VHDL

Wikipedia:

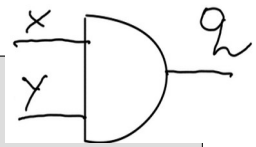
Very High Speed Integrated Circuit Hardware Description Language, kurz **VHDL**, ist eine Hardwarebeschreibungssprache, mit der es möglich ist, digitale Systeme textbasiert zu beschreiben.

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;
      use IEEE.std_logic_1164.all;
      use IEEE.std_logic_textio.all;
      use IEEE.std_logic_arith.all;
      use IEEE.numeric_bit.all;
      use IEEE.numeric_std.all;
      use IEEE.std_logic_signed.all;
      use IEEE.std_logic_unsigned.all;
      use IEEE.math_real.all;
      use IEEE.math_complex.all;
```

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). Eine Zuweisung außerhalb eines Prozesses hat eine implizite Sensitivität d.h., sie wird nur ausgewertet, wenn einen Signaländerung auftritt

z.B: x <= y -- y muss sich ändern

Architecture

Beschreibt das Innenleben einer Komponente

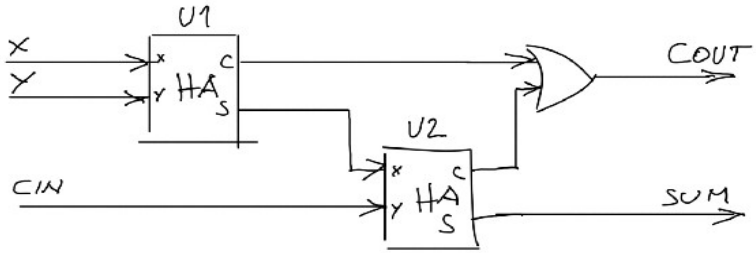
Verhaltensbeschreibung

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

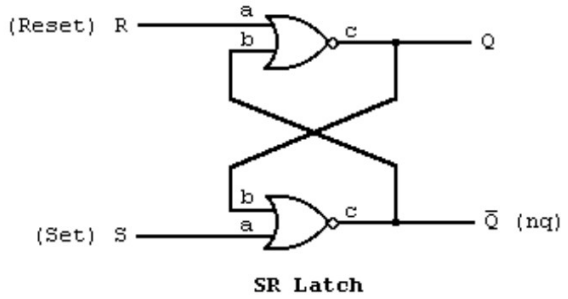
```
PORT MAP (X, Y, S1, S2)
```

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

Hier ist zu beachten, dass die Eingänge in die NOR-Gatter nicht direkt ein Ausgang (Q, nQ) verwendet werden darf. Es muss ein Puffer eingebaut werden (q <= q0) oder Q und nQ den Datentyp „inout“ erhalten. Achtung! Bei diesem Beispiel sind keine Signallaufzeiten angegeben, daher wird das

System abbrechen, wenn man in den verbotenen Bereich steuert (R=S=1) und diesen anschließend versucht abzuspeichern (R=S=0).

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

Inertial Delay (Trägheit) und **transport** Delay (Ideal)

- inertial Delay (Default) entspricht der Realität: wenn ein Puls kürzer ist als die Gatterlaufzeit (Trägheit) eines Gatters, dann wird er unterdrückt

-- transport Delay_ jeder noch so kleine Impuls wird weitergeleitet (Schlüsselwort „transport“)

```
entity tb is
end entity;

architecture behave of tb is
signal a,b ,c,x,in1,in2 : bit;
begin
 a <= '1' after 10 ns, '0' after 12 ns,'1' after 25 ns, '0' after 38 ns;

-- den ersten Puls von a kann man nicht sehen, weil die Transaktion bei der steigenden
Flanke (10ns)
-- durch die Transaktion bei der fallenden Flanke abgelöst wird.
-- b würde also bei 10ns auf  1 gehen, wird aber durch die neue Transaktion bei 12ns auf
Null gesetzt und
-- bleibt daher auf Null.
 b <= a after 5 ns;   --inertial delay
 c <= transport a after 5 ns;    --ideal transmission line, no pulses lost

x <= in1 and in2 after 10 ns;
```

```
in1 <= '1';
in2 <= '0','1' after 5 ns, '0' after 7 ns, '1' after 20 ns,'0' after 32 ns;




end behave;
```

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
```

Aber Achtung!

Zuweisungen an Signale erfolgen in der nächsten Zeitscheibe (wegen der Gatterlaufzeit);
Zuweisungen an Variable erfolgen sofort!

Richtig	Falsch

variable := a and b;
c <= variable or e;
d <= variable or f;
	c <= variable or e;
	d <= variable or f;
	variable := a and b;

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"

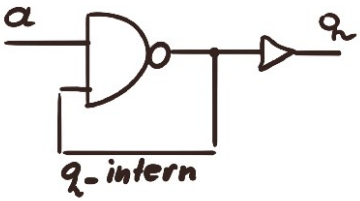
Das folgende Beispiel erzeugt einen 30ns breiten Puls bei 10ns, 110ns, 210ns usw.

```
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;
```

Der Simulator hängt sich auf: RS-FF

```
ENTITY rsff1 IS
 PORT (
 rn, sn : IN STD_LOGIC := '0';
 q, qn : INOUT STD_LOGIC);
END;

ARCHITECTURE arch OF rsff1 IS
 SIGNAL q_intern : STD_LOGIC := '1';
BEGIN
 q <= rn NAND qn;
 qn <= sn NAND q;
END arch;
```

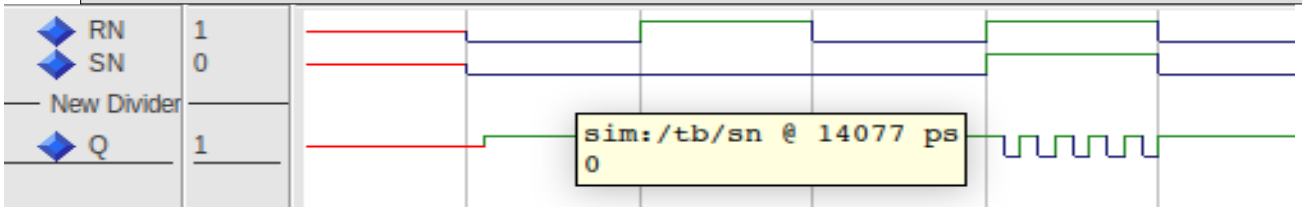
Die Gatter haben keine Laufzeit, der Ausgang wirkt sofort auf den Eingang zurück , das RS FF schwingt für den Testfall

```
ENTITY rsff1 IS
 PORT (
 rn, sn : IN STD_LOGIC := '0';
 q, qn : INOUT STD_LOGIC);
END;

ARCHITECTURE arch OF rsff1 IS
 SIGNAL q_intern : STD_LOGIC := '1';
BEGIN
 q <= rn NAND qn after 1 ns;
```

```
    qn <= sn NAND q after 1 ns;
END arch;
```

00 » 11 speichert den verbotenen Zustand ab ==> schwingt

```
PROCESS BEGIN
sn <= '0' AFTER 10 ns;
rn <= '0' AFTER 10 ns;
WAIT FOR 10 ns;
sn <= '0' AFTER 10 ns;
rn <= '1' AFTER 10 ns;
WAIT FOR 10 ns;
sn <= '0' AFTER 10 ns;
rn <= '0' AFTER 10 ns;
WAIT FOR 10 ns;
sn <= '1' AFTER 10 ns;
rn <= '1' AFTER 10 ns;
WAIT FOR 10 ns;

END PROCESS;
```



Mehrere Prozesse: warum ändert sich x und y nicht?

```
ENTITY tb IS
END;

ARCHITECTURE x OF tb IS
    SIGNAL x, y : BIT;
BEGIN
    p1 : PROCESS
    BEGIN
        x <= '1';
        WAIT FOR 7 ns;
        x <= '0';
    END PROCESS;
    p2 : PROCESS
        VARIABLE i : INTEGER;
    BEGIN
        -- x <= '0' after 2 ns; error "multiple sources"
        y <= '0';
        WAIT FOR 2.5 ns;
        y <= '1';
    END PROCESS;
END;
```

AW: x <= '0'; ändert x und veranlasst die sofortige Neuauswertung des Prozesses; es fehlt „wait for“

VHDL Die Programmiersprache

Datentypen

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. Sie gelten nur innerhalb des Prozesses in dem sie vereinbart 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;
```

Shared Variables (VHDL-93)

Vgl. globale Variable: Können in mehreren Prozessen gesehen werden.Ihr neuer Wert wird daher sofort wirksam, nicht erst nach Auswertung aller Prozesse. Das kann zu Problemen führen, wenn mehrere Prozesse gleichzeitig die Variable verändern.

VHDL Protected Type

Datenkapselung vgl. objektorientierte Programmierung.

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;
```

Typ Umwandlungen

Use ieee.numeric_std.all

Von	nach	Funktion
std_logic_vector	unsigned	unsigned(std_logic_vector)
unsigned	integer to_integer(unsigned)	
integer	unsigned	to_unsigned(integer, no_of_bits)
unsigned	std_logic_vector	std_logic_vector(unsigned)

integer in einen STD_LOGIC_VECTOR umwandeln

1. Umwandlung in unsigned
2. Umwandlung in STD_LOGIC_VECTOR bestimmter breite)

```
Address <= STD_LOGIC_VECTOR(to_unsigned(addr,4);
```

Beispiele:

Integer in einen STD_LOGIC_VECTOR umwandeln

1. Umwandlung in unsigned
2. Umwandlung in STD_LOGIC_VECTOR bestimmter breite)

addr_width und data_width sind Integer

STD_LOGIC_VECTOR(to_unsigned(2 ** addr_width - 1, data_width))

FIFO

FIFO als Ringpuffer. Der Speicher kann gleichzeitig geschrieben und gelesen werden. Es wird nicht geprüft, ob der Speicher leer oder voll ist.

```
LIBRARY ieee;
USE ieee.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY work;
USE work.txt_util.ALL;

ENTITY fifo IS
        GENERIC (
                addr_width : INTEGER := 3;
                data_width : INTEGER := 4);
```

```
        PORT (
                datain : IN STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0);
                dataout : OUT STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0);
                write_data, write_clk, read_clk : IN STD_LOGIC := '0'
        );
END ENTITY;

ARCHITECTURE behav OF fifo IS
        SIGNAL write_addr, read_addr : STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO 0) :=
(OTHERS => '0');
        TYPE memorytype IS ARRAY(0 TO 2 ** addr_width - 1) OF
STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0);
        SIGNAL memory : memorytype;
        --SIGNAL h : STD_LOGIC_VECTOR(data_width - 1 DOWNTO 0);

BEGIN
        writeP : PROCESS (write_clk)

                VARIABLE h : INTEGER;
        BEGIN
                IF (write_data = '1') THEN
                        IF rising_edge(write_clk) THEN
                                memory(to_integer(UNSIGNED(write_addr))) <= datain;
                                write_addr <= STD_LOGIC_VECTOR(unsigned(write_addr) +
1);
                                IF (write_addr > STD_LOGIC_VECTOR(to_unsigned(2 **
addr_width - 1, data_width))) THEN
                                        write_addr <= (OTHERS => '0');
```

```
                                END IF;

                                h := to_integer(UNSIGNED(write_addr));
                                print(str(h));
                                print(str(memory(h)) & " " & str(memory(1)) & " " &
str(memory(2)));

                                print (str(memory(to_integer(UNSIGNED(write_addr)))) & "
" & str(write_addr));
                        END IF;

                END IF;
        END PROCESS;

        readP : PROCESS (read_clk)
        BEGIN
                IF rising_edge(read_clk) THEN
                        dataout <= memory(to_integer(UNSIGNED(read_addr)));
                        read_addr <= STD_LOGIC_VECTOR(unsigned(read_addr) + 1);
                        IF (read_addr > STD_LOGIC_VECTOR(to_unsigned(2 ** addr_width -
1, data_width))) THEN
                                read_addr <= (OTHERS => '0');
                        END IF;

                END IF;
        END PROCESS;

END;
```

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