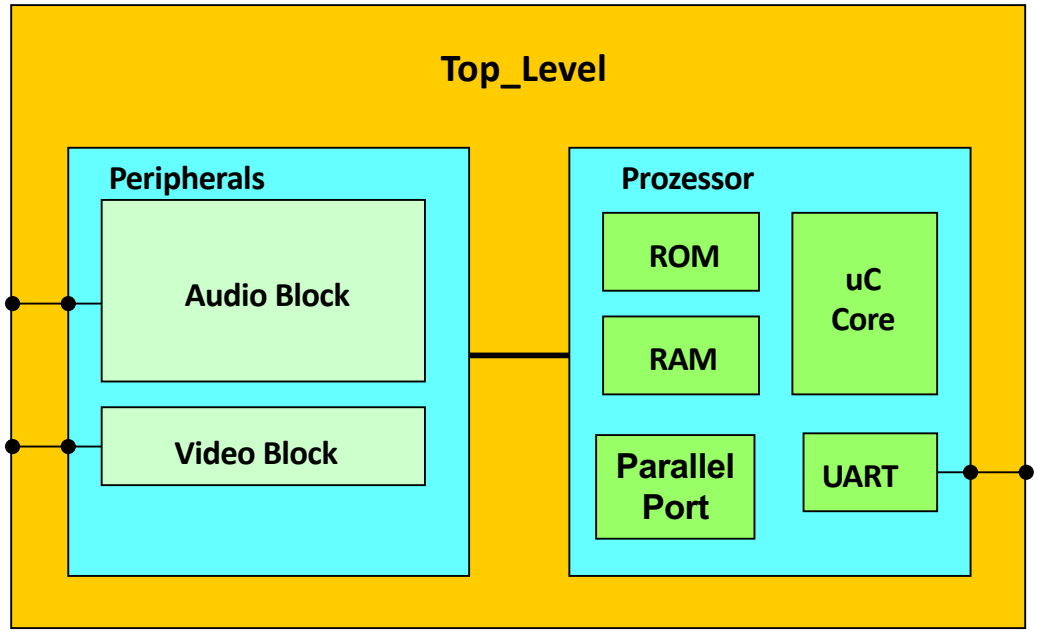
Hierarchisches VHDL Design

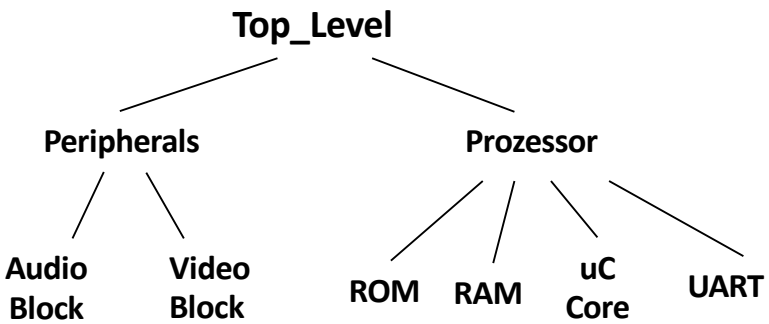
DTP2_Vorlesung_01

- Hierarchisches VHDL Design
  - Component Declaration, Component Instantiation, Port Map
- Generics
- Variablen
- For Loop
- Bedienung des emacs GNU-Editors

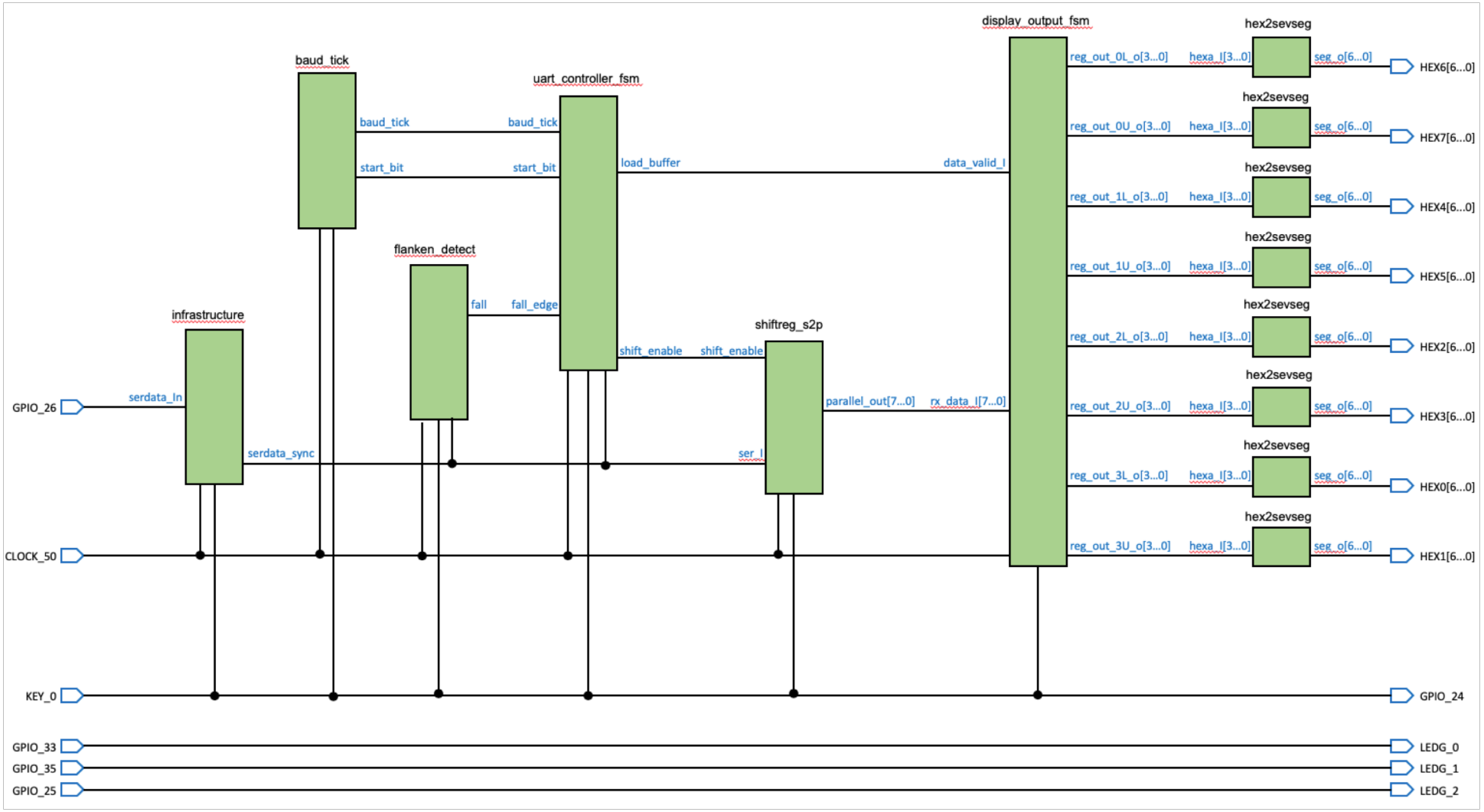
Matroschka Puppe

Hierarchisches VHDL Design oder Strukturale Modelierung



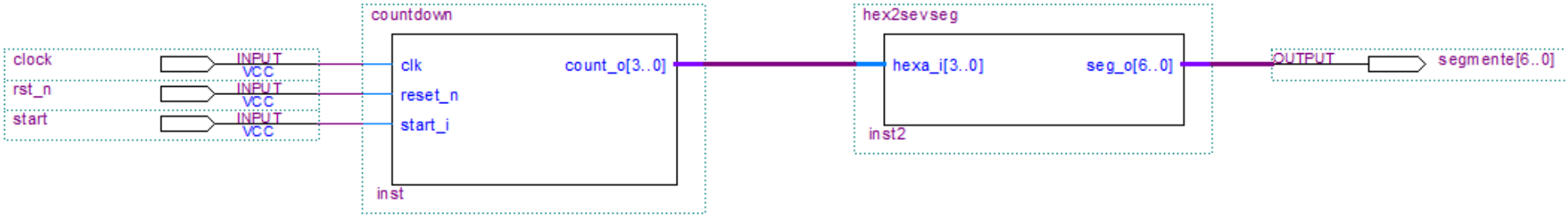


LAB1: VHDL TOP Level für UART



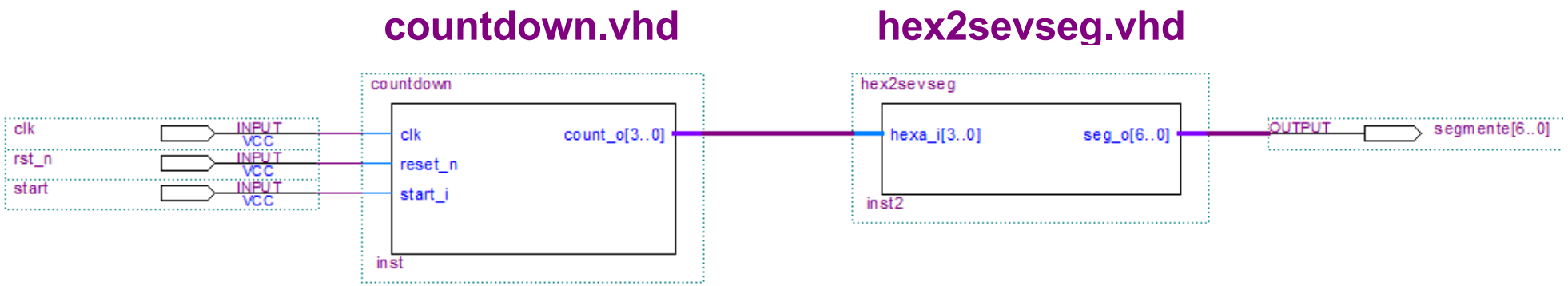
Beispiel für ein Hierarchisches Design

countdown.vhd **hex2sevseg.vhd**



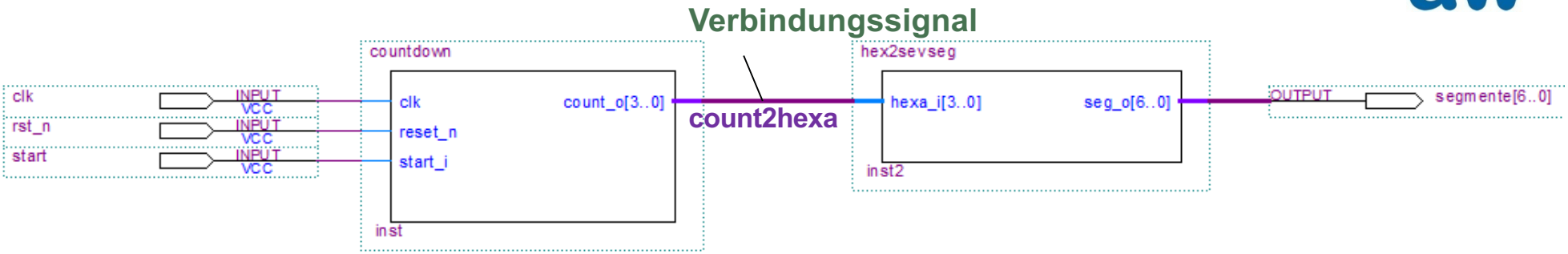
Dieser Top Level Schaltplan soll durch VHDL Code ersetzt werden

1. Top Level Entity




```
ENTITY top_level IS
  PORT(
       clock          : IN   std_logic;
       rst_n          : IN   std_logic;
       start          : IN   std_logic;
       segmente               : OUT  std_logic_vector(6 downto 0)
       );
END top_level ;
```

2. Component und Signal Deklarierungen

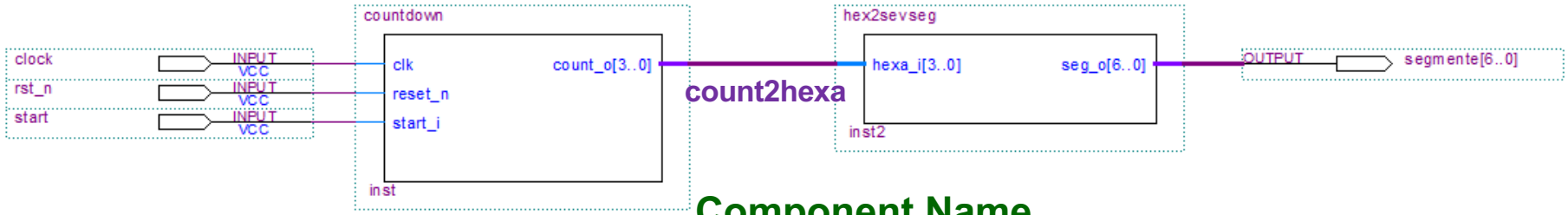



```
ARCHITECTURE struct OF top_level IS

        COMPONENT hex2sevseg
        PORT( hexa_i            : IN    std_logic_vector(3 downto 0);
              seg_o     : OUT   std_logic_vector(6 downto 0)
            );
        END COMPONENT;

        COMPONENT countdown
        PORT( clk,
              reset_n           : IN    std_logic;
              start_i           : IN    std_logic;
              count_o           : OUT   std_logic_vector(3 downto 0)
             );
        END COMPONENT;

        SIGNAL count2hexa       : std_logic_vector(3 downto 0);
BEGIN
```

Deklarierung Verbindungssignal:

3. Component Instanziierung

Instance = Realisierung einer Datengruppe



Component Name

Signal Namen in der übergeordneten Architektur

Instance Name (frei wählbar)

```
BEGIN
  inst_countdown_1: countdown
  PORT MAP (
            clk        =>      clock,
            reset_n    =>      rst_n,
            start_i    =>      start,
            count_o    =>      count2hexa
                );

inst_hex2sevenseg_1: hex2sevseg
  PORT MAP (
           hexa_i      =>      count2hexa,
           seg_o       =>      segmente
           );

END struct;
```

Signal Namen in der übergeordneten Architektur

Ports der untergeordneten Entity (siehe Component Declaration)

Komplettes Beispiel einer Top Level Architektur

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

```
ARCHITECTURE struct OF top_level IS

      COMPONENT hex2sevseg
      PORT(
      hexa_i      : IN   std_logic_vector(3 downto 0);
      seg_o       : OUT  std_logic_vector(6 downto 0));
      END COMPONENT;

      COMPONENT countdown
      PORT( clk,reset_n : IN    std_logic;
            start_i     : IN    std_logic;
            count_o     : OUT   std_logic_vector(3 downto 0));
      END COMPONENT;

      SIGNAL count2hexa : std_logic_vector(3 downto 0);
BEGIN

  inst_countdown_1: countdown
  PORT MAP (clk        =>    clock,
            reset_n    =>    rst_n,
            start_i    =>    start,
            count_o => count2hexa
            );

inst_hex2sevenseg_1: hex2sevseg
  PORT MAP (hexa_i     =>    count2hexa,
            seg_o      =>    segmente
            );

END struct;
```

Alternative Port-Map Methode (Positionale Methode)

```
architecture behv of test is
  signal sig_a, sig_b, sig_sum, sig_carry : std_logic;
  component adder -- Componenten-Deklarierung
    port(a          : in std_logic;
         b          : in std_logic;
         sum        : out std_logic;
         carry      : out std_logic);
    end component;

begin
  adder_1: adder port map (sig_a,sig_b,sig_sum,sig_carry);
end behv;
```

Übung1: Schaltplan der Hierarchie zeichnen

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY dekoder_einfach IS PORT (
    A: IN std_logic;
    B: IN std_logic;
    C: IN std_logic;
    D: IN std_logic;
    Z: OUT std_logic );
END dekoder_einfach ;

ARCHITECTURE structural OF dekoder_einfach IS
    SIGNAL int1: std_logic;
    COMPONENT nand2
        PORT (  in1,in2 : IN std_logic;
                outp : OUT std_logic );
    END COMPONENT;
    COMPONENT and3
        PORT (  in1,in2,in3 : IN std_logic;
                outp          : OUT std_logic );
    END COMPONENT;
BEGIN
    inst1: nand2 PORT MAP (
        in1      => A,
        in2      => B,
        outp     => int1    );
    inst2: and3 PORT MAP (
        in1      => int1,
        in2      => C,
        in3      => D,
        outp     => Z );
END ARCHITECTURE structural;
```

Generic

Definition

Mit «Generic» lassen sich Blöcke parametrisieren und die Wiederverwertbarkeit der Blöcke steigern.

Component Declaration:

```
component ''component_name''
  generic (''generic_liste'');
  port    (''port_liste'');
end component;
```

Component Instanziierung:

```
label: ''component_name''
generic map ( ''generic1'' => '' generic1_entity'' )
port map (
          ''component_port1'' => ''entity_port1'',
...
          ''component_portx'' => ''entity_portx''
          );
```

Generic

Beispiel Component Declaration mit Generic

```
ENTITY modulo_divider IS
GENERIC (width        : positive:= 5);
  PORT( clk,reset     : IN     std_logic;
        clk_div       : OUT    std_logic_vector(width-1 downto 0)
        );

END modulo_divider;

ARCHITECTURE rtl OF modulo_divider IS

SIGNAL cnt_folge:           unsigned(width-1 downto 0);
SIGNAL cnt_gegenwart:       unsigned(width-1 downto 0);

BEGIN
  logik : PROCESS(cnt_gegenwart)
  BEGIN
        cnt_folge <= cnt_gegenwart + 1 ;
  END PROCESS logik;
```

Generic

Beispiel Entity mit Generic

Default Wert, falls Generic bei der Instanziierung nicht verwendet wird

```
ENTITY modulo_divider IS
GENERIC (width        : positive:= 5);
  PORT( clk,reset     : IN     std_logic;
        clk_div       : OUT    std_logic_vector(width-1 downto 0)
        );

END modulo_divider;

ARCHITECTURE rtl OF modulo_divider IS

SIGNAL cnt_folge:           unsigned(width-1 downto 0);
SIGNAL cnt_gegenwart:       unsigned(width-1 downto 0);

BEGIN
  logik : PROCESS(cnt_gegenwart)
  BEGIN
        cnt_folge <= cnt_gegenwart + 1 ;
  END PROCESS logik;
```

Generic

Beispiel Component Instanziation mit Generic

```
architecture str of uart_top is

constant const_width : positive := 5

begin
modulo_divider_1 : modulo_divider
        generic map (width => const_width)
        port map (
            clk     => clock_50,
            reset_n => key_0,
            clk_div => clk_div
            );
```

Achtung:
Hier weder Komma
noch Strichpunkt

LOOP

Definition

- Sequential Statement used in: Process, Function and Procedure
- Der Parameter muss nicht deklariert werden
- Der Parameter darf im loop nicht verändert werden
- Synthese: Für jeden Schleifendurchlauf wird in der Kette der gleiche Block hardwaremäßig angehängt. Für Synthese dürfen keine «wait» statements enthalten sein.
- Simulation: Schleife wird zeitlich sequentiel durchlaufen

```
''eventuell_label'': for ''parameter'' in
''range'' loop
  sequential statements
end loop ''eventuell_label'';
```

LOOP

Beispiel 1

Achtung: 0 to 7 oder 7 downto 0

```
for i in 0 to 7 loop
  w(i) <= a(i) and b;
end loop;
```

LOOP

Beispiel 2

```
loop2: process(all)
begin
     z <= "0000";
     for i in z'range loop
          if (a = i) then
               z(i) <= '1';
          end if;
     end loop;
end process loop2;
```

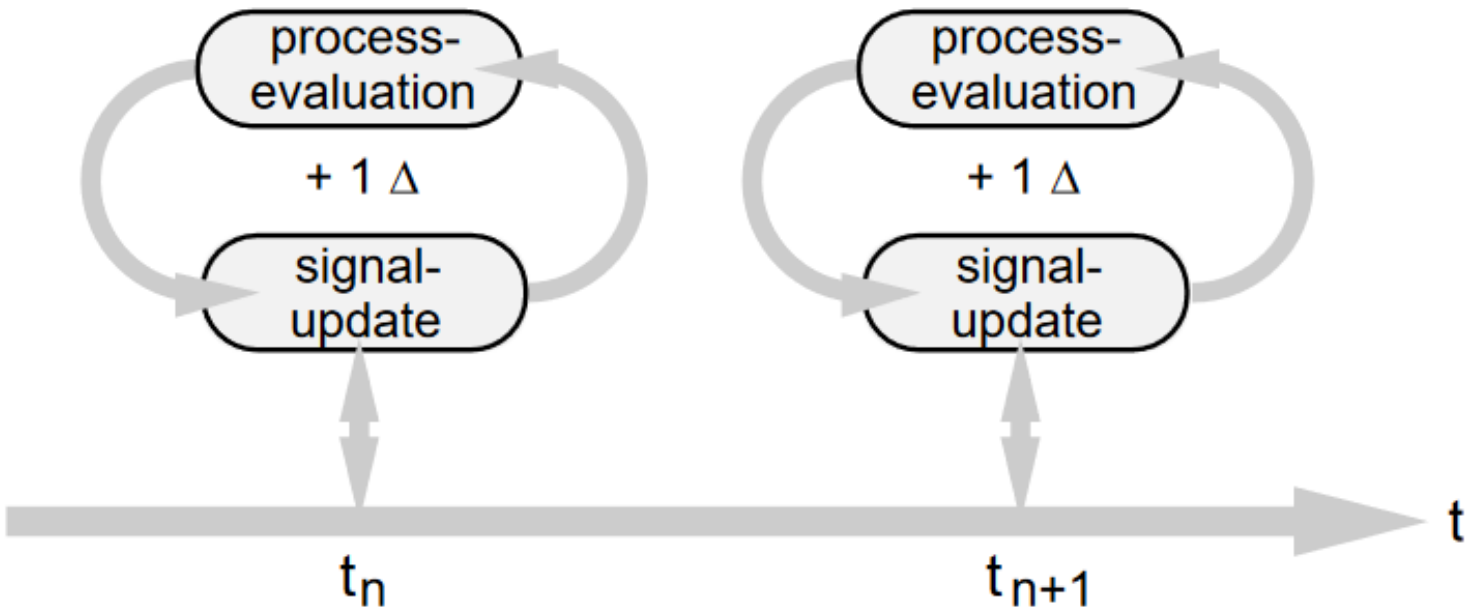
LOOP

Beispiel 3

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

Variablen

Ablauf von Prozessen



Quelle: Lehman, Wunder, Selz: «Schaltungsdesign mit VHDL»

Syntax der Variablen

Zürcher Hochschule für Angewandte Wissenschaften

Variablen kann man benutzen, um innerhalb eines Prozesses oder einer Function oder Procedure einen Wert temporär zu speichern

```
VARIABLE variablen_name : type [:= initialisierung];
```

Variablenzuweisung:

```
''Variablenname'' := ''Wert'';
```

Zuweisungsoperator

Beispiel:

```
VARIABLE summe : std_logic_vector (2 downto 0) := "101" ;
```

Verwendung einer Variablen

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

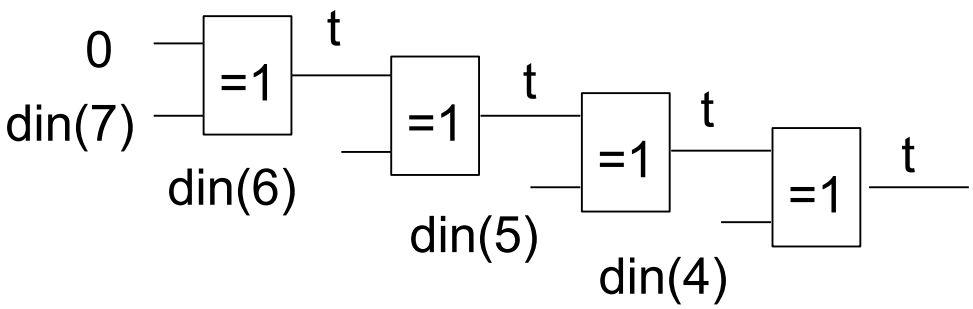
Beispiel 1

```
architecture comb of voter is
signal a,b,c: std_logic;
begin
  comb_voter : process (all) is

    variable x : std_logic_vector(2 downto 0) := "000";

  begin  -- process comb_voter
    x := a & b & c;
    case x is
      when "110"  => y <= '1';
      when "101"  => y <= '1';
      when "011"  => y <= '1';
      when "111"  => y <= '1';
      when others => y <= '0';
    end case;
  end process comb_voter;
end architecture comb;
```

Verwendung einer Variablen

Beispiel 2

```
architecture comb of parity_generator is

  signal din : std_logic_vector(7 downto 0);
  signal parity_out : std_logic;

begin

  parity : process (all) is
    variable t : std_logic := '0';
  begin
    for i in 7 downto 0 loop
      t := t xor din(i);
    end loop;  -- i
    parity_out <= t;
  end process parity;
end architecture comb;
```



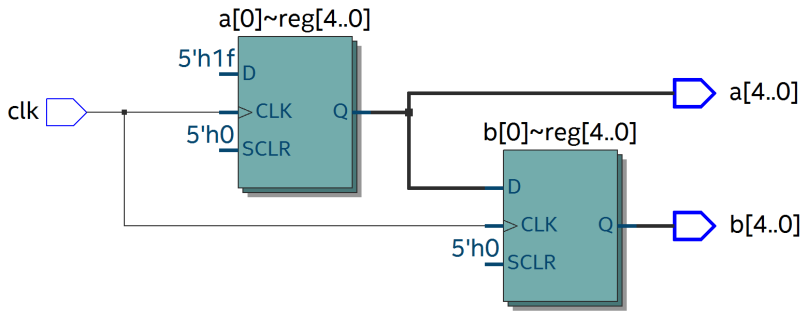
Variablen

- Kann man nur in Prozessen oder Unterprogrammen benutzen.
- Werden deshalb zwischen den Zeilen „Prozess“ und „Begin“ definiert.
- Variablen müssen vor Gebrauch initialisiert werden.
- Sind Global nicht verfügbar (ausserhalb des Prozess)
- Will man Variablen ausserhalb eines Prozesses weiterbenutzen, muss ein Signal zugewiesen werden. sig <= var;
- Sind nach der Zuweisung sofort gültig, nicht erst beim Verlassen des Prozesses.
- Können als Index oder temporärer Speicher innerhalb eines Prozesses benutzt werden.
- Zuweisung durch :=

Variable vs. Signal

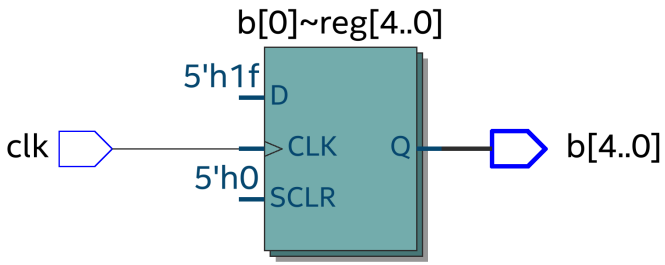
Variante 1:

```
signal a,b : std_logic_vector(4 downto 0);

process (CLK)
    begin
        if (rising_edge(clk)) then
            a <= "11111";
            b <= a;
        end if;
end process;
```



Variante 2:

```
process (CLK)
    begin

variable a : std_logic_vector(4 downto 0);

        if (rising_edge(clk)) then
            a := '11111';
            b <= a;
        end if;
end process;
```



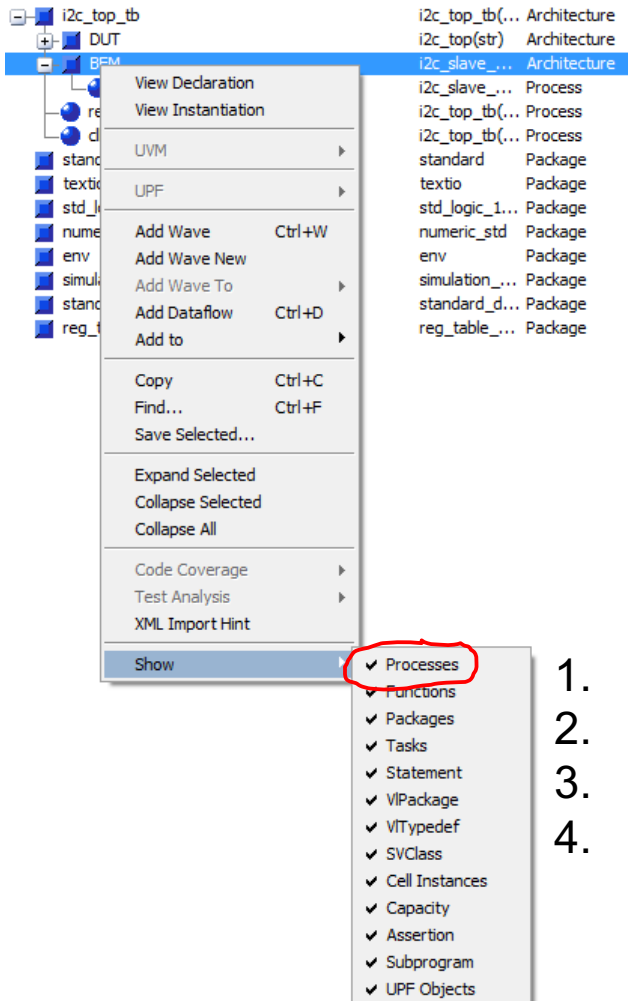
Signale vs. Variablen

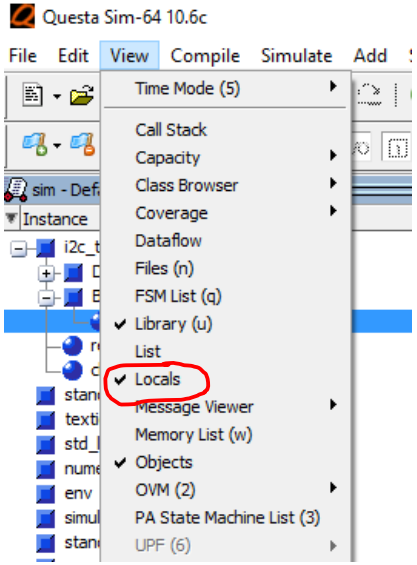
Zürcher Hochschule
für Angewandte Wissenschaften

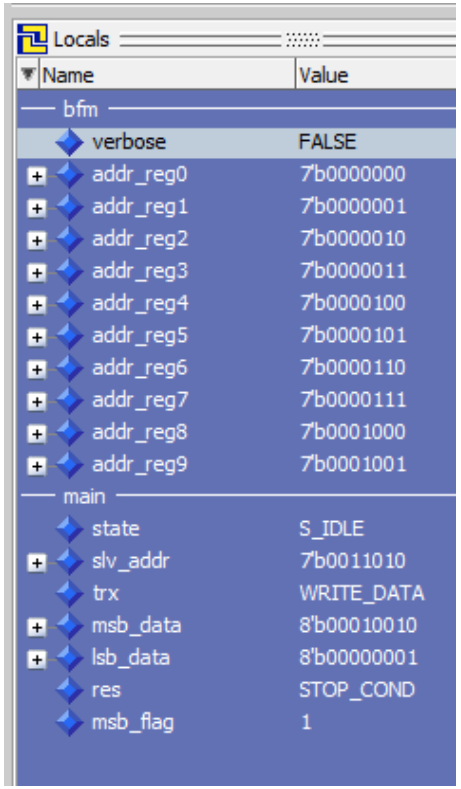
zh
aw

Signale	Variablen
Existieren global in der gesamten Architektur	Nur lokal im Prozess lesbar
Werden am Ende des Prozesses aktualisiert.	Werden immer sofort aktualisiert
Werden zwischen „Architectur“ und „Begin“ deklariert	Werden zwischen „Process“ und „Begin“ deklariert
Zuweisung v <= ‘1‘	Zuweisung v := ‘1‘
Erzeugt ein FF nach rising_edge(clk)	Erzeugt <u>kein</u> FF nach rising_edge(clk)

Variablen im Simulator anzeigen lassen



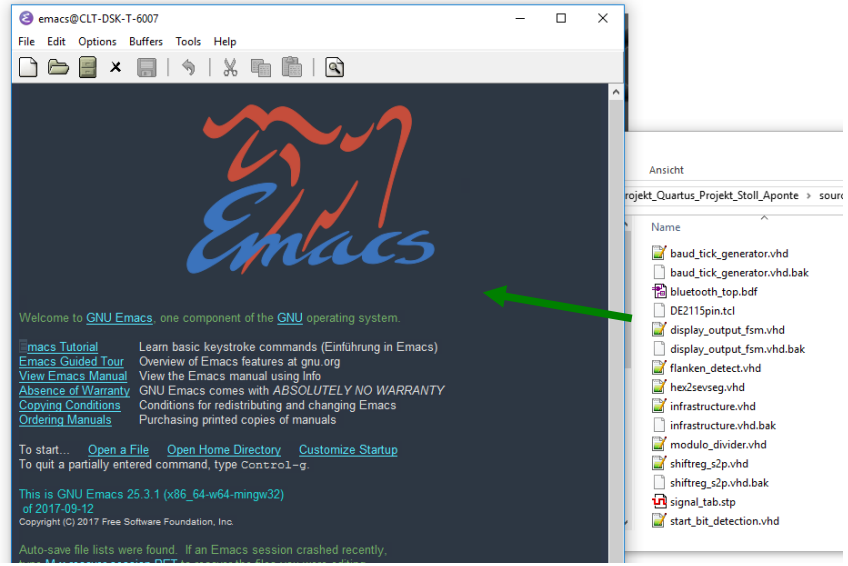




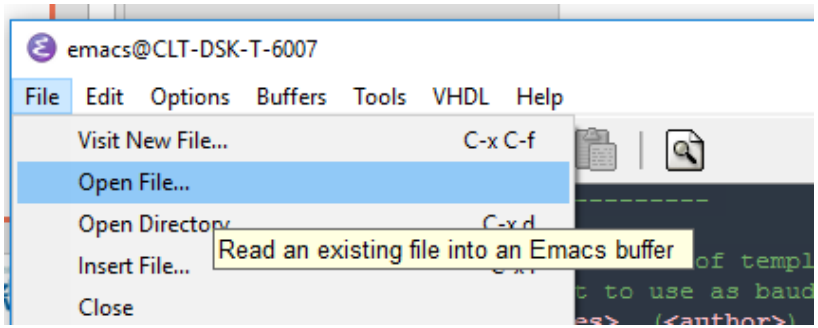
1. Select Entity
2. Rechte Maustaste → Show →Processes
3. Select Process
4. View → Locals

Dateien öffnen

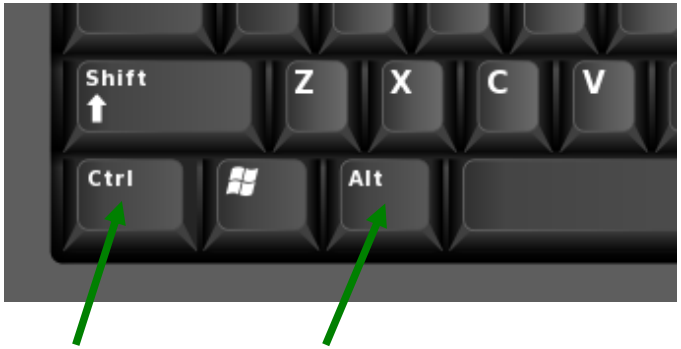
1. Methode: Drag and Drop



2. Methode: File → Open File



emacs Tastatur Abkürzungen



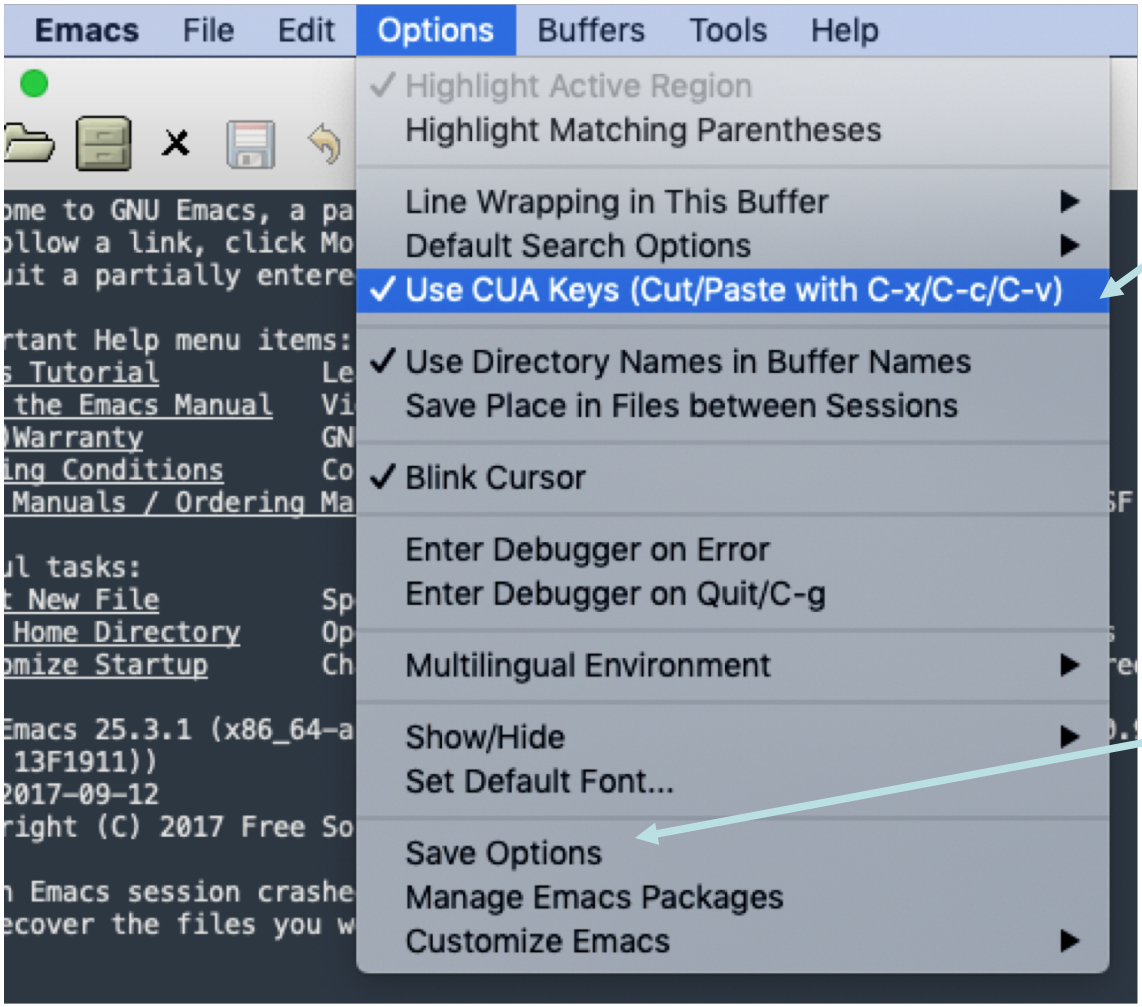
C-… **(Control)**

M-… **(Meta)**

Die gebräuchlichsten Abkürzungen:

C-x C-s	Save
M-w	Copy - Markierten Text in Kopier-Buffer speichern
C-y	Einfügen
C-k	Kill, löscht Zeile bis zum Ende
C-x u	Undo
C-x C-b	Beautify
C-s	Search forward
C-r	Search backward
C-g	Quit Menu

Emacs einrichten, damit C-x, C-c, C-v und C-z so funktioniert wie bei anderen Editoren



Hacken setzen

Sichert Einstellung dauerhaft

File Menu

Tastatur Abkürzungen

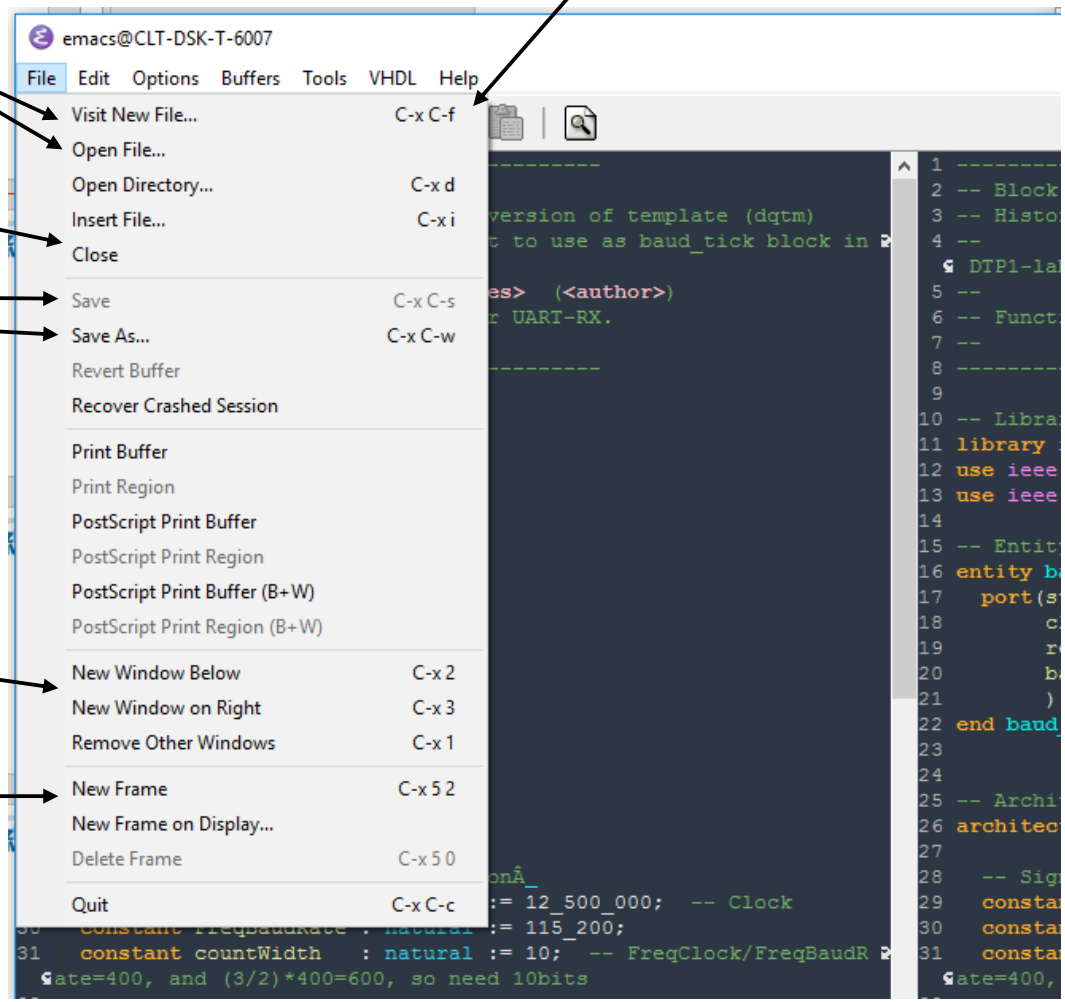
Datei zum Editieren öffnen…

Datei schliessen

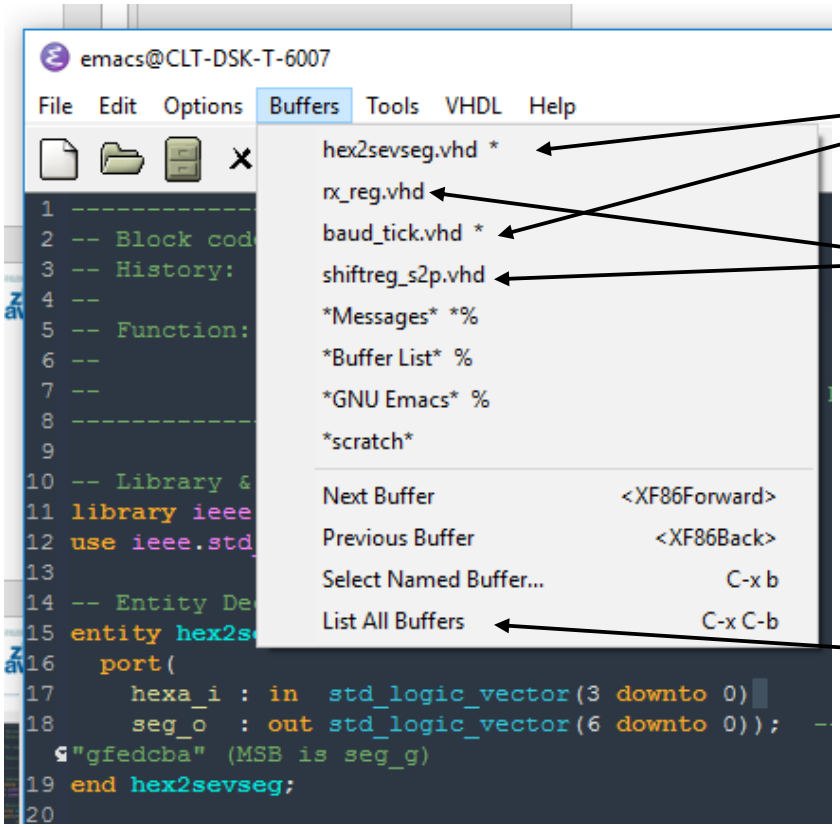
Datei sichern

emacs Fenster unterteilen

Weiteres emacs Fenster öffnen



Buffer Menu



Modifizierte Dateien, noch nicht gesichert

Gesicherte Dateien im Buffer

Im Buffer geöffnete Dateien werden
mit Herkunfts-Pfad gelistet

C-x C-s	Ausgewählte Datei sichern
C-x s	Alle im Buffer modifizierten Dateien sichern **Achtung:** Erfordert quittieren mit Yes

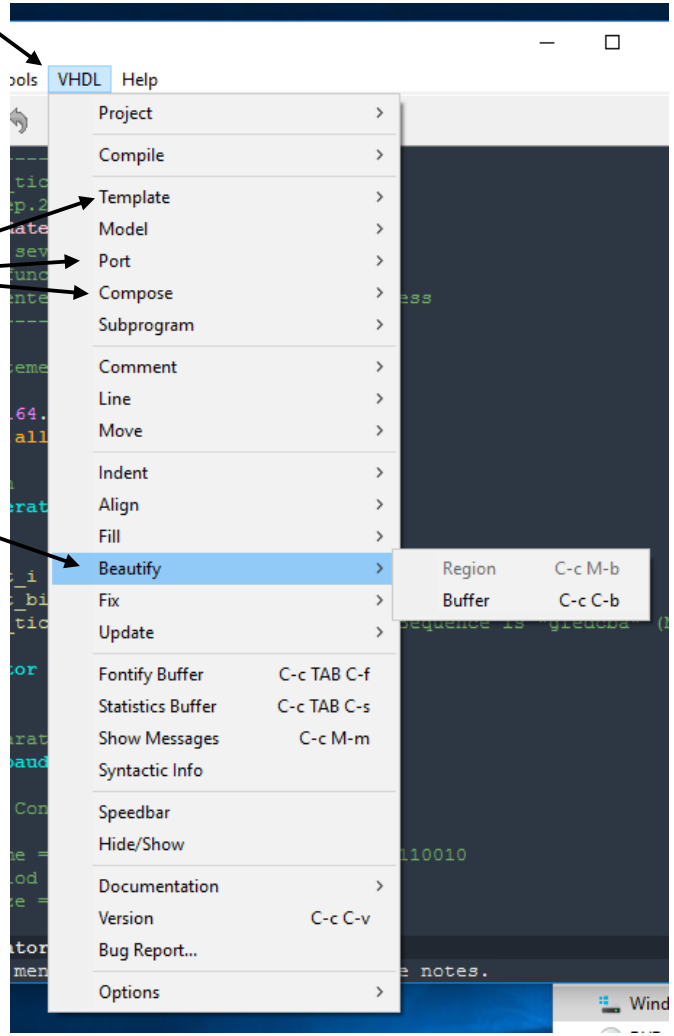
VHDL Features

Menus nur sichtbar, wenn
Datei mit VHDL Code bereits geöffnet

VHDL Features

Automatisches Generieren
von Code Passagen
(siehe weitere Folien)

Rückt Code automatisch
an die richtige Stelle



Generierung einer neuen Entity/Architektur

VHDL→Compose→New Component

Interactive Inputs:

File Name: <file name>

Title: <title for header>

Description: <short description>

```
1  ------------------------------------------------------------------------------
2  -- Title      : UART TOP
3  -- Project    :
4  ------------------------------------------------------------------------------
5  -- File       : uart_top.vhd
6  -- Author     :   <gelk@CLT-DSK-T-6007>
7  -- Company    :
8  -- Created    : 2018-02-06
9  -- Last update: 2018-02-06
10 -- Platform   :
11 -- Standard   : VHDL'08
12 ------------------------------------------------------------------------------
13 -- Description:
14 ------------------------------------------------------------------------------
15 -- Copyright (c) 2018
16 ------------------------------------------------------------------------------
17 -- Revisions  :
18 -- Date        Version  Author  Description
19 -- 2018-02-06  1.0      gelk    Created
20 ------------------------------------------------------------------------------
21
22 library ieee;
23 use ieee.std_logic_1164.all;
24
25 ------------------------------------------------------------------------------
26
27 entity uart_top is
28
29   generic (
30     );
31
32   port (
33     );
34
35 end entity uart_top;
36
37 ------------------------------------------------------------------------------
38
39 architecture str of uart_top is
40
41   ----------------------------------------------------------------------------
42   -- Internal signal declarations
43   ----------------------------------------------------------------------------
44
45   ----------------------------------------------------------------------------
46   -- Component declarations
47   ----------------------------------------------------------------------------
48
49 begin  -- architecture str
50
51   ----------------------------------------------------------------------------
52   -- Component instantiations
53   ----------------------------------------------------------------------------
54
55 end architecture str;
56
57 ------------------------------------------------------------------------------
```

Anzeigen und ersetzen von gleichen Textpassagen

1. Wort zum suchen oder ersetzen mit Maus selektieren (Wörter mit «_», auf «_» Klicken, damit ganzes Wort selektiert wird)
2. C-; lässt gleiche Textpassagen aufleuchten
3. Die Textpassage lässt sich jetzt mit dem Keyboard verändern, aber Achtung, alle Highlight Passagen werden gleichzeitig geändert
4. Mit C-; das Highlight Feature wieder ausschalten

```
    -- for ILA /4 => 108, 163
    signal count, next_count : integer range 0 to 65

Begin Architecture
IN

-----------------------------------------
- Concurrent Assignments
-----------------------------------------
    comb : PROCESS(ALL)
    BEGIN
            IF start_bit_i = '1' THEN
                    next_count <= 163;
            ELSIF count = 0 THEN
                    next_count <= 108;
            ELSE
                    next_count <= count - 1;
            END IF;
    END PROCESS comb;

- Implementation option: concurrent comb logic with w
eg_proc : PROCESS(ALL)
EGIN
            if reset_i = '0' THEN
                    count <= 0;
            elsif (rising_edge(clk_i)) THEN
                    count <= next_count;
            END IF;
```

Electric Mode

VHDL→Options→Mode→Electric Mode
schaltet Electric Mode ein

1. Type: If <space>

```
proc1: process (all) is
begin  -- process proc1
  if <condition>

end process proc1;
-\**-  modulo_divider.vhd   16% L69
condition: start_bit = '1'
```

2. Type: <condition>

IF/ELSE Statement wird automatisch vervollständigt:

```
proc1: process (all) is
begin  -- process proc1
  if start_bit = '1' then

  end if;

end process proc1;
```

Electric Mode funktioniert für: process, signals, if/else, case ...

Componet Declaration und Instantiation erstellen

VHDL→Compose→Place Component

VHDL→Compose→Copy Component

```
-- Entity Declaration
ENTITY baud_tick_generator IS
  PORT(
        clk_i            : IN   std_logic;
                reset_i          : IN     std_logic;
                start_bit_i      : IN   std_logic;
                baud_tick_o      : OUT  std_logic); --
END baud_tick_generator ;
```

baud_tick_generator.vhd

```
46   ---------------------------------------------
47   -- Component declarations
48   ---------------------------------------------
49
50   component baud_tick_generator is
51     port (
52       clk_i       : IN  std_logic;
53       reset_i     : IN  std_logic;
54       start_bit_i : IN  std_logic;
55       baud_tick_o : OUT std_logic);
56   end component baud_tick_generator;
57
58 begin  -- architecture str
59
60   ---------------------------------------------
61   -- Component instantiations
62   ---------------------------------------------
63
64   -- instance "baud_tick_generator_1"
65   baud_tick_generator_1: baud_tick_generator
66     port map (
67       clk_i       => clk_i,
68       reset_i     => reset_i,
69       start_bit_i => start_bit_i,
70       baud_tick_o => baud_tick_o);
71
72 end architecture str;
```

uart_top.vhd

Vorgang für die weiteren Components wiederholen

Component Verbindungssignale erzeugen

1. In `port map` klicken

2. VHDL→Port→Copy

```
  ---------------------------------------------
  -- Component instantiations
  ---------------------------------------------

  -- instance "baud_tick_generator_1"
  baud_tick_generator_1: baud_tick_generator
    port map (
      clk_i       => clk_i,
      reset_i     => reset_i,
      start_bit_i => start_bit_i,
      baud_tick_o => baud_tick_o);

end architecture str;
```

uart_top.vhd

1. Zwischen `architecture` und `begin` klicken

2. VHDL→Port→Paste as Signals

```
42 architecture str of uart_top is
43
44   ---------------------------------------------
45   -- Internal signal declarations
46   ---------------------------------------------
47   signal clk_i       : std_logic;
48   signal reset_i     : std_logic;
49   signal start_bit_i : std_logic;
50   signal baud_tick_o : std_logic;
51   ---------------------------------------------
52   -- Component declarations
53   ---------------------------------------------
54
55   component baud_tick_generator is
56     port (
57       clk_i       : IN  std_logic;
58       reset_i     : IN  std_logic;
59       start_bit_i : IN  std_logic;
60       baud_tick_o : OUT std_logic);
61   end component baud_tick_generator;
62
63 begin  -- architecture str
64
65   ---------------------------------------------
66   -- Component instantiations
67   ---------------------------------------------
68
69   -- instance "baud_tick_generator_1"
70   baud_tick_generator_1: baud_tick_generator
71     port map (
72       clk_i       => clk_i,
73       reset_i     => reset_i,
74       start_bit_i => start_bit_i,
75       baud_tick_o => baud_tick_o);
76
77 end architecture str;
78
```

uart_top.vhd