# Formal methods for fun and profit

# VHDL


Ilja Kipermann

Sommersemester 2005

Leitung Jun. Prof. Beckert

Universität Koblenz-Landau

# Einleitung

Früher: Manuelles Zeichnen von Belichtungsmasken

Heute: Hardwarebeschreibungssprachen

Beschreibung *was* ein Hardwarebaustein tut nicht *wie* er es tut.

Anwendungsgebiete:

FPGAs (Field programmable gate arrays)
ASICs: (Application Specific Integrated Circuits)

# Geschichte von VHDL

Man benötigte eine Sprache die nicht nur maschinen-, sondern auch menschenlesbar ist.

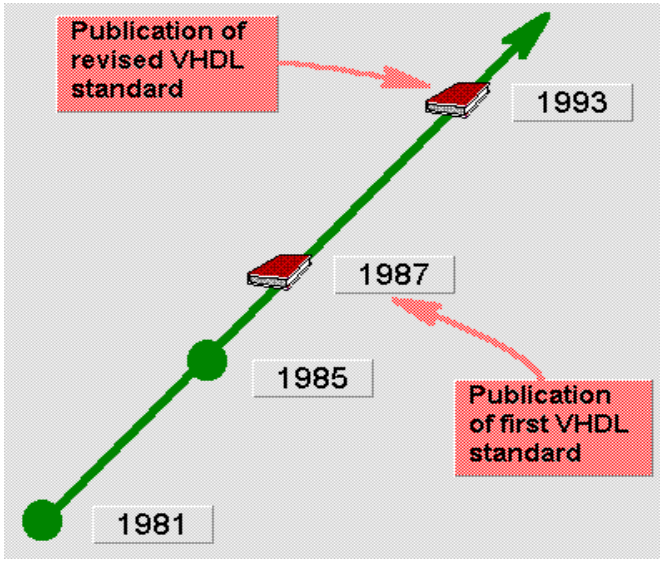
ende 70er - Anregung durch das U.S. Verteidigungsministerium

1983-85 – Entwicklung der Grundlagen von Intermetrics, IBM and Texas Instruments

1987 – Publikation als IEEE Standard

1993 – Aktualisierter Standard (VHDL 1076-1993)

1999 – Erweiterung um analoge Elemente (VHDL-AMS)

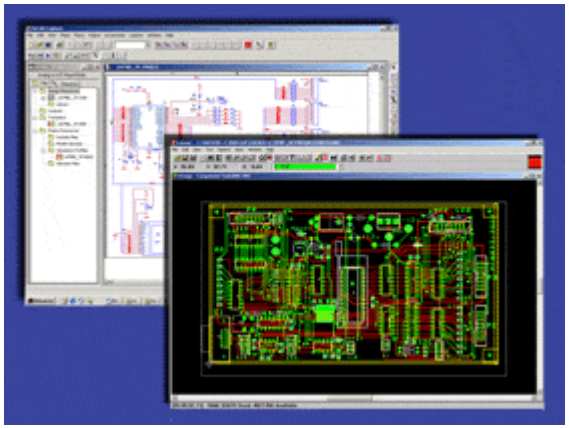



# Was ist VHDL?

- VHDL ist eine Hardwarebeschreibungssprache.
- Beschreibt das Verhalten und die Struktur elektronischer Bausteine.
- Präzise und Komplett definiert im Language Reference Manual (LRM)
- IEEE-Standard und nicht propietär

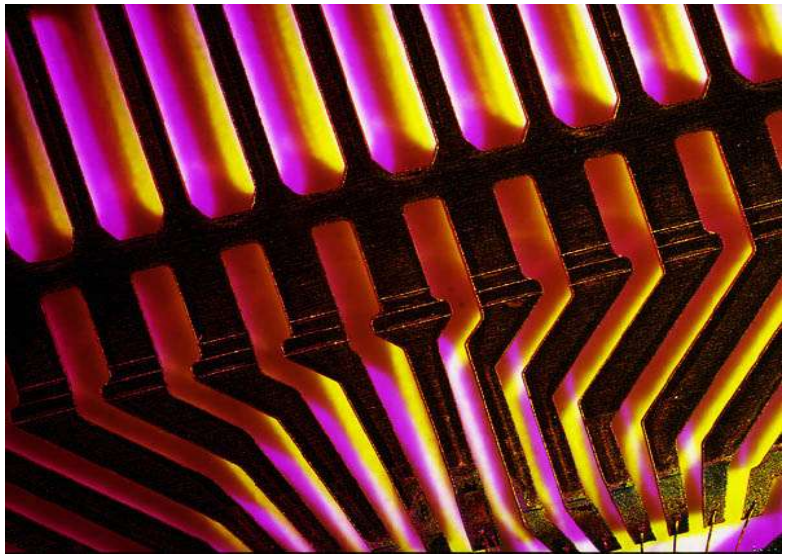
# Was ist VHDL?

- VHDL ist eine Sprache
- Zum effektiven Einsatz benötigt man eine
  - Sprache
  - geeignete Methodik
  - Satz an Softwarewerkzeugen
- Wichtigste Methoden: Simulation und Synthese
- VHDL komplett simulierbar
- Das LRM definiert eindeutig wie ein Simulator die Sprache verarbeitet



# Konzepte von VHDL

- Zweierlei  Arten von Anweisungen bei VHDL: Sequenzielle, Nebenläufige
- Nebenläufige Anweisungen können Parallelität echter Hardware nachbilden
- 3 Wichtige Modellierungsmethodiken: Abstraktion, Modularität, Hierarchie



# Abstraktion

- Verschiedene Teile des Modells mit verschiedenen Detailstufen
- Nur simulationsrelevante Module nicht so detailliert wie Funktionsrelevante Module
- Es wird zwischen wichtigen und (aktuell) unwichtigen Informationen unterschieden
- Abstraktionsgrad kann im Verlaufe des Entwicklungsprozesses verfeinert werden

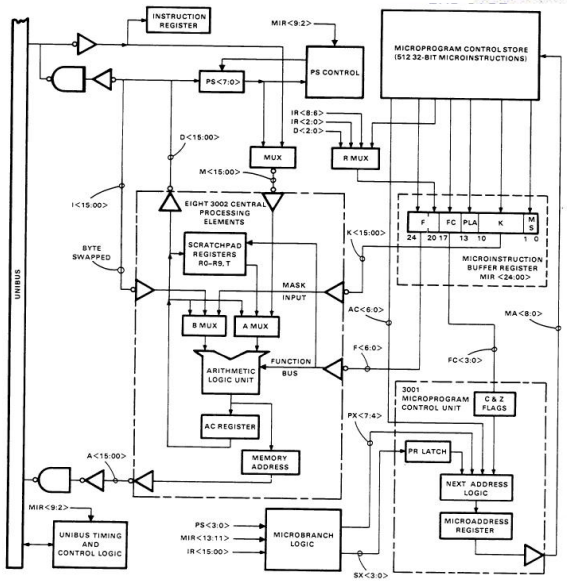
# Modularität und Hierarchie

- Erlaubt es große Funktionsblöcke in kleinere abgeschlossene Module zu Unterteilen
- Aus Modulen, die auch Untermodule besitzen können kann ein System aufgebaut werden
- Verschiedene Hierarchieebenen
- Ein oder mehrere Module mit unterschiedlichem Abstraktionsgrad bilden eine Hierarchieebene, die darin enthaltenen Untermodule bilden die Hierarchieebene darunter.
- Teilbeschreibungen aus unteren Ebenen werden als Instanzen an übergeordnete Module weitergegeben

# Abstraktionsebenen

- VHDL legt keinen Beschreibungsstil fest.
- Bottom up, top down, middle out
- Drei verschiedene Abstraktionsebenen
  - Verhaltensmodellierung
  - Register-Transfer-Level (RTL)
  - Gatterebene
- Hauptunterschied: Timing

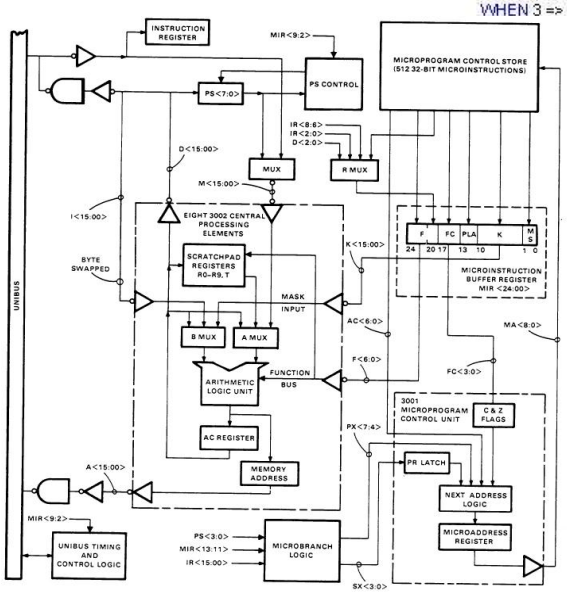
```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
                clock_2: IN STD_LOGIC;
                sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```




# Entwicklungsprozess

1. Informelle Spezifikation
2. Simulierbare formale Beschreibung (Verhaltensmodellierung)
3. Erste Simulation
4. Abstraktionsverfeinerung (RTL)
5. Synthese
6. Einzelne Bausteine zu Gesamtschaltung verbinden ( Gatterebene)
7. Tatsächliche Physik ( ausserhalb VHDL-Blickwinkels)

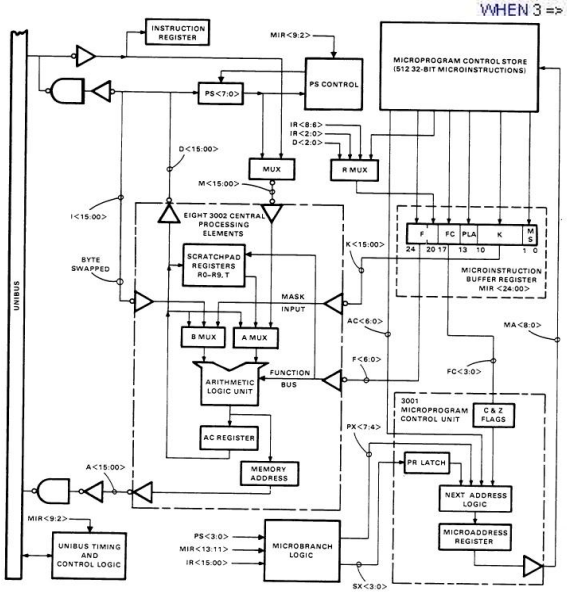
```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
              clock_2: IN STD_LOGIC;
              sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```




# Verhaltensmodellierung

- Satz von Anweisungen für ein bestimmtes Ergebnis
- Simulierbar, Performanz und Funktionalität prüfen
- nicht Synthetisierbar
- berücksichtigt keinen Takt

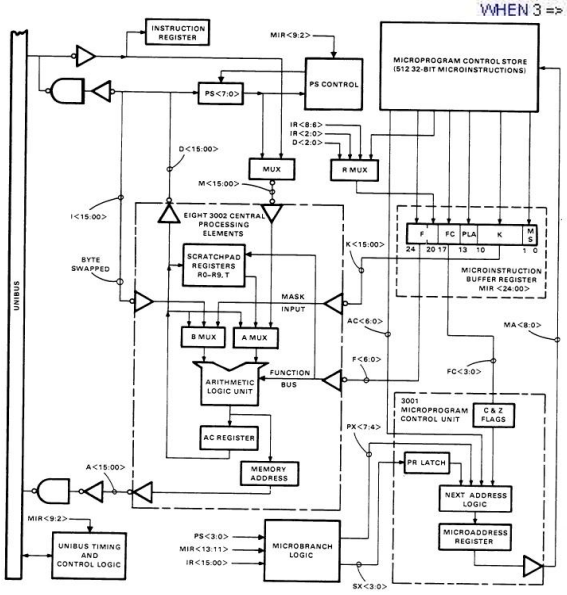
```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
              clock_2: IN STD_LOGIC;
              sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```




# Register-Transfer-Ebene

- Beschreibung des Datenflusses zwischen Registern, in welchen Informationen zwischen einzelnen Zeitschritten gespeichert werden
- spezifiziert, wie und wo Information gespeichert und während einer Operation weitergeleitet wird
- berücksichtigt explizit den Takt
- alle Operationen getimed um zu bestimmten Taktzyklen ausgeführt zu werden
- keine Delays
- Simulierbar
- Eingabeformat für die Synthese

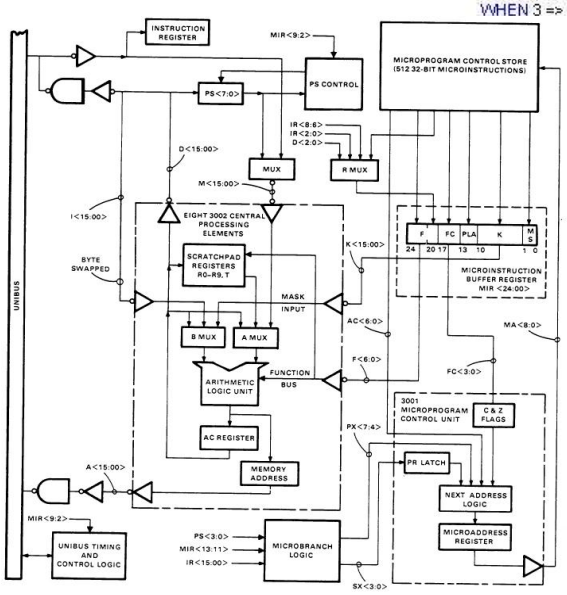
```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
              clock_2: IN STD_LOGIC;
              sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```




# Maschinelle Synthese

- automatisiertes Werkzeug
- schafft aus RTL-Modell eine Schaltungsdarstellung auf Gatter-Ebene
- VHDL-Code für bestimmtes Werkzeug zugeschnitten
- manchmal manuelle Synthese zur Flächen-, Energie- oder Taktratenoptimierung. Zeitbedarf und Gewinn abwägen
- Ausgabe: Beschreibung der Schaltung bezüglich tatsächlicher Komponenten und die logische Struktur zwischen ihnen

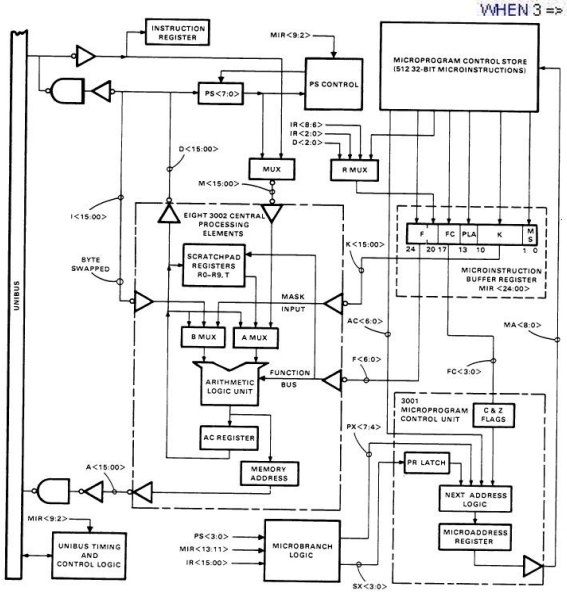
```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
              clock_2: IN STD_LOGIC;
              sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```




# Gatterebene

- Beschreibung vom Netzwerk von Gattern und Registern
- instanzieert über eine Technologiebibliothek, in der technologiespezifische Delays enthalten sind
- Beschreibung des Bausteins in seiner Funktionsweise mit Takt und Verzögerungsinformationen

```
USE std.textio.all;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ctr IS
        PORT (reset: IN STD_LOGIC;
              clock_2: IN STD_LOGIC;
              sel_digit: OUT STD_LOGIC_VECTOR(0 to 1));
END ctr;
ARCHITECTURE rtl OF ctr IS
BEGIN
PROCESS (clock_2, reset)
        VARIABLE count: integer RANGE 0 TO 3;
BEGIN
        IF (reset = '0') THEN
                count := 0;
                sel_digit <= "00";
        ELSIF rising_edge(clock_2) THEN
                IF count = 3 THEN
                        count := 0;
                ELSE
                        count := count + 1 ;
                END IF;
                CASE count IS
                        WHEN 0 => sel_digit <= "00";
                        WHEN 1 => sel_digit <= "01";
                        WHEN 2 => sel_digit <= "10";
                        WHEN 3 => sel_digit <= "11";
```



# 8-bit Komparator

```
ENTITY compare IS
        PORT( A, B: IN bit_vector(0 TO 7);
                EQ: OUT bit);
END compare;
ARCHITECTURE compare1 OF compare IS
BEGIN
        EQ <= 1 WHEN (A = B) ELSE 0;
END compare1;
```

- Akzeptiert zwei 8-bit Inputs, vergleicht sie, und gibt ein 1-bit Ergebniss aus. 1 bei Übereinstimmung 0 bei Differenz

- mindestens eine *Entity* und eine *Architecture*
- *Entity* beschreibt die Schaltung von “Ausserhalb”
- Zu jeder *Entity* eine *Architecture*
- *Architecture* beinhaltet tatsächliche Verhaltensbeschreibung

# ENTITY

```
ENTITY compare IS
        PORT( A, B: IN bit_vector(0 TO 7);
                EQ: OUT bit);
END compare;
ARCHITECTURE compare1 OF compare IS
BEGIN
        EQ <= 1 WHEN (A = B) ELSE 0;
END compare1;
```

- Das komplette Interface für die Schaltung
- Namen, Datentypen und Ports
- Alle Informationen um diesen Teil mit anderen zu verbinden oder Input-Stimuli zu entwickeln für eine Simulation
- tatsächliche Funktionsweise nicht enthalten

# ARCHITECTURE

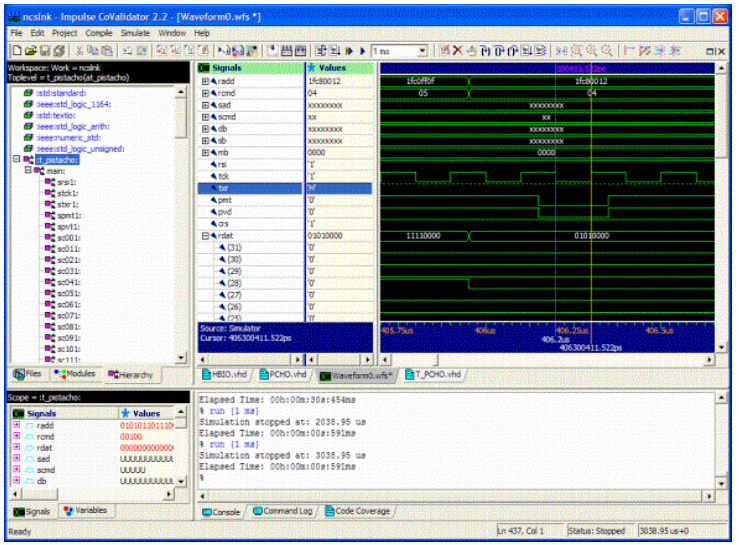
```
ENTITY compare IS
        PORT( A, B: IN bit_vector(0 TO 7);
                EQ: OUT bit);
END compare;

ARCHITECTURE compare1 OF compare IS
BEGIN
        EQ <= 1 WHEN (A = B) ELSE 0;
END compare1;
```

- Name notwendig da mehrere Architectures möglich
- Durch Hierarchien und Unterprogramm-Features ist es möglich Komponenten von tieferen Abstraktionsebenen einzufügen
- Beim Schritt zur nächsten Abstraktionsebene wird der Code zum RTL level umgeschrieben und erweitert

# Formale Methoden und VHDL

- gängige Praxis auf Korrektheit zu überprüfen: Simulation mit Testbenches mit zufallsgenerierten oder manuellen Eingaben
- Bugs sind sehr teuer
- sicherheitskritische Anwendungen
- Formale Verifikation sehr wichtig
- Aufgrund fehlender Semantik auf  Verhaltensebene kaum durchführbar
- Verschiedene Ansätze der formalem Verifikation mit VHDL, jedoch noch eher ein Forschungsgebiet



*Simulationswerkzeug*

# Formal equivalence checking

- Register-Transfer-Ebene als Referenz
- Synthese zu Gatter-Ebene
- Theoretisch muss ein Synthesewerkzeug logische Äquivalenz garantieren:
  In der Praxis: Bugs, manuelle Optimierungen
- Idee: Anstatt Äquivalenz einzelner Ausdrücke mit Hilfe der symbolischen Logik Ausdrücke benutzen die eine ganze Bandbreite an Inputs representieren. Äquivalenz gilt dann für alle In- und Outputs.
- Nachteile:

  Algorithmen sehr komplex und funktionieren nicht so gut bei größeren Designs. Design-Segmentierung schwierig.

  Eventuell bereits schwerwiegende Bugs auf RTL-Ebene

# Model Checking

- Aus einem System wird durch Abstraktion ein Modell und durch Formalisierung aus einer Spezifikation eine logische Formel gewonnen.
- Modell und Formel an Model Checker übergeben, der überprüft ob das Modell der Spezifikation genügt
- Im Idealfall positives Ergebnis oder Gegenbeispiel, was die Fehlerkorrektur vereinfacht
- Leider nicht möglich direkt an VHDL anzusetzen da das Design als endlicher Automat vorliegen muss.
- Gängigster Ansatz: VHDL in Binery Decision Diagramme übersetzen und darauf das Model Checking Verfahren anwenden.
- Vorgang sehr komplex und daher Kosten- und Zeitaufwendig
- Gefahr der Fehler bei der Übersetung

# Theroem Proving

- Behauptungen auf der Grundlage von Beweissystemen mit Hilfe von Programmen verifizieren.
- logische Sprache basierend auf der klassischen Logik
- VHDL-Code aufgrund fehlender Semantik nicht geeignet und muss für das Verfahren umgeschrieben werden.
- Sehr viel manuelle Arbeit
- Gut geeignet um verwendete Algorithmen zu verifizieren, jedoch bislang zu hart und kompliziert für die komplette Verifikation

# Vor- und Nachteile von VHDL

## Vorteile

- Vielseitigkeit
- Programmunabhängigkeit
- Technologieunabhängigkeit
- Modellierungsmöglichkeiten

## Nachteile

- Modellierung analoger Systeme
- Komplexität
- Synthese-Sprachumfang

# Alternativen und Asublick

- Verilog
  - prinzipiell ähnliche Möglichkeiten
  - leichter erlernbar da C-ähnlich
  - schnellere Simulation
  - nicht so ausgereift und weniger abstrakt
  - VHDL Flexibler, aber komplizierter
  - VHDL in Europa verbreiteter, Verilog in Nordamerika und Japan

```
module simple_register(in, out, clr_n, clk, a);

  // port declarations
  input          clr_n;
  input          clk;
  input [7:0]    in;
  input          a;
  output [7:0]   out;

  // signal declarations
  reg [7:0]      out;
  wire           a;

  // implement a register with asynchronous clear
  always @(posedge clk or negedge clr_n)
  begin
    if (clr_n == 0)  // could also be written if (!clr_n)
      out <= 0;
    else
      out <= in;
  end

  // continuous assignment
  assign a = !out[0];

endmodule
```

# Alternativen und Asublick

- Neu auf dem Markt: **SystemVerilog**
  - Systemmodellierung und Simulation auf hoher Abstraktionsebene
  - erweiterter Verilog-Standard und direkte Verbindung zu C
  - Einbindung von VHDL Elementen
- Arbeitsgruppe zur weiterentwicklung von VHDL : **VHDL-200X**

SystemVerilog