then present_state ∅ <= next_state; end if; end process; end behavioral;</pre>	<pre>next_state : process(inp, ∅ present_state) begin next_state <= wait_for_1; -- default value case present_state is when wait_for_0 => if (inp = '0') then next_state <= pulse; else next_state <= wait_for_0; end if; when pulse => if (inp = '1') then next_state <= wait_for_1; end if; when wait_for_1 => if (inp = '1') then next_state <= wait_for_0; end if; when others => if (inp = '1') then next_state <= wait_for_1; end if; end case; end process;</pre>
--	--	--

```

-- Edge detection Test Bench
library ieee;
use ieee.std_logic_1164.all;

entity edgedetneg_tb is
end edgedetneg_tb;

architecture tb of edgedetneg_tb is
    constant f : integer := 1000;
    -- Frequency f in HZ
    constant T : time := 1 sec / f; -- Clock
    Period T
    component edgedetneg
        port(
            clk : in std_ulogic;
            nrst : in std_ulogic;
            inp : in std_ulogic;
            oup : out std_ulogic
        );
    end component;
    signal clk_tb : std_ulogic;
    signal nrst_tb : std_ulogic;
    signal inp_tb : std_ulogic;
    signal oup_tb : std_ulogic;
    signal oup_exp_tb : std_ulogic;
begin
    dut : edgedetneg
        port map(
            clk => clk_tb,
            nrst => nrst_tb,
            inp => inp_tb,
            oup => oup_tb
        );
    stimuli_clk : process
    begin
        clk_tb <= '0';
        loop
            wait for T / 2;
            clk_tb <= not clk_tb;
        end loop;
        wait;
    end process;
    stimuli_nrst : nrst_tb <= '0', '1' after 3
ms, '0' after 14.2 ms, '1' after 14.4 ms;
    stimuli_inp : process
    begin
        inp_tb <= '1'; --Eingangspulse während
reset
        wait for T;
        inp_tb <= '0';
        wait for 0.5 * T; --kurzer Impuls während
reset
        inp_tb <= '1';
        wait for T;
        inp_tb <= '0'; --langer Impuls während
reset
        wait until rising_edge(nrst_tb);
        wait until rising_edge(clk_tb);
        wait for 0.1 * T; --ergibt zusammen mit
Statement oben einen aktiven Puls während
Wechsel von reset
        inp_tb <= '1';
        wait for 0.5 * T;
        inp_tb <= '0';
        wait for 1.5 * T;
        inp_tb <= '1';
        wait for 1.5 * T;
        inp_tb <= '0';
        wait for 0.5 * T;
    end process;
    stimuli_oup_exp : process
    begin
        loop
            oup_exp_tb <= '0';
            if (nrst_tb = '0') then -- Prüfen
bei reset

```

```

wait until rising_edge(nrst_tb);
wait until rising_edge(clk_tb);
        if inp_tb = '0' then
            wait until rising_edge(clk_tb);
        end if;
    end if;
        if (inp_tb = '1') then -- Prüfen bei
einem negativen Puls
            while (inp_tb = '1') and (nrst_tb =
'1') loop
-- Zusatzbedingung für den Fall, dass während
Wartezeit der asynchrone Reset ausgelöst wird
                wait until (rising_edge(clk_tb) or
falling_edge(nrst_tb));
            end loop;
            if nrst_tb = '1' then
                oup_exp_tb <= '1';
            end if;
            wait until rising_edge(clk_tb);
            oup_exp_tb <= '0';
            if inp_tb = '1' then
--Flankendetektor ignoriert Eingangspulse
                wait until rising_edge(clk_tb);
-- während der Ausgabe eines Ausgangspulses
            end if;
        else
            wait until rising_edge(clk_tb);
        end if;

    end loop;
    wait;
end process;

evaluation : process
begin
    wait for 0.9 * T; --Evaluation geschieht
kurz vor der nächsten Clockflanke, welche bei
absolut 1.5*T liegt.
    loop
        assert (oup_tb = oup_exp_tb) report
"Error" severity error;
        wait for T;
    end loop;
end process;
end tb;

```

```

-- Bidirektionaler Bus
library ieee;
use ieee.std_logic_1164.all;

entity bus_driver is
port(
    sel : in std_ulogic;
    data_out : in std_ulogic_vector(7 downto 0);
    my_bus : inout std_logic_vector(7 downto 0);
    data_in : out std_ulogic_vector(7 downto 0));
end bus_driver;

architecture behavioral of bus_driver is

begin

-- Bus_Komponente
-- (Label (read: , blah:) dürfen überall davor
-- gesetzt werden)
    read: data_in <= to_stdulogicvector(my_bus);
    write: process(sel, data_out)
    begin
        if sel = '1' then
            my_bus <= to_stdlogicvector(data_out);
        else
            my_bus <= (others => 'Z');
        end if;
    end process write;
end behavioral;

```

```

-- Motor project Top File
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_bit.all;
-- Hier sind die Ein- und Ausgänge des ◊
motorproject_top definiert:
entity motorproject_top is
  port(
    clk      : in  bit;
    reset    : in  bit;
    lb0      : in  bit;
    lb1      : in  bit;
    bt_select : in  bit;
    led_left  : out bit;
    led_right : out bit;
    led_pulse : out bit;
    led_time  : out bit;
    led_distance : out bit;
    led_velocity : out bit;
    sseg      : out bit_vector(7 downto 0);
    sseg_an   : out bit_vector(3 downto 0)
  );
end motorproject_top;

architecture structural of motorproject_top is

  -- Hier deklarieren Sie interne Signale:
  signal lb_int      : bit_vector(1 downto 0);
  signal op_time_int : bit_vector(7 downto 0);
  signal distance_int : bit_vector(7 downto 0);
  signal velocity_int : bit_vector(7 downto 0);
  signal binary_int  : bit_vector(7 downto 0);
  signal sel_int     : bit_vector(1 downto 0);
  signal nrst_int    : bit;
  signal clkslow_int : bit;
  signal bt_select_debounce_int : bit;

  -- Hier deklarieren Sie interne Komponenten:
  component analysis
    port(clk      : in  bit;
         nrst     : in  bit;
         lb       : in  bit_vector(1 downto 0);
         dir_left : out bit;
         dir_right : out bit;
         pulse    : out bit;
         op_time  : out bit_vector(7 downto 0);
         distance : out bit_vector(7 downto 0);
         velocity : out bit_vector(7 downto 0));
  end component analysis;

  component clockdivider
    port(clk      : in  bit;
         nrst     : in  bit;
         clkslow  : out bit);
  end component clockdivider;

  component debounce
    port(clk      : in  bit;
         nrst     : in  bit;
         inp      : in  bit;
         oup      : out bit);
  end component debounce;

  component display
    port(clk      : in  bit;
         nrst     : in  bit;
         inp      : in  bit_vector(7 downto 0);
         sseg     : out bit_vector(7 downto 0);
         sseg_an  : out bit_vector(3 downto 0));
  end component display;

  component mux3x8
    port(sel      : in  bit_vector(1 downto 0);
         inp0     : in  bit_vector(7 downto 0);
         inp1     : in  bit_vector(7 downto 0);
         inp2     : in  bit_vector(7 downto 0);
         oup      : out bit_vector(7 downto 0));
  end component mux3x8;

  component switchdisplay
    port(clk      : in  bit;
         nrst     : in  bit;
         button    : in  bit;
         sel      : out bit_vector(1 downto 0));
  end component switchdisplay;

begin
  -- Hier instanziiieren und verbinden Sie ihre◊
  Instanzen:
  analysis_1 : component analysis
    port map(clk      => clkslow_int,
             nrst     => nrst_int,
             lb       => lb_int,
             dir_left => led_left,
             dir_right => led_right,
             pulse    => led_pulse,
             op_time  => op_time_int,
             distance => distance_int,
             velocity => velocity_int);

  mux3x8_1 : component mux3x8
    port map(sel      => sel_int,
             inp0     => op_time_int,
             inp1     => distance_int,
             inp2     => velocity_int,
             oup      => binary_int);

  display_1 : component display
    port map(clk      => clkslow_int,
             nrst     => nrst_int,
             inp      => binary_int,
             sseg     => sseg,
             sseg_an  => sseg_an);

  debounce_1 : component debounce
    port map(clk      => clkslow_int,
             nrst     => nrst_int,
             inp      => bt_select,
             oup      => bt_select_debounce_int);

  switchdisplay_1 : component switchdisplay
    port map(clk      => clkslow_int,
             nrst     => nrst_int,
             button    => bt_select_debounce_int,
             sel      => sel_int);

  clkdivider_1 : entity work.clockdivider
    port map(clk      => clk,
             nrst     => nrst_int,
             clkslow  => clkslow_int);

  -- Hier noch die normalen Zuweisungen:
  led_time      <= sel_int(0) nor sel_int(1);
  led_distance  <= sel_int(0) and not sel_int(1);
  led_velocity  <= not sel_int(0) and sel_int(1);
  nrst_int      <= not reset;

end architecture structural;

```