VHDL

Wikipedia:

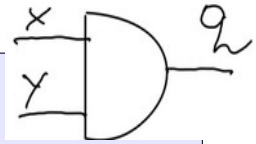
Very High Speed Integrated Circuit Hardware Description Language, kurz **VHDL**, ist eine Hardwarebeschreibungssprache, mit der es möglich ist, digitale Systeme textbasiert zu beschreiben.

Inhalt

VHDL – Die Komponenten

Ein einfaches Beispiel zuerst

Blockschaltbild AND Gate



Beschreibung AND Gate

```
-------------------------------------------------
-- AND gate (ESD book figure 2.3)
-- two descriptions provided
-------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------

entity AND_ent is
port( x: in std_logic;
      y: in std_logic;
      q: out std_logic
);
end AND_ent;

-------------------------------------------------
```

```
architecture behav1 of AND_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='1') and (y='1')) then
          q <= '1';
      else
          q <= '0';
      end if;
    end process;
end behav1;

architecture behav2 of AND_ent is
begin
    q <= x and y;
end behav2;

-------------------------------------------------
```

IEEE Libraries

VHDL standard packages

https://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html

```
  library IEEE;

        package _standard                    … predefined, data types bit, bool, ...

        use IEEE.std_logic_1164.all;         … std_logic Type
        use IEEE.std_logic_textio.all;       … text messages for std_logic types

use IEEE.std_logic_arith.all;          … +,-,ABS, <=, >=, /= , SHL, SHR

A set of arithemtic, conversion, and comparison functions for SIGNED, UNSIGNED,
SMALL_INT, INTEGER, STD_ULOGIC, STD_LOGIC, and STD_LOGIC_VECTOR.                --
```

Datentypen

std_logic

```
TYPE std_logic IS (     'U',  -- Uninitialized
                        'X',  -- Forcing  Unknown
                        '0',  -- Forcing  0
                        '1',  -- Forcing  1
                        'Z',  -- High Impedance
                        'W',  -- Weak     Unknown
                        'L',  -- Weak     0
                        'H',  -- Weak     1
                        '-'   -- Don't care
                      );
```

```
   ----------------------------------------------------------------------
   -- resolution function
   ----------------------------------------------------------------------
   CONSTANT resolution_table : stdlogic_table := (
   --      ---------------------------------------------------------
   --      |  U    X    0    1    Z    W    L    H    -        |   |
   --      ---------------------------------------------------------
           ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
           ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
           ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
           ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
           ...
```

```
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');
type INTEGER is range -2147483647 to 2147483647;
type STRING is array (POSITIVE range <>) of CHARACTER;
type BIT_VECTOR is array (NATURAL range <>) of BIT;

type TIME is range $- to $+
        units
                fs;                     -- femtosecond
                ps      =  1000 fs;     -- picosecond
                ns      =  1000 ps;     -- nanosecond
                us      =  1000 ns;     -- microsecond
                ms      =  1000 us;     -- millisecond
                sec     =  1000 ms;     -- second
                min     =  60 sec;      -- minute
                hr      =  60 min;      -- hour;
  end units;


SUBTYPE UX01Z   IS resolved std_ulogic RANGE 'U' TO 'Z'; -- ('U','X','0','1','Z')
```

Entity

Beschreibt die Schnittstelle einer Komponente (Inputs, Outputs, Konfiguration).

Prozesse/Sensitivity Liste

Alle Prozesse laufen nebeneinander gleichzeitig. Wenn eine Sensitivity Liste angegeben ist, dann wird der Prozess nur ausgewertet, wenn eine Änderung in einem der Signale der Liste auftritt (beschleunigt die Auswertung).

Architecture

Beschreibt das Innenleben einer Komponente

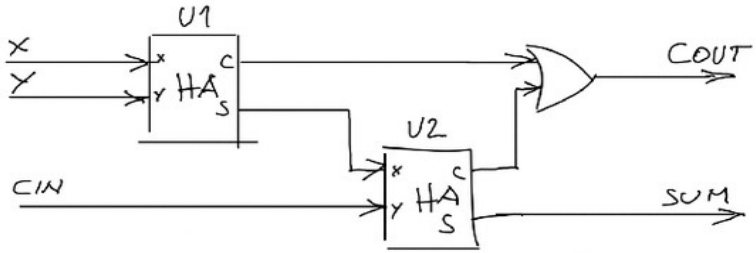
Verhaltensbeschreibung

Das Verhalten der Schaltung wird rein funktional beschrieben.

```
ENTITY halfAdder IS
  PORT (X, Y: INBit;SUM, COUT: OUTBit);
END halfAdder;

ARCHITECTURE behavioural OF halfAdder IS
BEGIN
    SUM<= X XOR Y ;
    COUT<= X AND Y ;
END behavioural ;
```

Strukturbeschreibung

Die Schaltung wird durch Instanzierung von Komponenten aufgebaut und "verdrahtet".




```
ENTITY fullAdder IS
   PORT (X, Y, CIN : IN Bit;SUM, COUT : OUT Bit);
END fullAdder;

ARCHITECTURE structural OF fullAdder IS

  SIGNAL S1, S2, S3;

  COMPONENT halfAdder
       PORT (X, Y: IN Bit;  SUM, COUT : OUT Bit);
  END COMPONENT;

BEGIN
   U1 : halfAdder PORT MAP (X=>X, Y=>Y, SUM=>S1, COUT => S2);
   U2 : halfAdder PORT MAP (S1, CIN, SUM, S3);
   COUT <= S2 OR S3;
END structural;
```

Signal

Interne "Leiterbahn".

Port Map: Komponente "anschließen"

Verkürzte Schreibweise: Reihenfolge der Komponente einhalten

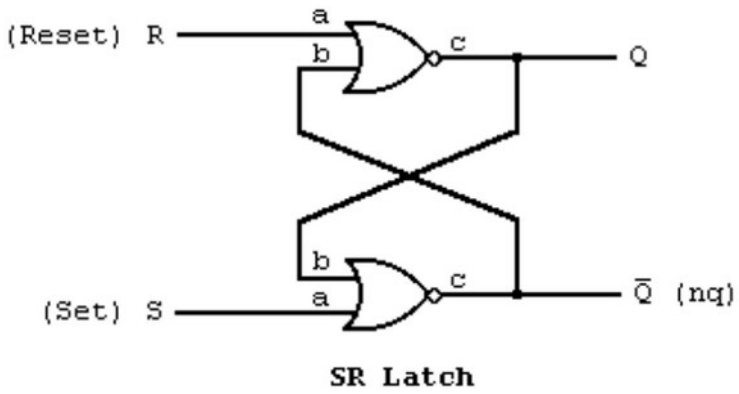
```
PORT MAP (X, Y, S1, S2)
```

Lange Form: Reihenfolge egal

```
PORT MAP (X=>X, Y=>Y, SUM=>S1, COUT=>S2)
```

Beispiele

```
entity rsff is
  port (s,r: in bit;
        q,nq: out bit);
end latch;

architecture dataflow of rsff is
  signal q0 : bit :=
'0';
  signal nq0 : bit :=
'1';
begin
  q0<=r nor nq0;
  nq0<=s nor q0;
  nq<=nq0;
  q<=q0;
end dataflow;
```



SR Latch

Testbench

Eine Testbench ist ein VHDL Code, der eine Schaltung simuliert (und testet). Es wird eine leere Entität erzeugt, die Testbench, und die zu simulierende Schaltung (unit under test UUT) wird mit internen Signalen "angeschlossen" (port map), anschließend werden die Internen Eingangs-Signale auf bestimmte Bitmuster (Pattern) gesetzt und die Reaktion der UUT beobachtet (bzw. mittels ASSERT Befehlen auch automatisch ausgewertet).

Testbench Beispiel

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

entity TB_tridr is
end TB_tridr;

architecture TB of TB_tridr is

    component tristate_dr is
    port(    d_in:  in std_logic_vector(7 downto 0);
             en:    in std_logic;
             d_out: out std_logic_vector(7 downto 0)
    );
    end component;

    signal T_d_in, T_d_out: std_logic_vector(7 downto 0);
    signal T_en: std_logic;
```

```
    begin

        U_UT: tristate_dr port map (T_d_in, T_en, T_d_out);

    process
    begin

      T_d_in <= "11001100";
      T_en <= '1';
      wait for 20 ns;
      assert(T_d_out = T_d_in) report "Error detected!"
      severity warning;

      ...

      wait;  -- wait forever

    end process;
end TB;
```

"wait for" und "after"

P1: clk1 steckt auf 1 fest

P2: Änderungen bei 60 120, 160 220, 260 320 usw.

P3: Änderungen bei 100, 200, 300 usw.

```
entity clks is
 port(clk1,clk2,clk3:out bit);
end;
architecture behav of clks is
begin
  P1: process
      begin
        clk1 <= '1';
        wait for 100ns;
        clk1 <= '0';
      end process;
  P2: process
      begin
        wait for 50 ns; -- process is suspended for 50ns
        clk2 <= '1' after 10 ns;
        wait for 50 ns;
        clk2 <= '0' after 20 ns;
      end process;
```

```
  P3: process
      begin
        clk3 <= '1';
        wait for 100ns;
        clk3 <= '0';
        wait for 100ns;
      end process;

end behav;
```

VHDL verstehen

Alles passiert gleichzeitig

… daher ist die Reihenfolge der Befehle egal. Die beiden Architekturen sind gleichwertig.

```
-- kner 2020
-- order of commands does not matter
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY e_feedback IS
        PORT (  a,b : IN std_logic;q   : INOUT std_logic );
END e_feedback;

ARCHITECTURE arch1 OF e_feedback IS
        SIGNAL  q_intern : std_logic;
BEGIN
      q_intern <= a and b;
      q <= q_intern;
END arch1;

ARCHITECTURE arch OF e_feedback IS
        SIGNAL  q_intern : std_logic;
BEGIN
      q <= q_intern;
      q_intern <= a and b;
END arch
```

Prozesse werden gleichzeitig evaluiert

Der Simulator hängt sich auf: Änderungen innerhalb einer Zeitscheibe

P1: alles passiert in der Zeit 0ns: clk1 ändert sich und damit muss der Prozess ständig neu evaluiert werden, der Prozess hängt
P2: wir haben die Zeit 0ns, die Auswertung erfolgt, das Ergebnis würde später geschrieben, aber soweit kommt es nicht, weil die Änderung sofort eine Neuevaluierung des Prozesses erfordert usw., der Prozess hängt

```
  P1: process
      begin
        clk1 <= '1';
        clk1 <= '0';
      end process;



P2: process
      begin
        clk1 <= '1' after 10ns;
        clk1 <= '0' after 20ns;
      end process;
```

Lösung des Problems: die Evaluierung aussetzen mit z.B: "wait for 100 ns"

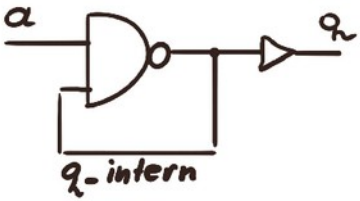
Das folgende Beispiel erzeugt einen 30ns breiten Puls bei 10ns, 110ns, 210ns usw.

```
P3: process
     begin
      clk3 <= '1' after 10ns, '0' after 40ns;
      wait for 100 ns;
     end process;
```

Der Simulator hängt sich auf: Rückkopplungen

Der Simulator muss immer neu evaluieren, wenn sich ein Signal geändert hat.

Wenn a von 0 auf 1 geht, ändert sich q von 1 auf 0 und wird an den Eingang des Gatters zurückgeführt; das führt dazu, dass q wieder 1 wird usw. Die Schaltung schwingt, solange a auf 1 bleibt.



Wenn man im folgenden Beispiel die Zeitverzögerung entfernt, dann hängt sich der Simulator auf, weil dann eine Änderung am Eingang des Nand-Gatters zu einem neuen Ausgangssignal führt und dieses wieder als Eingang ins Gatter ausgewertet werden muss. Die Gatterlaufzeit beträgt 0ns. Die Schaltung schwingt mit unendlicher Frequenz.

Wenn man die Verzögerung einführt, dann schwingt die Schaltung in der Simulation, wenn man das Eingangssignal von 0 auf 1 ändert

>force a 0
>run 50
>force a 1
>run 50

```
-- kner 2020
-- Demonstrates feedback problem: this circuit is oscillating
-- hangs when no delay introduced

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY e_feedback IS
        PORT (
                a : IN std_logic;
                q   : INOUT std_logic
        );
END e_feedback;

ARCHITECTURE arch OF e_feedback IS
        SIGNAL  q_intern : std_logic;
BEGIN
      q_intern <= a nand q_intern after 10 ns;
      q <= q_intern;
END arch;
```

VHDL Die Programmiersprache für Parallel Programming

Subtypen

Die Anzahl der möglichen Werte soll eingeschränkt werden.
natural … natürliche Zahlen 0,1,2 …
positiv … positive Zahlen 1,2,3,…

```
subtype small_int is integer range –128 to 127;
subtype LCD_display_string is string(1 to LCD_display_len);
subtype digit is bit_vector(3 downto 0);
subtype X01  is std_ulogic range 'X' to '1';
```

Array Typen

Sammlung von Werten vom gleichen Typ. Index zeigt die Position innerhalb des Arrays.

```
type word is array (0 to 31) of bit;
type word is array (31 downto 0) of bit;
```

```
subtype coeff_ram_address is integer range 0 to 63;
type coeff_array is array (coeff_ram_address) of real;
variable coeff : coeff_array;
coeff(0) := 0.0
```

Initialisierung

```
type point is array (1 to 3) of real;
constant origin : point := (0.0, 0.0, 0.0);
variable view_point : point := (10.0, 20.0, 0.0);
```

Array Attribute

```
A'left
A'right
A'range
A'reverse_range
A'length
```

```
type A is array (1 to 4) of boolean;

A'left = 1
A'right = 4
A'range ... 1 to 4
A'reverse_range ... 4 down to 1
A'length = 4
```

Unbeschränkte Typen müssen bei der Instanzierung eines Objektes angegeben werden:

```
type sample is array (natural range <>) of integer;
variable short_sample_buf : sample(0 to 63);
```

Variable und Konstante

Variable und Konstante dürfen nicht von mehreren Prozessen zugegriffen werden.

```
constant number_of_bytes : integer := 4;
constant number_of_bits : integer := 8 * number_of_bytes;
constant e : real := 2.718281828;
constant prop_delay : time := 3 ns;
constant size_limit, count_limit : integer := 255
```

```
variable index : integer := 0;
variable sum, average, largest : real;
variable start, finish : time := 0 ns;
...
index := index + 1;
```

Sequenzielle Statements

Sequenzielle Statements können nur innerhalb von Prozessen verwendet werden!

```
if … then … else … end if;
if … then … elsif … elsif … else … end if;
case … is … when … when … when … end case;
loop … exit when … end loop;
loop … if … then exit; end if … end loop;
while … loop … end loop;
for … in 0 to 10 loop … end loop;
```

Assert

… prüft, ob ein Ausdruck erfüllt ist (z.B: ob das richtige Signal ausgegeben wird) und liefert bei Bedarf eine Meldung, wenn eine Schaltung nicht das gewünschte Ergebnis liefert (z.B: weil ein Logik- oder Hardware-Fehler aufgetreten ist).

```
assert initial_value <= max_value
     report "initial value too large" severity warning;
```

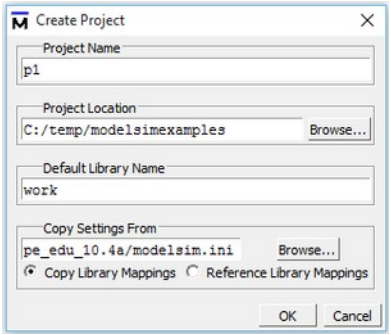
Entwicklungswerkzeuge

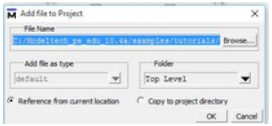
ModelSim Student Edition

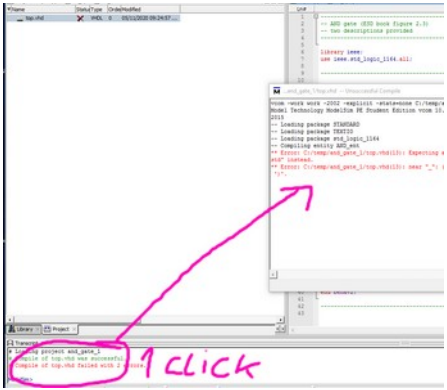
https://www.mentor.com/company/higher_ed/modelsim-student-edition-eval

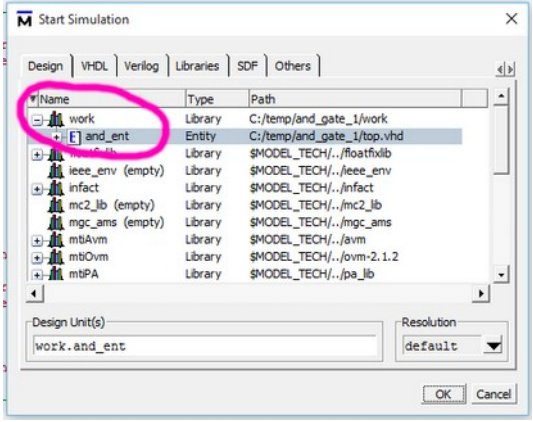
* Registrieren
* Downloaden
* Installieren
* Lizenzfile downloaden und in den Installationsordner kopieren

Modelsim Projekt anlegen

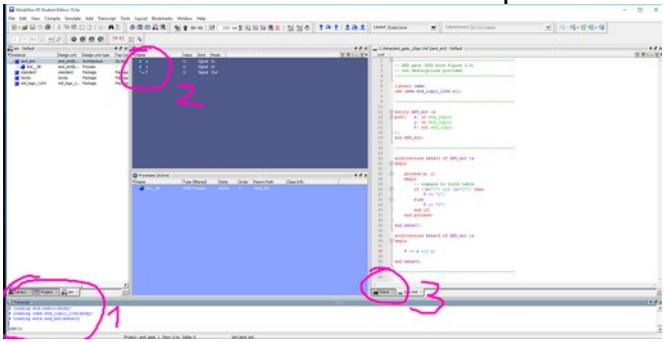
1. //File/New/Project
2. Add Existing File 

oder Create New File
3. //Compile/All 

bei Fehlermeldung auf die rote Meldung doppelklicken 



4. //Simulate/Start Simulation
5. Simulationskommandos in Transcript-Fenster aus einem Editor per Copy&Paste einfügen.



Beispiel Simulator Kommando File

```
delete wave *
add wave *
restart
force -freeze x 1 0, 0 50  -r 100  #start with 1 at 0ns    repeat all 100ns
force -freeze bus 8'h1A 0        # set bus to 1A hex at 0 ns
examine x                              # show value of x
force y 1
run 300
noforce y
force y U
force y W
force y X #with comment
force y H
force y L
history
```

GHDL

Eine Alternative zu Xilinx und ModelSim.

Da der Xilinx Compiler sehr groß ist (>10GByte) verwende ich hier den GHDL Compiler.

```
$ ghdl -a adder_tb.vhd                    …  analyze / compile
$ ghdl -a *.vhdl

$ ghdl -e adder_tb                        …  elaborate entity/link

$ ghdl -r adder_tb                        …  run

$ ghdl -r adder_tb --vcd=adder.vcd        …  for graphical output
$ gtkwave adder.vcd
```

bei Fehlermeldungen versuchen:

$ghdl -a --ieee=synopsys -fexplicit adder_tb.vhd

$ghdl -r --ieee=synopsys -fexplicit adder_tb --stop-time=500ns --vcd=vcd.out

GTKWAVE

Nach dem Aufruf ist das Anzeigfenster leer! Es müssen zuerst die Signale in das "Signals" Fenster gezogen werden und anschließend "Zoom fit".

Das folgende TCL Skript zeigt alle Signale an und zoomt die gesamte Zeit. Der Aufruf erfolgt für das Viewer-File x.vcd über die Kommandozeile:

>gtkwave -S x.tcl x.vcd

TCL-Script "x.tcl":

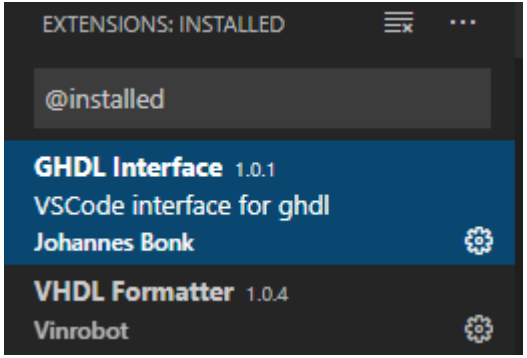
```
set nfacs [ gtkwave::getNumFacs ]
set dumpname [ gtkwave::getDumpFileName ]
set dmt [ gtkwave::getDumpType ]

puts "number of signals in dumpfile '$dumpname' of type $dmt: $nfacs"

set clk48 [list]

for {set i 0} {$i < $nfacs } {incr i} {
    set facname [ gtkwave::getFacName $i ]

    puts "facname: $facname"
```

```
    set indx [ string first "." $facname  ]
    if {$indx == -1} {
        lappend clk48 "$facname"
    }
}

set ll [ llength $clk48 ]
puts "number of signals found : $ll"

set num_added [ gtkwave::addSignalsFromList $clk48 ]
puts "num signals added: $num_added"

gtkwave::/Time/Zoom/Zoom_Full
```

MS Code und VHDL

Voraussetzung: ghdl und gtkwave sind installiert

Extensions:



GtkWave Integration in das GHDL Interface funktioniert bei mir nicht richtig.

GHDL Skript

PowerShell Skript. Dieses Skript compiliert das Projekt und ruft GtkWave mit allen Signalen automatisch auf.

Das Skript liegt im Pfad, es wird als Parameter "entity" die Entity des Testbench-Files angegeben.

>ghdl.ps1 -entity CKT_TB

ghdl.ps1:

```
param (    [string]$entity )
$l = Get-Location
echo $l
rm "$entity.vcd"
ghdl -a  $l\*.vhd
ghdl -e $entity
ghdl -r $entity --vcd=$entity.vcd
gtkwave  -S $PSScriptRoot/gtkwave.tcl "$entity.vcd"
```

Quellen

Ganz super: Xilinx XST User Guide; enthält Erklärungen und jede Menge Beispiele

https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf

gute Einführung und viele Beispiele:

VHDL Tutorial: Learn by Example-- by Weijun Zhang, July 2001

die Beispiele laufen mit erweiterten ghdl Schaltern: ghdl -a --ieee=synopsys -fexplicit tb_ALU.vhd

GHDL … ein freier VHDL Compiler

XILINX ISE WebPack … freier Compiler von Xilinx (alt)

XILINX Vivado WebPack … freier Compiler von Xilinx (neu)

https://course.ccs.neu.edu/cs3650/ssl/TEXT-CD/Content/Tutorials/VHDL/vhdl-tutorial.pdf