1 Allgemein

1.1 Entity

```
entity <entity_name> is

    -- Konstanten fuer Entity
    generic(<generic_name> : <type> := <value>;
    <other_generics> );

    -- Ein- und Ausgaenge
    port(   <port_name> : <mode> <type>;
    <other_ports>);

end <entity_name>;
```

1.2 Architecture

```
architecture <arch_name> of <entity_name> is
-- component declaration
-- signal declaration
-- constant declaration
-- variable declaration
begin
-- architecture body
-- component instantiation and logic
end <arch_name>;
```

1.3 Komponenten

```
component <component_name>
    -- Konstanten fuer Entity
    generic(<generic_name> : <type> := <value>;
            <other_generics> );

    -- Ein- und Ausgaenge
    port(   <port_name> : <mode> <type>;
            <other_ports>);
end component;
```

```
<instance_label>: <component_name>
    generic map (
        <generic_name> => <value>,
        <other_generics> )
    port map (
        <comp_port_name> => <signal_name>,
        <other_ports> );
```

Oder ohne Komponenten-Deklaration

```
<instance_label>: entity work.<component_name>
    generic map (
        <generic_name> => <value>,
        <other_generics> )
    port map (
        <comp_port_name> => <signal_name>,
        <other_ports> );
```

2 Variabeln

```
signal   sig_1   : STD_LOGIC := '0';
variable var_1   : STD_LOGIC := '1';
constant const_1 : STD_LOGIC := '1';

sig_1 <= '1';
var_1 := '1';
```

```
signal sig_2 : STD_LOGIC_VECTOR( N-1 downto 0 );
sig_2 <= (others => '0');
```

```
signal sig_1 : STD_LOGIC_VECTOR( 3 downto 0 );
signal sig_2 : STD_LOGIC_VECTOR( 3 downto 0 );

sig_2( 1 downto 0 ) <= sig_1( 3 downto 2 );
```

```
signal sig_1 : STD_LOGIC_VECTOR( 3 downto 0 );
signal sig_2 : STD_LOGIC_VECTOR( 1 downto 0 );
signal sig_3 : STD_LOGIC_VECTOR( 1 downto 0 );

sig_1 <= sig_2 & sig_3;
```

```
type <name> is array (n downto 0) of <typ> (m downto 0);
```

3 Kombinatorische Logik

```
with a select b <=
"1000" when "00",
"0100" when "01",
"0010" when "10",
"0001" when "11",
"0000" when others;
```

```
b <= "1000" when a = "00" else
"0100" when a = "01" else
"0010" when a = "10" else
"0001" when a = "11" else
"0000";
```

4 Sequentielle Logik

```
<optional_label>: process (<optional sensitivity list>)
    <declarations>
begin
    --sequential statements
end process <optional_label>;
```

```
case a is
    when "00"   => b <= "1000";
    when "01"   => b <= "0100";
    when "10"   => b <= "0010";
    when "11"   => b <= "0001";
    when others => b <= "0000";
end case;
```

```
if a = "00" then
    b <= "1000";
elsif a = "01" then
    b <= "0100";
elsif a = "10" then
    b <= "0010";
elsif a = "11" then
    b <= "0001";
else
    b <= "0000";
end if;
```

```
for I in 0 to 3 loop
-- Code einfuegen
end loop;
```

5 Moore-Automat

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
    port (  clk, reset, x1 : IN std_logic;
            outp : OUT std_logic);
end entity;

architecture moore of fsm is
    type state_type is (s1,s2,s3,s4);
    signal state: state_type ;
begin

state_conv: process (clk,reset)
begin
    if (reset ='1') then
        state <=s1;
    elsif (rising_edge(clk)) then
        case state is
            when s1 =>
                if x1='1' then
                    state <= s2;
                else
                    state <= s3;
                end if;
            when s2 => state <= s4;
            when s3 => state <= s4;
            when s4 => state <= s1;
        end case;
    end if;
end process state_conv;

out_fn : process (state)
begin
    case state is
        when s1 => outp <= '1';
        when s2 => outp <= '1';
        when s3 => outp <= '0';
        when s4 => outp <= '0';
    end case;
end process out_fn;
end moore;
```

6 Mealy-Automat

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fsm is
    port (  clk, reset, x1 : IN std_logic;
            outp : OUT std_logic);
end entity;

architecture mealy of fsm is
    type state_type is (s1,s2,s3,s4);
    signal state: state_type ;
begin

state_conv: process (clk,reset)
begin
    if (reset ='1') then
        state <=s1;
    elsif (rising_edge(clk)) then
        case state is
```

```
            when s1 =>
                if x1='1' then
                    state <= s2;
                else
                    state <= s3;
                end if;
            when s2 => state <= s4;
            when s3 => state <= s4;
            when s4 => state <= s1;
        end case;
    end if;
end process state_conv;

out_fn : process (state, x1)
begin
    case state is
        when s1 => outp <= x1;
        when s2 => outp <= x1;
        when s3 => outp <= not x1;
        when s4 => outp <= not x1;
    end case;
end process out_fn;
end mealy;
```