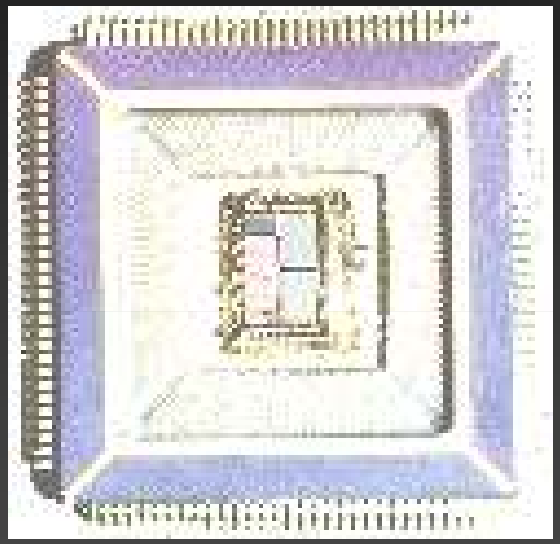


Verifikation

ECS Group, TU Wien

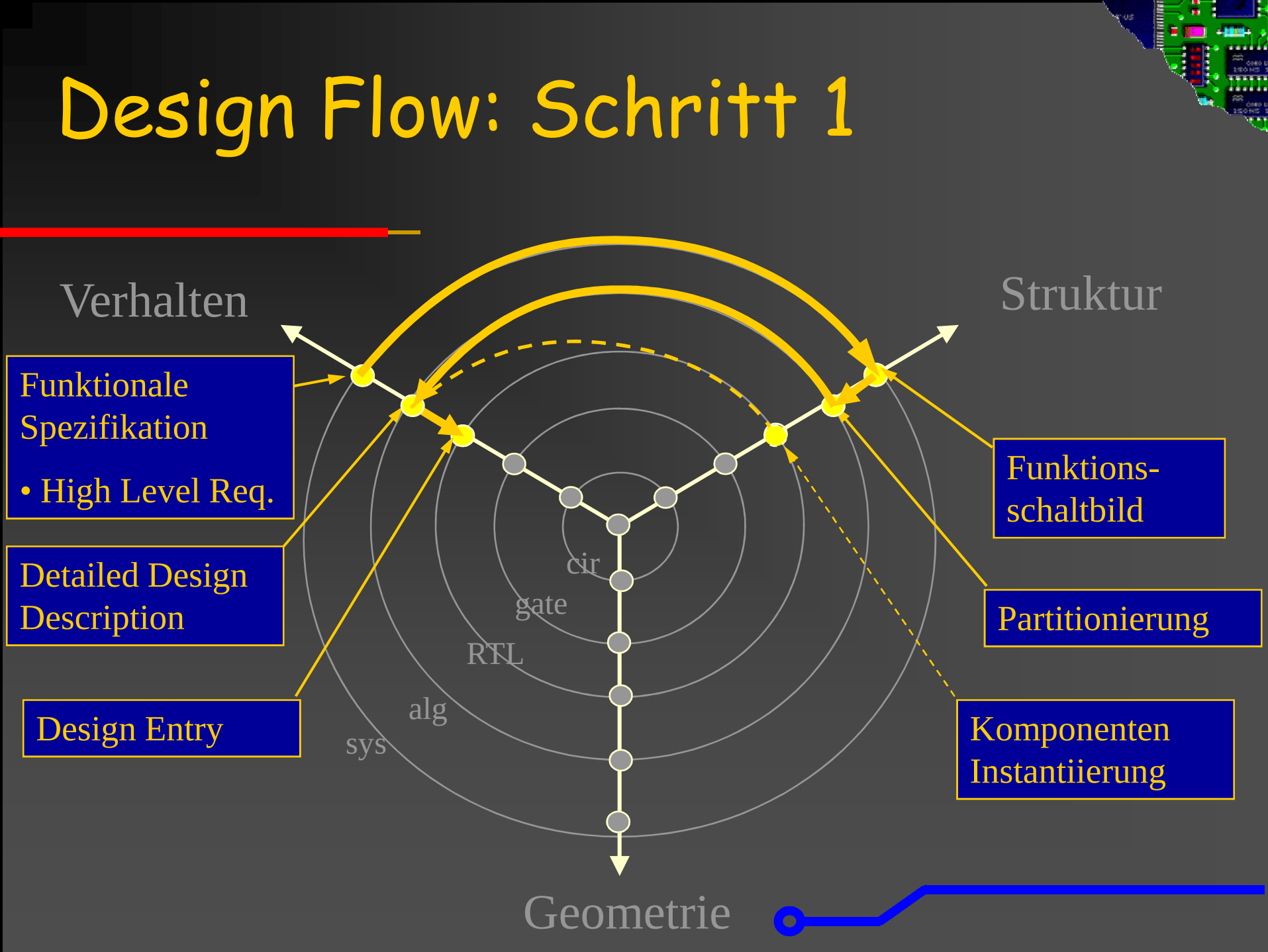
Überblick

- Abstraktionsebenen
- Testbench
- Modelsim

Überblick Hardware Modeling

- Hardware Specification
  - Functional Specification
  - High Level Requirements
  - Detailed Design Description
- Realisation
  - Hardware Description
  - Hardware Implementation
- Verification
  - Review
  - Formal Verification
  - Simulation

Design Flow: Schritt 1



High Level Simulationsmodelle

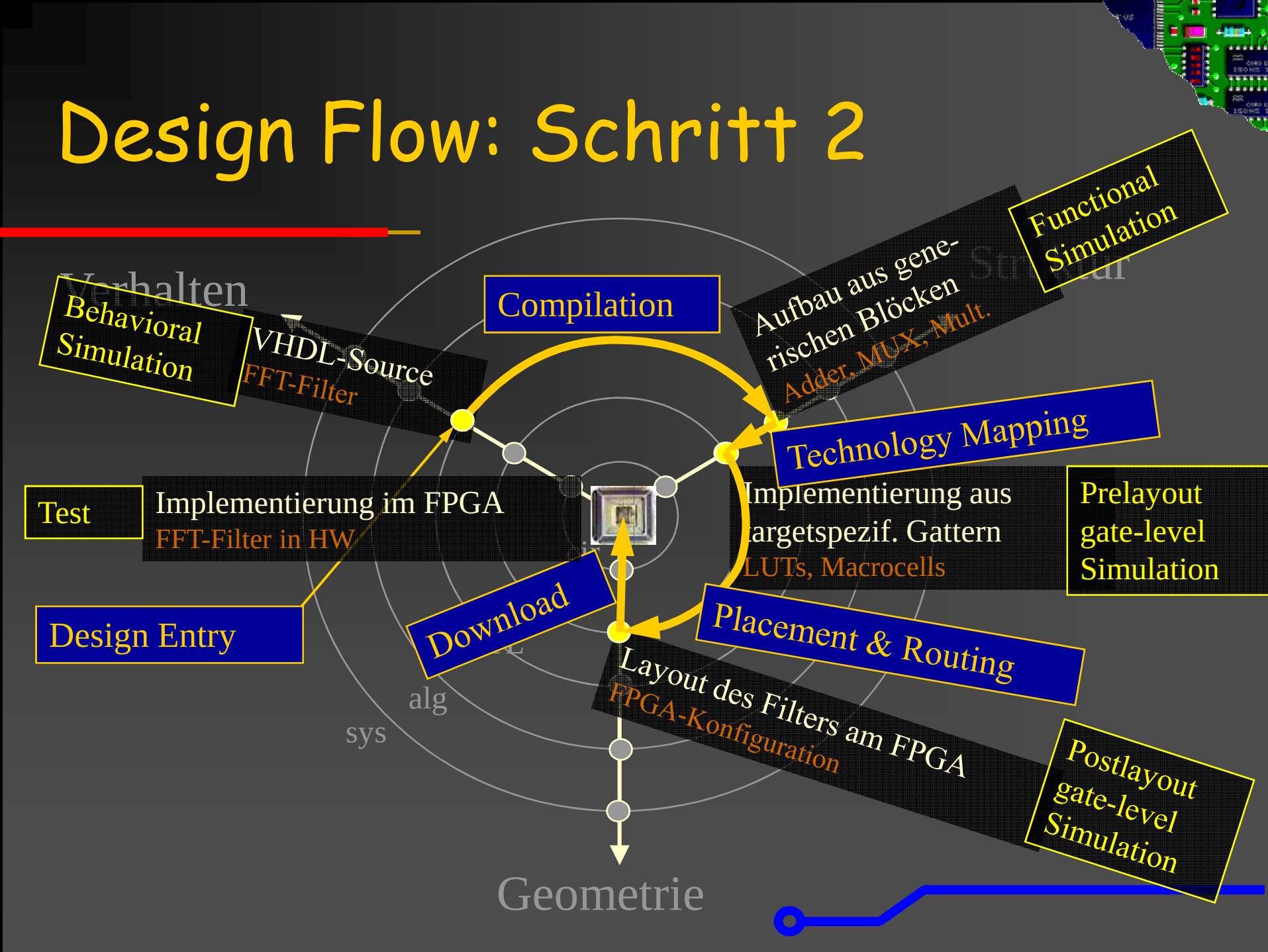
- Functional Model
  - Untimed : Absolut keine Timinginformation
  - Timed : Grobes Timing enthalten, z.B. Eventabfolge
- Transaction Level Model (TLM)
  - Bus Cycle Accurate: Beschreibt die Kommunikation mit Timing
  - Pin Cycle Accurate: Beschreibt das Interface
- Register Tranfer Level (RTL)
  - Alles ist Clockzyklen genau modelliert

=> Beachte Unterschied Verifikation und Model Checking

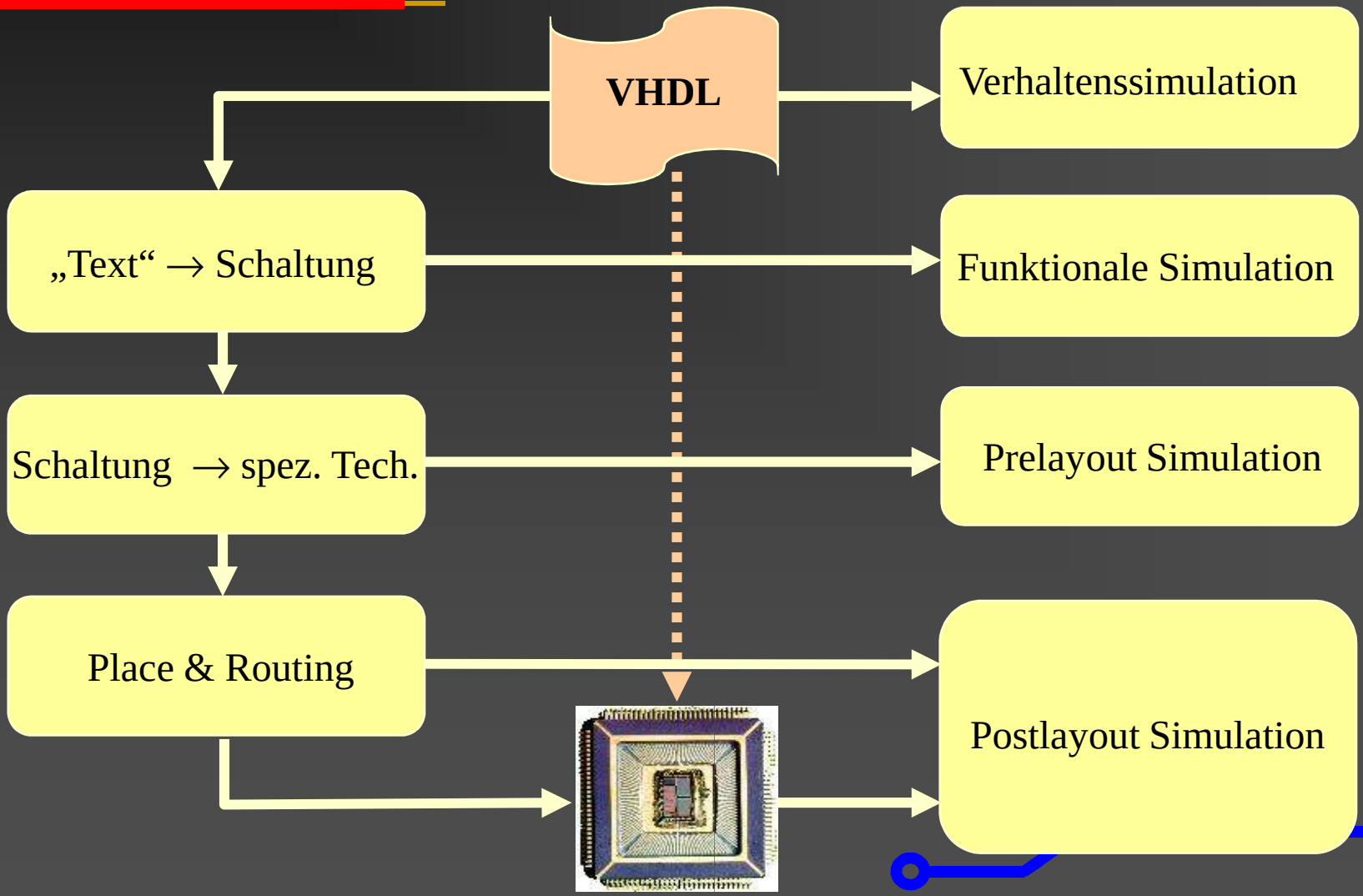
SystemC vs. VHDL

- SystemC ist eine Erweiterung von C/C++ mit Hardware-Libraries
- Hardwarebeschreibung auf algorithmischer Ebene
- Erlaubt schnelles Erstellen von Systemmodellen und erste Evaluation
- Standard bietet TLM an
- Notwendige Tools für Simulation: *GCC*
- Modelsim: Co-Simulation (SystemC und VHDL) möglich
- Nur Subset synthetisierbar

Design Flow: Schritt 2



Design Flow: Schritt 2



Simulationen

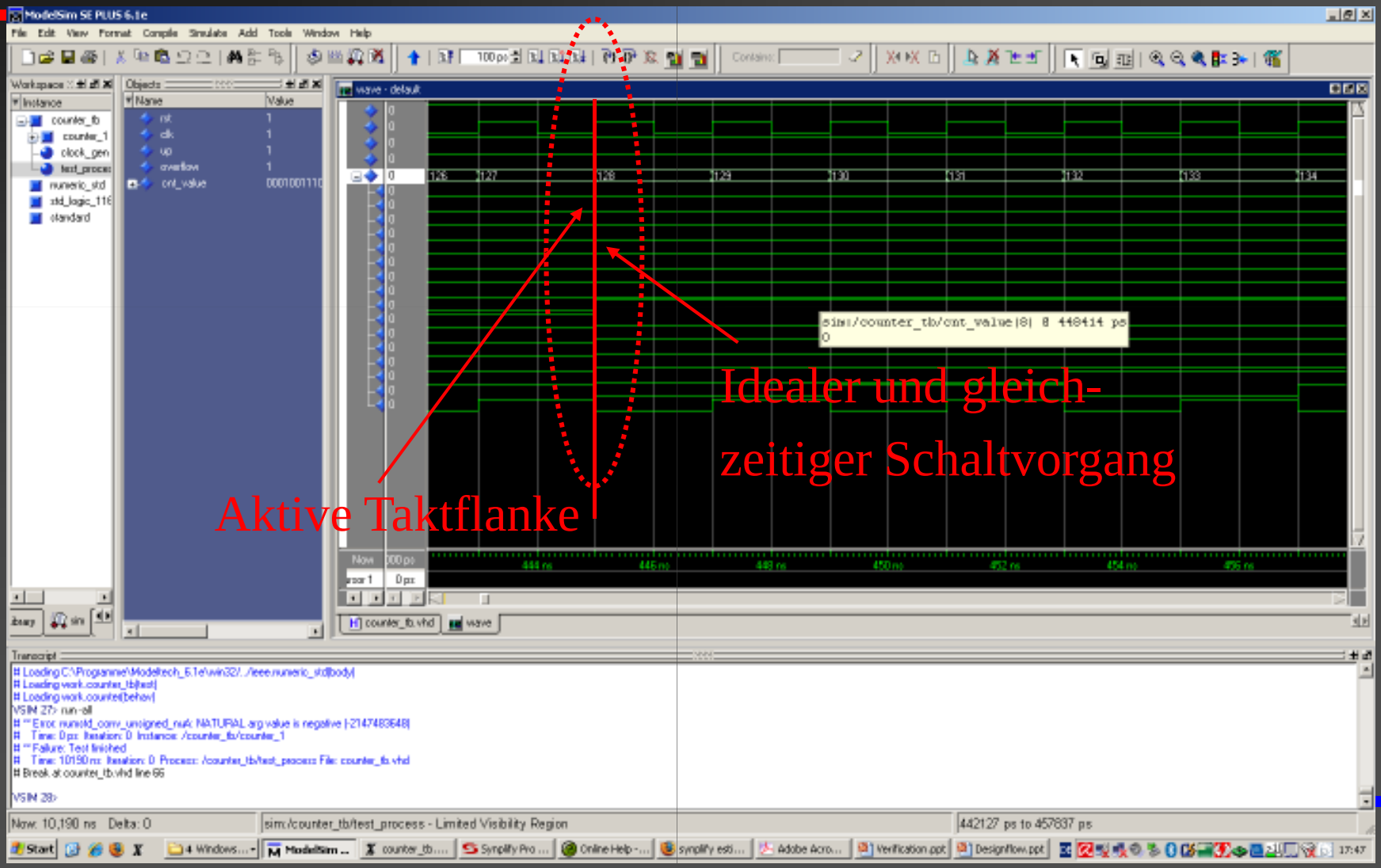
- Verhaltenssimulation
  - reine Funktionalität – keine Timinginformationen
  - VHDL-Code muss nicht synthetisierbar sein
- Funktionale Simulation
  - Darstellung mit generischen Gattern
  - Verifikation mit Gattern
- Prelayout Simulation
  - Technologiespezifische Gatter
  - Optional Unit-Delays
- Postlayout Simulation
  - Nach Place&Route mit echtem Timing

Beispiel: Counter

```
reg: process (clk, rst)
 begin
   if rst = '0' then
    cnt_value_int <= 0;
   elsif clk'event and clk = '1' then
    cnt_value_int <= cnt_value_nxt;
   end if;
 end process reg;

 cnt_nxt: process (up, cnt_value_int)
 begin
   cnt_value_nxt <= cnt_value_int;
   if up ='1'  then
     cnt_value_nxt <= cnt_value_int +1;
   end if;
 end process cnt_nxt;
```

Beispiel: Verhaltenssimulation



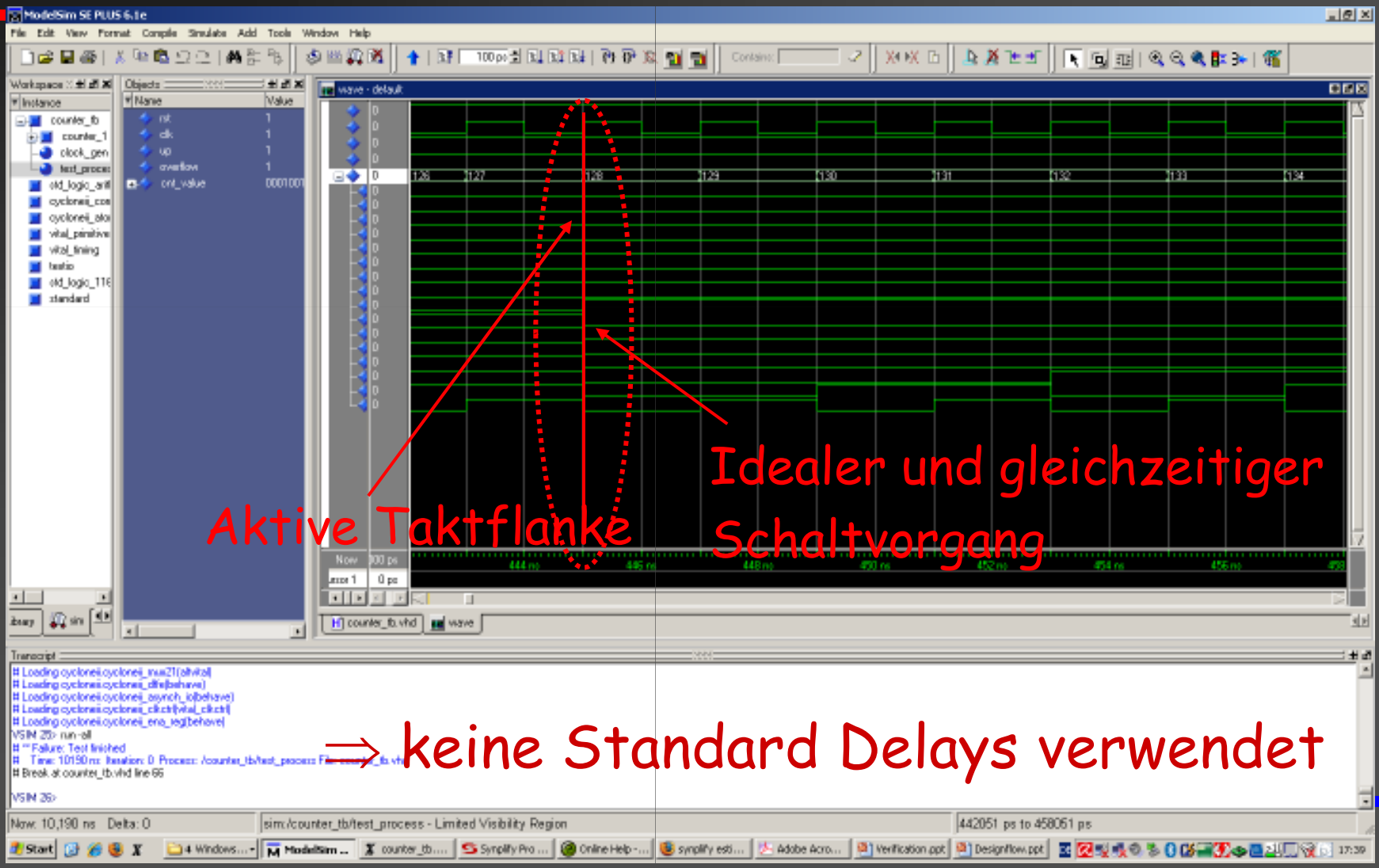
Beispiel: Funktionale Simulation

Synplify generiert keine VHDL
Beschreibung der generischen Hardware

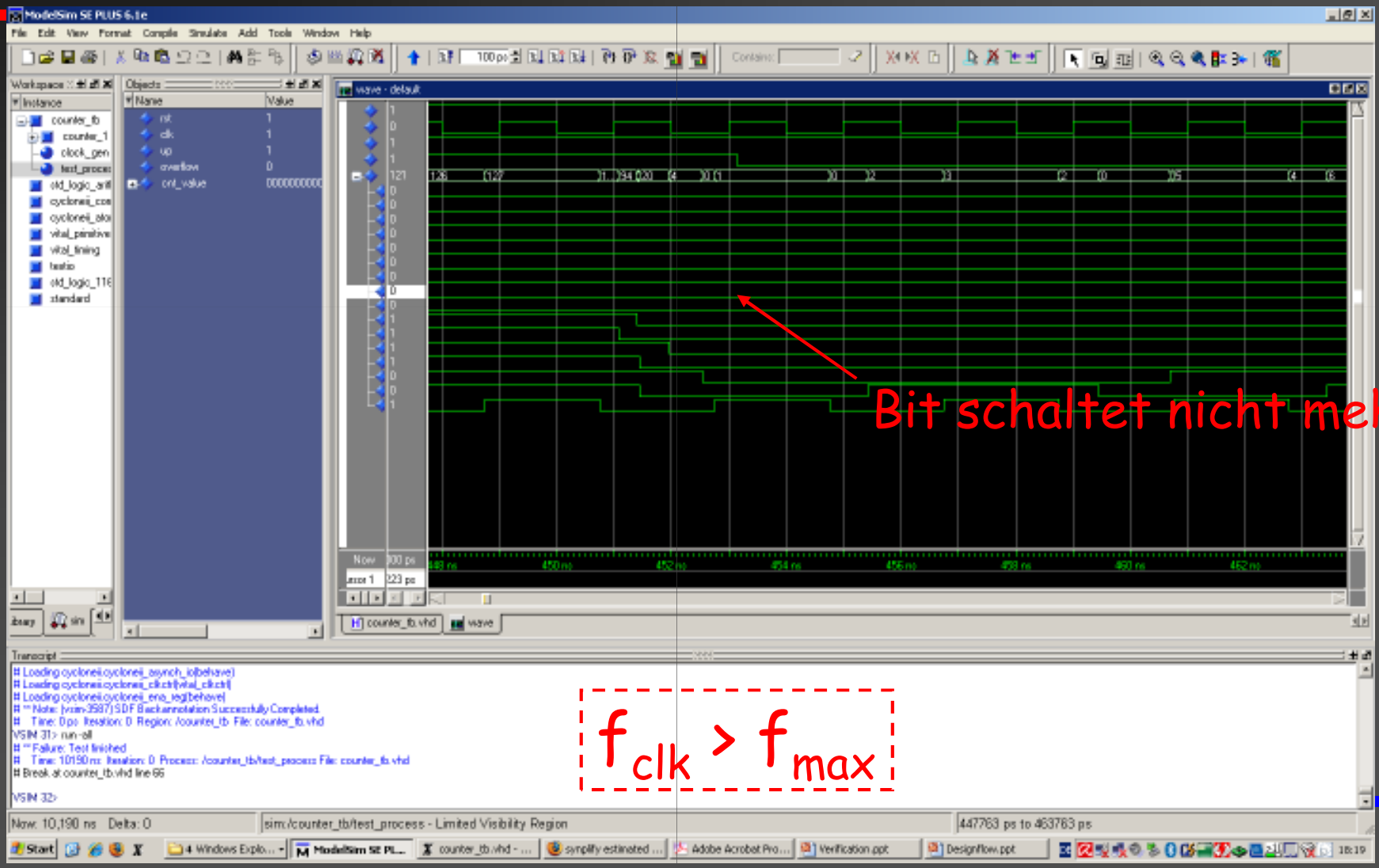
☹

Funktionale Simulation nicht
durchführbar

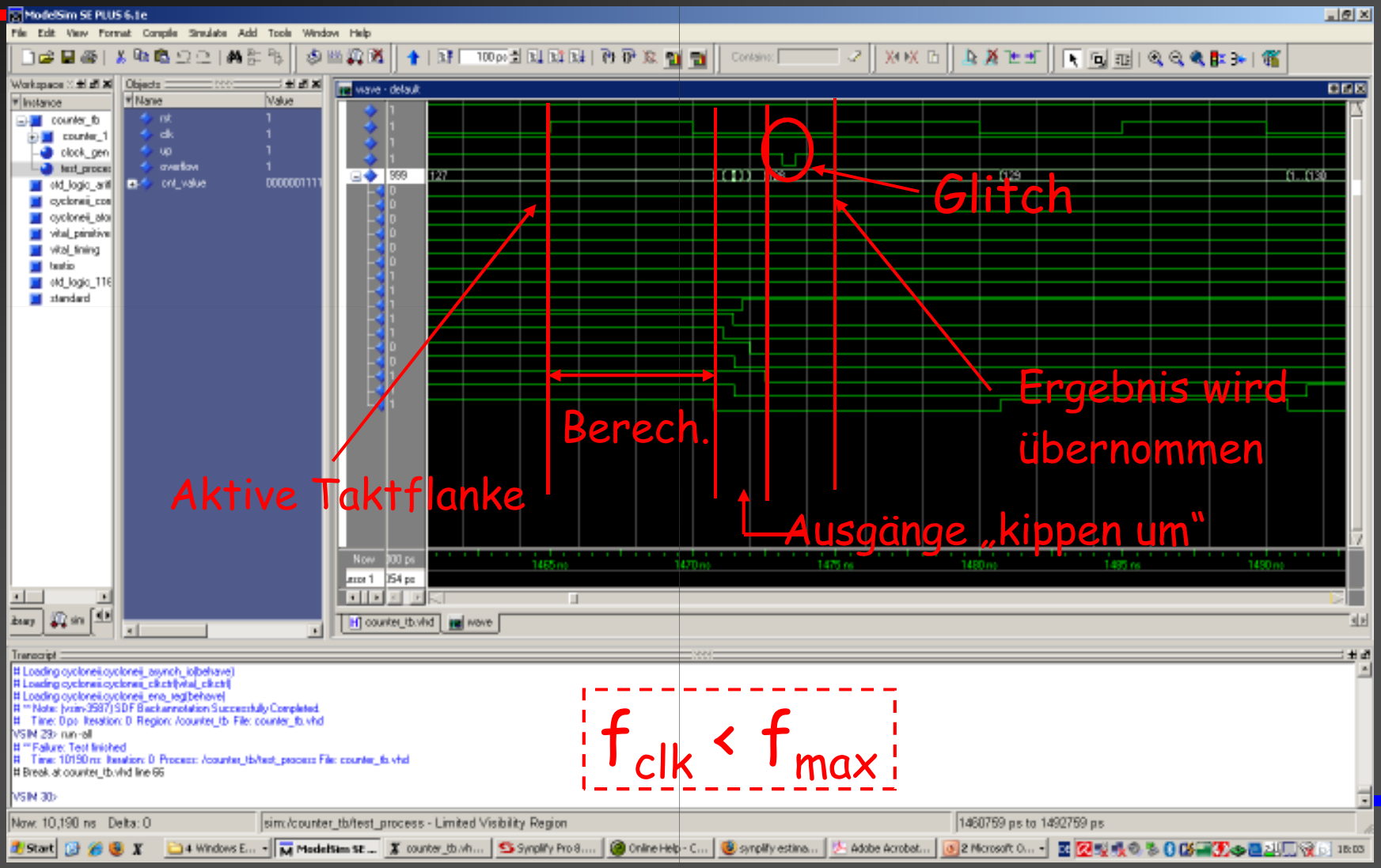
Beispiel: Prelayout Simulation



Beispiel: Postlayout Simulation (1)



Beispiel: Postlayout Simulation (2)



Postlayout Simulation

- Verzögerungen zu den Ausgängen
  - „Verschiebung“ der Signale
- Automatisierte Testbenchen
  - Abtastzeitpunkt beachten
- Geschwindigkeit der „Teststimuli“ beachten

Verifizieren eines Designs

- Zum Verifizieren eines Designs müssen
  (i) Stimuli angelegt und
  (ii) Antwort protokolliert werden

- Zwei Möglichkeiten
  - Direktes Anlegen und Visualisieren der Signale
    - Simulation mit Quartus oder „Force“Befehl im Modelsim
  - Schreiben einer Testbench

Testbench

- Vorteil
  - Übersichtlich und reproduzierbar
  - Komplexere Stimuli generieren
  - Flexibler in der Auswertung
- Nachteil
  - Erhöhter Aufwand beim Erstellen
  - Auf spezifische Testcase zugeschnitten

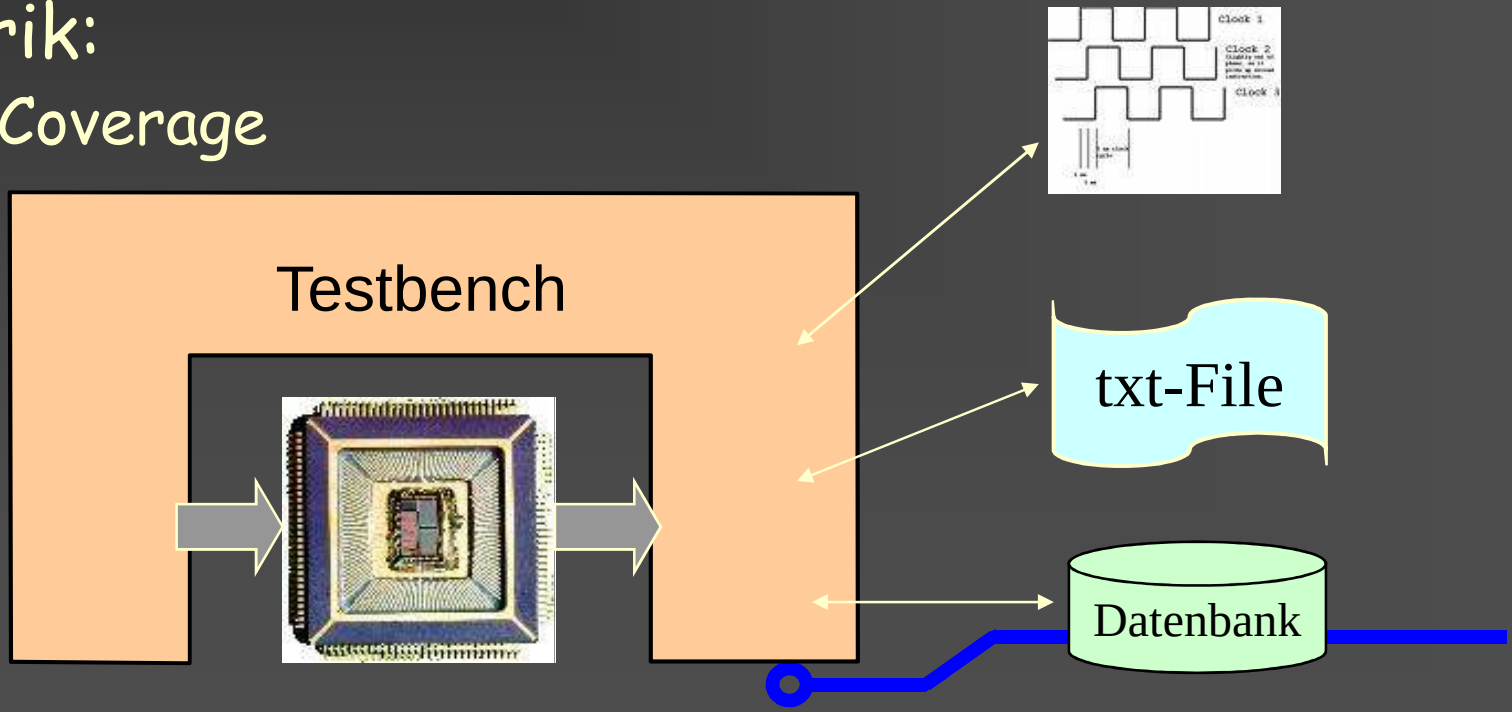
Einfache Testbench

Aufgaben

- Erzeugen von Input Vektoren
- Ausgabe der Reaktion des Device Under Test (DUT)

Metrik:

- Coverage



Testbench (2)

Prelayout Simulation:

- Abstrakte Hardware
- Keine Timinginf.
- Verhalten entspricht dem des realen Chips

Verhaltenss. Funktionale S. Prelayout S. Postlayout S.

Aufbau einer Testbench(1)

- Besteht aus Entity, Architecture und Configuration
- Hat selbst keine Ein- und Ausgänge (ist abgeschlossen)
- In Architecture werden dieTestvektoren der Reihen nach angelegt
- Muss NICHT synthetisierbar sein

Testbench Entity

```
entity xyz_tb is
end entity xyz_tb;
```

⇒ das war‘s schon

Testbench Architecture - Einfach

```
architecture behav of xyz_tb is
    begin

        -- Instantiieren des DUTs

test: process
     begin
        reset <= '1'
        clock <= ,0'
        wait for 10 ns
        clock <= '1'
        -- Applizieren der Stimuli
        wait for 10 ns;
        …

      end process test;
end behav;
```

Testbench Architecture - Hilfsmittel

- Mehrere Prozesse verwenden
  - z.B. ein Prozess zum Generieren eines Clock Signals
- Definition der Taktperiode in einer Konstanten
- Funktion zum Warten für eine bestimmte Anzahl von Clock Zyklen
- Verwenden von for-Schleifen zum Generieren der Stimuli
- Abbruchbedingung mit Assertion

Testbench Beispiel (1)

```
architecture mux1 of hdl_bsp is
  mux : process (A, B, C)

  begin
    if C = '0' then
      Y <= A;
    else
      Y <= B;
    end if;
  end process mux;
end mux1;
```

```
architecture mux2 of hdl_bsp is
  mux : process (A)

  begin  -- process mux1
    if C = '0' then
      Y <= A;
    else
      Y <= B;
    end if;
  end process mux;
end mux2;
```

Testbench Beispiel (2)

```
entity tb_mux is
end tb_mux;

architecture testbench of tb_mux is

  constant clk_period: time:= 100 ns;
  signal CLK : std_ulogic;
  signal A, B, C, Y1 : std_ulogic;
  signal Y2: std_ulogic;

  component hdl_bsp
    port (
      A, B, C : in  std_ulogic;
      Y       : out std_ulogic);
  end component;
```

```
begin

  mux_ok : hdl_bsp
    port map (
      A   => A,
      B   => B,
      C   => C,
      Y   => Y1);

  mux_nosens : hdl_bsp
    port map (
      A   => A,
      B   => B,
      C   => C,
      Y   => Y2);
```

Testbench Beispiel (3)

```
clkgenerator : process
 begin
    CLK <= '0';
    wait for clk_period/2;
    CLK <= '1';
    wait for clk_period/2;
 end process clkgenerator;

 test1 : process
   procedure icwait(cycles: Natural) is
     begin
     for i in 1 to cycles loop
       wait until CLK='1' and CLK'event;
     end loop;
    end ;
```

=> Process zum Generieren des Clock Signal

Sensitivity List fehlt

=> Procedure zum Warten für beliebige Anzahl von Clock Cycles

Testbench Beispiel (4)

```
begin
   A <= '1';
   B <= '0';
   C <= '0';
   icwait(1);
   C <= '1';
   icwait(2);
   B<= '1';
   icwait (1);
   C <= '0';
     ...
   icwait (3);
   assert false
     report "Test finished"
     severity error;
 end process test1;
end testbench;
```

Anlegen der Testsignale

einen Clock Cycle warten

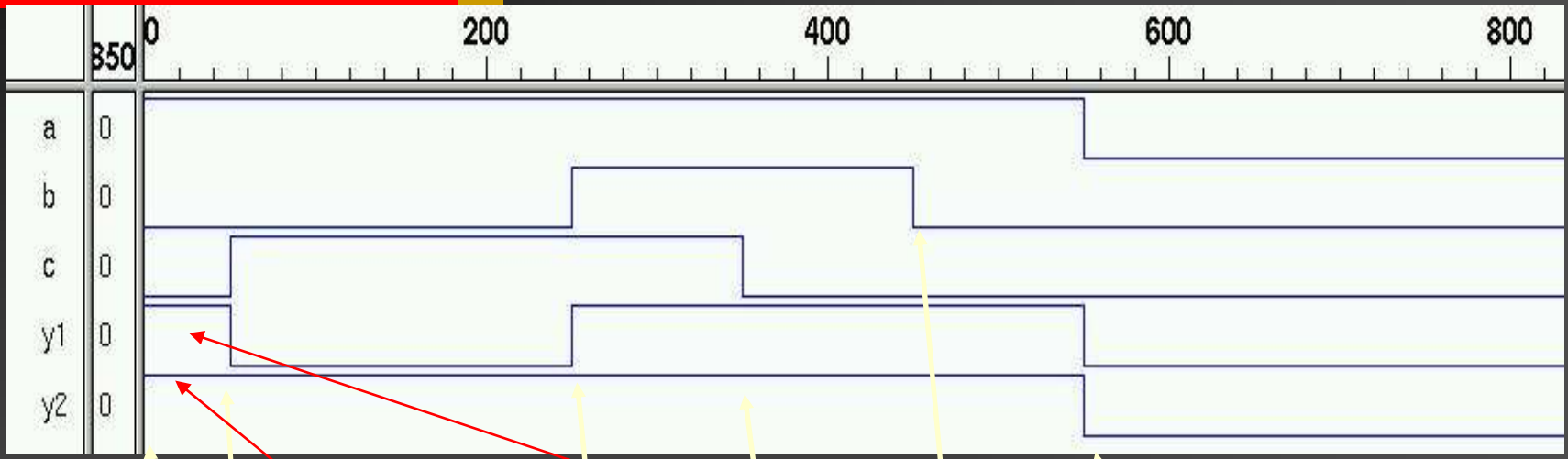
nächstes Testsignal

=> Abbruchbedingung

Testbench Configuration (1)

```
configuration cfg_tb_mux of tb_mux is
  for testbench
    for mux_ok     : hdl_bsp use entity work.hdl_bsp(mux1);
    end for;
    for mux_nosens : hdl_bsp use entity work.hdl_bsp(mux2);
    end for;
  end for;
end cfg_tb_mux;
```

Ergebnis




```
architecture mux2 of hdl_bsp is
 mux : process (A)
begin  -- process mux1
  if C = '0' then
    Y <= A;
  else
    Y <= B;
  end if;
end process mux;
end mux1;
```

```
C <='1';
```

```
A <= '1';
B <= '0';
C <= '0';
```

```
B <= '1';
```

```
architecture mux1 of hdl_bsp is
 mux : process (A, B, C)
begin  -- process mux1
  if C = '0' then
    Y <= A;
  else
    Y <= B;
  end if;
end process mux;
end mux1;
```

```
B <= '0';
```

```
C <= '0';
```

```
A <= '0';
```

Ergebnis

Befehle sind ident !

jedoch

(Verhaltens-)Simulation

verschieden !

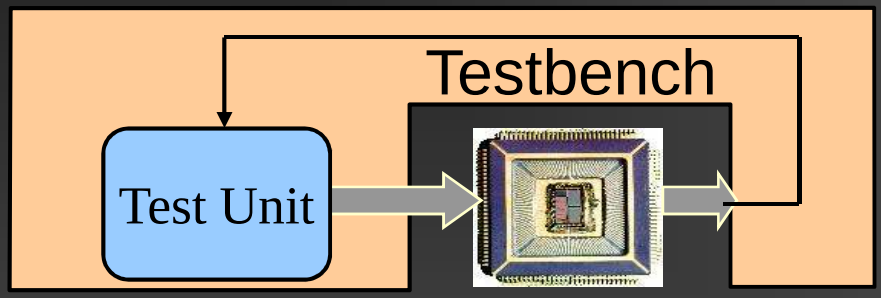
=>

Sensitivity List unterschiedlich

Komplexere Testbenchansätze

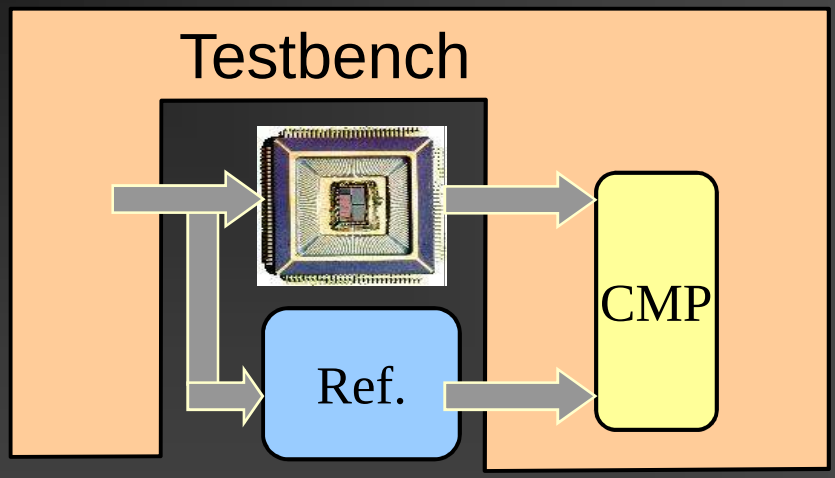
- Zusätzliche Entity zum Generieren der Stimuli
- Automatischer Vergleich mit:
  - Refenenzdaten
  - „Golden Node“
- Lesen und Schreiben der Stimuli von bzw. in Files

Testbench mit Test Unit



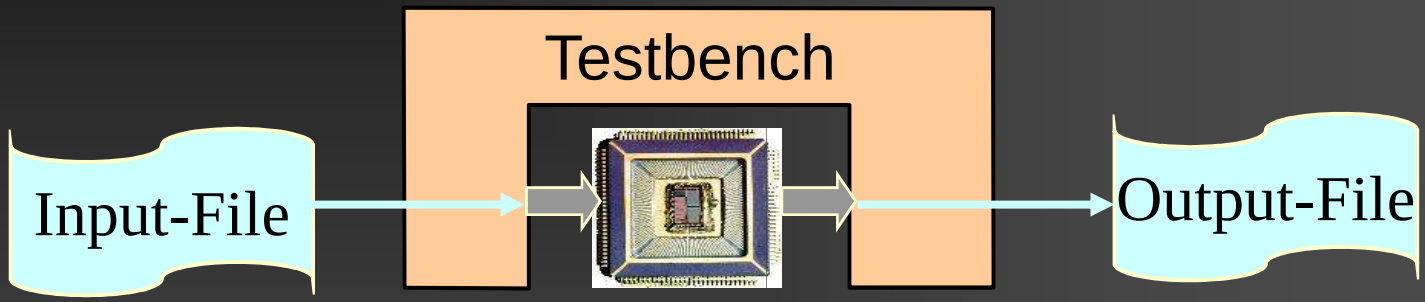
- Test Unit muss nicht synthetisierbar sein
- Kann benutzt werden, um noch nicht implementierte Teile des Designs zu emulieren
- Kann auf Outputs des DUTs reagieren

Automatisierter Vergleich



- Reduzierter Aufwand für Auswertung
- Referenz muss verifiziert sein
- Referenz muss nicht eine Design Unit sein
- Untersuchungen bzgl. Fehlertoleranz geeignet

IO-Files (1)



- Input und Output Files sind normale Text Files
- Input: Kann von der Applikation generiert werden
- Output: Einfachere Analyse im Vergleich zu einem Waveform Output möglich

IO-Files (2)

- Package für Text Ein-Ausgabe
  - “use.std.textio.all”
- Files definieren
  - file file_Stimuli, file_Response: text;
- Zusätzliche notwendige Variablen
  - variable string_Stimuli : string(stimuli'length downto 1);
  - variable line_Stimuli : line;

Daten einlesen

```
FILE_OPEN( file_stimuli, "../Stimuli/input.txt),
  READ_MODE);

while not endfile(file_Stim) loop
    readline(file_Stim, line_Stimuli);
    -- eventl. richtige Anzahl an Zeichen überprüfen
    read(line_Stimuli, string_Stimuli)
   -- Sting Konvertieren in geforderten Datentype
   -- Stimulus anlegen
 end loop;

FILE_CLOSE (file_Stim);
```

Daten schreiben

```
FILE_OPEN( file_Response, "../Stimuli/input.txt),
   WRITE_MODE);

while not test_finished loop
   -- Datentype aufnehmen
   -- Datentype konvertieren in String
   string_Stimuli := stdvec_to_str(output_std_vector);
   write(line_Stimuli, string'(string_Stimuli));
   writeline(file_Response, line_Stimuli);
end loop;

FILE_CLOSE (file_Response);
```

Assertion

- Einfügen von “Check Points” im Design selbst
- Implementieren keine Funktion
- Nicht synthetisierbare Anweisung
- Unterschiedliche “Severity Levels”
- Ausgabe erfolgt im “Report-Window” des Simulators
  - kann in File umgeleitet werden
- Kann zum Abbruch einer Simulation benutzt werden

Coverage

- Metrik zum Bewerten der Qualität des Tests
- Requirement Coverage
  - Gibt es für jedes Requirement einen Test Case ?
  - Meist nicht automatisiert
- Code Coverage
  - Werden alle Teile meines VHDL Codes während des Test ausgeführt ?
  - Tool Support vorhanden

Code Coverage (1)

- Bei der Verhaltenssimulation
- Coverage = Anzahl der Verwendung dieser Codezeile/Abfrage während Simulation
- Mehrere Test Cases:
  - Coverage Statistiken können zusammengelegt werden

Code Coverage (2)

- Wird 100% Coverage nicht erreicht
  - Testvektoren unvollständig
  - Zeile nicht erreichbar → überflüssiger Code

- Code, der NICHT ausgeführt werden soll:
  - Assertions
  - Case Anweisung: *others*-Zweig
  - …

Code Coverage (3)

Coverage off / coverage on Direktive

```
…
counter_nxt <= counter + 1;
assert counter < 12
    report "Counter Overflow" severity error
…
```

```
…
counter_nxt <= counter + 1
-- coverage off
assert counter < 12
   report "Counter Overflow" severity error
-- coverage on
…
```

} Nicht berücksichtigt

Code Coverage Arten

- Statement Coverage
- Expression Coverage
- Path Coverage
- (Entry/Exit Coverage)

Statement Coverage

```
15          architecture mux1 of hdl_bsp is
16
17              begin  -- mux1
18                mux : process (A, B, C, D)
19                begin  -- process mux1
20                  if C = '0' and D='1' then
21                    Y <= A;
22                  else
23                    Y <= B;
24                  end if;
25                end process mux;
26          end mux1;
```

7
4
3
3
7

Expression Coverage

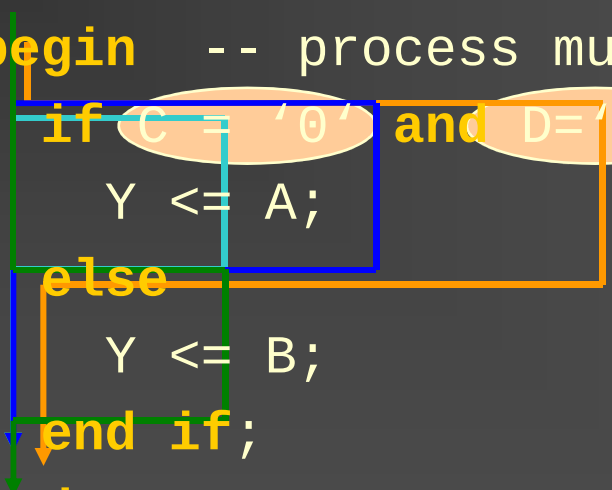
```
15  architecture mux1 of hdl_bsp is
16
17      begin  -- mux1
18        mux : process (A, B, C, D)
19        begin  -- process mux1
20          if                  and                 then
21            Y <= A;
22          else
23            Y <= B;
24          end if;
25        end process mux;
26  end mux1;
```

7
4
3
3
7

Path Coverage

```
15      architecture mux1 of hdl_bsp is
16
17          begin  -- mux1
18            mux : process (A, B, C, D)
19            begin  -- process mux1
20              if C = '0' and D='1' then
21                Y <= A;
22              else
23                Y <= B;
24              end if;
25            end process mux;
26      end mux1;
```

7 4 3 3 7



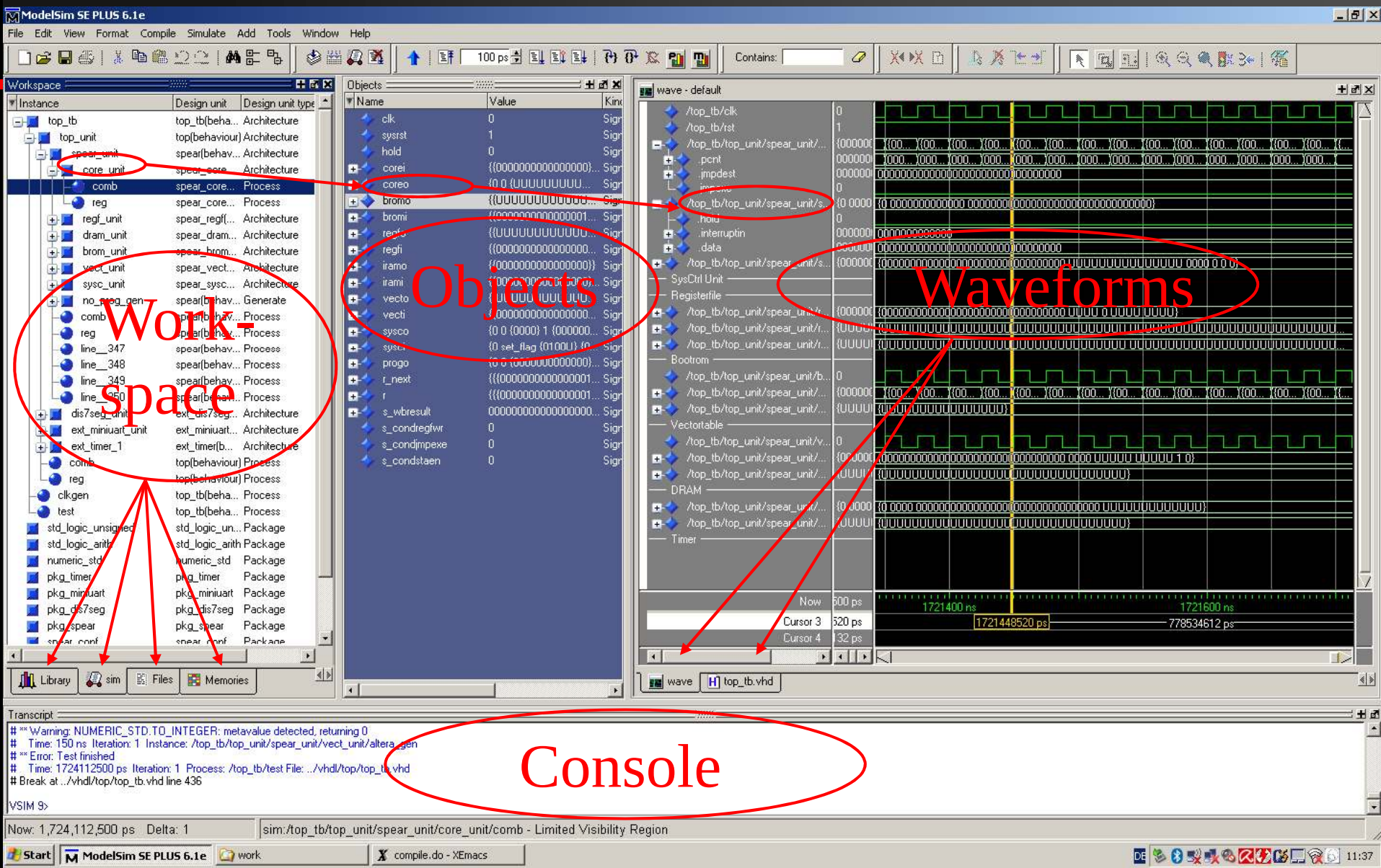
Testbench: Zusammenfassung

- Zur Simulation benötigen wir eine Testbench
- Verschiedene Testbench - Ansätze möglich
- Gleiche Testbench für Verhaltens-, Funktionale, Prelayout- und Postlayout-Simulation verwenden
- Wichtig - **ALLE** Input Signals definieren
- Coverage ist Metrik zur Überprüfung der "Qualität" der Test Cases.

Modelsim

- Unterstützt verschiedene HDL's
  - VHDL, VERILOG, SystemC, ...
- Co-Simulation möglich
- Benutzerschnittstelle:
  - Command line
  - Grafisches Interface
- Kann mit und ohne Testbench verwendet werden

Modelsim GUI



VCOM Befehl

- Übersetzen des VHDL Codes

```
vcom  -work work ../vhdl/counter.vhd
```

Befehl — Ziellibrary — VHDL File

- Argument zum Aktivieren der Code Coverage

  -cover bcest

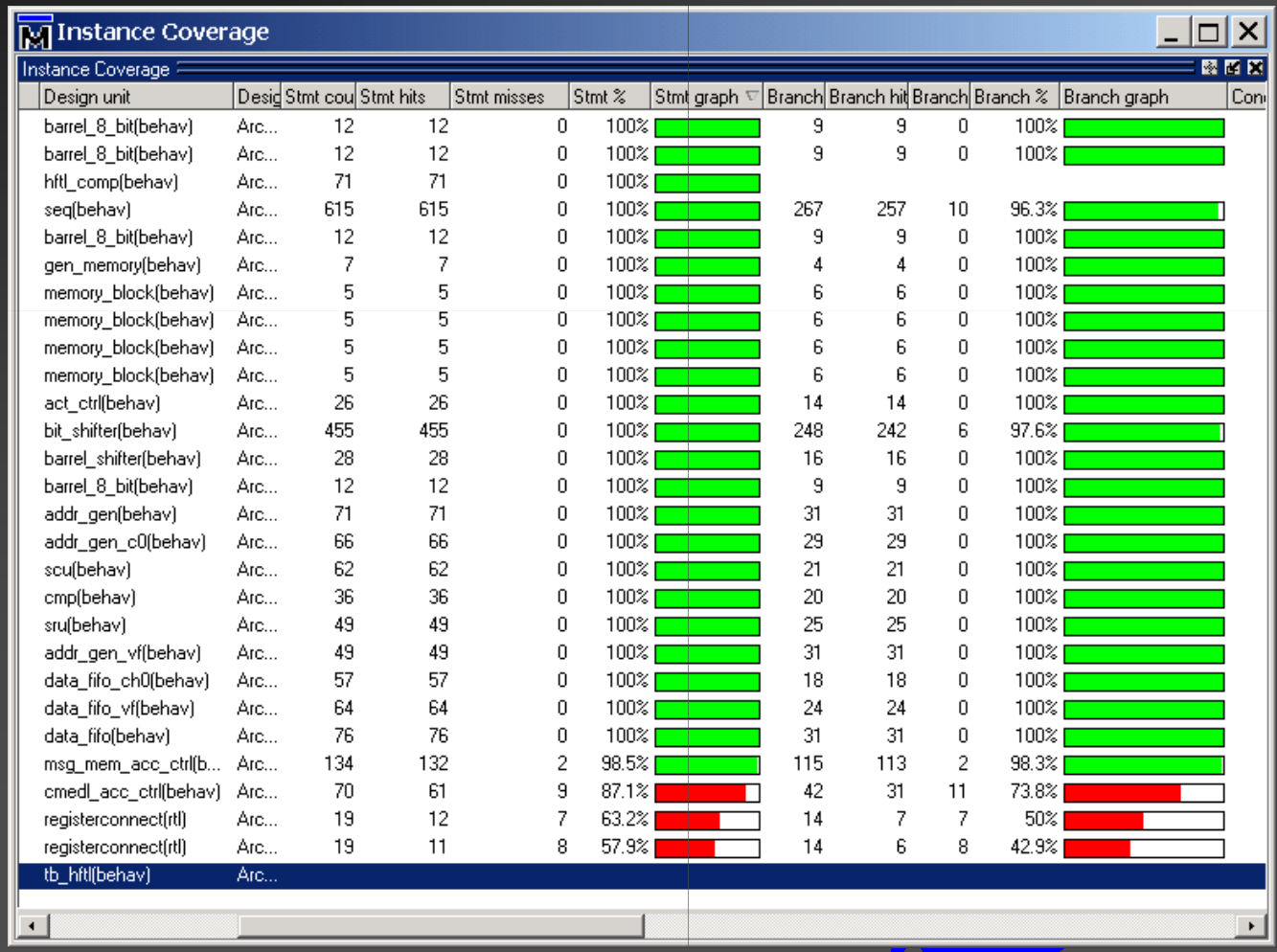
  wobei b .. branch, c .. condition, e expression, s .. statements, t .. toggle

- Mehrere Files → Reihenfolge beachten

VCOM -Weitere Argumente

vcom [-87] [-93] [-2002] [-amsstd | -noamsstd] [-bindAtCompile] [-bindAtLoad]
[-check_synthesis] [-debugVA] [-explicit] [-f <filename>] [-force_refresh
<design_unit>]
[-gen_xml <design_unit> <filename>] [-help] [-ignoredefaultbinding] [-
ignorevitalerrors]
[-just abcep] [-line <number>] [-lint] [-no1164] [-noaccel <package_name>]
[-nocasestaticerror] [-nocheck] [-nodbgsym] [-noindexcheck] [-nologo] [-
nonstddriverinit]
[-noothersstaticerror] [-norangecheck] [-note <msg_number> [,<msg_number>, ...]]
[-novital] [-novitalcheck] [-nowarn <category_number>] [-O0 | -O1 | -O4 | -O5]
[-pedanticerrors] [-performdefaultbinding] [-quiet] [-rangecheck] [-refresh] [-s]
[-skip abcep] [-source] [-time] [-version]
[-suppress <msg_number>[,<msg_number>,...]]
[-error <msg_number>[,<msg_number>,...]]
[-warning <msg_number>[,<msg_number>,...]]
[-fatal <msg_number>[,<msg_number>,...]]
[-work <library_name>] <filename>
[+acc[=<spec>][+<entity>[(architecture)]]] [-novopt] [-vopt]
[-cover <stat>] [-coverAll] [-coverExcludeDefault] [-covernosubs]
[-nocoverGenerate] [-nodebug[=ports]]

Beispiel: Code Coverage Output



Design unit	Desig	Stmt cou	Stmt hits	Stmt misses	Stmt %	Stmt graph	Branch	Branch hit	Branch	Branch %	Branch graph	Con
barrel_8_bit(behav)	Arc...	12	12	0	100%		9	9	0	100%		
barrel_8_bit(behav)	Arc...	12	12	0	100%		9	9	0	100%		
hftl_comp(behav)	Arc...	71	71	0	100%							
seq(behav)	Arc...	615	615	0	100%		267	257	10	96.3%		
barrel_8_bit(behav)	Arc...	12	12	0	100%		9	9	0	100%		
gen_memory(behav)	Arc...	7	7	0	100%		4	4	0	100%		
memory_block(behav)	Arc...	5	5	0	100%		6	6	0	100%		
memory_block(behav)	Arc...	5	5	0	100%		6	6	0	100%		
memory_block(behav)	Arc...	5	5	0	100%		6	6	0	100%		
memory_block(behav)	Arc...	5	5	0	100%		6	6	0	100%		
act_ctrl(behav)	Arc...	26	26	0	100%		14	14	0	100%		
bit_shifter(behav)	Arc...	455	455	0	100%		248	242	6	97.6%		
barrel_shifter(behav)	Arc...	28	28	0	100%		16	16	0	100%		
barrel_8_bit(behav)	Arc...	12	12	0	100%		9	9	0	100%		
addr_gen(behav)	Arc...	71	71	0	100%		31	31	0	100%		
addr_gen_c0(behav)	Arc...	66	66	0	100%		29	29	0	100%		
scu(behav)	Arc...	62	62	0	100%		21	21	0	100%		
cmp(behav)	Arc...	36	36	0	100%		20	20	0	100%		
sru(behav)	Arc...	49	49	0	100%		25	25	0	100%		
addr_gen_vf(behav)	Arc...	49	49	0	100%		31	31	0	100%		
data_fifo_ch0(behav)	Arc...	57	57	0	100%		18	18	0	100%		
data_fifo_vf(behav)	Arc...	64	64	0	100%		24	24	0	100%		
data_fifo(behav)	Arc...	76	76	0	100%		31	31	0	100%		
msg_mem_acc_ctrl(b...	Arc...	134	132	2	98.5%		115	113	2	98.3%		
cmedl_acc_ctrl(behav)	Arc...	70	61	9	87.1%		42	31	11	73.8%		
registerconnect(rtl)	Arc...	19	12	7	63.2%		14	7	7	50%		
registerconnect(rtl)	Arc...	19	11	8	57.9%		14	6	8	42.9%		
tb_hftl(behav)	Arc...											

VLIB und VMAP Befehle

- VLIB erzeugt eine „Design Library“

  vlib altera_mf

- Verbindung der „Design Library“ mit einem physikalischen Pfad

  vmap altera_mf work

VSIM Befehl

- Starten der Simulation

```
vsim -t ps work.counter_tb
```

Befehl — Zeitbasis — VHDL File

- WeitereArgumente
  - -coverage … Aktiviert Coverage für Simulation
  - -assertfile <filename> … leitet Assertion Ausgabe in Datei um
  - -sdfmin,-sdftyp oder -sdfmax … Timinginformationen hinzufügen

VSIM Argumente

vsim [-assertfile <filename>] [-c] [-colormap new] [-display <display_spec>]
[-debugDB=<db_pathname>] [-do "<command_string>" | <macro_file_name>]
[+dumpports+collapse] [+dumpports+direction] [+dumpports+no_strength_range]
[+dumpports+unique] [-f <filename>] [-g<Name>=<Value>…] [-G<Name>=<Value>…]
[-gblso <filename>] [-gui] [-help] [-i] [-installcolormap] [-keeploaded] [-keeploadedrestart]
[-keepstdout] [-l <filename>] [-lib <libname>] [-L <library_name> …]
[-Lf <library_name>…] [-msgmode both | tran | wlf] [-multisource_delay min | max | latest]
[+multisource_int_delays] [-nocompress] [+no_notifier] [+no_tchk_msg]
[+notimingchecks] [-printsimstats] [+pulse_int_e/<percent>] [+pulse_int_r/<percent>]
[-quiet] [+sdf_iopath_to_prim_ok] [-sdfmin | -sdftyp | -sdfmax[@<delayScale>]
[<instance>=]<sdf_filename>] [-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn]
[+sdf_verbose] [-t [<multiplier>]<time_unit>] [-tag <string>] [-title <title>]
[-trace_foreign <int>] [+transport_int_delays] [-vcdstim [<instance>=]<filename>]
[-version] [-view [<dataset_name>=]<WLF_filename>]
[-viewcov [<dataset_name>=]<UCDB_filename>]
[-error <msg_number>[,<msg_number>,…]]
[-fatal <msg_number>[,<msg_number>,…]]
[-note <msg_number>[,<msg_number>,…]]
[-suppress <msg_number>[,<msg_number>,…]]
[-warning <msg_number>[,<msg_number>,…]]
[-wlf <filename>] [-wlfcachesize <n>] [-wlfcollapsedelta] [-wlfcollapsetime]
[-wlfcompress] [-wlfdeleteonquit] [-wlflock] [-wlfnocollapse] [-wlfnocompress]
[-wlfnodeleteonquit] [-wlfnolock] [-wlfnoopt] [-wlfopt] [-wlfsimcachesize <n>]
[-wlfslim <size>] [-wlftlim <duration>]
[-geometry <geometry_spec>] [-name <name>]
[-sync] [-visual <visual>]
[-voptargs="<args>"] [-vopt_verbose] [-vopt | -novopt] [-no_autoacc]
[-elab <filename>] [-elab_cont <filename>] [-elab_defer_fli] [-load_elab <filename>]
[-filemap_elab <HDLfilename>=<NEWfilename>] [-compress_elab] [-restore <filename>]
478 ModelSim SE Reference Manual, v6.3
Commands
[-memprof] [-memprof+call] [-memprof+file=<filename>]
[-memprof+fileonly=<filename>] [-memprof+inst=<inst_name>]
[-autoexclusions=[fsm | none]] [-coverage] [-coverCountNone]
[-dpioutoftheblue 1 | 0] [<license_option>]

⇒ Handbuch !!

ADD WAVE Befehl

- Hinzufügen von Signal im Waveform Fenster

```
add wave  -format logic /counter_tb/clk
```

Befehl — Anzeigeformat — VHDL File

- WeitereArgumente
  - -label clock clk … clk wird zu clock im Waveform W.
  - -divider <divider_name> … führt einen Separator ein
  - -radix … definiert die Darstellung des Signals

FORCE Befehl

- Modelsim kann ohne Testbench verwendet werden
- Signale können mit dem FORCE Befehl direkt getrieben werden

force object value

Befehl — Signalname — Wert

- WeitereArgumente
  - -repeat … Generieren von periodischem Signalverlauf
  - 200 ns … Setzt das Signal nach 200 ns

do- Files

- Modelsim spezifische Skripts
- Files enthalten
  - Modelsim Befehle
  - TCL Programm
- Aufruf:
  - Argument von VSIM
  - Direkt in Console

Beispiel: do-File

```
vlib work
vmap work work
vcom -work work ../vhdl/counter/counter.vhd
vcom -work work ../vhdl/counter/counter_tb.vhd
vsim -t ps work.counter_tb
add wave -noupdate -format logic /counter_tb/clk
add wave -noupdate -format Logic /counter_tb/reset
(force reset 0)
run -all
```

Modelsim: Zusammenfassung

- Umfangreiches Simulationstool für versch. HDLs
- Benutzerinterface: Command Line oder GUI
- Simulation kann mit und ohne Testbench erfolgen
- Simulation kann mit Do-Files automatisiert werden