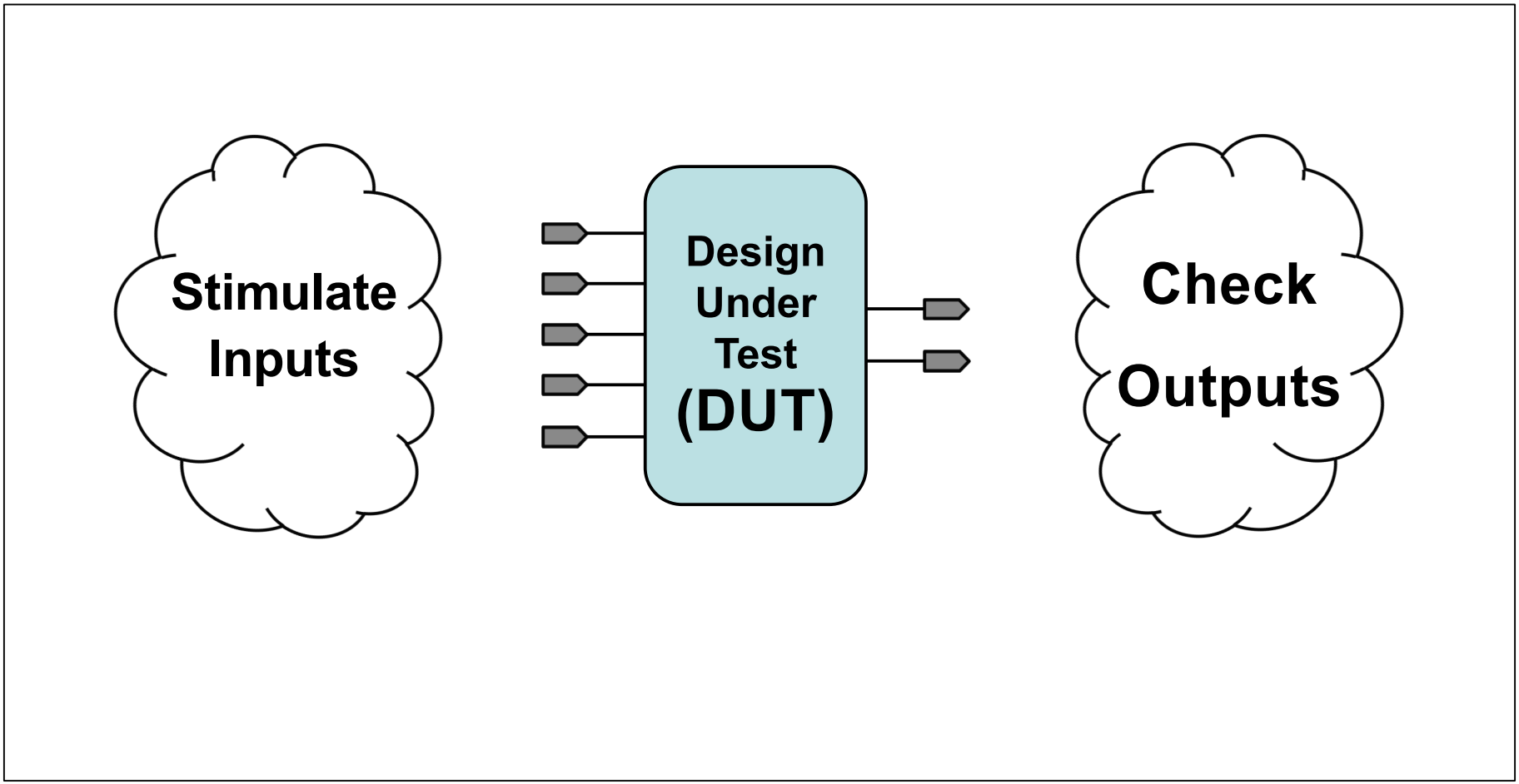
Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

VHDL Simulation

Testbench Architecture



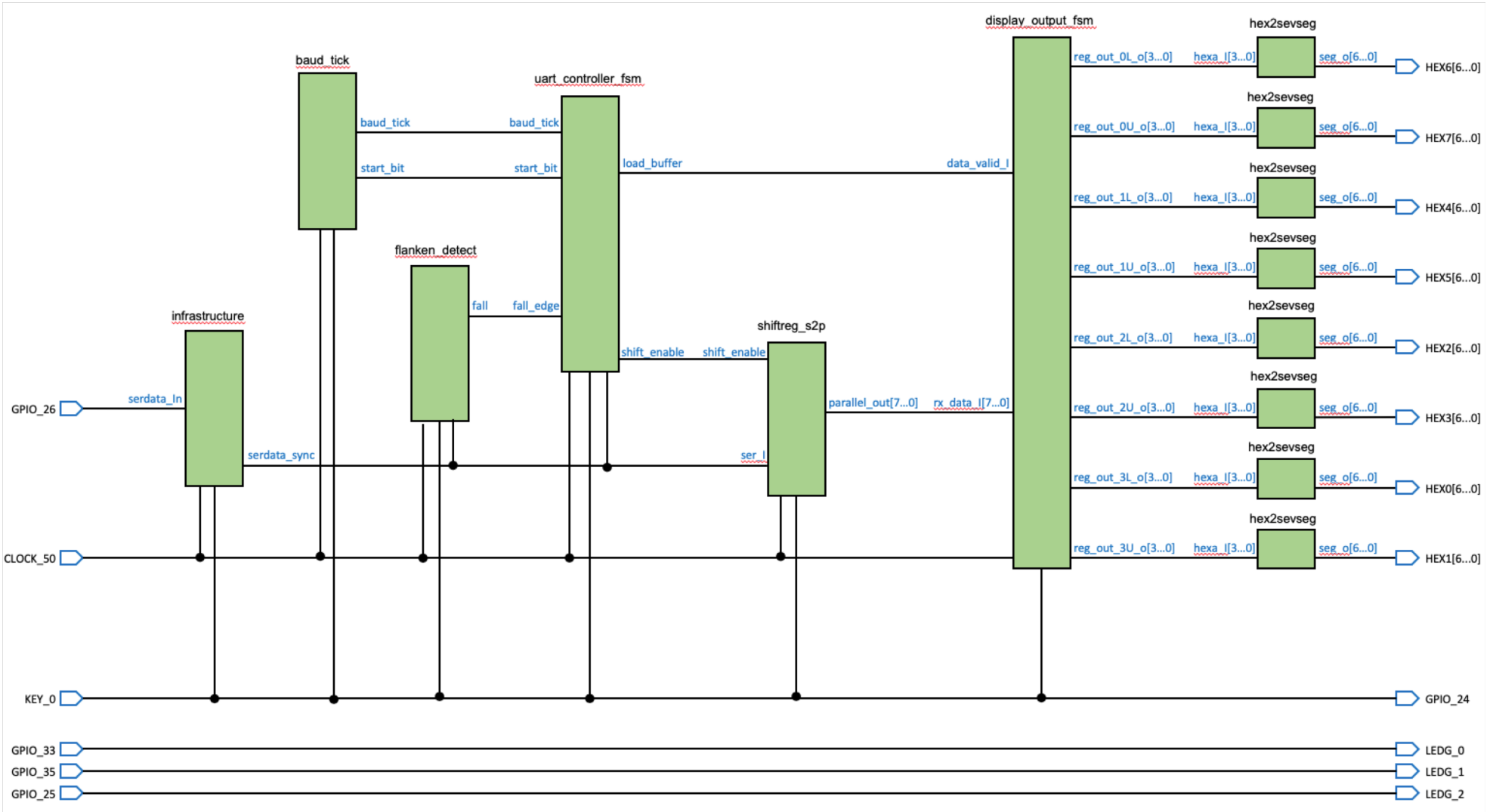
Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

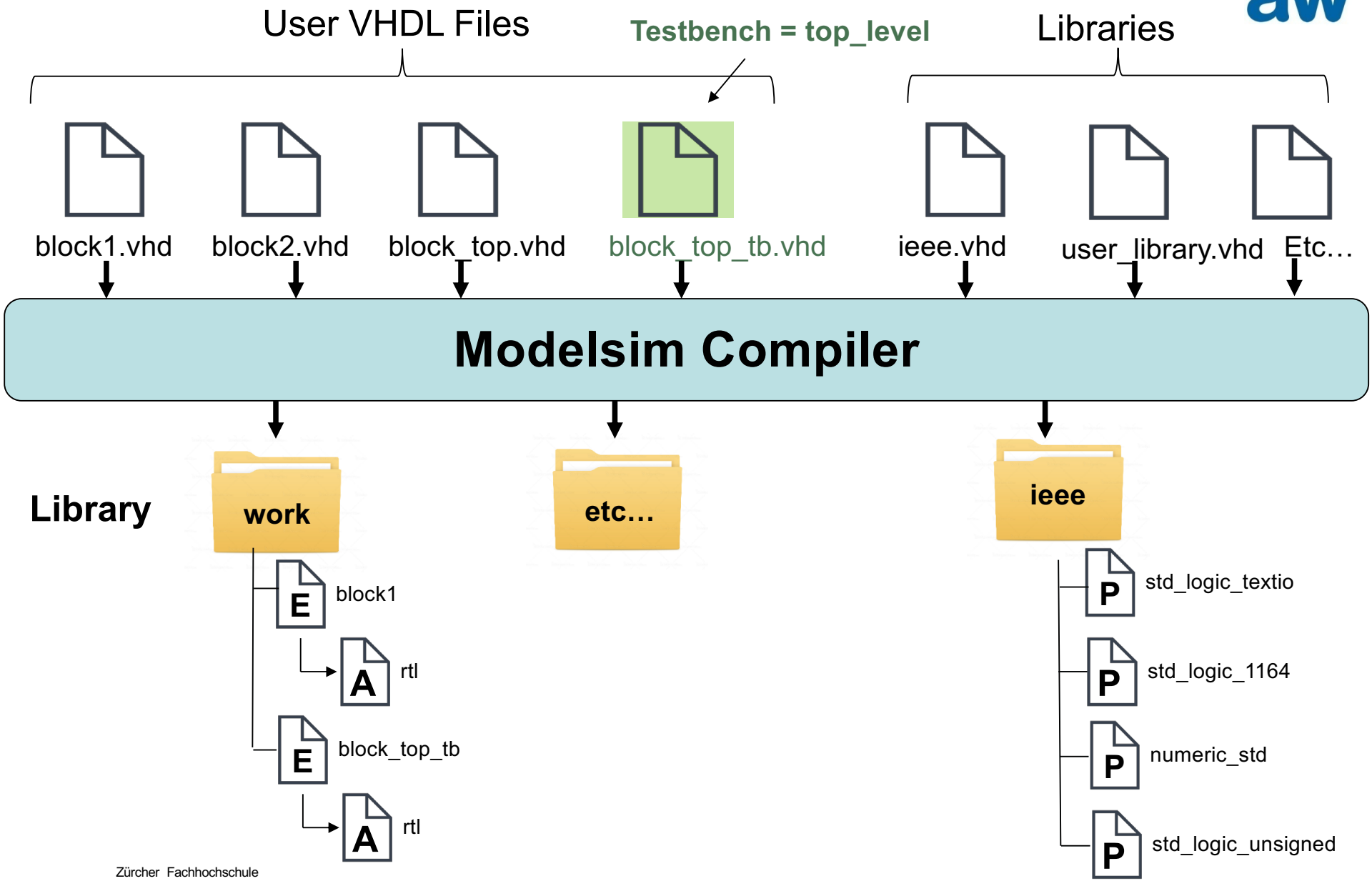
DTP2_Vorlesung_01

- Einführung
- Procedures
- Packages
- Procedure Based Testbenches
- Simulation Specific VHDL Commands
- Testbench Code
- File Organisation
- Setting Up Modelsim Projects

VHDL TOP Level für UART: uart_top



Modelsim VHDL Simulator

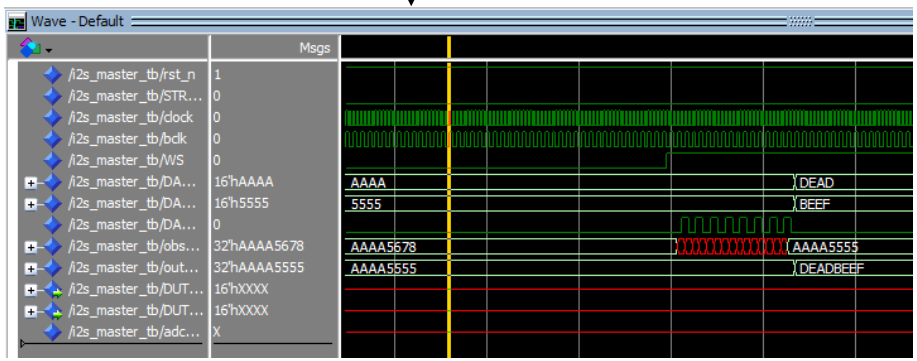


Modelsim VHDL Simulator

Library

Modelsim Simulator

```
# Refreshing C:/Lehre_18/dtp2/Reference_synthi_18/synthi_18/simulations/i2s_master
# Loading work.i2s_master(top)
# Refreshing C:/Lehre_18/dtp2/Reference_synthi_18/synthi_18/simulations/i2s_master
# Loading work.modulo_divider(rtl)
# Refreshing C:/Lehre_18/dtp2/Reference_synthi_18/synthi_18/simulations/i2s_master
# Loading work.frame_decoder(rtl)
# Refreshing C:/Lehre_18/dtp2/Reference_synthi_18/synthi_18/simulations/i2s_master
# Loading work.p2s(rtl)
# Refreshing C:/Lehre_18/dtp2/Reference_synthi_18/synthi_18/simulations/i2s_master
# Loading work.s2p(rtl)
# LOGIC HAS BEEN RESET FOR 400 ns
# AAAAAAAA WRITTEN TO I2S BUS(CODEC ANALOG OUT)
# AAAAAAAA RETURNED FROM DUT, O.K.
# 12345678 WRITTEN TO I2S BUS(CODEC ANALOG OUT)
# 12345678 RETURNED FROM DUT, O.K.
# 87654321 WRITTEN TO I2S BUS(CODEC ANALOG OUT)
# 87654321 RETURNED FROM DUT, O.K.
# HEX VALUE 12345678 APPLIED to DUT
# AUDIO DATA RECEIVED CORRECT BY CODEC 12345678
# HEX VALUE AAAA5555 APPLIED to DUT
# AUDIO DATA RECEIVED CORRECT BY CODEC AAAA5555
# HEX VALUE DEADBEEF APPLIED to DUT
# AUDIO DATA RECEIVED CORRECT BY CODEC DEADBEEF
#  CONGRATULATIONS ALL TESTS PASS
# Break in Subprogram end_simulation at ../../support/simulation_pkg.vhd line 62

VSIM 2>
```



Text Output

Waveform Output

Simulator compile.do Script

```
# create work library
vlib work

# compile project files
vcom -2008 -explicit -work work ../../support/simulation_pkg.vhd
vcom -2008 -explicit -work work ../../support/standard_driver_pkg.vhd
vcom -2008 -explicit -work work ../../support/user_driver_pkg.vhd
…
vcom -2008 -explicit -work work ../../../source/flanken_detect.vhd
vcom -2008 -explicit -work work ../../../source/hex2sevseg.vhd
vcom -2008 -explicit -work work ../../../source/modulo_divider.vhd
vcom -2008 -explicit -work work ../../../source/uart_top_tb.vhd


# run the simulation
vsim -novopt -t 1ns -lib work work.uart_top_tb
do ./wave.do
run 50 ms
```

VHDL Procedures

Definition

Procedures helfen den Code kompakt und übersichtlich zu halten. Sie enthalten eine Befehls-Sequenz die man beliebig oft aufrufen kann.

```
procedure "procedure_name" (parameter_list) is
  declarations
begin
  sequential statements
end "procedure_name";
```

VHDL Procedures

Beispiel

```
procedure parity_generator
   (signal din : in  std_logic_vector;
    signal par : out std_logic) is

  variable t : std_logic := '0';
begin
   for i in din'range loop
     t := t xor din(i);
   end loop;
   par <= t;
end parity_generator;
```

Procedure Aufruf

```
...
parity_generator(databus,par_bit);
...
```

VHDL Package

Definition

Typen, Objekte oder typspezifische Operatoren lassen sich in Bibliotheken einbinden Bibliotheken lassen sich in jedes VHDL-Modell einbinden. Solche gemeinsamen Definitionen werden in Packages gemacht.

```
package "package_name" is
  declarations
end package "package_name" ;
```

```
package body "package_name" is
  descriptions
end package body "package_name" ;
```

Benutzung in Entity Architektur Datei

```
use work.package_name.all;
```

VHDL Package

Beispiel einfach 1

```
package tone_gen_pkg is

 type t_midi_array is array (0 to 9) of t_note_record;
 constant N_CUM: natural :=19;
 constant N_LUT: natural :=8;
 constant L:              natural := 2**N_LUT;
 constant N_RESOL:        natural := 13;
 constant N_AUDIO :       natural := 16;

end package tone_gen_pkg;
```

Aufruf

```
use work.tone_gen_pkg.all;
```

Definition eines «Records» im Package

Beispiel einfach 2

```
package simulation_pkg is

  type test_vect is record
    arg1          : std_logic_vector(7 downto 0);
    arg2          : std_logic_vector(7 downto 0);
    arg3          : std_logic_vector(7 downto 0);
    arg4          : std_logic_vector(7 downto 0);
    obs_data      : std_logic_vector(31 downto 0);
    fail_flag     : boolean;
    clock_period : time;
  end record;

end package simulation_pkg;
```

Referenzierung später in Architektur

```
signal tv       : test_vect;
```

Zugriff auf einzelnes Record Element

```
tx_sig <= tv.arg1;
```

Package mit Procedure

Beispiel mit Procedure

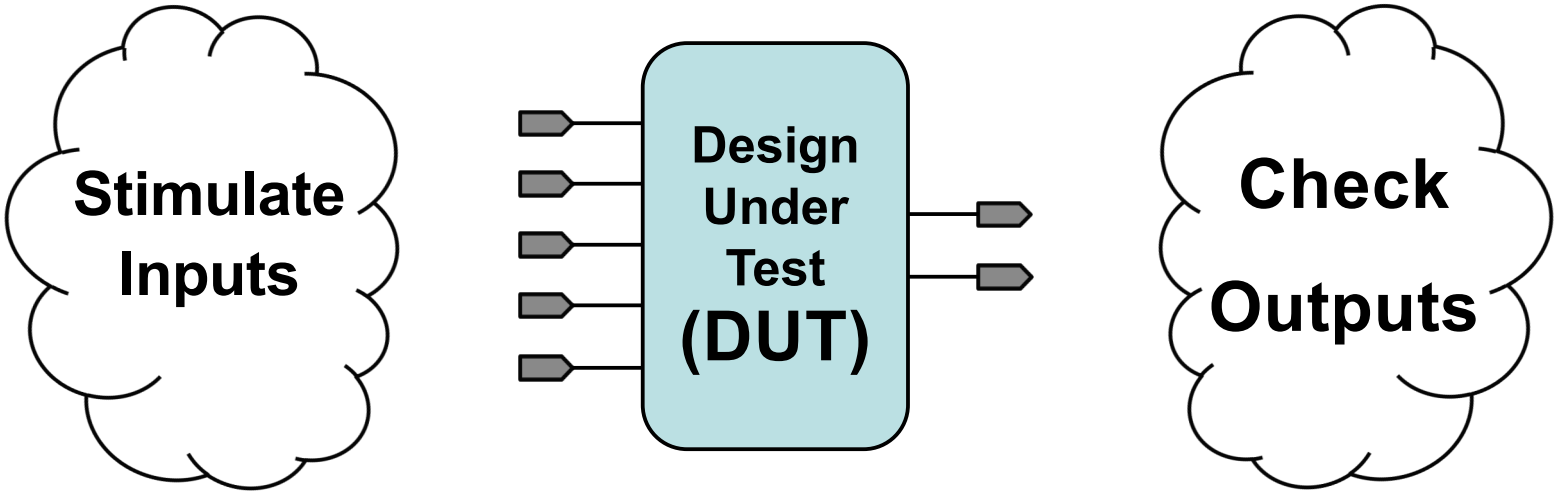
```
package my_package is
  procedure parity_generator
    (signal din : in  std_ulogic_vector;
     signal par : out std_ulogic);
end my_package;
```

Procedure Declaration

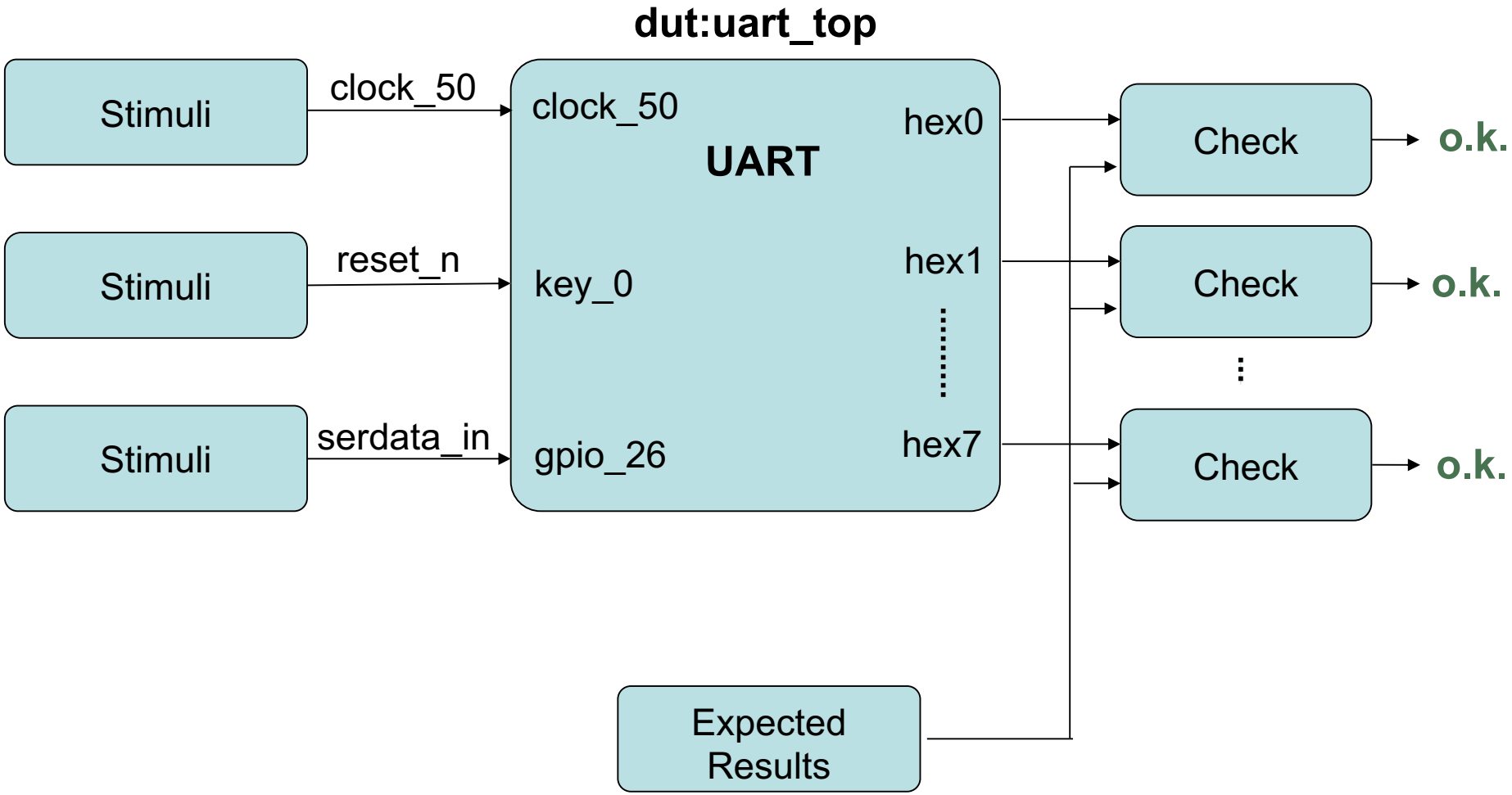
```
package body my_package is;
    procedure parity_generator
      (signal din : in  std_logic_vector;
       signal par : out std_logic) is
       variable t : std_logic := '0';
    begin
      for i in din'range loop
        t := t xor din(i);
      end loop;
         par <= t;
    end procedure parity_generator;
end package body my_package;
```

Procedure Definition im Package Body

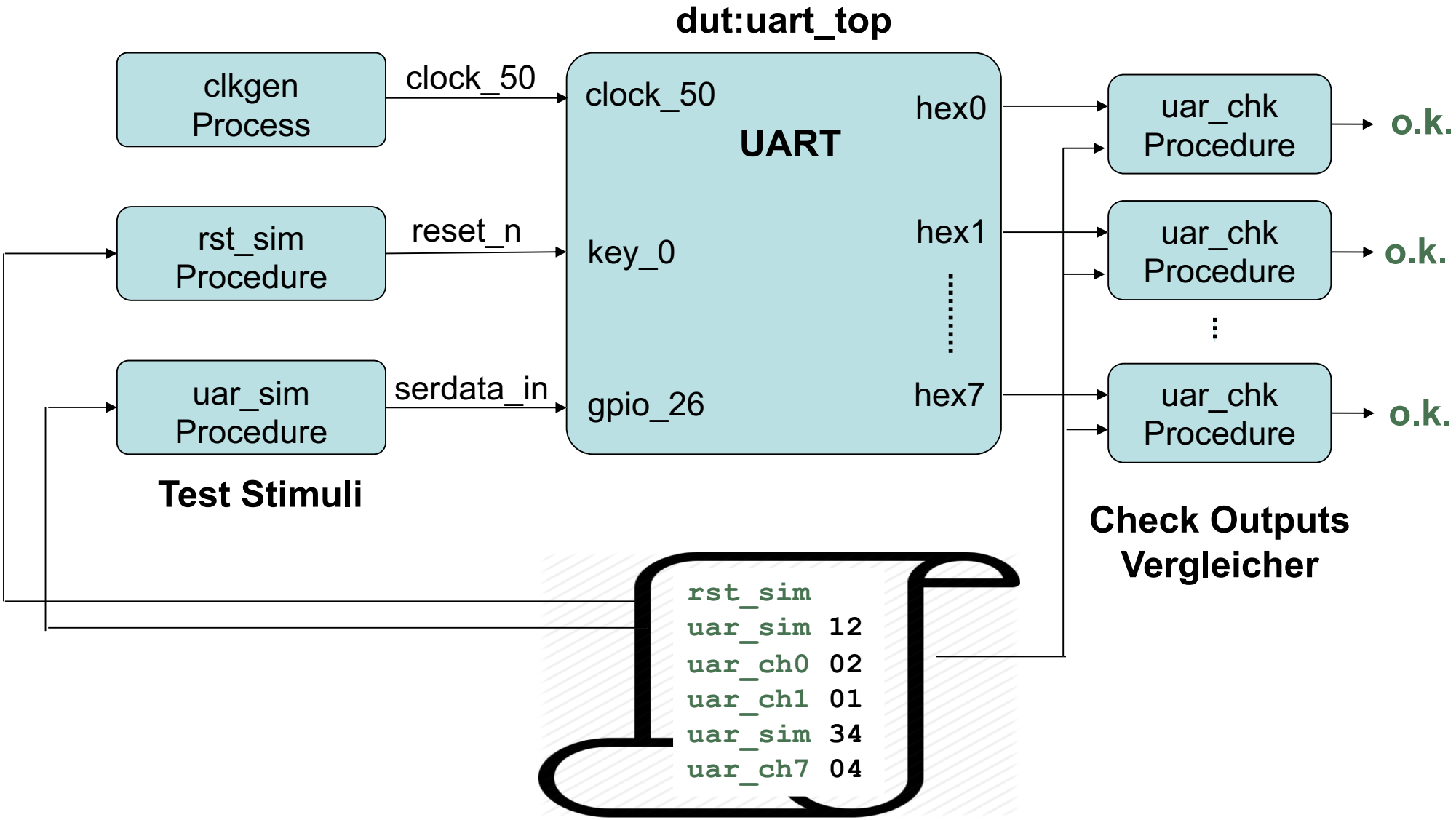
Testbench Architecture



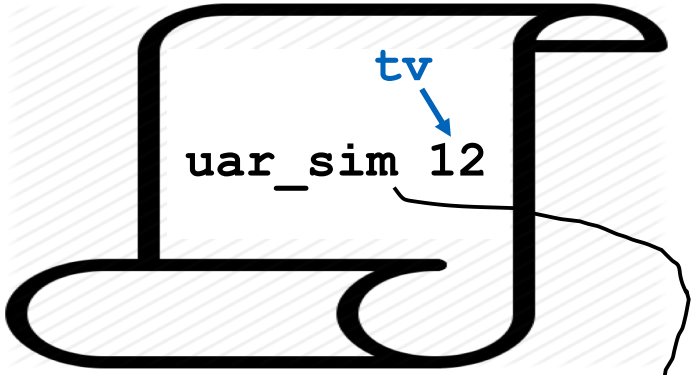
Testbench für UART



Procedure Based Testbench



Procedure Based Testbenches



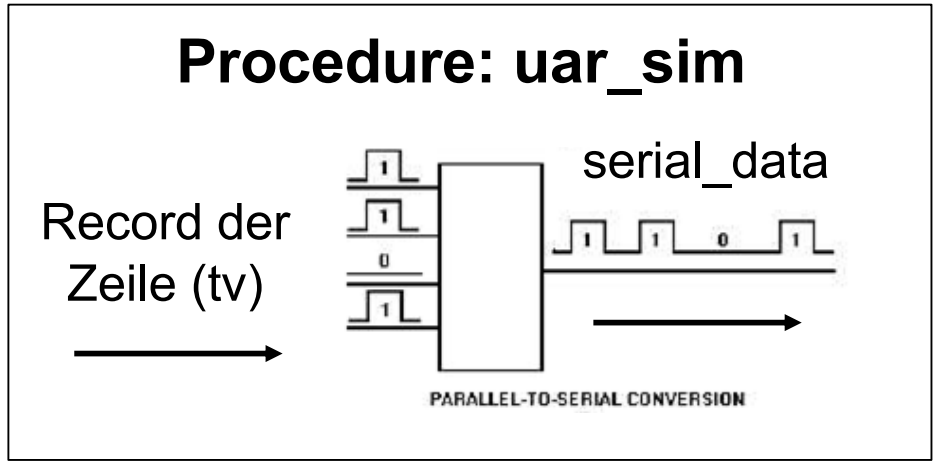
Test Vector Script: testcase.dat

VHDL Record mit Kommando und Argumenten

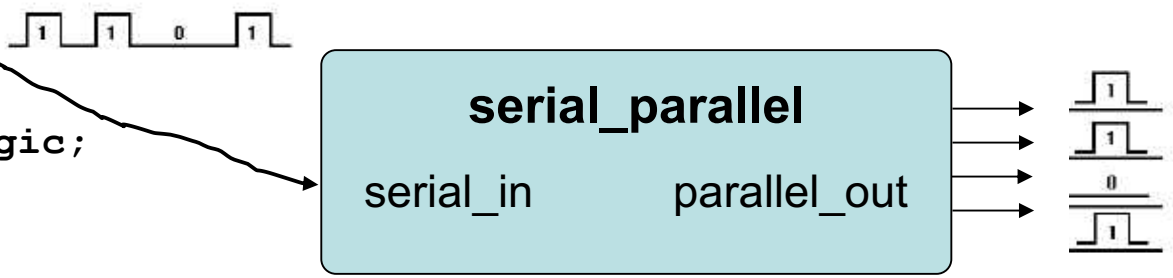
```
uar_sim(tv, serial_data_pin);
```

```
signal serial_data : std_logic;
```

Procedure: uar_sim



DUT



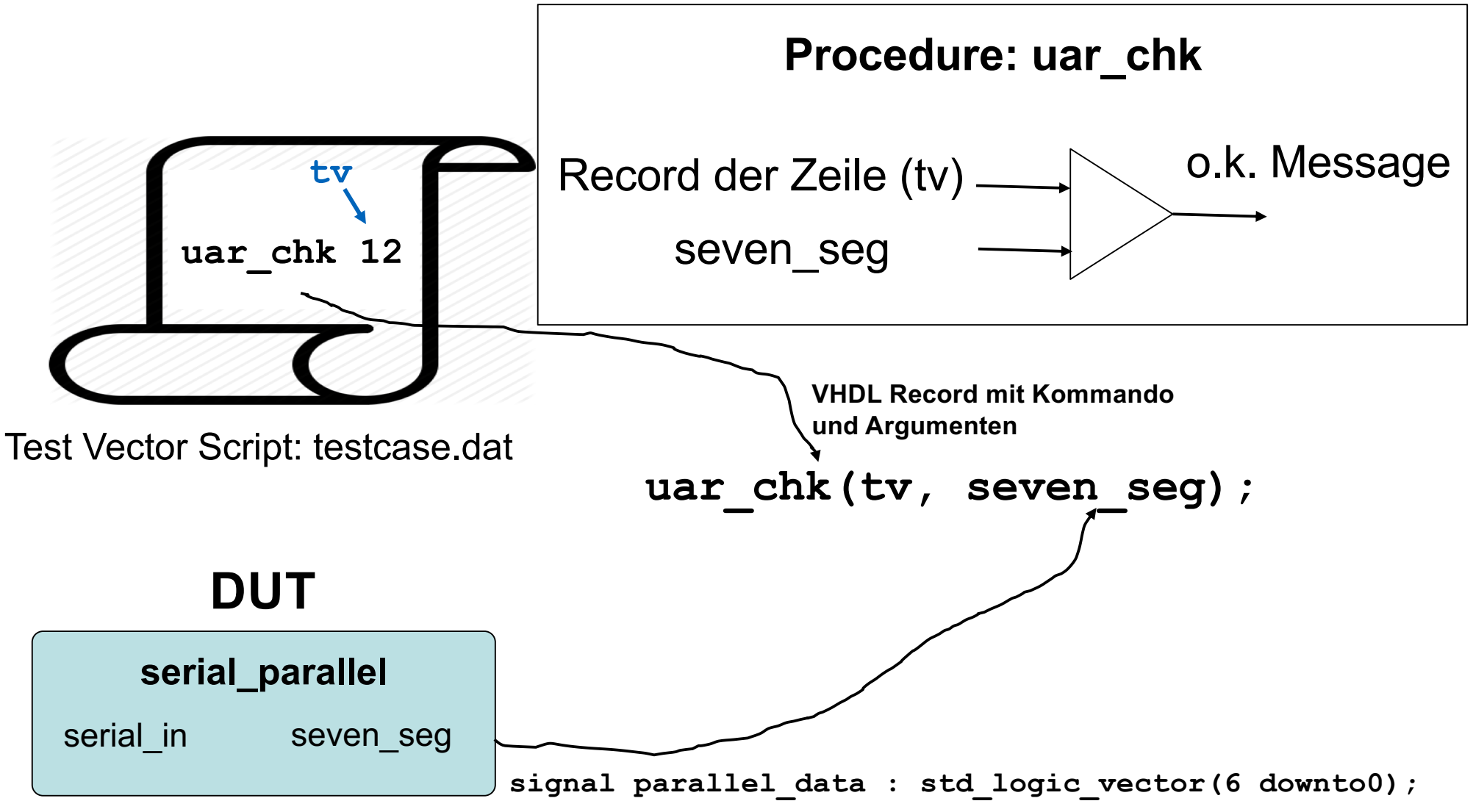
Serielle Signalerzeugung mit «For Loop»

uar_sim Procedure

```
.
.
.
txloop : for i in 0 to 7 loop
    tx_sig <= tv.arg1(i);
       wait for baud_period;
end loop txloop;
.
.
.
```

standard_driver_pkg.vhd

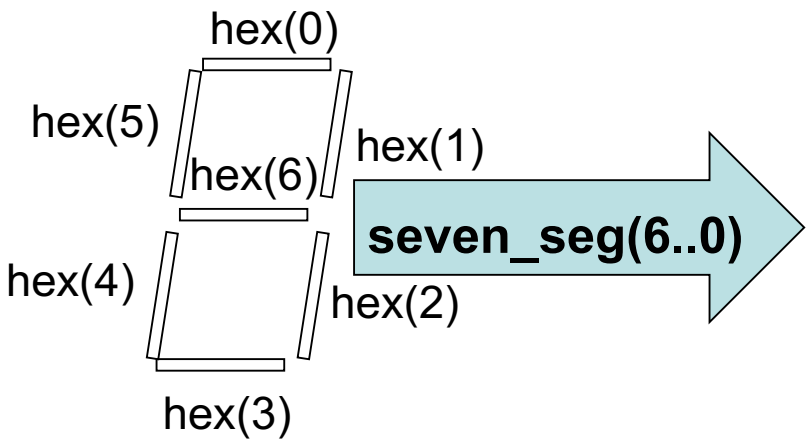
Procedure Based Testbenches



Aufbau der uar_chk Procedure

Led on = ‘0’
Led off = ’1’

Tabelle 7-segm to binär



7-Segment Ausgang vom UART block

seven_seg(6..0)

"1000000“	=> 0x00
"1111001“	=> 0x01
"0100100“	=> 0x02
"0110000“	=> 0x03
"0011001“	=> 0x04
"0010010“	=> 0x05
"0000010“	=> 0x06
"1111000“	=> 0x07
"0000000“	=> 0x08
"0010000“	=> 0x09
"0001000“	=> 0x0a
"0000011“	=> 0x0b
"1000110“	=> 0x0c
"0100001“	=> 0x0d
"0000110“	=> 0x0e
"0001110”	=> 0x0f

Vergleicher

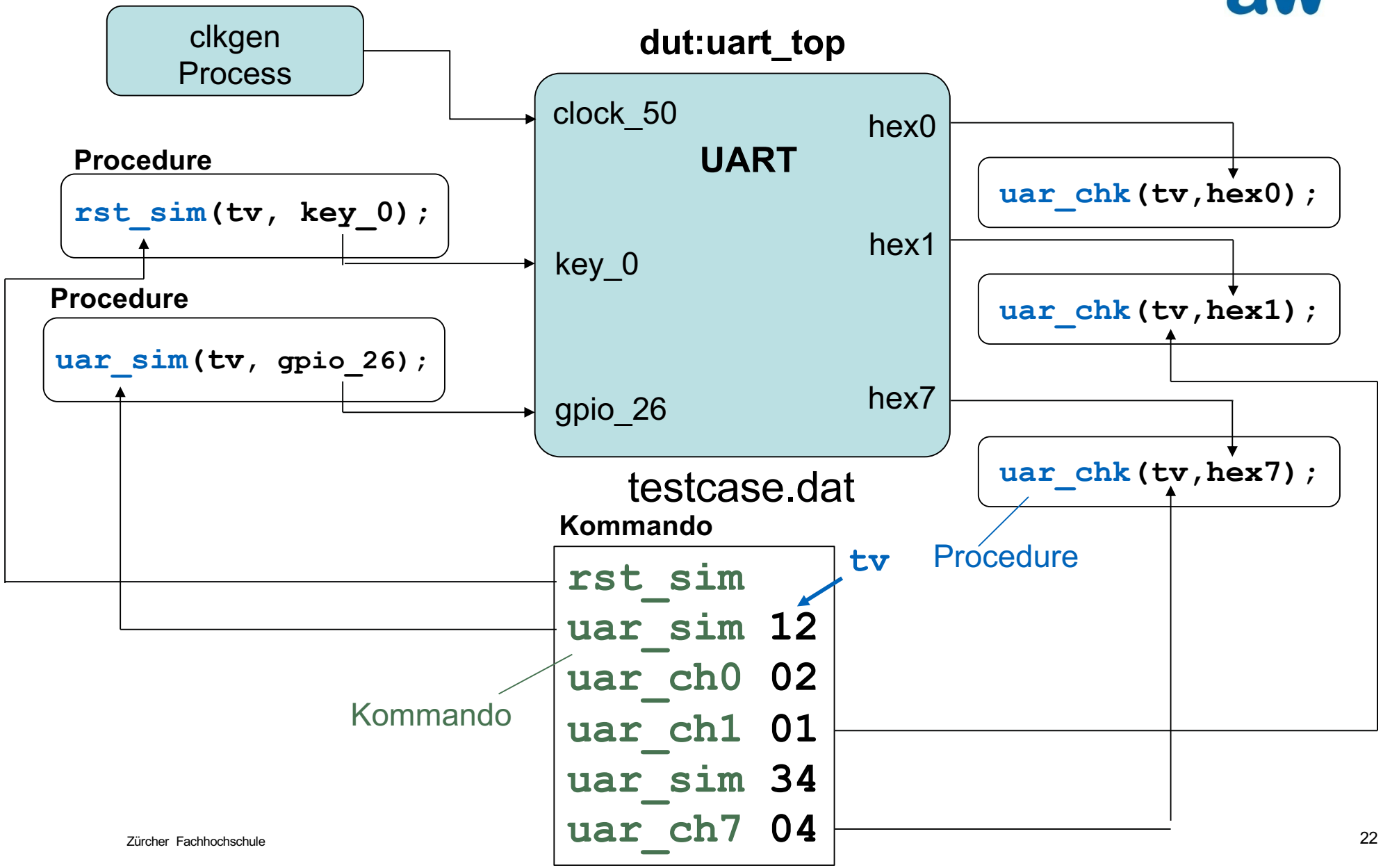
(7..0)

tv.arg1(7..0)

Argument von uar_chk Kommando-Zeile

o.k. /
not o.k.

Procedure Based Testbenches



Kommando löst Procedure in Testbench aus

```
if cmd = string'("rst_sim") then
  rst_sim(tv, key_0);

elsif cmd = string'("uar_sim") then
  uar_sim(tv, gpio_26);

elsif cmd = string'("uar_ch0") then
  uar_chk(tv, hex0);

elsif cmd = string'("uar_ch1") then
  uar_chk(tv, hex1);

elsif cmd = string'("uar_ch2") then
  uar_chk(tv, hex2);

else
  assert false
    report "Unknown Command" severity failure; --
end if;
```

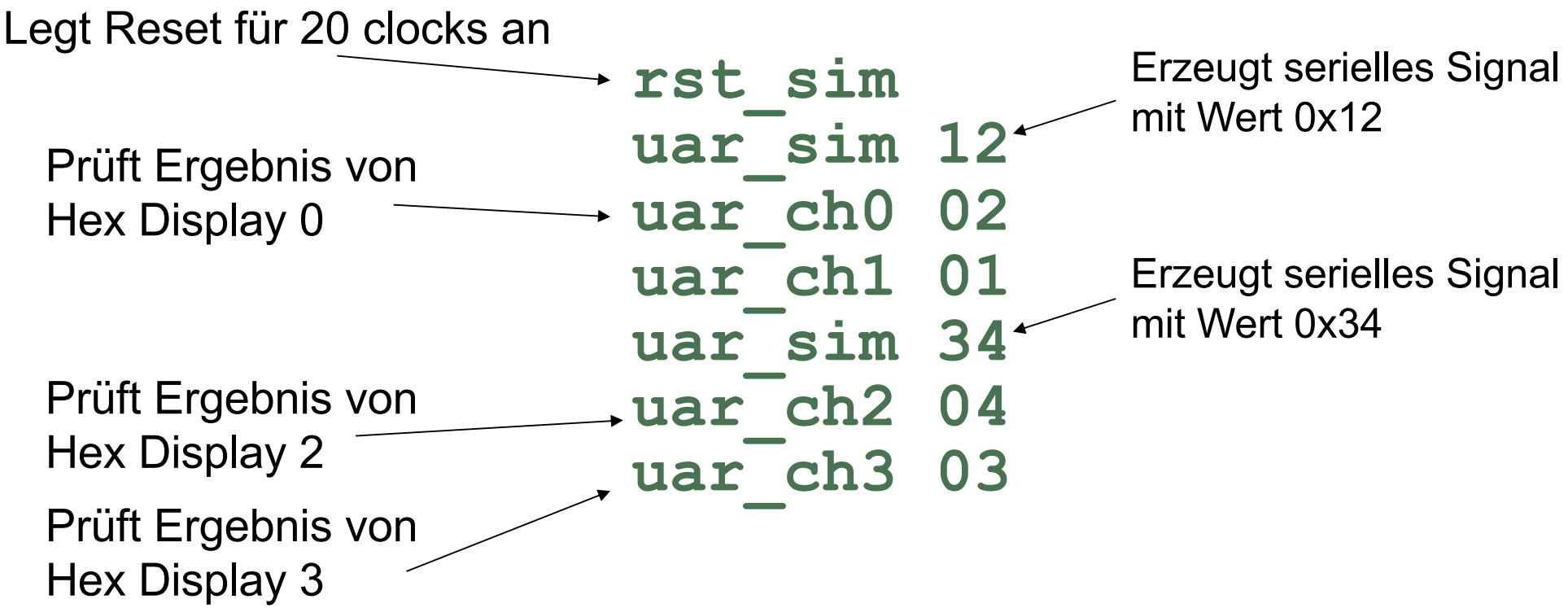
Procedure

Kommando

Unterschiedliche Kommandos können die gleichen Procedures auslösen

Test Script: testcase.dat

Beispiel einer Testsequenz zum Testen des UART



Test Script: testcase.dat

Aufbau eines Kommandos in DTP2 Simulationen

```
Kommando arg1 arg2 arg3 arg4
```

Kommando:
Definieren die Test-Procedure.
Test-Procedures erzeugen Stimuli oder führen Checks aus.
String mit genau 7 Zeichen

Argument 1 bis Argument 4:
Werden in der Procedure zu Stimuli umgewandelt bzw. dienen als Vergleichswerte für Checks.
Min. ein, max vier Argumente möglich.
Je Argument genau ein Byte mit Hex Wert.
MSB arg1, LSB arg4

Beispiel

```
gpi_sim 12 aa 5f 3e
```

Erzeugt einen Stimulus mit dem Hex Wert 0x12aa5f3e, der an den Eingang eines DUT angelegt wird.

Verfügbare Test Procedures

Zürcher Hochschule
für Angewandte Wissenschaften

zhaw

Command	Function	Arguments	Input/Output Signals
rst_sim	Resets Simulation	None	Output: low active reset signal std_logic
gpi_sim	General purpose stimulus signal	arg1 – MSB arg2 arg3 arg4 – LSB	Output: 32 general purpose bits std_logic_vector(31 downto 0)
gpo_chk	General Purpose Check	arg1 – MSB arg2 arg3 arg4 – LSB	Input: 32 general purpose bits std_logic_vector(31 downto 0)
uar_sim	UART serial signal generation	arg1- Byte hex value arg2 – 01= 31’250kBd default=115’200kBd arg3 – 01= stop bit = 0 inserts stop error	Output: serial signal std_logic
uar_chk	Seven Segment display check	arg1- Byte Hex value with leading 0	Input: 8-bit hex value with one leading zero std_logic_vector(7 downto 0)
i2s_sim	Generates 32-bit serial i2s signal	arg1-MSB arg2 arg3 arg4-LSB	Output: i2s Serial Signal std_logic
i2s_chk	Checks a 32-bit serial i2s signal against a 32-bit value	arg1-MSB arg2 arg3 arg4-LSB	Input: i2s serial signal, bclk, ws all std_logic
run_sim	Runs simulation for n clock cycles	arg1-MSB Number of simulation clk cycles arg2 arg3 arg4-LSB	None

Wait Statements

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

```
wait until condition;
```
Warten bis ein bestimmter Zustand eintritt

```
wait on signal_list;
```
Warten bis ein bestimmte(s) Signal(e) wechselt

```
WAIT FOR time;
```
Eine bestimmte Zeit warten

```
wait;
```
Unbestimmt Warten

Beispiele:

```
wait until CLK = '1';
wait for 10 ns;
wait on a,b;
```

WAIT Statements

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

- WAIT Statements sind sequentielle Statements und dürfen nur im Prozess vorkommen
- Beim Ausführen des WAIT Statements wird der Process unterbrochen und die zugewiesenen Signale werden aktualisiert
- Nach Ausführen der WAIT Bedingung wird der Prozess an der Stelle fortgefahren, wo er unterbrochen wurde
- WAIT ist nicht synthetisierbar

Erzeugung des Taktes für die Simulation

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

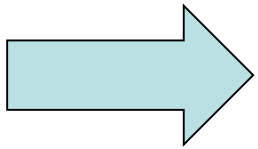
```
…
CONSTANT  clock_period      : time := 20 ns;
…
    clkgen : PROCESS
    BEGIN
        clock_50 <= '0';
        WAIT FOR clock_period/2;
        clock_50 <= '1';
        WAIT FOR clock_period/2;
    END PROCESS clkgen;
…
END struct;
```

(Ohne Sensitivity Liste)

Einfache Generierung von Stimuli

Primitve Erzeugung von Stimuli

```
stimuli: PROCESS
          BEGIN
          key_0 <= '0';
          gpio_26 <= '1';

          WAIT FOR 12 * clk_halfp;
          key_0 <= '1';
          WAIT FOR baud_period;
          gpio_26 <= '1';
          WAIT FOR baud_period;
          gpio_26 <= '0';
          WAIT FOR baud_period;
          gpio_26 <= '0';
          WAIT FOR baud_period;
          gpio_26 <= '1';
          WAIT FOR baud_period;
          gpio_26 <= '1';
          WAIT FOR baud_period;
          gpio_26 <= '1';
          WAIT FOR baud_period;
          gpio_26 <= '0';
          WAIT FOR baud_period;
          gpio_26 <= '0';
          WAIT FOR baud_period;
          gpio_26 <= '1';
          WAIT;
          END PROCESS stimuli;
END struct;
```



Erzeugung von Stimuli mit Procedure

```
if cmd = string'("rst_sim") then
        rst_sim(tv, key_0);

elsif cmd = string'("uar_sim") then
        uar_sim(tv, gpio_26);

end if;
```

Report

Definition

Mit Report kann eine Meldung auf der
Simulator Konsole ausgegeben werden und die Simulation
gegebenenfalls abgebrochen werden

```
report "string" severity "severity_level";
```

Beispiel

```
report "diese Meldung ist ein Hinweis" severity note;
report "Hier ist etwas schlimmes geschehen" severity failure;
```

Verschiedene Möglichkeiten für *severity*:
«note», «error» or «warning» druckt nur eine Nachricht auf die Konsole
«failure» gibt eine Nachricht auf die Konsole und stoppt die Simulation

Assert und Report

Definition

Prüft eine “Bedingung” auf “true” oder “false”. Ist der Wert *falsch*, ist die Bedingung erfüllt und der “report” string wird ausgegeben.

```
assert "bedingung" report "string" severity "severity_level";
```

«note», «error»,«warning», «failure» geben Nachricht auf Konsole aus.
«failure» gibt Nachricht aus und stoppt die Simulation

Beispiele

```
assert (a > c) report "a muss grösser c sein" severity note;

assert (true)  report "diese Meldung wird nie ausgegeben" severity note;

assert (false) report "diese Meldung wird immer ausgegeben" severity note;
```

Erstellung von Verifikationstests

Die Testbench, welche in den folgenden Folien im Detail erklärt wird, wird mit dem emacs Editor automatisch erzeugt. Die Testbench muss nur noch mit Test Procedures ergänzt werden.

1. **Automatische Erstellung der Testbench**
2. **Einbauen von passenden, fertig vorbereiteten Test-Procedures oder Möglichkeit selbst Test-Procedures zu erstellen**
3. **Erstellen des Test-Scripts testcase.dat**
4. **Ausführen der Tests**

Testbench VHDL Code

Libraries

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.all;
use std.textio.all;
use work.simulation_pkg.all;
use work.standard_driver_pkg.all;
use work.user_driver_pkg.all;
```

Procedures und Data Types
benutzt in Testbench

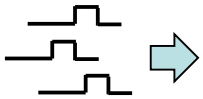
Mitgegebene Treiber
Für Synthesizer Projekt

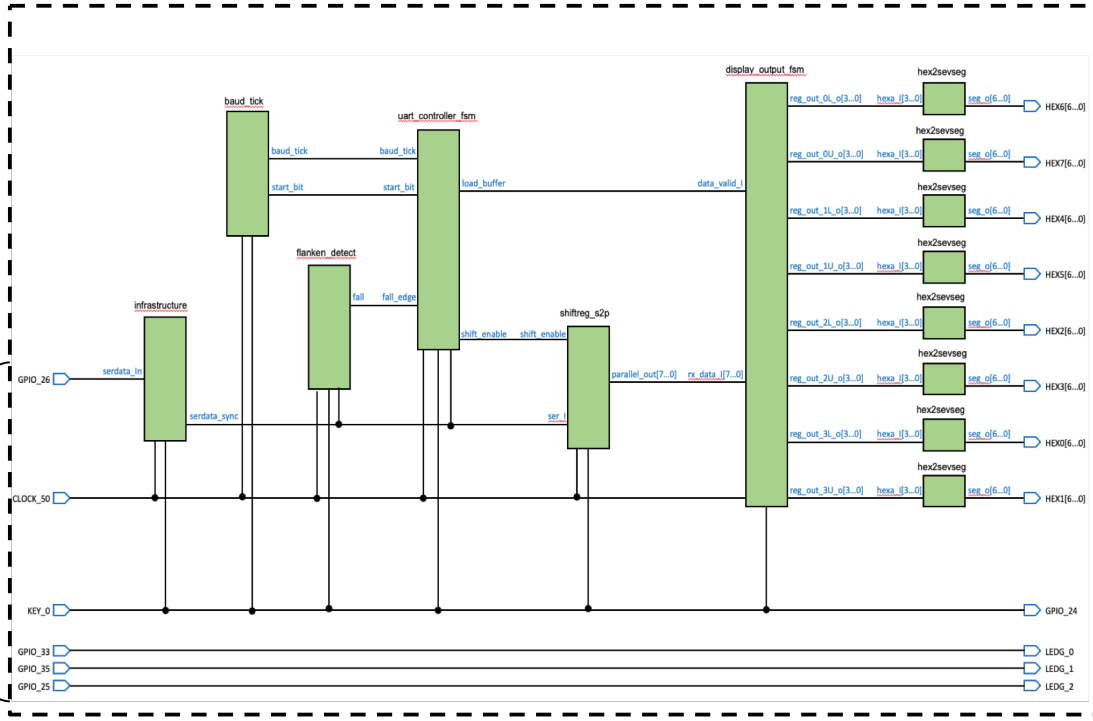
Für selbst erstellte
oder abgewandelte Treiber

Testbench eine Stufe über Top_Level

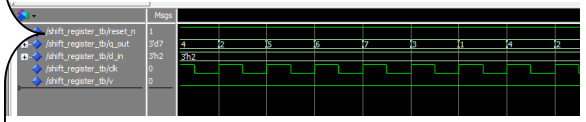
uart_top_tb

Stimuli





Graphische Ausgabe der Simulation

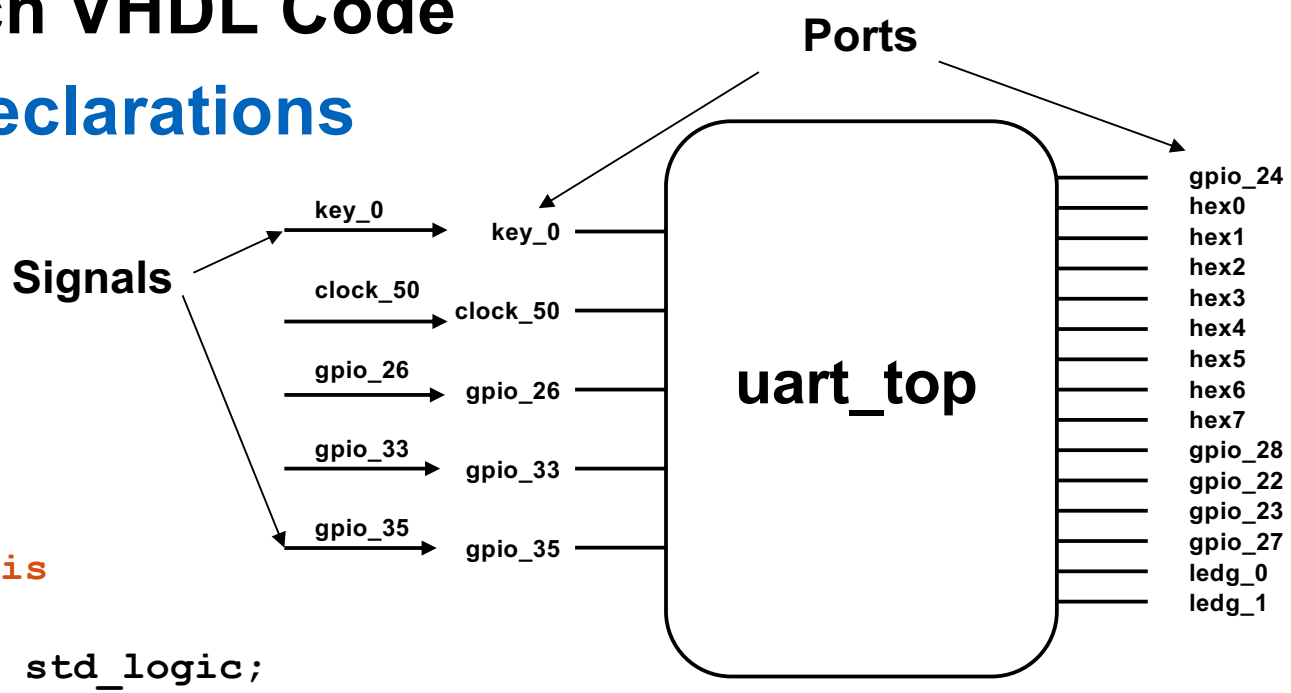


```
ENTITY uart_top_tb IS

END uart_top_tb;
```

Eine Testbench hat keine I/O

Testbench VHDL Code

Signal Declarations




```
component uart_top is
   port (
     clock_50 : in  std_logic;
     gpio_26  : in  std_logic;
     key_0    : in  std_logic;
     hex0     : out std_logic_vector(6 downto 0);
     …
     );
 end component uart_top;

architecture struct of uart_top_tb is
  signal clock_50 : std_logic;
  signal gpio_26  : std_logic;
  signal key_0    : std_logic;
  signal hex0     : std_logic_vector(6 downto 0);
```

Da eine Testbench keine I/Os hat, wird per DUT port ein Signal benötigt.

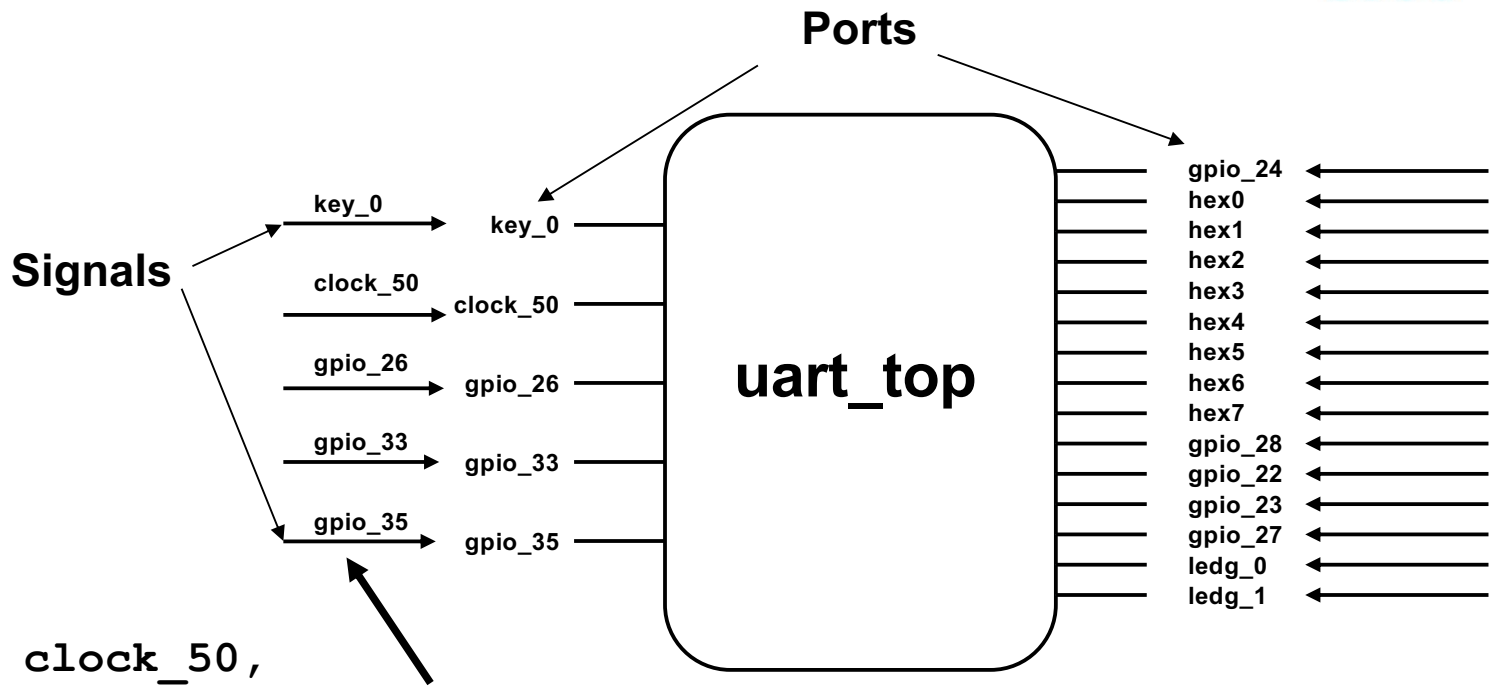
Testbench VHDL Code

Declaration of Clock

```
constant clock_freq   : natural := 50_000_000;
constant clock_period : time    := 1000 ms/clock_freq;
```

Testbench VHDL Code

DUT Instanziation




```
DUT: uart_top
   port map (
     clock_50 => clock_50,
     gpio_25  => gpio_25,
     gpio_26  => gpio_26,
     gpio_33  => gpio_33,
     gpio_35  => gpio_35,
     key_0    => ...
```

Verbindungsignale
DUT zu Testbench

Zürcher Hochschule
für Angewandte Wissenschaften

Testbench VHDL Code

Process zum Verarbeiten von testcase.dat wird von emacs automatisch generiert

```
readcmd : process

 variable cmd         : string(1 to 7); --stores test command
 variable line_in     : line;           --stores to be processed line
 variable tv          : test_vect;      --stores arguments 1 to 4
 variable lincnt      : integer := 0;--counts line number in testcase.dat
 variable fail_counter : integer := 0; --counts failed tests

begin
    -- Open the Input and output files
    FILE_OPEN(cmdfile, "../testcase.dat", read_mode);
    FILE_OPEN(outfile, "../results.dat", write_mode);

    -- Start the loop
    loop
```

Testbench VHDL Code

Einlesen der Kommandozeile
Wird von emacs automatisch generiert

```
-- Check for end of test file and print out results at the end

    if endfile(cmdfile) then    -- Checks end of command file
      end_simulation(fail_counter); -- Prints Results to console
      exit;
    end if;

-- Read all the argumnents and commands

    readline(cmdfile, line_in);
    lincnt := lincnt + 1;
    next when line_in'length = 0;      -- Skip empty lines
    next when line_in.all(1) = '#';    -- Skip lines starting with #
                                       -- for comment lines
```

Fülle Variable tv mit Argumenten der Zeile

Fülle Variable cmd mit Kommando

```
    read_arguments(tv, line_in, cmd);
    tv.clock_period := clock_period;--set Clock period for driver calls
```

Testbench VHDL Code

Aufruf der Testroutinen (Testbench spezifisch)

```
if cmd = string'("rst_sim") then
  rst_sim(tv, key_0);

elsif cmd = string'("uar_sim") then
  uar_sim(tv, gpio_26);

elsif cmd = string'("uar_ch0") then
  uar_chk(tv, hex0);

elsif cmd = string'("uar_ch1") then
  uar_chk(tv, hex1);

elsif cmd = string'("uar_ch2") then
  uar_chk(tv, hex2);

else
  assert false
    report "Unknown Command" severity failure; --
end if;
```

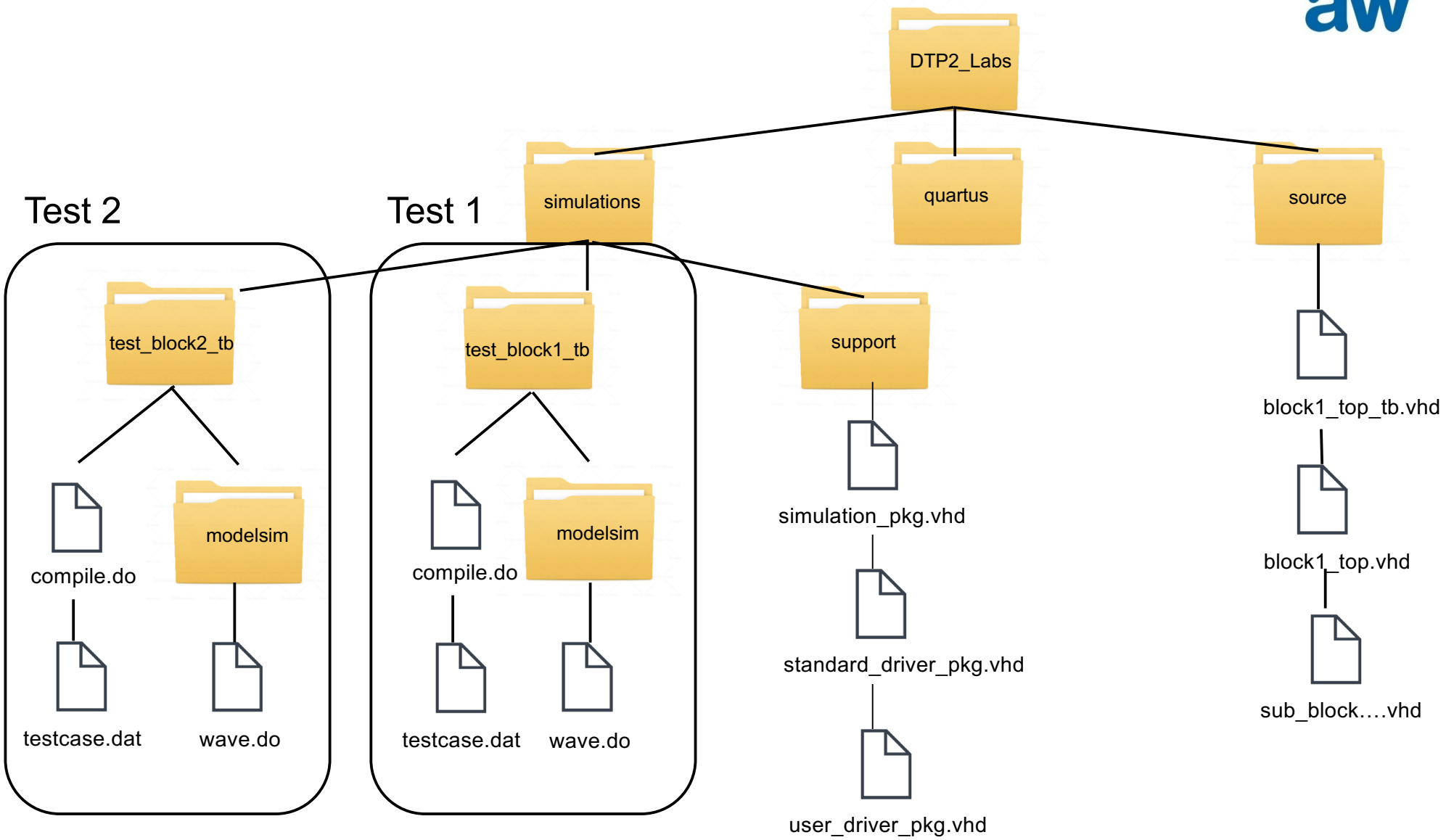
Eine Reset Procedure gibt es für jeden Test

Beim check des Hex-Displays wird wiederholt die gleiche Procedure aufgerufen, aber die getesteten Pins ändern sich

Testbench VHDL Code

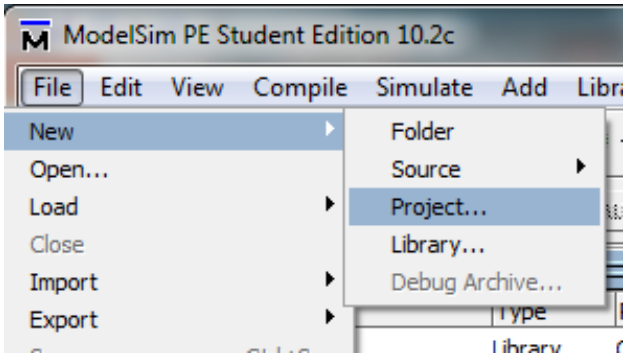
Beende Verarbeitung der Command-Line
Wird von emacs automatisch generiert

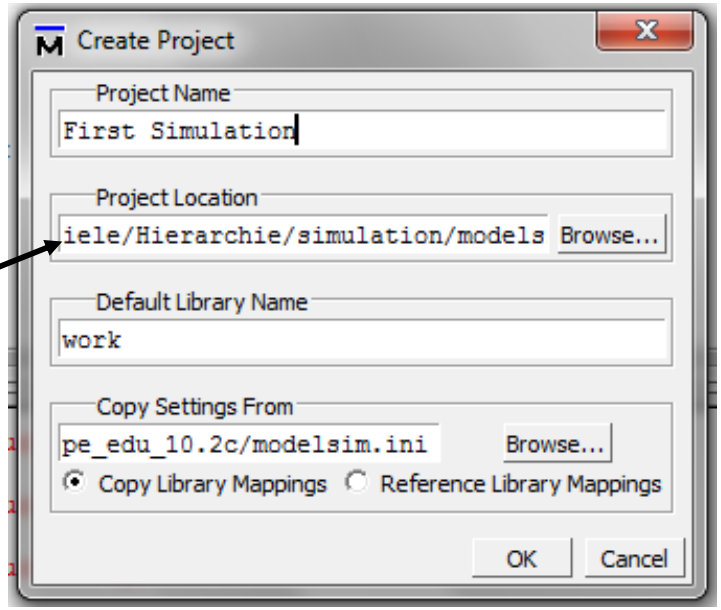
```
  if tv.fail_flag = true then --count failures in tests
        fail_counter := fail_counter + 1;
  else fail_counter := fail_counter;
  end if;

 end loop; --finished processing command line

  wait; --to avoid infinite loop simulator warning

end process;
```


DTP2_Labs
simulations
quartus
source
Test 2
Test 1
test_block2_tb
test_block1_tb
support
compile.do
modelsim
testcase.dat
wave.do
compile.do
modelsim
testcase.dat
wave.do
simulation_pkg.vhd
standard_driver_pkg.vhd
user_driver_pkg.vhd
block1_top_tb.vhd
block1_top.vhd
sub_block….vhd

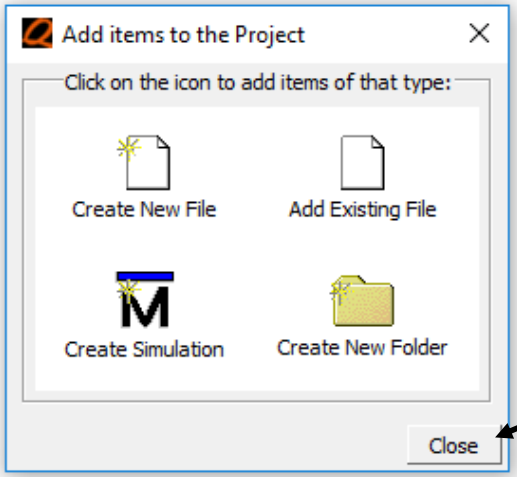
Simulator Starten

Creating a new Project



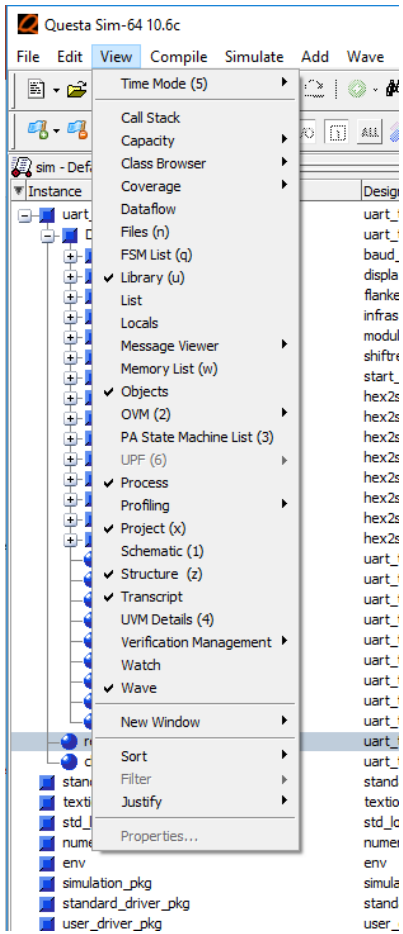


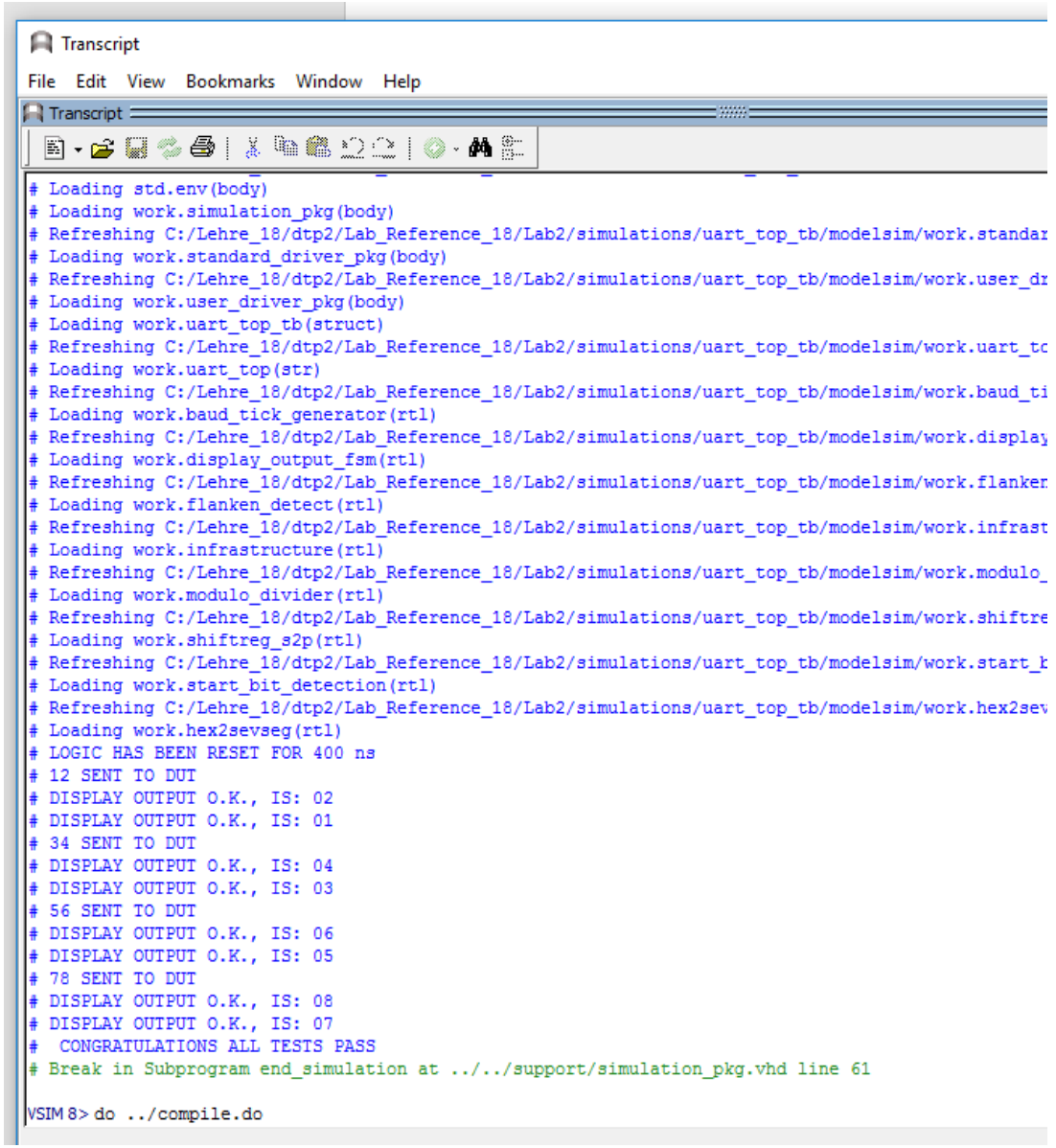
Arbeitsverzeichnis ins Projektverzeichnis legen



Hier nur «close» drücken, keine Dateien hinzufügen

Transcript Window



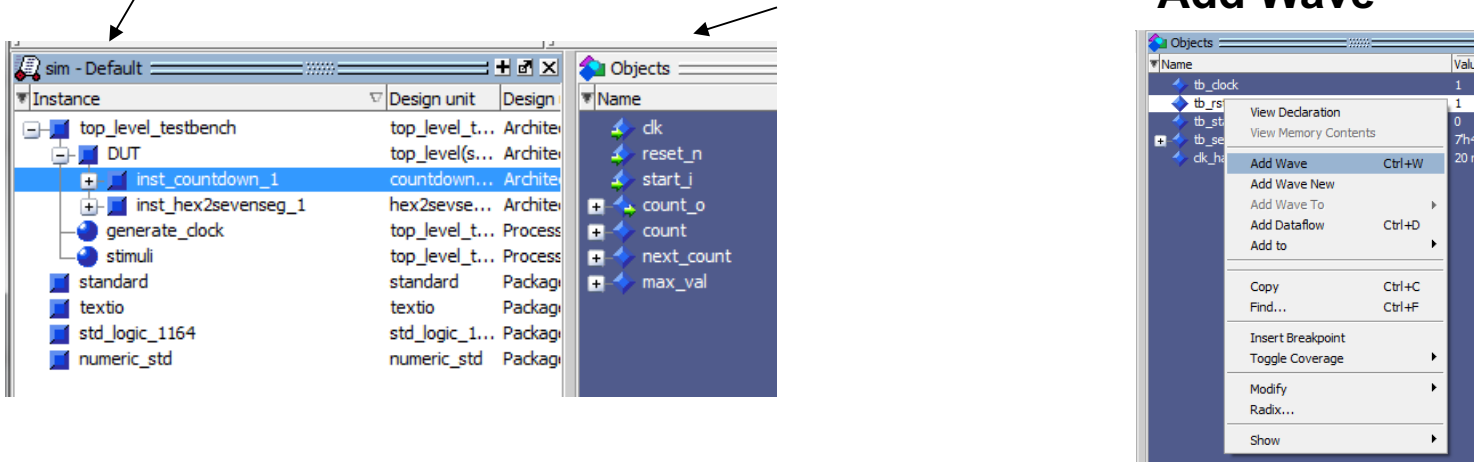


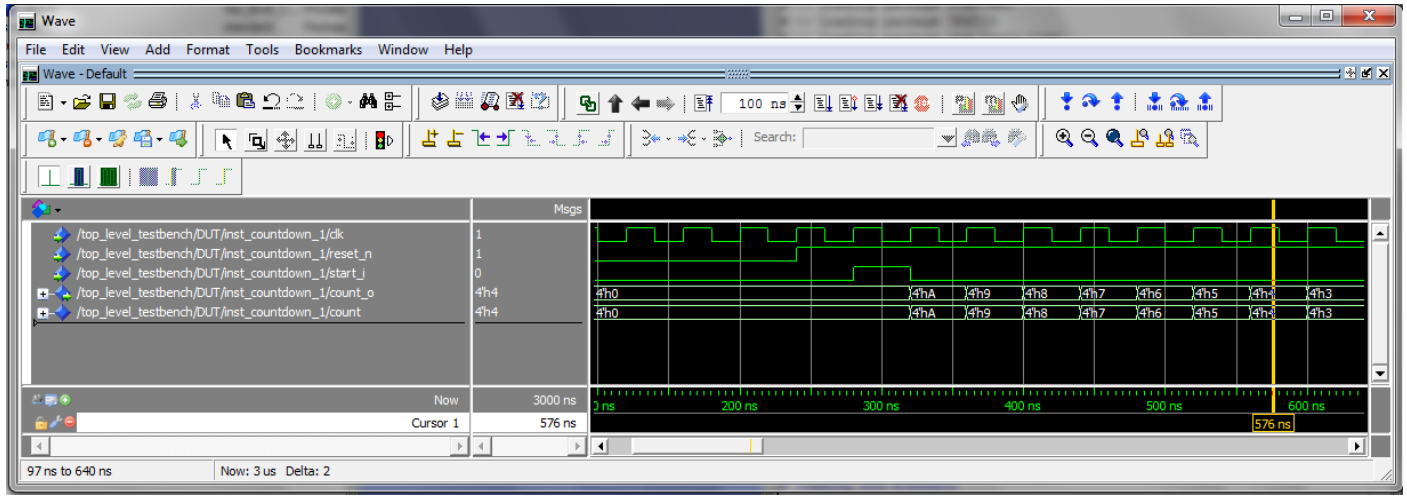
Waveform Window

Structure Window
(Design Hierarchie Browser)

Objects Window
(zu simulierende Signale)

Add Wave





Library Window