

VHDL Simulation

in ORCAD

**B
U
L
M
E**  **Höhere Technische
Bundes-Lehr- und
Versuchsanstalt
BULME Graz – Götting**

V1.0

F. Wolf

Graz, Jänner 2002

Inhaltsverzeichnis

| | | |
|----------|--|----------|
| 1 | <i>Einleitung</i> | 1 |
| 1.1 | Simulation und Verifikation | 2 |
| 1.2 | Entwurfsqualität | 2 |
| 1.3 | Begriffe in der Elektronik und ihre Äquivalenz zu VHDL | 3 |
| 1.4 | Testbench | 3 |
| 1.5 | Warum ORCAD ? | 4 |
| 2 | <i>Hochstarten der Simulationsumgebung</i> | 5 |
| 2.1 | Anlegen eines neuen Projektes | 5 |
| 2.2 | Anlegen des Quellcode in VHDL | 5 |
| 2.3 | Simulation | 7 |
| 2.4 | Starten der Simulation | 7 |
| 2.5 | Kompilieren | 10 |
| 2.6 | Anhang VHDL-Files | 11 |
| 2.6.1 | Modell | 11 |
| 2.6.2 | Testbench | 13 |

VHDL Simulation

in ORCAD

1 Einleitung

Bei einer Simulation von VHDL müssen die Eingangssignale, die einer Komponente hineinführen, stimuliert werden (Testbench). Anschließend müssen die Ausgangssignale, die von der Komponente nach außen führen, verifiziert werden.

Wird rein das VHDL Modell simuliert spricht man von einer „funktionalen Simulation“, also ohne die Prüfung der Zeitverhalten der Schaltung. Erhält die Netzliste die bei jeder Simulation generiert wird, schon Verzögerungszeiten für die aus der Bibliotheken eingesetzten Elemente (Gate Level Simulation), können wir bereits an dieser Stelle eine Timing-Simulation durchführen (Einheitsverzögerungszeiten) . Die Simulation in dieser Phase wird Pre-Layout-Simulation genannt. Da die exakte Verzögerungszeiten erst nach der Verdrahtung feststellbar sind, liefert die Pre-Layout-Simulation nur eine angenäherten Eindruck über das Zeitverhalten der Schaltung.

Werden bei der Simulation Fehler entdeckt, so wird die Schaltung korrigiert und die Verifikation nochmals durchgeführt. Nach zufriedenstellender Verifikation wird nun die Synthese durchgeführt. Mittels Synthese wird aus der VHDL-Beschreibung eine Entwurfsdarstellung auf Logikebene erzeugt entweder in Form einer Netzliste oder einer weiteren VHDL-Beschreibung. Dabei werden die Elemente in dem CPLD (FPGA) platziert und verdrahtet (Place&Route). Dazu werden meistens automatische Werkzeuge verwendet, die über Vorgabe (Constraints) gesteuert werden. Die endgültige Laufzeiten der Signale werden nach dem Platzieren und Verdrahten berechnet Diese Laufzeiten und auch Angaben über die Platzierung können automatisch in den Schaltplan, bzw. in die entsprechenden Netzliste eingetragen werden (Backannotation).

Mit diesen endgültigen Laufzeiten können wir jetzt mit einer Post-Layout-Simulation das zeitliche Verhalten der Schaltung genau untersucht werden.

Der schaltplanbasierte Entwurf stößt bei der immer größer werdenden CPLD's (FPGA's) schnell an seine Grenzen. Spätestens dann ist der Übergang zu einem Entwurf mit Hardwarebeschreibungssprachen (meistens VHDL) notwendig.

1.1 Simulation und Verifikation

Die Simulation dient zur Überprüfung des Schaltungsentwurfs. Mit Hilfe von den Entwicklern erstellten Eingangsdaten-Kombinationen (Input Stimuli, Testbench ...) wird getestet, ob die Schaltung die erwarteten Ergebnisse an ihren Ausgängen liefert. Damit lassen sich eventuelle Fehlern nachweisen. Leider kann man in der Regel jedoch die Fehlerfreiheit einer Schaltung nicht vollständig nachweisen. Diese wäre nur dann möglich, wenn wir die Schaltung auf alle möglichen Eingangskombinationen überprüfen würden.

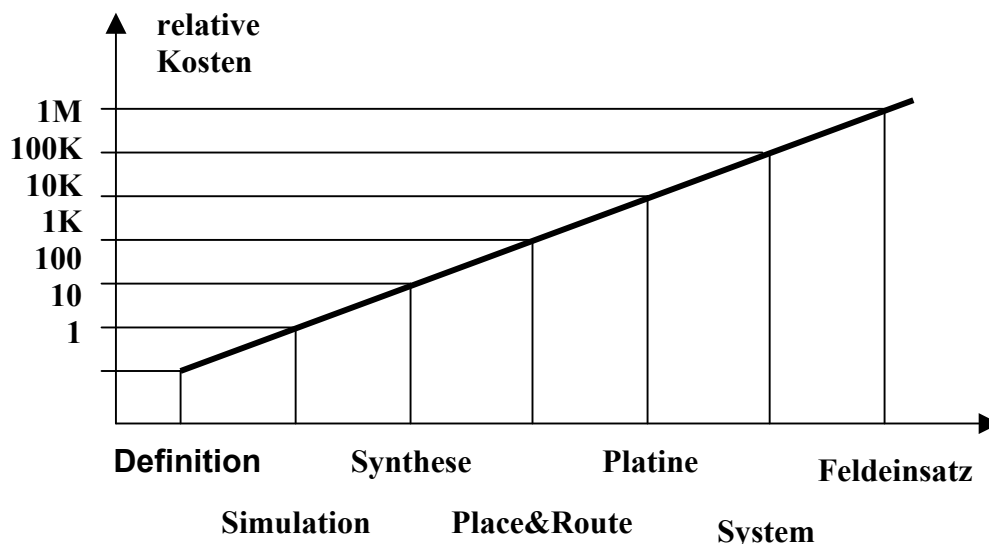
Verifikation ist der Nachweis, das ein aus Komponenten einer niedrigeren Ebene zusammengesetztes System das in der Spezifikation beschriebene Verhalten zeigt. Die automatische Verifikation ist noch eine Forschungsthema, und wird in der Regel durch eine Simulation von Fallbeispielen ersetzt.

1.2 Entwurfsqualität

Die Überprüfung des Entwurfs auf mögliche Fehler ist für den Erfolg eines Entwicklungsprojektes unbedingt notwendig. Wie die Erfahrungen aus der Praxis zeigen, enthalten leider Schaltungen ab einer gewissen Komplexität mit hoher Wahrscheinlichkeit Fehler. Die Überprüfung erfolgt in der Entwicklungsphase mit Simulation und Verifikation. Naher beim Prototypenaufbau kommt der erste Test im Labor oder sogar schon in einer realen Umgebung. In der Produktionsphase werden die Bauteile dann nicht mehr auf Entwurfsfehler, sondern auf Produktionsfehler getestet.

Neben einem fehlerfreien Entwurf ist ein weitere sehr wichtige Ziel, Entwicklungsfehler so früh wie möglich zu entdecken. Ganz einfach gesagt, je später ein Fehler entdeckt wird, desto teurer ist seine Beseitigung (und eventuell die verursachten Schaden).

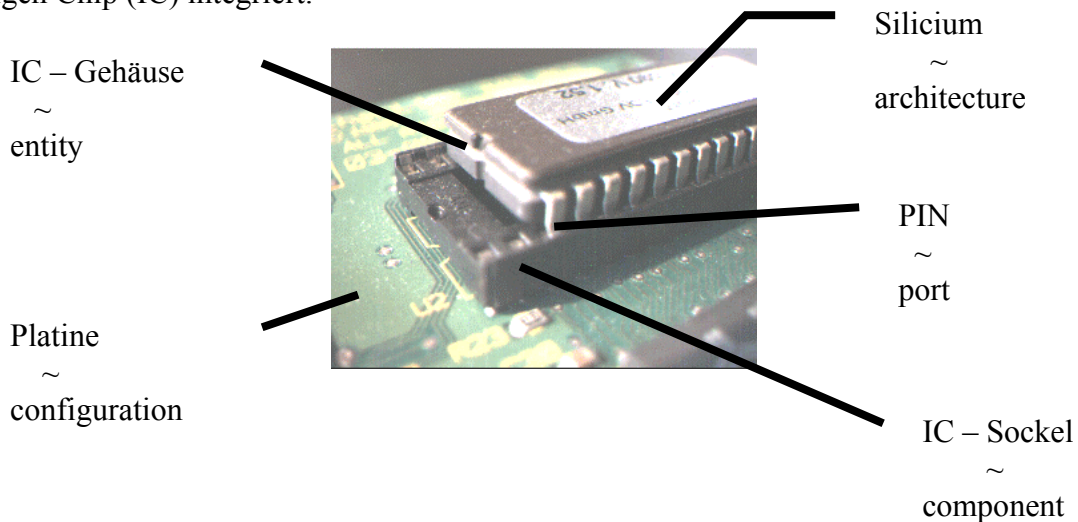
Ein wichtiger Faustregel: bei jedem Entwurfsschritt kann ein Kostenfaktor von ca. 10 auftreten:



1.3 Begriffe in der Elektronik und ihre Äquivalenz zu VHDL

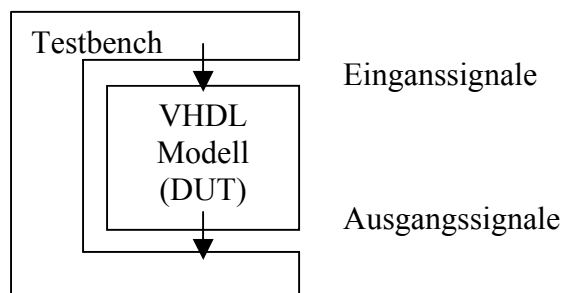
| Elektronik | VHDL |
|---------------------------------------|-------------------------|
| Silizium (als Technologie für den IC) | architecture |
| IC | entity |
| Pin | port |
| Norm, Bauteilkatalog | package |
| unbestückter IC-Sockel | component instantiation |
| Leiterbahn mit zugehörigen Lötungen | signal |
| Bestückungsplan | configuration |

In dem untenstehenden Bild ist nochmals der Zusammenhang zwischen den Begriffen in der Elektronik und der Gegenüberstellung zu VHDL dargestellt. Es soll extra drauf hingewiesen werden, dass dies nur eine bildliche Darstellung entspricht. VHDL als Gesamtheit ist in einem einzigen Chip (IC) integriert.



1.4 Testbench

Eine Homogene und von einem Simulationsprogramm unabhängige Simulationsumgebung erhält man, wenn man die zu testende Komponente (Modell) in eine sogenannte „Testbench“ (Bench: engl. Arbeitstisch, Werkbank) einbindet. Eine Testbench ist wiederum ein VHDL Programm, welches Stimuli für die zu testende Komponente erzeugt und in zeitlicher sinnvoller Abfolge die von den Komponente erzeugten Ausgangssignale prüft. Die zu testente Komponente wird als „DUT“ Device under test bezeichnet. Die Entity einer Testbench ist immer leer, da es nach außen hin zum Simulator keine Ein- und Ausgangssignale gibt.



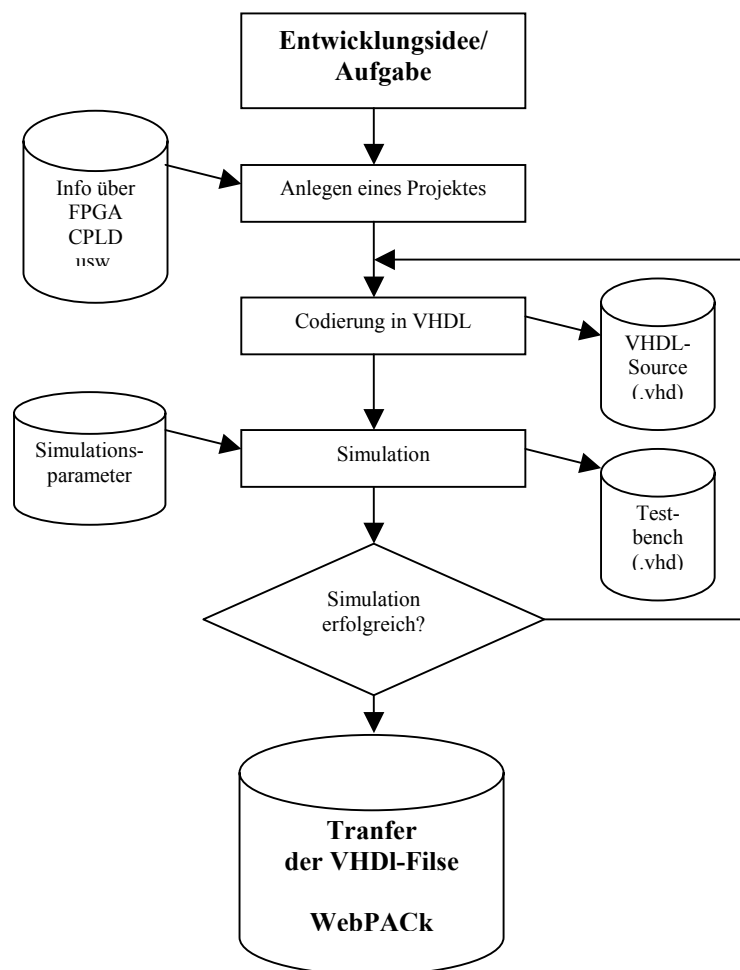
Struktur einer Simulationsumgebung

1.5 Warum ORCAD ?

ORCAD eignet sich sehr gut als Simulator hier an dieser Schule, weil es fast überall als Vollversion vorhanden ist. ModelSim wird von Xilinx in der Designumgebung WebPACK mitgeliefert, hat aber einige Nachteile. So muss für jeden PC an dem simuliert werden soll, direkt unter Xilinx per Internet einen Licens-File für ModelSim XE/Starter 5.3d bestellt werden. Diese Lizenz gilt nur an den einen PC auf dem man die Bestellung durchgeführt hat. Hat man dies erfolgreich getan, so ist man bei dieser ModelSim Version nur berechtigt eine Simulation durchzuführen. Bei jeder weitem Simulation muss der Simulator neu gestartet werden. Aus diesen Gründen ist eine Modelentwicklung unter ORCAD vorteilhaft.

Für die Synthese bzw. das Herunterladen in die Zielhardware ist WebPACK hervorragend geeignet.

Schematische Darstellung der Entwicklungsschritte mit ORCAD

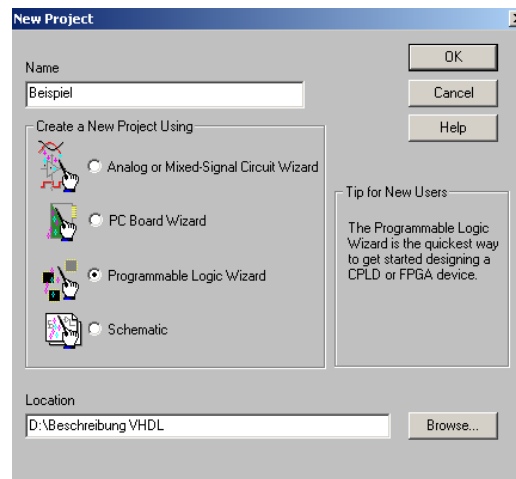


2 Hochstarten der Simulationsumgebung

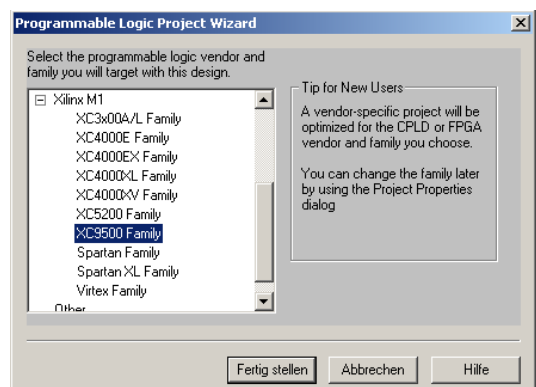
2.1 Anlegen eines neuen Projektes

ORCAD wird über das Windowsstartmenü gestartet
Danach wird ein neues Projekt angelegt

FILE ⇒ NEW ⇒ Project



Wählen Sie die Bausteinfamilie aus, für die das Projekt angelegt werden soll



2.2 Anlegen des Quellcode in VHDL


Design ⇒ New VHDL-File

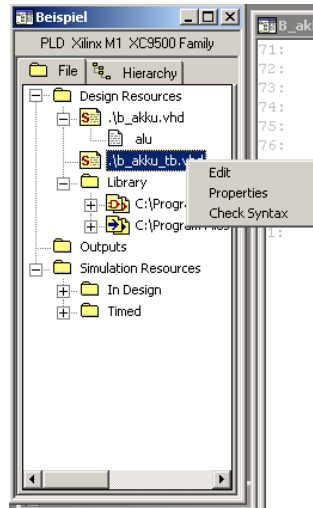
Es erscheint ein Fenster, Speichern unter, dort ist der Dateityp VHDL-File (*.vhd) schon ausgewählt ! Dieser Schritt muss zweimal durchgeführt werden.

1. Modell
2. Testbench

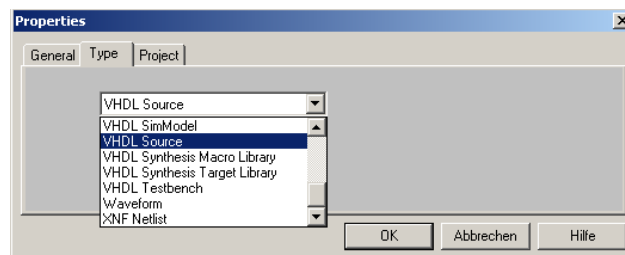
Als vorteilhaft ist es den Namen der Testbench so zu bezeichnen, das er „_tb“ (Testbench) als bestandteil des Namens enthält.. z.B.: B_akku_tb.vhd

Definition der VHDL-Files

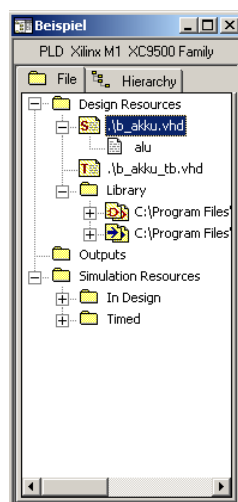
Als Standard ist Source „S“ als Filetyp definiert. Mit der rechten Maustaste  auf den File klicken, hier B_akku_tb.vhd. Es erscheint ein neues Fenster:



Properties auswählen , und als Type VHDL Testbench definieren.



Es muss nun folgendes im Projektfenster angezeigt werden:



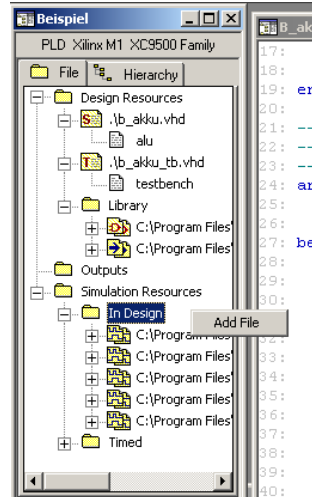
S ...steht für Source File

T...steht für Testbench

2.3 Simulation

Nun fügt man die beiden Files (Source, Testbench) unter SimulationResources dazu.

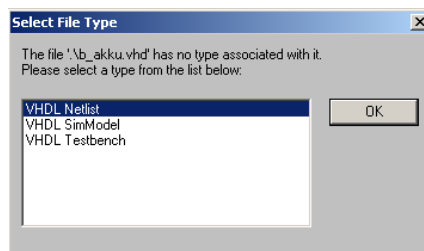
SimulationResources ⇒ In Design  Rechte Maustaste ⇒ Add File dazu.



Nun wird definiert:

Source File → VHDL SimModel

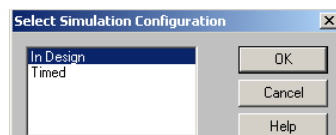
Testbench → VHDL Testbench



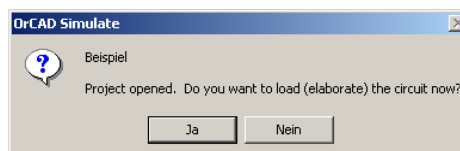
2.4 Starten der Simulation

Tools ⇒ Simulate

Es erscheint dieses Fenster

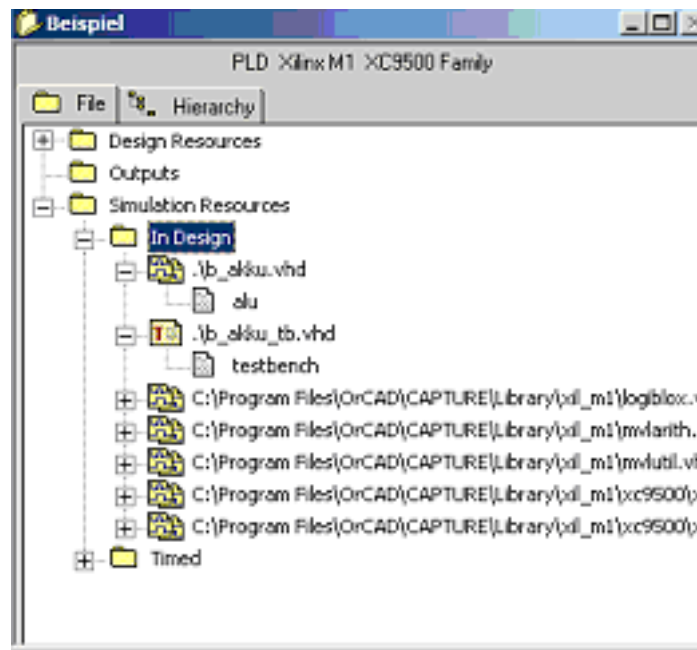


Hier gibt es die Möglichkeit eine Zeitsimulation oder ein das Funktionsmodell zu simulieren: Wir wählen Sie „In Design“ es erscheint:

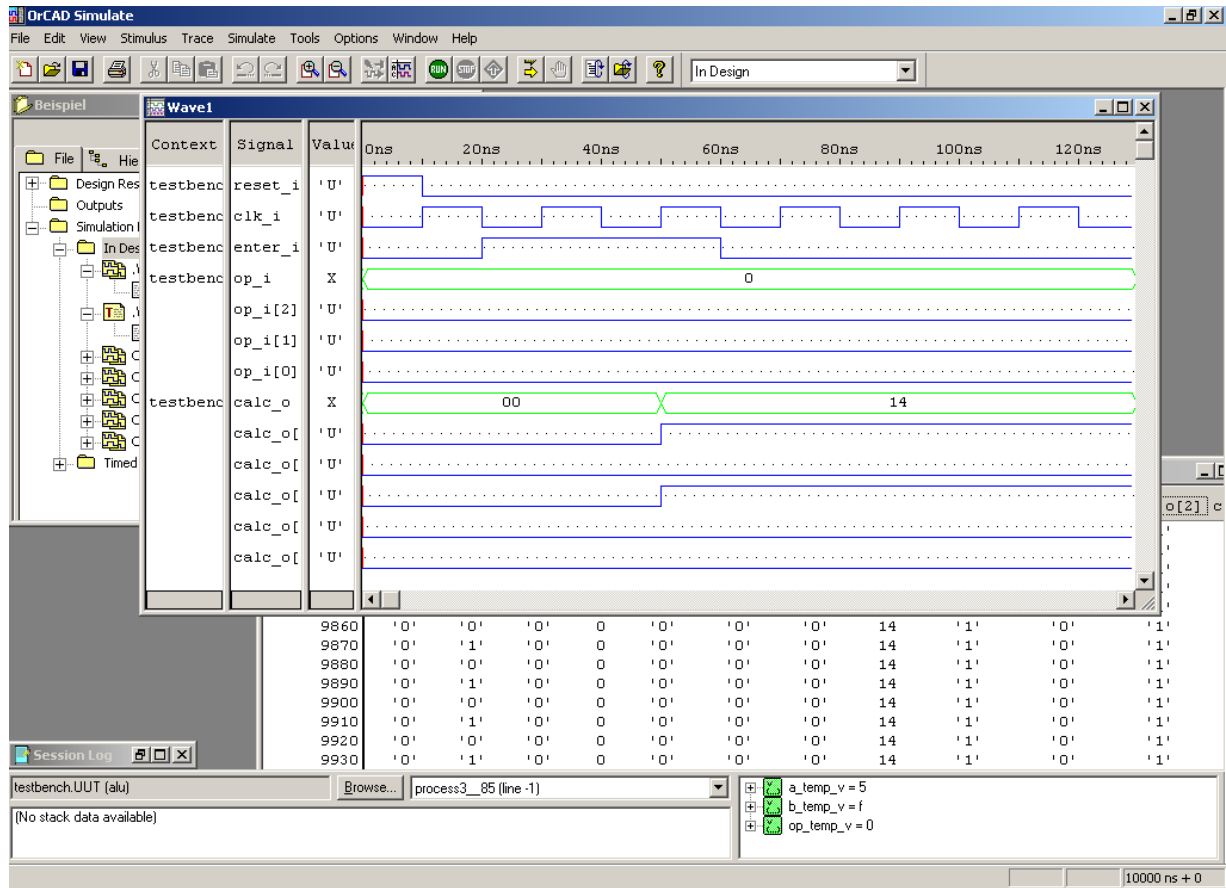


hier wählen wir „Ja“

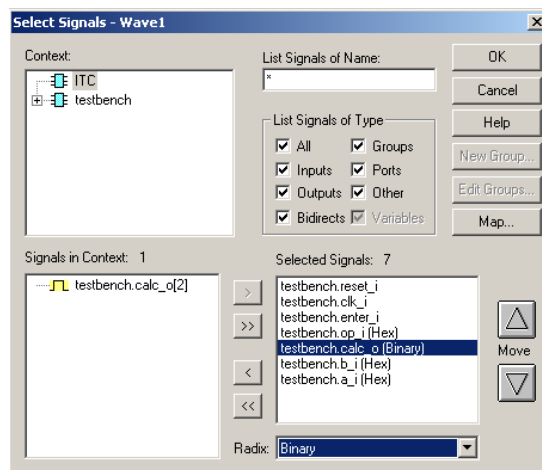
Es erscheint nun die Simulationsumgebung, gestartet wir die Simulation mit RUN



Es erscheine nun ein Fenster, wo man die Simulationsergebnisse ansehen kann. Das Ergebnis der Simulation wird zumeinen als Liste, zum anderen als Waveform Übersicht angezeigt. Die Dauer der Simulationszeit können Sie unter Option ⇨ Preferences ändern. Default sind 10 000 ns eingestellt.



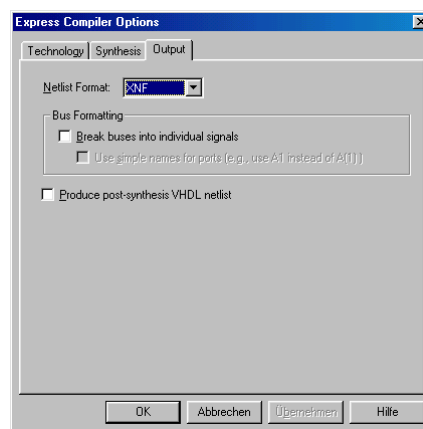
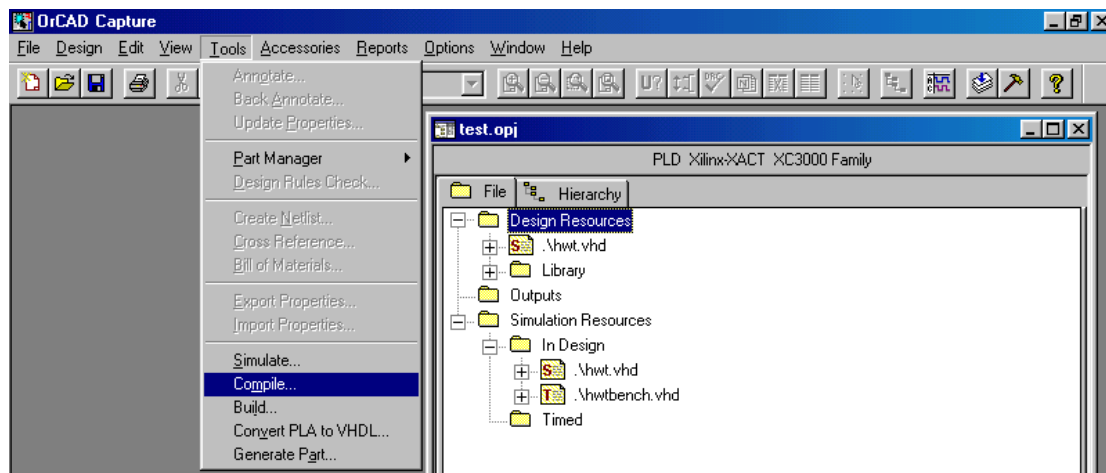
Unter Trace ⇒ Edit Signal Trace können nun Signale dazu addiert werden oder der Radix HEX, OCDAL oder Binary ausgewählt werden



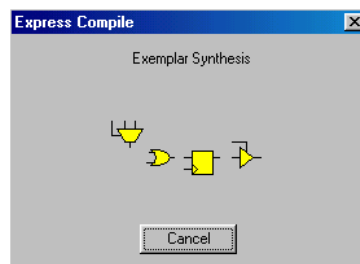
2.5 Kompilieren

Nachdem die von Ihnen erstellte Schaltung simuliert und ihre Funktion überprüft wurde, können Sie nun die Netzliste erstellen. Die Netzliste ist die Schnittstelle zum Programm Xilinx Xact 6. Gehen Sie beim Kompilieren folgendermaßen vor:

Starten des Compilers



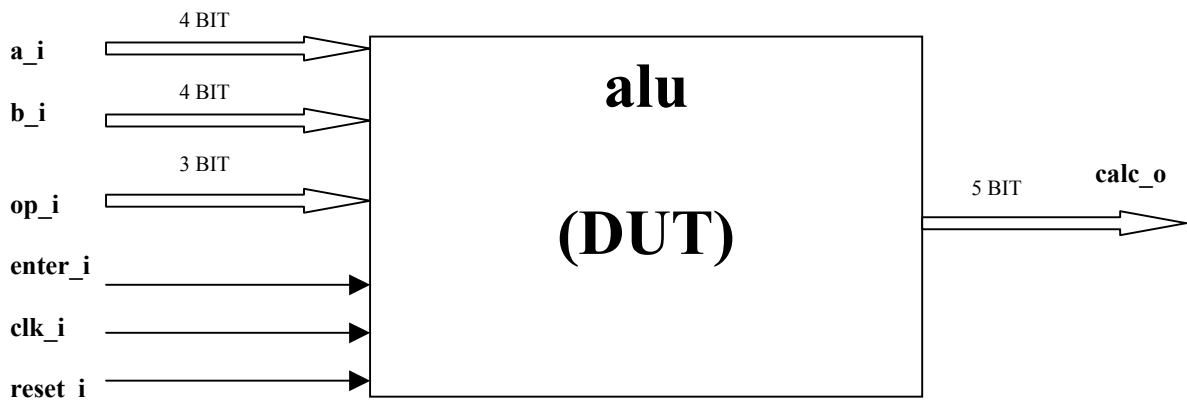
Im „*Express Compiler*“-Fenster sind alle Optionen zum kompilieren bereits eingestellt. Achten Sie lediglich darauf, dass die Netzliste im *XNF-Format* erstellt wird. Mit OK starten Sie den Kompiliervorgang.



Als Ergebnis wird die Netzliste im Unterordner *Outputs* des angelegten Projektes als XNF-Datei abgelegt.

2.6 Anhang VHDL-Files

Hier ist als Beispiel ein einfaches Modell einer ALU inklusive Testbench dargestellt. Es soll durch Eingabe zweier 4 Bit Vektoren (a_i , b_i) verschiedene Operationen durchgeführt werden. Die Operation kann durch eine 3 Bit Vektor (op_i) eingestellt werden. Durch Betätigen von $enter_i$ wird die Berechnung durchgeführt, und das Ergebnis am Ausgang ($calc_o$) sichtbar. Weiters ist ein Takt (clk_i) und ein Reset ($reset_i$) verwendet worden.



2.6.1 Modell

```

-- Version 1.0
-- 20.10.2001
-- design & simulated with orcad
--F. Wolf
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-----
-- ***** ENTITY *****
-----
entity alu is
  port (
    a_i    : in std_logic_vector(3 downto 0);
    b_i    : in std_logic_vector(3 downto 0);
    calc_o : out std_logic_vector(4 downto 0);
    op_i   : in std_logic_vector(2 downto 0);
    enter_i : in std_logic;
    clk_i  : in std_logic;
    reset_i : in std_logic
  );
end alu;

-----
-- ***** ARCHITECTURE *****
-----

```

```

-----
architecture behave of alu is
  type states is (value_input, calculate);
  signal com_valid : states;
begin
  process (clk_i, reset_i)
    variable a_temp_v : std_logic_vector(4 downto 0);
    variable b_temp_v : std_logic_vector(4 downto 0);
    variable op_temp_v : std_logic_vector(2 downto 0);
  begin
    if (reset_i='1') then
      calc_o <=(others =>'0');
      a_temp_v :=(others =>'0');
      b_temp_v :=(others =>'0');
      op_temp_v :=(others =>'0');
      com_valid <= value_input;
    elsif (clk_i'event and clk_i='1') then
      case com_valid is
        -----
        when value_input =>
          if enter_i='1' then
            a_temp_v(3 downto 0) := a_i;
            b_temp_v(3 downto 0) := b_i;
            op_temp_v := op_i;
            com_valid <= calculate;

          else
            com_valid <= value_input;
          end if;
        -----
        when calculate =>
          case op_temp_v is
            -----
            when "000" => --- algebra summation
              calc_o <= a_temp_v + b_temp_v;
            -----
            when "001" => --- algebra subtraktion
              calc_o <= a_temp_v - b_temp_v;
            -----
            when "010" => --- logic and
              calc_o <= a_temp_v and b_temp_v;
            -----
            when "011" => --- logic or
              calc_o <= a_temp_v or b_temp_v;
            -----
            when "100" => --- logic xor
              calc_o <= a_temp_v xor b_temp_v;
            -----
            when "101" => --- logic nand
              calc_o <= a_temp_v nand b_temp_v;
            -----
            when "110" => --- logic nor
              calc_o <= a_temp_v nor b_temp_v;
            -----
            when "111" => --- logic not a_i
              calc_o <= not a_temp_v;
            -----
          when others => null;
        end case;
        -----
      when others => null;
    end case;
  end if;
end process;
-----

```

```

    end process;
end behave;

```

```

-----
-- ***** CONFIGURATION *****
-----
configuration alu_conf of alu is
    for behave
        end for;
end alu_conf;

```

2.6.2 Testbench

```

-- Version 1.0
-- 20.10.2001
-- design & simulated with orcad
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

LIBRARY ieee;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY testbench IS
END testbench;

-----
-- ***** ARCHITECTURE *****
-----
ARCHITECTURE testbench_arch OF testbench IS
    COMPONENT alu
        PORT (
            a_i      : in std_logic_vector(3 downto 0);
            b_i      : in std_logic_vector(3 downto 0);
            calc_o   : out std_logic_vector(4 downto 0);
            op_i     : in std_logic_vector(2 downto 0);
            enter_i  : in std_logic;
            clk_i    : in std_logic;
            reset_I  : in std_logic
        );
    END COMPONENT;

    SIGNAL a_i      : std_logic_vector(3 downto 0);
    SIGNAL b_i      : std_logic_vector(3 downto 0);
    SIGNAL calc_o   : std_logic_vector(4 downto 0);
    SIGNAL op_i     : std_logic_vector(2 downto 0);
    SIGNAL enter_i  : std_logic;
    SIGNAL clk_i    : std_logic;
    SIGNAL reset_i  : std_logic;

BEGIN

    UUT : alu
    PORT MAP (

```

```

        a_i    => a_i,
        b_i    => b_i,
        calc_o => calc_o,
        op_i   => op_i,
        enter_i => enter_i,
        clk_i  => clk_i,
        reset_i => reset_i
    );

-----
-- ***** C L O C K *****
-----
    PROCESS
    BEGIN
        -----
        clk_i <= transport '0';
        -----
        WAIT FOR 10 ns;
        clk_i <= transport '1';
        -----
        WAIT FOR 10 ns;
        clk_i <= transport '0';
    END PROCESS;

-----
-- ***** F O R   S T I M U L I *****
-----
    PROCESS
    BEGIN
        -----
        enter_i  <= transport '0';
        a_i      <= transport std_logic_vector("0000");
        b_i      <= transport std_logic_vector("0000");
        op_i     <= transport std_logic_vector("000");
        reset_i  <= transport '1';
        -----
        WAIT FOR 10 ns;
        reset_i <= '0';
        -----
        WAIT FOR 10 ns;
        a_i      <= transport std_logic_vector("0101");
        b_i      <= transport std_logic_vector("1111");
        op_i     <= transport std_logic_vector("000");
        enter_i  <= transport '1';
        -----
        WAIT FOR 40 ns;
        enter_i <= transport '0';
        -----
        wait;
        WAIT FOR 90 ns;
        code_i <= transport std_logic_vector("11000001");
        -----
        WAIT FOR 40 ns;
        enter_i <= transport '1';
        -----
        WAIT ; -- global
        -----
    END PROCESS;
END testbench_arch;

```

```
-----  
-- ***** CONFIGURATION *****  
-----  
CONFIGURATION alu_cfg OF testbench IS  
  FOR testbench_arch  
  END FOR;  
END alu_cfg;
```