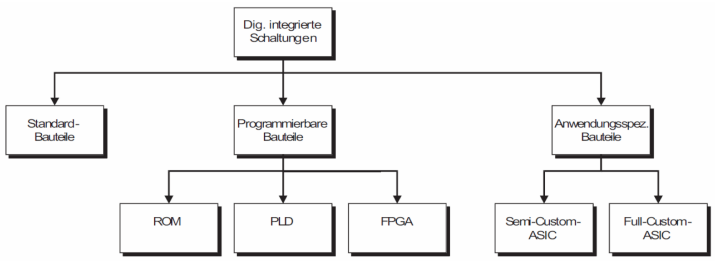
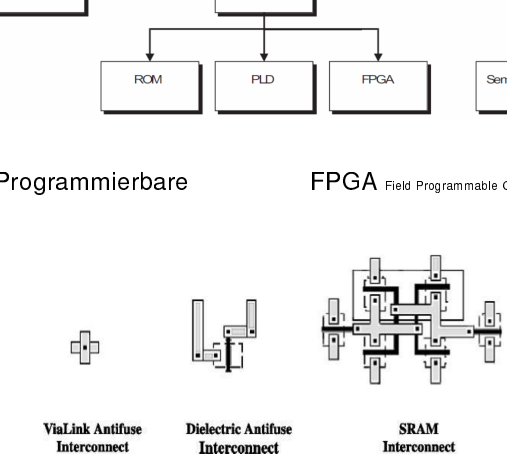
1 Grundlagen Digitaltechnik II

1975: "The Moore's Low"(Gorden E. Moore, Inter Co-Founder): Anzahl Transistoren auf 1 Chip verdoppelt sich alle 2 Jahre

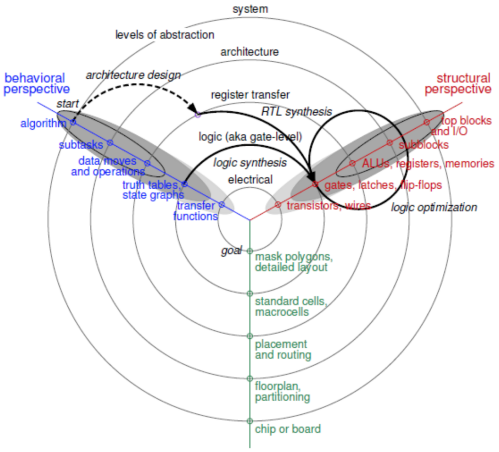


SDT-Bauteile	Digitale Gates Flipflops	AND, OR, ...
Programmier-bare	ROM Read Only Memory	PROM (one time programmable) EPROM (eresable with UV) EEPROM (eresabel with Voltage) FLASCH (Spezielle EEPROMs)
	PDL Programable Logic Device	PAL Programmable Array Logic GAL Generic Array Logic CPLD Complex CLD

Programmierbare	FPGA Field Programmable Gate Array	Im Gegensatz zu PLDs hat es eine regelmässige Strucktur, die in Logik- (bei Xilinx CBL) und I/O Blöcke (bei Xilinx IOB) Man kann Pullup- und Pulldouwnwiderstände hinzuschalten Schaltpegel sind TTL und CMOS kompatibel	
		SRAM - Zelle	Antifuse - Technologie
		Mehrfach Programmierbar Löscht wenn Speisung unterbrochen hohe Kapazität unprogrammiert geringer Leitwert programmiert Stärker verbreitet kann kleine Daten Speichern Startprogramm(e) oft auf PROMS	Verbindung durch Durch-schweissen der Isolation viel kleiner als SRAM nicht wiederverwendbar sicherer als SRAM → Einsatz im Weltraum
Anwendungsspeziefische	Semi-Costom	Vorgegebenes Grunddesign→Nutzer macht nur noch Netzmaske billiger da in Grossen Mengen produziert kann selten optimal ausgenutzt werden	
	Full-Custom	Spezifische Schaltungen → Spezialanfertigung → hohe Kosten Nur in Grossen Mengen rentabel kann selten optimal ausgenutzt werden	



2 Digitaler Designe-Flow

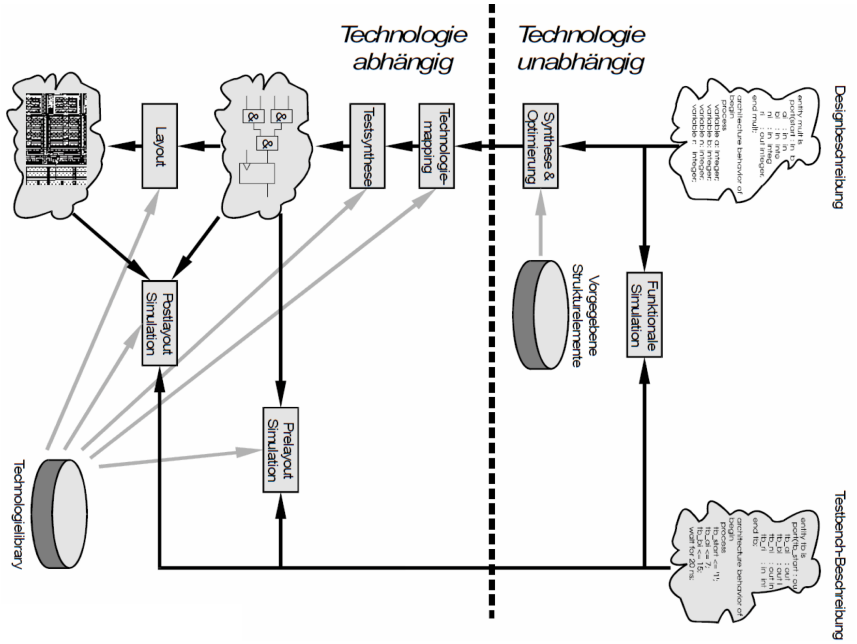


Y-Model von Gajski

1. Verhaltenssicht (blau) → Wie muss sich das System Verhalten?
2. Struckturelle Sicht (rot) → Welche elektronische Schaltungen/Baublöcke
3. Physikalische Sicht (grün) → Wo und wie platziere ich meine Blöcke?

2.1 Desigenprozess

1. Wichtigste Fragen:
   - Welches verhalten will der Kunde
   - Gewünschte Schnittstellen
   - Randbedingungen: Grösse, Kosten, . . .
   - Soft-, Hardware oder mixed Lösung gewünscht
   - Gibt es schon etwas ähnliches auf dem Markt
   - . . .
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
   → in VHDL
2. Synthese und Optimierung
   → Übersetzung in generische Netzliste (noch kein timing)
3. Tegnologie Mapping
   → auf real existierende Baublöcke anpassen
4. Place & Route
   → Hardware wird "Programmiert“ (real timing)
5. Simulation
   → mit Hilfe einer Test-Bench (meistens mit einem Zeitdiagramm)

3 VHDL

Überblick
Kreatives erkennen → Mensch
Fleissarbeit → CAD Systeme (Computer Aided Design)
"Verilog“ und “SystemC“ sind alternativen (weniger verbreitet)
Vorteile von VHDL:
· Fokus auf Beschreibung des Verhaltens
· geprägt von kreativen, erfahrungsbasierten Design
· Realisierung von wiederverwendbaren Blöcken

Geschichte
1980: DoD (US Department od Defence) beauftragt IBM, Intermetrics und TI eine gemeinsame Hardwaresprache zu definieren
Namen setzt sich von VHSIC (Very High Speed Integrated Circuits) des Dod und HDL (Hardware Description Languege) zusammen
1987: DoD verzichtet auf das Exklusivrecht
→ IEEE führt den STD weiter (neuster: IEEE 1076.1)

Charakteristische Elemente
· Hierarchie und Konnektivität
· Beschreibung und Interpretation **paralleler** Abläufe
· Beschreibung elektrischer Signale
· Beschreibung des Zeitlichen Verhaltens
· Parametisierbarkeit von Modellen

Eigenschaften
· nicht "case sensitiv" Gross- Kleinschreibung
· Identifier alphanumerisch
  · immer mit Buchstabe beginnen
  · keine "_“ am Ende oder doppelt
  · keine Schlüsselwörter verwenden

```
1 <>   -- identifier
2 []   -- Optional Element
3 {}   -- kann beliebig oft
          wiederholt werden
4
```

4 Key Concept I Hierarchie und konektivität

4.1 Library

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

```
1 -- deklaration der Bibliotek
2 library ieee;
3 -- Konstante aus Bibliothek lesen
4 <variable>:=<library_name>.<consr_name>;
5 -- Bibliothek sichtbar machen
6 use <library_name>.<package_name>.<element_name>; -- oder
7 use <library_name>.all;
8
```

4.2 Entity

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

```
-- Syntax
entity <ENTITY_NAME> is
    port (
        {<PORT_NAME>: <mode> <type>;}
    );
end <ENTITY_NAME>;

```

4.3 Architecture

mögliche architecture_names:
· Behavioral : Verhaltensbeschreibung (ähnlich wie Prog.sprachen)
· Structural : Strukturbeschreibung (Schaltschema beschrieben)
· TB : Test-Bench

```
-- Syntax
architecture <architecture_name> of <entity_name> is
    [Type_,Subtype_,Constant_,Signal_,Component_declaration]
begin
    {architecture_body: concurrent actions}
end [architecture_name];

```

Beispiel für Structural

```
-- Deklaration
component <component_name>
    port (
        {port_name: <port_mode><port_type>});
end component;

signal <internal_signal_name>: <sygnal_type>;

```

```
-- Initialisierung
<instance_name>: <component_name>
    port map (  -- explizit
        <instance_port_name> => <external_signal_name>,
        <instance_port2_name> => <external_signal2_name>);
        -- or implizit
        (<external_signal_name>, <external_signal2_name>)

```

5 Key Concept II Nebenläufige Prozesse und Prozessiteration

5.1 Signale

```
--  Definition
signal <signal_name1> {,<signal_name2>}: <singal_type> [:=
     initial_value]
-- Std zuweisung
y <= x;
-- Unbedingte Signalzuweisung
[label:] <signal_name> <= Expression {, Exression};
Y <= '1'; ["100101";] [A and B;]    -- Beispiele

```

```
-- Selektive Signalzuweisung
[label:] with <select_signal> select
    <dest_sig> <=   {surce_sig_1 when select_value_1,}
                    source_sig_n when others;
-- Bedingte Signalzuweisung
[label:] <dest_sig> <=  {expr_1 when condition_1 else}
                        expr_n

```

5.2 Nebenläufigkeit Prozesse

```
--  Definition
[<label:] process [(<Sensitivitaetsliste>)]
     -- falls keine, wait nicht vergessen
    <Deklarationsteil> -- Prozess intern
begin
    {<sequentielle Anweisungen>}
end process [<lable>];
-- nur Variabeln im Prozesses aenderbar Bsp.
variable <var_name>: <type> [ := expr];

```

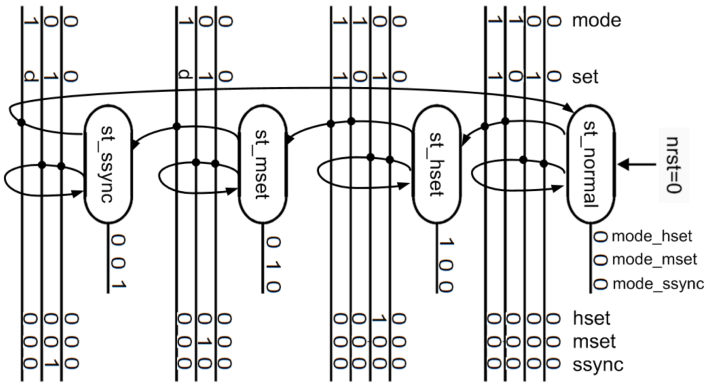
```
-- sequenzielle Anweisungen - if
if condition_a then
    {sequential statements};
{elsif condition_b then
    {sequential statements};}
else
    {sequential statments};
end if;

```

```
-- case
case expression is
    when choice_1   => {sequential
     statement};
    {when choice_n  => {sequential
     statement};}
    when others     => {sequential
     statement};
end case;

```

5.3 Sequentielle Schaltung



- FSM ähnliche Zustände wie Z-Diagramm
- empfohlen die Übergänge mit einem CLK zu lösen

```
if CLK'event and (CLK = '1') then {} -- pos falnke ('0' neg)
```

- Signale in synchronem Prozess auf linker Seite -> FF Flip-Flop
- in Sensitivitätsliste nur CLK (und RST)
- if Anweisungen mit CLK immer am Schluss des Prozesses
- Aufzählungstyp (z.B. für Prozessname) Anzahl Bits kann auch definiert werden

```
type <enum_type_name> is (type_element{, type_element});
```

- hinterlegte Bits für Aufzählungstyp selber bestimmen

```
attribute <ENUM> : <ENUM_type>; -- ENUM encoding
attribute <ENUM> of <aufz_type> : type is expression
subtype <aufz_type> is <const_type>; -- constant encoding
constant <aufz_type_element> : <aufz_type> := expression
```

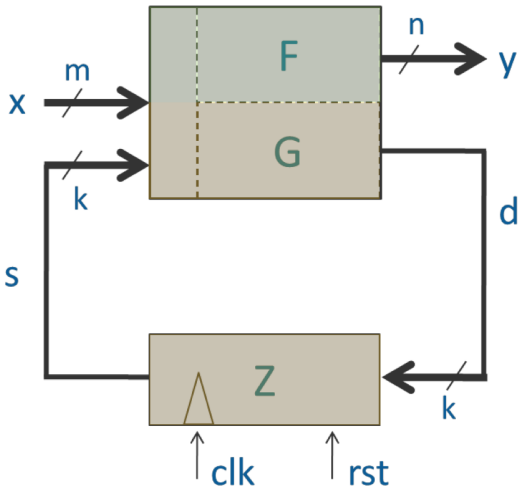
5.4 Endliche Z-Maschinen

```
-- Prozess G = next_state_logic: Kombinatorischer Prozess
    next_state_logic: process (INP, present_state)
    begin
        {Kombinatorische Anweisungen} -- Bestimmung des next_state
    end process;

-- Prozess Z = state_register:
    state_register: process (CLK, RST)
    begin
        {Sequentielle Anweisungen} -- Aktualisiert bei CLK den present_state
    end process;

-- Prozess F = Output_Logic: 1. Mealy
    output_logic: process (INP, present_state)
    begin
        {output_bestimmung} -- Ausgang von IMP und present_state abhaengig
    end process
-- 2. Moore
    output_logic: process (present_state)
    begin
        {output_bestimmung} -- Ausgang nur vom present_state abhaengig
    end process
-- 3. Medwedjew
    output_logic: OUP <= present_state;

```



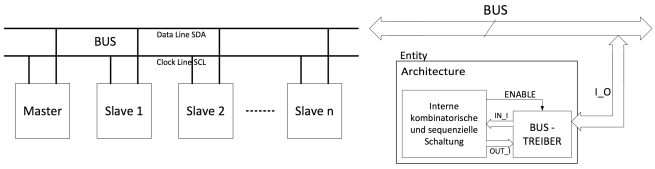
6 Key Concept III Diskreter Ersatz für elektrische Signale

use.std_logic_1164.all enthält folgende 2 Datentype

std_logic	std_ulogic
· solved logic	· unsolved logic
· erlaubt mehrere Treiber an einem Signal. Siehe Tabelle → Kurzschluss Gefahr!	· erlaubt auch mehrere Treiber es darf aber nur einer aktiv sein → Kein Kurzschluss möglich
· Subklasse von std_ulogic	· erkennt in der Sim. nicht initialisierte Signale bezeichnet sie als „U“
· erlaubt Modellierung von Schwachen Signalen (L und H)	· erlaubt Modellierung von Schwachen Signalen (L und H)
· erlaubt DONT'CARES (Signalwert '-')	· erlaubt DONT'CARES (Signalwert '-')
· erlaubt bidirektionale Busse	· nur einfache Busse erlaubt
· grosser Simulationsaufwand	· kleinerer Simulationsaufwand als std_logic
· Vektorform: std_logic_vector	· Vektorform: std_ulogic_vector

6.1 Busse

- Master bestimmt welcher Slave spricht
- Alle Teilnehmer (Slaves) können den Bus jederzeit abhören
- Unaktive Treiber auf „Z“ um den Datenverkehr nicht zu stören
- Beschreibung des Zeitlichen Verhaltens
- Parametisierbarkeit von Modellen



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’

6.2 Codierungsempfehlung

Falls immer möglich (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_vektor ist trotz Unterklasse erst seit 2008 mit std_ulogic_vektor kompatibel (dieser neuer STD ist noch nicht weit verbereitet!).

7 Arithmetick und Datentypen

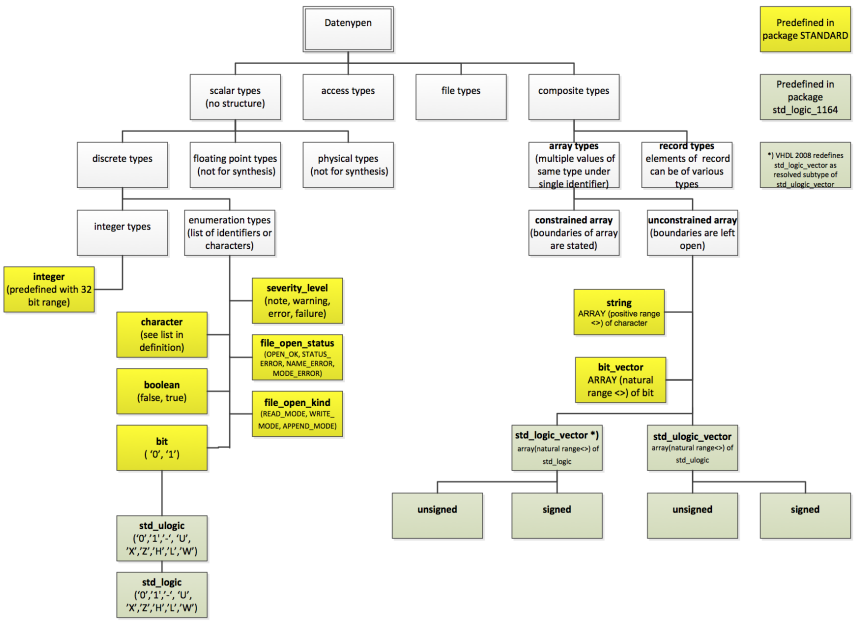
Logische Operatoren

für bit, bit_vector, std_(u)logic, std_(u)logic_vector

mit STD Library	mit ieee.numeric_std ieee.numeric_bit
· and, nand	· + ,-
· or, nor	· signed, unsigned
· xor, xnor	· abs, mod, rem
· not	· *, /, **

7.1 Datentypen

```
1  -- Aufzaelungstyp
2  type <my_type> is ({my_value_1,} my_value_n);
3  type traffic_light is (rot, bruen, orange);-- Bsp.
4  -- Physikalische Typen
5  tyle PHYS_NAME is range RANGE_OF_VALUES
6  units
7      BASE_UNIT;
8      {MULTIPLES;}
9  end units
10 tyle CAPACITANCE is range 0 to 1E30 -- Bsp
11 units
12     pF;
13     nF = 1000 pF;
14     uF = 1000 nF;
15 end units
16 -- Arraytyp
17 type ARRAY_NAME is array
18     (UPPER_LIM downto LOWER_LIM) of BASE_TYPE;
19 type VECTOR_1 is array  -- Bsp
20     (9 downto 0) of integer;
21
```



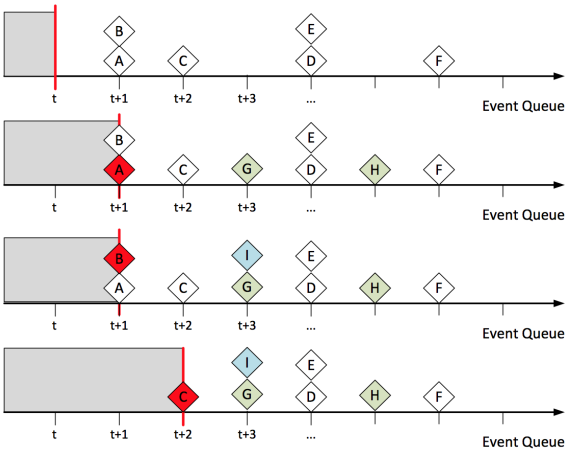

```
1  -- Resize
2  function RESIZE (ARG: signed; NEW_SIZE: natural)
3  return signed;
4  y <= resize(A,y’length) + resize(B,y’length);
5  -- Type_cast
6  target_signal <= target_type (source_signal);
7  -- Type_convercion
8  target_signal <= to_<target_type> (source_signal);
```

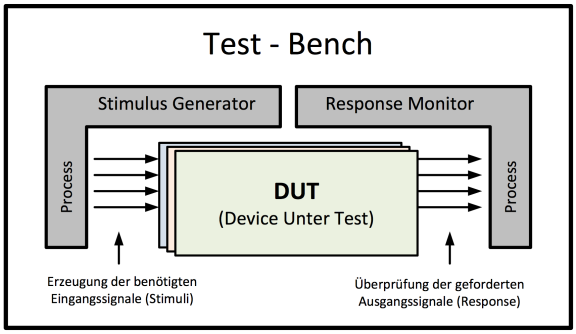
Konversionsfunktion	Argumentyp
To_bit	- std_ulogic
To_StdULogic	- bit
To_bitvector	- std_ulogic_vector - std_logic_vector
To_StdULogicVector	- bit_vector - std_logic_vector
To_StdLogicVector	- bit_vector - std_ulogic_vector

8 Key Concept IV Zeitliche Darstellung bei Simulation

- Die Simulation / Funktionstest wird mit Hilfe einer Test-Banch gemacht
- Die Test-Banch sollte Wiederverwendbarsein
- Test-Bench unabhängig des verwendeten verfahren des DUT
- Um so früher ein Fehler bemerkt wird desto „billiger“ ist er
- VHDL-Simulatoren arbeiten mit einer Event-Queue (sie Bild)
- (SPICE-Simulatoren arbeiten mit DGL → brauchen viel mehr Rechenleistung)
- Delta-Zyklus: (auf Bild t's)
  1. Update Request: wartet auf Trigger
  2. Prozessausfürung: 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

```vhdl
B <= transport A after tp;      -- tp in type "time"
D <= inertial C after tp;       -- "inertial " nicht noetig
```





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

```vhdl
-- for loop
[label:] for <parameter> in <range>
    loop
    {sequential statements}
end loop [label];
-- assert
[label:] assert condition [report
    string_expretion]
    [severity warning | error | ..];
```

```vhdl
-- 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;
```

9 Key Concept V Parametisierbarkeit von Modellen

Das Ziel von Parametern und Modellen ist es, dass man einen ähnlichen block wiederverwerten kann. z.B wenn man einen Zähler macht der auf 100 zählt, währe es schön wenn man den selben „Block“ auch für einen der auf 230 zählt brauchen kann.

```vhdl
-- Entidy declaration mit generic Parameter:
entity entity_name is
    generic ({generic_name: type_name [:=
     default_value]});
    port    (port_list);
end entity_name;

-- Component Declaration mit generic Parameter
component component_name
    generic (generic_list);
    port    (port_list);
end component;

```

```vhdl
-- Zaehler Beispiel:
entity count_x is
    generic (max_count : integer := 127);
    port    (clk, rst, ena  : in std_logig;
             oup            : out std_logic_vector
                    (integer(ceil(log2(real(max_count)))) -1 downto 0));
end count_x

architecture RTL of count_x is
    constant word_with: integer :=(integer(ceil(log2(real(max_count)))));
    signal present_count, next_count : unsigned(word_width-1 downto 0);

begin
    output_logic : oup <= std_logic_vector(present_count);

    next_state_logic: process(present_count_ ena)
    begin
        next_count <= present_count;
        if ((present_count = max_cpunt) and (ena = '1')) then
            next_count <= (others => '0');
        elsif (ena = '1') then next_count <= present_count + 1;
        end if;
    end process;
    -- Fortsetzung auf naechster Seite
```

Fixierung von Designparameter

Parameter zu Designzeit : Konstanten
Parameter zu Compilezeit : Generic
Parameter zu Laufzeit : Signale

```
-- Fortsetzung:
    register : process(clk, rst)
    begin
        if (rst = '1') then present_count <= (others => '0');
        elsif (clk'event and (clk = '1')) then present_count <=
     next_count;
        end if;
    end process;
end architecture RTL;
```

10 Beispiele

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;

entity edgedetpos_random_tb is
end edgedetpos_random_tb;

architecture tb of edgedetpos_random_tb is
    constant f : integer := 1000;       -- Frequency in HZ
    constant T : time    := 1 sec / f;

    component edgedetpos
        port(
            clk  : in  std_ulogic;
            nrst : in  std_ulogic;
            inp  : in  std_ulogic;
            oup  : out std_ulogic
        );
    end component;

    for all : edgedetpos use entity work.edgedetpos(
     behavioral);

    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 : edgedetpos
        port map(
            clk  => clk_tb,
            nrst => nrst_tb,
            inp  => inp_tb,
            oup  => oup_tb
        );

    stimuli_clk : process
    begin
        clk_tb <= '0';
        wait for T / 2;
        clk_tb <= '1';
        wait for T / 2;
    end process;

    stimuli_nrst : nrst_tb <= '0', '1' after 3 ms;

    stimuli_inp : process
        variable t      : time;
        variable t_real : real;
        variable seed1  : positive := 1;
        variable seed2  : positive := 1;

    begin
        inp_tb <= '0';
        loop
            UNIFORM(seed1, seed2, t_real);
            t := (t_real * 0.007) * 1 sec;
            wait for t;
            inp_tb <= not inp_tb;
        end loop;
    end process;

stimuli_oup_exp : process
    begin
        loop
            oup_exp_tb <= '0';
            if (nrst_tb = '0') then
                wait until (nrst_tb'event and nrst_tb = '1');
                wait until (clk_tb'event and clk_tb = '1');
            end if;
            if (inp_tb = '0') then
                while (inp_tb = '0') loop
                    wait until (clk_tb'event and clk_tb='1');
                end loop;
                oup_exp_tb <= '1';
                wait until (clk_tb'event and clk_tb = '1');
                oup_exp_tb <= '0';
            else
                wait until (clk_tb'event and clk_tb = '1');
            end if;
        end loop;
        wait;
    end process;

    evaluation : process
    begin
        wait until (clk_tb'event and clk_tb = '1');
        wait for 0.5 us;
        loop
            assert (oup_exp_tb = oup_tb) report "Error oup"
     severity error;
            wait for 1 us;
        end loop;
        wait;
    end process;

end tb;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity edgedetpos is
    port(
        clk  : in  std_ulogic;
        nrst : in  std_ulogic;
        inp  : in  std_ulogic;
        oup  : out std_ulogic
    );
end;

architecture behavioral of edgedetpos is
    type state_type is (reseteststate, wait_for_1, pulse,
     wait_for_0);

    signal present_state : state_type;
    signal next_state    : state_type;

begin
    -- short form for output logic. Code version with process
     : see edgedetneg
    Output_logic:   oup <= '1' when present_state = pulse
     else '0';

    Next_state_logic : process(inp, present_state)
    begin
        next_state <= wait_for_0;       -- default state
        case present_state is
            when wait_for_1 =>
                if (inp = '1') then
                    next_state <= pulse;
                else
                    next_state <= wait_for_1;
                end if;
            when pulse =>
                if (inp = '0') then
                    next_state <= wait_for_1;
                -- else                       -- due to default
                -- next_state <= wait_for_0;
                end if;
            when wait_for_0 =>
                if (inp = '0') then
                    next_state <= wait_for_1;
                -- else
                -- next_state <= wait_for_0;
                end if;
            when others =>
                if (inp = '0') then
                    next_state <= wait_for_1;
                -- else
                -- next_state <= wait_for_0;
                end if;
        end case;
    end process;

    registers : process(nrst, clk)
    begin
        if (nrst = '0') then
            present_state <= reseteststate;
        elsif (clk = '1') and clk'event then
            present_state <= next_state;
        end if;
    end process;

end behavioral;
```

```vhdl
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;
```

```vhdl
architecture TEST of BUS_DRIVER is

begin

-- Bus_Komponente
    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 Test;
```

```vhdl
entity mux3x8_top is
    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 mux3x8_top;

architecture RTL of mux3x8_top is
    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;

    for all: mux3x8 use entity work.mux3x8(
     PROCESS_IF_NO_Default);
begin
     U1: mux3x8
        port map(sel  => sel,
                 inp0 => inp0,
                 inp1 => inp1,
                 inp2 => inp2,
                 oup  => oup);
end architecture RTL;
```

```vhdl
entity or2 is
  port(
    a, b : in bit;
    y : out bit);
  end;

architecture behavioral of or2 is
begin
   y <= a or b;
end behavioral;
```

```vhdl
-- Beispiel ENUM ENCODING
type STATE_TYPE is (ST_NORMAL, ST_HSET, ST_MSET, ST_SSYNC);
-- Explizite codierung mit ENUM_ENCODING
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of STATE_TYPE:
    type is "0001␣0010␣0100␣1000"; -- one hot
signal PRESENT_State, NEXT_State : STATE_TYPE;

                -------------------------
                -- State     | Encoding --
                ------------|------------
                -- st_normal | 0001     --
                -- st_hset   | 0010     --
                -- st_mset   | 0100     --
                -- st_ssync  | 1000     --
                -------------------------
```