MULTIPROJEKT CHIP-GRUPPE

BADEN - WUERTTEMBERG

WORKSHOP FEBRUAR 1992

KARLSRUHE

HERAUSGEBER : FACHHOCHSCHULE ULM

INHALTSVERZEICHNIS

ANACAD
COMPUTER SYSTEMS GMBH

Wintersemester-Workshop 1991/92 an der FH Karlsruhe

Autor : **Dipl.-Ing. Peter Zdun; ANACAD**

Themenschwerpunkt: **Systemsimulation**

1. Thema : **Simulation analoger und gemischt analog-digitaler Systeme in einer CAD-Umgebung**

Kurzfassung: **Der Vortrag soll Vorgehensweise und Stand bei der Mixed-Signal-Simulation aufzeichnen, sowie den Einsatz der "Analogue Behavioral Simulation" erläutern. Desweiteren wird auf Aspekte der Modulierung und der Simulation von nichtlinearen dynamischen Systemen eingegangen.**

Die Unterstützung von Simulationswerkzeugen ist beim Entwurf mikroelektronischer Schaltungen für das konkurrenzfähige Entwickeln von elektronischen Schaltungen unabdingbar geworden und im Hinblick auf die VLSI-Technologie unverzichtbar. Allgemeines Ziel der Simulation ist die Funktionalität eines Schaltkreises zu untersuchen und genaue Angaben über sein Zeitverhalten zu analysieren.

Zu Beginn der 70´er Jahre wurde vorwiegend die elektrische Simulation zur Gewinnung von detaillierten Informationen über das Verhalten von Schalt-kreisen eingesetzt. Bei dieser Vorgehensweise wurden die dynamischen Verhaltensmerkmale durch nichtlineare Differentialgleichungen beschrie-ben. Die Simulation basierte auf der Lösung der resultierenden Gleich-ungssysteme, ausgehend von einer Menge initialer Bedingungen und in Abhänigkeit einer Menge elektrischer Eingangs-größen. Der wohl bekann-teste Analogsimulator SPICE war ursprünglich für die elektrische Simula-tion von Schaltkreisen in der Größenordnung von 100 Transistoren konzi-piert. Der hohe Aufwand der elektrischen Simulation und eine ständig ste-igende Komplexität der zu entwickelnden Schaltungen bzw. Systeme erfordert jedoch neue Methoden. Die Einführung von Abstraktionsebenen bei dem Entwurf elektronischer Schaltungen und der Bedarf einer Validie-rung von entsprechenden Schaltkreismodellen breits in früher Phase des Entwurfs führte zur Entwicklung neuer Simulatoren.

Noch heute gibt es Abschnitte im Ablauf der Schaltungsentwicklung, die von den herkömmlichen Werkzeugen nur unzureichend unterstützt werden. Die Simulation von Baugruppen, die intern sowohl analoge als auch digitale Funktionen ausführen sollen, sah bisher so aus, daß der analoge Teil mit einem konventionellen Analogsimulator und getrennt davon der digitale Teil mit einem Digitalsimulator simuliert wurde.

Führt man sich jedoch die Tatsache vor Augen, daß bestimmte Analog- und Digitalschaltungen im Zusammenwirken nur eine bedingte funktionale Zuverlässigkeit aufweisen, wird es zur zwingenden Notwendigkeit, diese Mixed-Signal-Strukturen als eine Schaltung zu betrachten und auch als eine Schaltung zu simulieren.

Ein Beispiel dafür ist im Bereich der Hochfrequenztechnik zu finden. Hier tritt häufig der Fall auf, daß Störsignale vom Digitalteil auf den Analogteil übersprechen und somit das Signal/Rauschverhältnis negativ beeinflussen. Wenn diese dadurch entstehenden Auswirkungen auf das Übertragungsverhalten vom Gesamtsystem nicht erkannt werden und demzufolge keine Abhilfe geschaffen werden kann, könnte dies dazu führen, daß sich die Entwicklungszeit merklich erhöht.

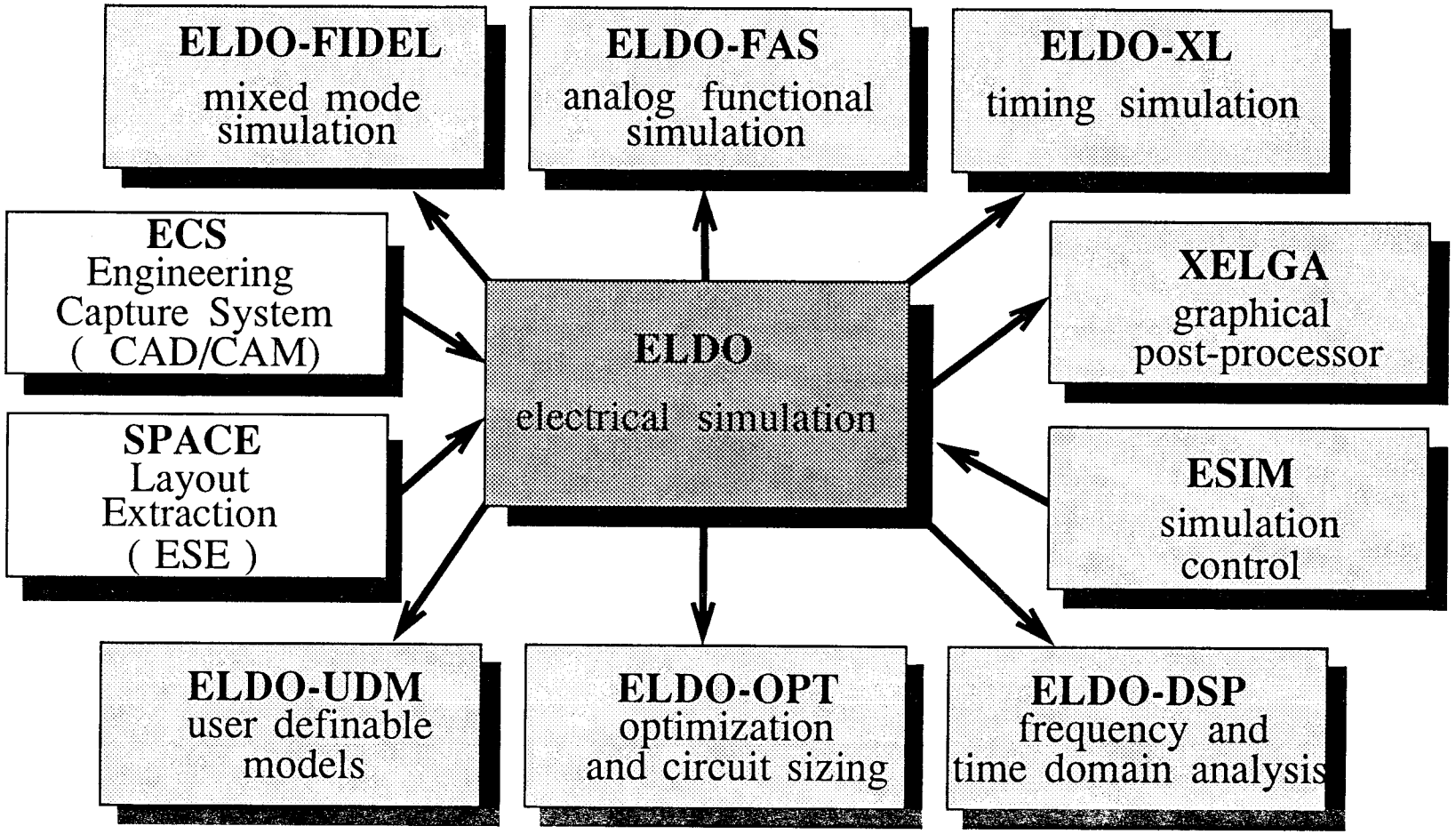
In diesem Beitrag wird der Mixed-Signal-Simulator FIDELDO und das für die Systemsimulation konzipierte Tool ELDO-FAS vorgestellt. FIDELDO ist ein Simulator, der in der Lage ist, komplexe analoge, digitale sowie analog-digital gemischte Schaltungen zu simulieren. Er ist modular aufgebaut und sein Kern besteht aus dem Analogsimulator ELDO.

ELDO ist ein schneller Circuit Simulator der dritten Generation zur Simulation von komplexen analogen Schaltungen, die aus tausenden MOS- oder BIPOLAR-Transistoren bestehen können. Seine Schnelligkeit beruht auf der Tatsache, daß die gesamte Schaltung in Blöcken partioniert wird. Die Unterteilung wird nach dem Grad der Kopplung vorgenommen. Für stark gekoppelte Bereiche, aufgebaut mit vorwiegend Bipolar-Transis-toren, erfolgt die Lösung der Netzwerkgleichungen mit einem Newton-Raphson Verfahren und für schwach gekoppelte Bereiche, aufgebaut mit vorwiegend MOS-Transistoren, mit dem One-Step-Relaxation (OSR) Ver-fahren. Die Iterationen zwischen den Blöcken werden durch OSR kon-trolliert. Dieses von herkömmlichen Methoden abweichende Verfahren führt zu einer höheren Simulationsgeschwindigkeit, zu einer höheren Ge-nauigkeit, zu besseren Konvergenzeigenschaften sowie zur Fähigkeit, sehr große Schaltungen simulieren zu können. Die Syntax der Eingabe ist SPICE kompatibel.

Aus der Integration des Analogsimulators ELDO und des Digitalsimulators FIDEL resultiert der Mixed-Signal-Simulator FIDELDO. FIDELDO kann hierarchisch beschriebene Schaltungen behandeln und Elemente können in ihrem s- bzw. z-Verhalten beschrieben werden. Analoge Signale werden in der Ereignis gesteuerten Umgebung manipuliert und digitale Elemente können aus dem Gate-, dem Register Transfer- und dem funktionalen Niveau beschrieben werden.

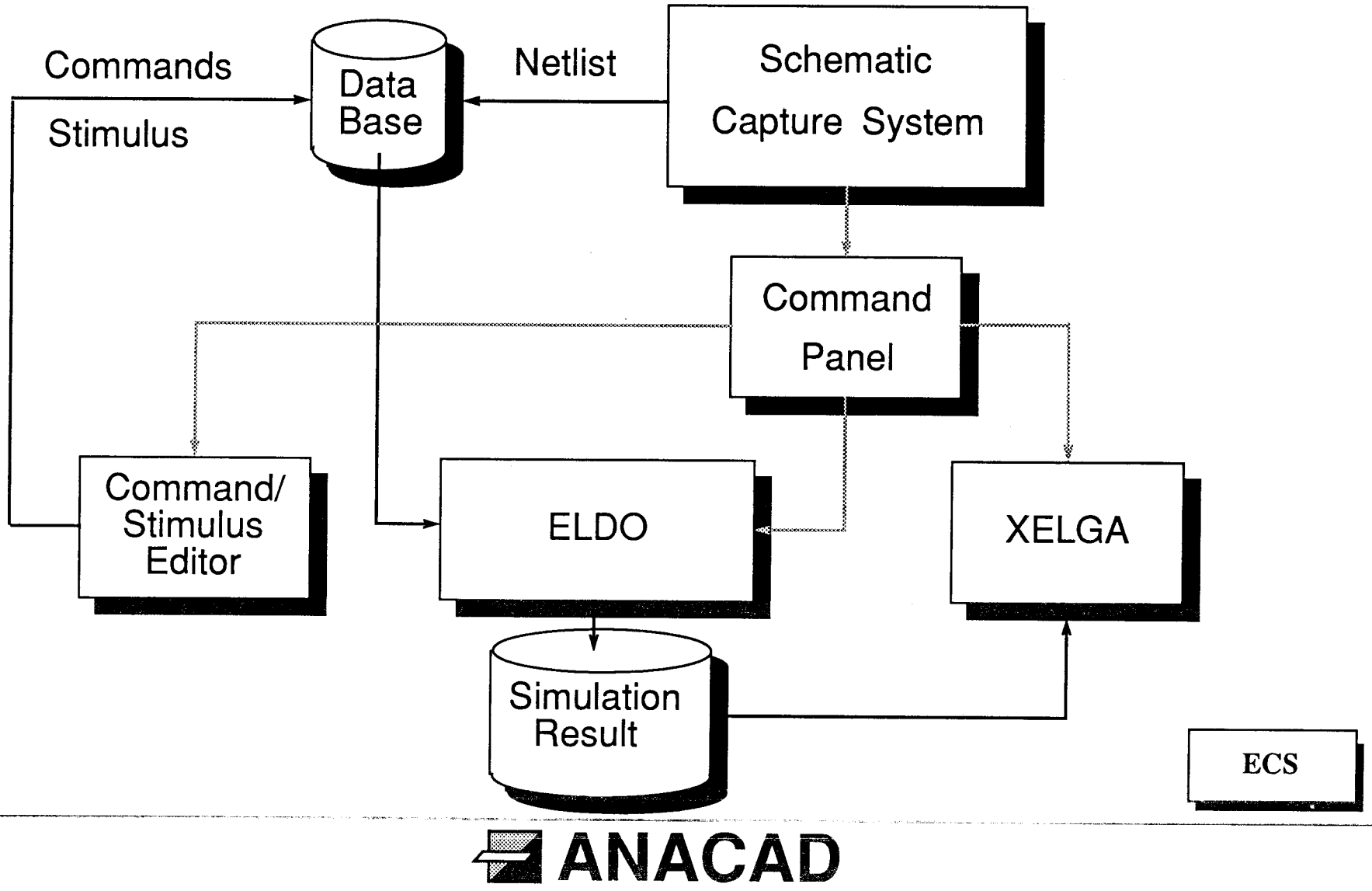
Der Vorteil der Behavioral-Language-Models für die Funktionsbeschreibung von digitalen Schaltungselementen steht dem Entwickler ebenso zur Verfügung, wie die Möglichkeit der Beschreibung von beliebigen nichtlinearen dynamischen Systemen unter Nutzung von ELDO-FAS. Damit können u. a. komplette Regelkreise, bestehend aus elektrischen und nichtelektrischen Komponenten, einheitlich beschrieben und simuliert werden.

ELDO Simulationssystem



ANACAD

Simulationsumgebung



ANACAD

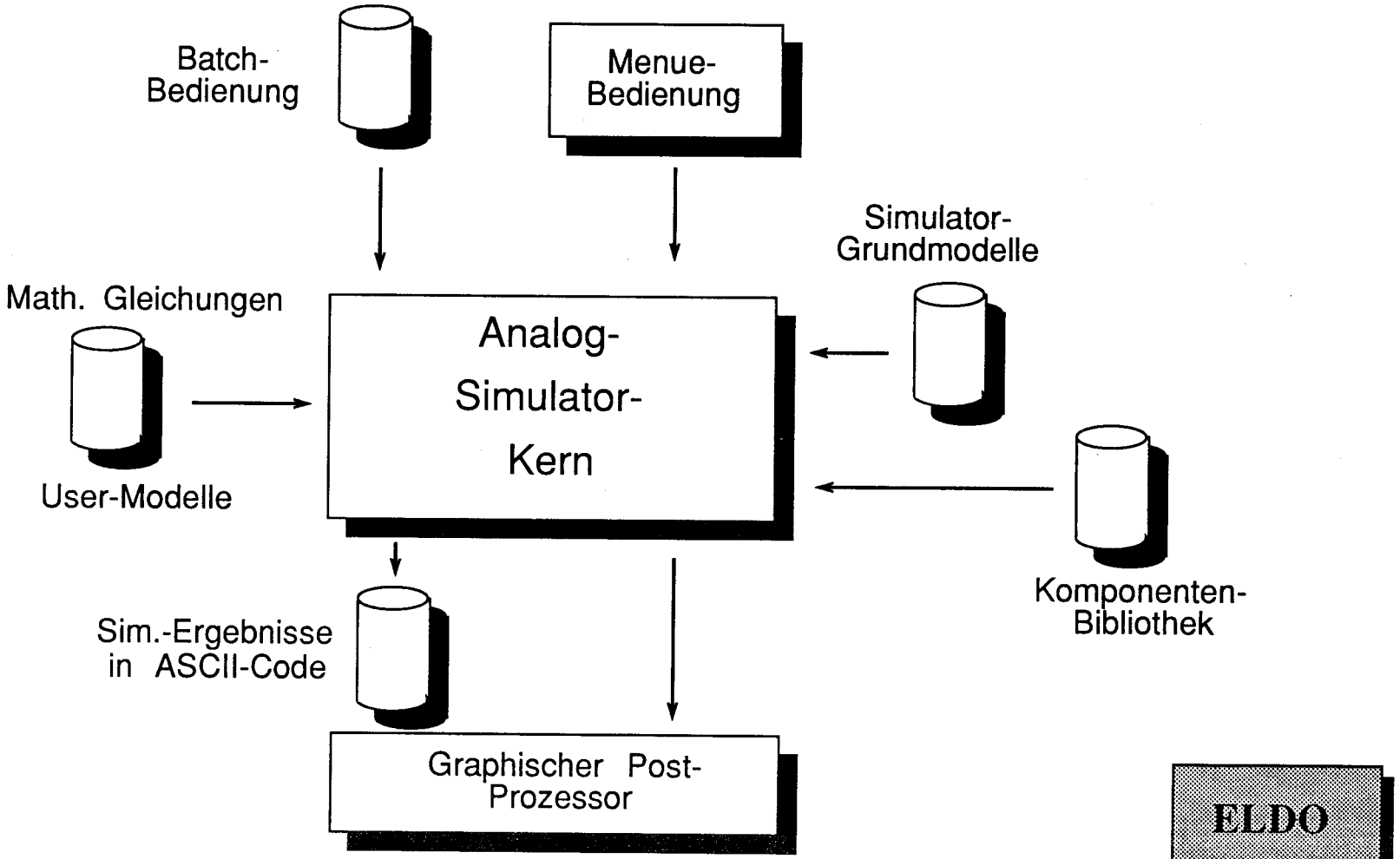
Nachteile traditioneller Analog-Simulatoren

- kein Eingriff in Modellgleichungen möglich
- kein Selbstdefinieren von Modellen möglich
- keine Systemsimulation möglich
- keine elekt./mechan. (Motor) oder elekt./optisch. (Leuchtdiode) Elemente möglich
- kein komfortabler und intelligenter graphischer Post-Prozessor vorhanden
- keine komfortable und interaktive Bedienung vorhanden
- schlechte Performance
- keine Kopplung mit Digital-Simulatoren

ELDO

ANACAD

Blockschaltbild eines modernen Simulators



ANACAD

Der Analogsimulator ELDO

- Kompatibel zu SPICE 2G6
- alle Standard-MOS-Modelle, BSIM I & II, Bipolar, GaAs-FET
- AC, TRAN, MC, NOISE, SENS, PZ
- C-Interface zur Erstellung eigener MODELLE
- Bauteile aus dem S-und Z-Bereich

ELDO

Die Besonderheiten des ELDO Systems

- schnelle Simulation von VLSI - Schaltkreisen

- genaue Simulation von VLSI - Schaltkreisen

- exzellente Konvergenzeigenschaften

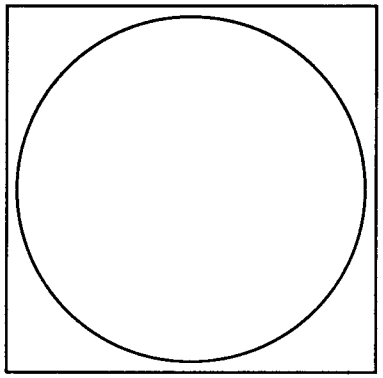
- Mixed-Signal-Simulation in Zusammenarbeit mit anderen Digital-Simulatoren

ELDO

ANACAD

Der Algorithmus

SPICE



circuit

ein Block

↓

eine Matrix

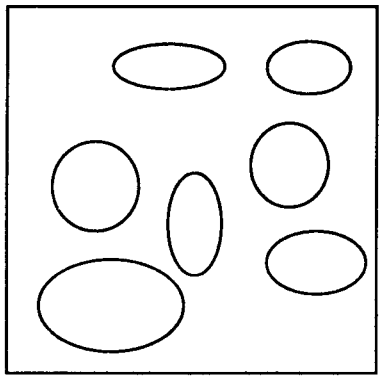
↓

Newton Algorithmus

↓

Ergebnis

ELDO



circuit

mehrere Blöcke

↓

mehrere Matrizen

↓

OSR oder Newton Algorithmus

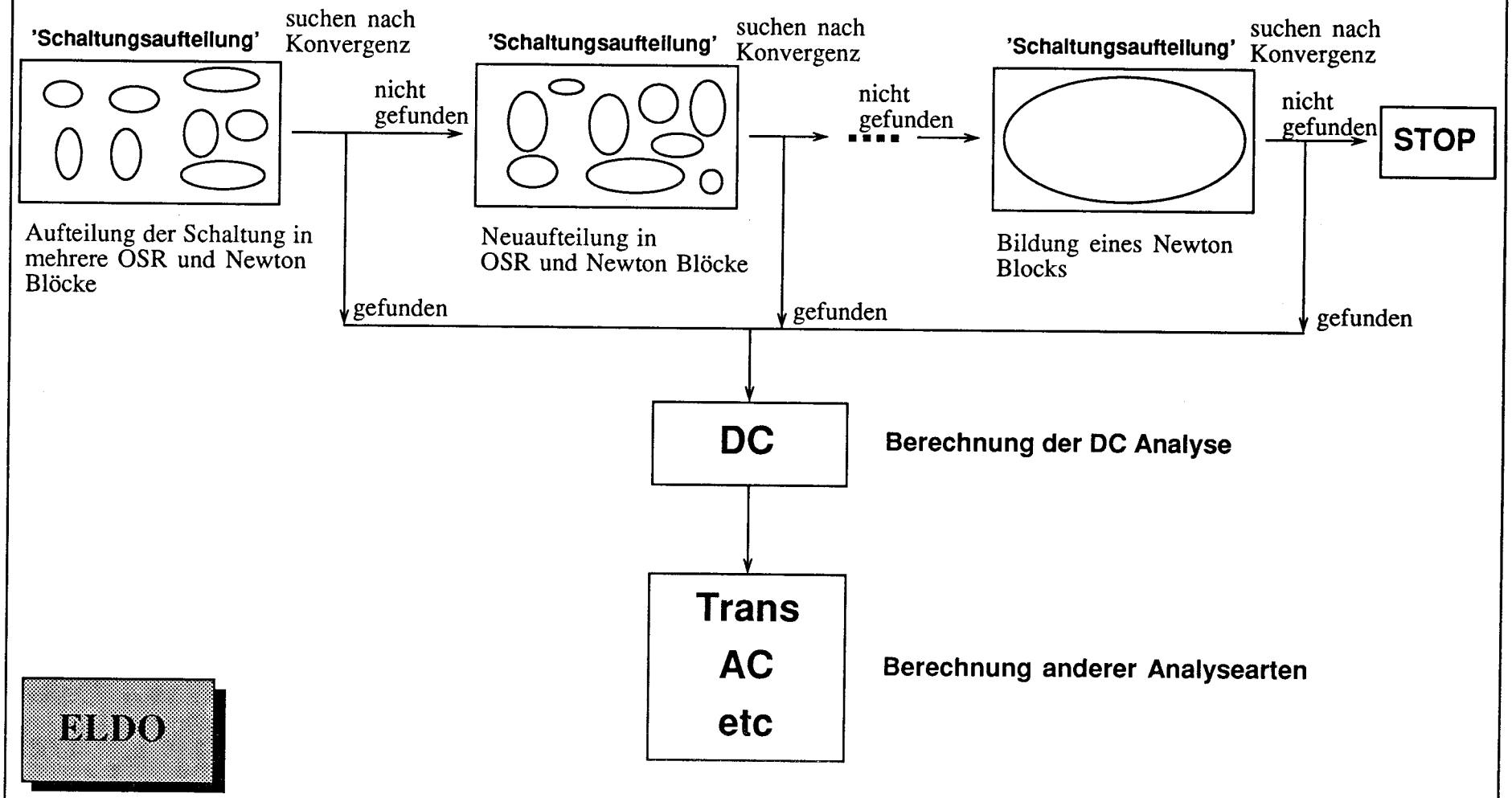
↓

Ergebnis

ELDO

ANACAD

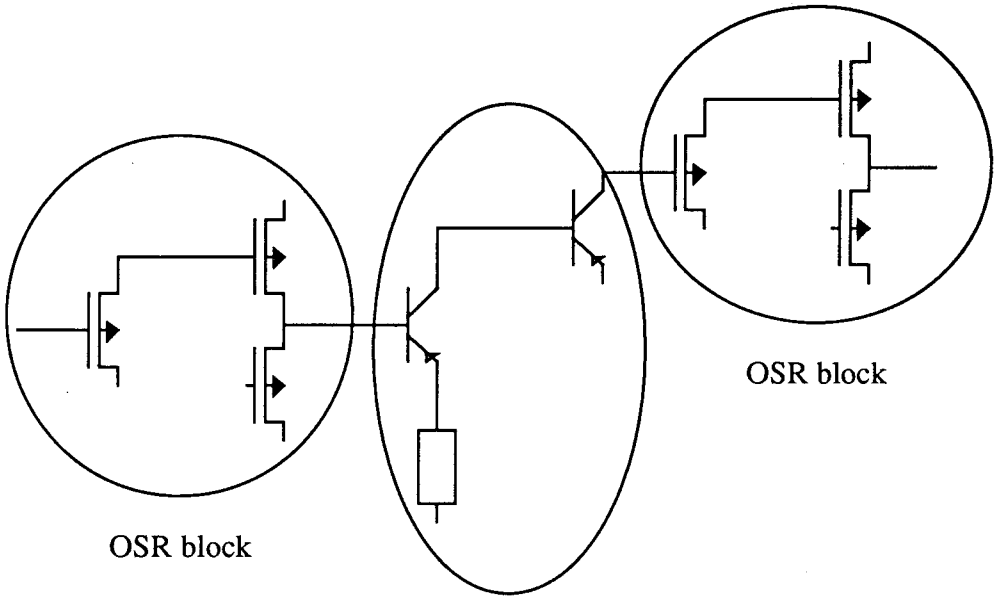
DC Analyse



ANACAD

Block-Aufbereitung

Balancing OSR Algorithmus



ELDO

ANACAD

Block - Aufteilung

Schaltungselemente mit

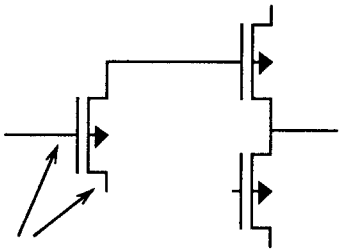
schwacher Kopplung
z.B. MOS - Transistoren

starker Kopplung
z.B. Bipolar - Transistoren

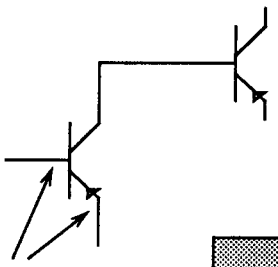
werden berechnet durch

OSR Algorithmus

Newton Algorithmus



schwache Kopplung



starke Kopplung

ELDO

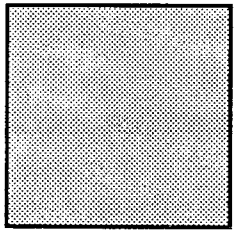
ANACAD

Vorteile des kombinierten Algorithmus

- weniger Berechnungen nötig, da keine Bestimmungen der Ableitungen beim OSR notwendig sind
- Berechnung verschiedener kleinerer Newton-Blöcke ist schneller als die eines Grossen
- CPU-Zeit nimmt ungefähr mit n^2 zu, n - Anzahl der Knoten
- die maximale Newton-Block-Größe ist limitiert

SPICE

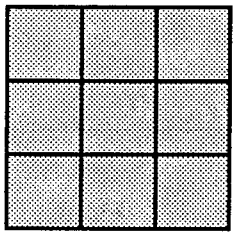
1 Newton-Block



CPU-Zeit $\approx n^2$

ELDO

9 Newton-Blöcke



CPU-Zeit $\approx 9 \left(\frac{n}{9} \right)^2$ + Zeit der OSR-Iteration

ELDO

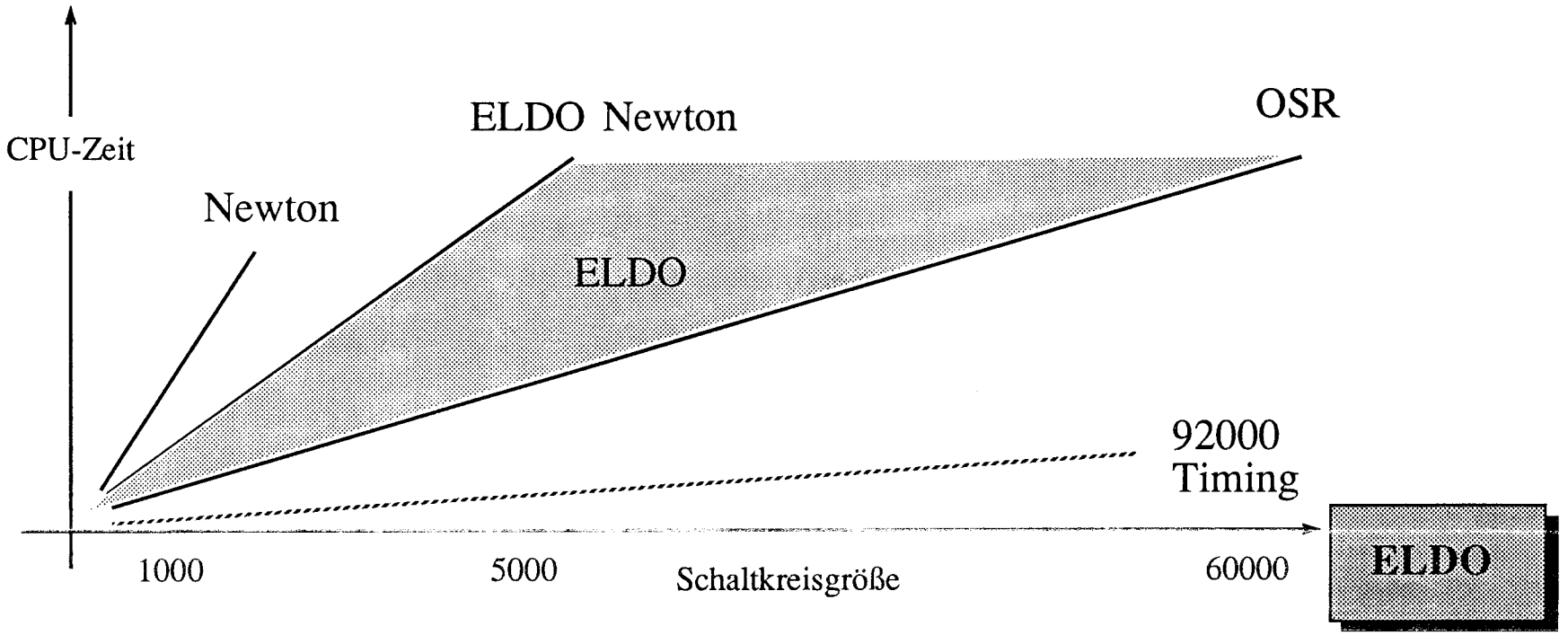
ANACAD

Vorteile des kombinierten Algorithmus

Schaltkreise mit 90.000 Transistoren können simuliert werden.

Schaltkreissimulationen mit 1.000 Transistoren sind Routineberechnungen.

Sehr gute Konvergenzeigenschaften.



ANACAD

S-Domain-Filter

Syntax **FNSxx IN OUT N0 N1 N2 N3 ... , D0 D1 D2 D3 ...**

Beschreibung

Dieses Kommando beschreibt ein S-Domain-Filter mittels der Übertragungsfunktion :

$$H(s) = \frac{N0 + N1*s + N2*s^2 + N3*s^3 ...}{D0 + D1*s + D2*s^2 + D2*s^3 ...}$$

Parameter

FNSxx	Name des S-Domain-Filters.
IN, OUT	Ein- und Ausgangsknoten.
$N_0 ... N_i$	Zählerkoeffizienten der Übertragungsfunktion.
$D_0 ... D_i$	Nennerkoeffizienten der Übertragungsfunktion.

Beispiel: S-Domain-Filter

ELDO

Syntax **FNSxx IN OUT N0 N1 N2 N3 ... , D0 D1 D2 D3 ...**

Übertragungsfunktion: Butterworth-Filter:

filter 2nd order $$H(s) = \frac{1}{1 + 1.414*s + s^2}$$

filter 3rd order $$H(s) = \frac{1}{1 + 2*s + 2*s^2 + s^3}$$

filter 4th order $$H(s) = \frac{1}{1 + 2.613*s + 3.414*s^2 + 2.613*s^3 + s^4}$$

filter 5th order $$H(s) = \frac{1}{1 + 3.2361*s + 5.2361*s^2 + 5.2361*s^3 + 3.2361*s^4 + s^5}$$

filter 6th order $$H(s) = \frac{1}{1 + 3.8637*s + 7.4641*s^2 + 9.1416*s^3 + 7.4641*s^4 + 3.8637s^5 + s^6}$$

ANACAD

Die Modell-Frage

UTMOST II (SILVACO) und IC-CAP (HP) unterstützen die Parameterextraktion für SPICE-Modelle.

IC-CAP unterstützt die Parameterextraktion für analoge funktionale Modelle.

ELDO

ANACAD

Einige Kundenschaltkreise

Schaltkreis	Größe	Firma
EEPROM - digital	24.000 MOS	SGS-THOMSON (STM)
Flat Panel LCD Driver - analog	19.000 MOS	France Telecom
Flash ADC Video Converter - analog/digital	3.000 BICMOS	TMS
Digital Video Dematrix - analog/digital	25.000 MOS 1.000 bipolar	TCE
Wiegand Bridge Reader - analog	1.040 bipolar	Doduco GmbH
D2-MAC and HD_MAC Equalizer - digital	128.000 MOS	France Telecom

ELDO

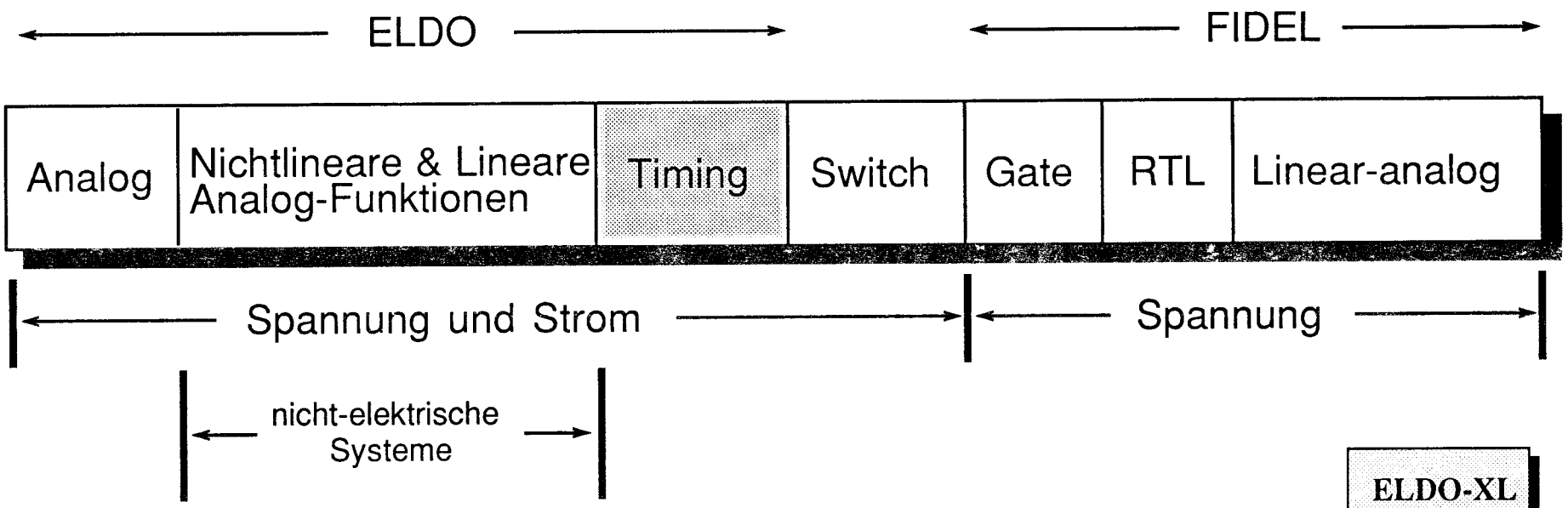
ANACAD

Kooperationen

CAD-CAM Group	*ECS*
Cadence	*Verilog und OPUS*
GenRad	*HILO*
MENTOR Graphics	*LSIM und FALCON*
SGS-THOMSON	*MOZART*
Silvaco	*UTMOST*
Hewlett Packard	*IC CAP*
CAD-UL	*ARIADNE*

ELDO - XL

ELDO-XL / ELDO : ungefähr 50 mal schneller für komplexe Schaltungen



ANACAD

ELDO - FAS : Behavioral-Simulation

Hochsprache zur Beschreibung dynamischer Systeme

Vordefinierte Konstrukte aus fünf Bereichen

Integriert in ein Gesamtsystem mit Verbindung zu ELDO

C-Programme einbindbar

ELDO-FAS

ANACAD

Die FIDEL Modelle

FIDEL

Beschreibungssprache

amodel

Analog-Modelle
Elektrisch, mechanisch und andere dynamische Systeme

Zeitschrittgesteuert

smodel

Struktur-Modelle
Mixed Mode Simulation von elektr. Systemen (volt only)

Ereignisgesteuert

fmodel

Funktions-Modelle
Mixed Mode Simulation von elektr. Systemen (volt only)

Ereignisgesteuert

ELDO-FAS

ANACAD

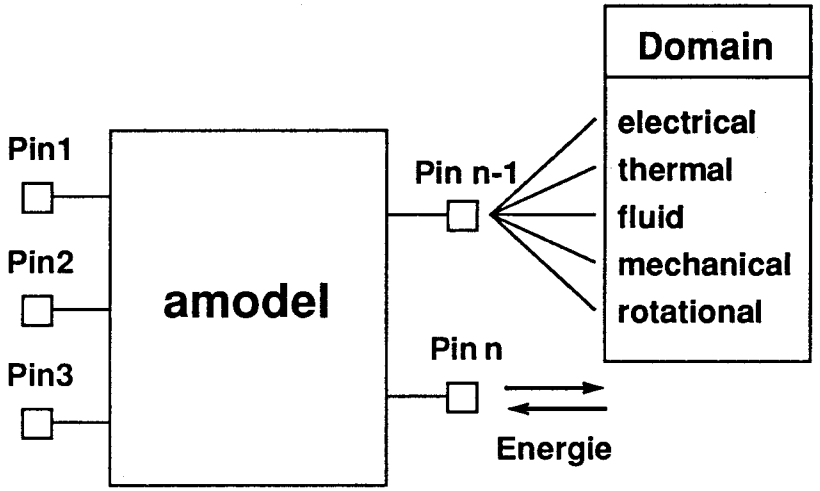
Ein Behavioral Modell

... ist eine n-Pin "Black Box"

... an jedem Pin wird Energie übertragen

... und jeder Pin kann zu jedem Bereich zugeordnet werden

... und die im Modell-Body ermittelten Ergebnisse liefern die Werte an die Pins



ELDO-FAS

ANACAD

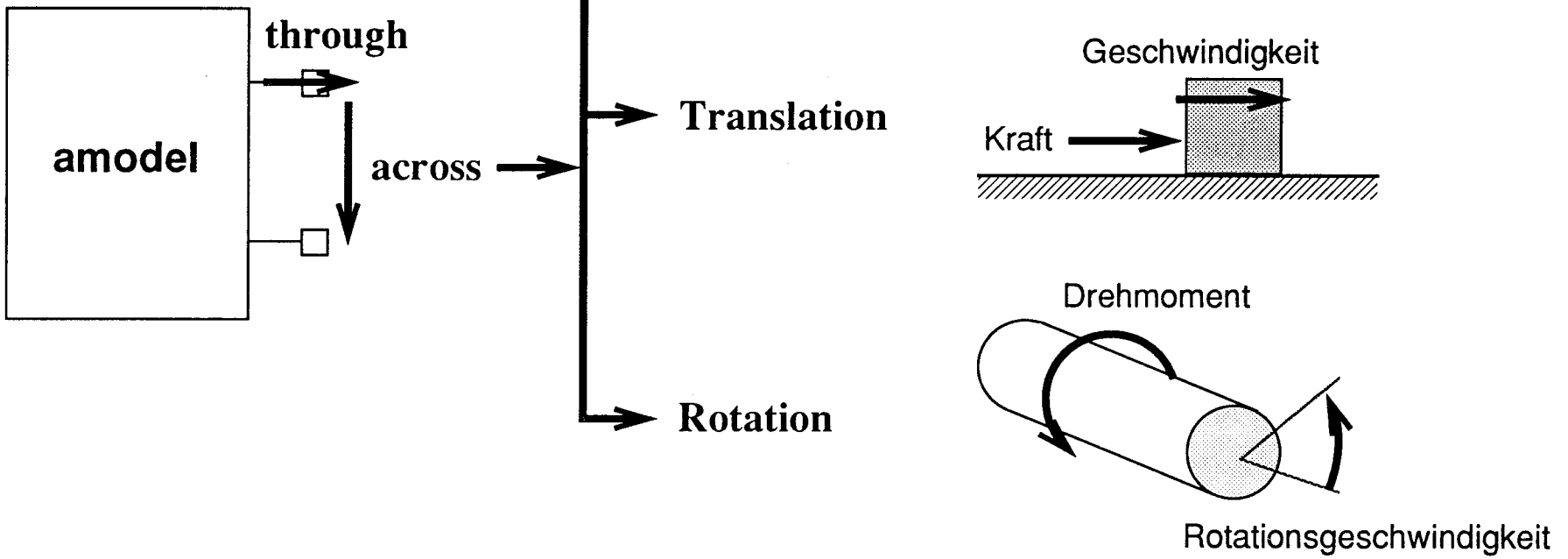
Die unterschiedlichen Bereiche

Bereich	'across' Werte (relative Werte)	'through' Werte (absolute Werte)
Elektrizität	Spannung	Strom
Rotation	Winkelgeschwindigkeit	Drehmoment
Translation	Geschwindigkeit	Kraft
Strömung	Druck	Durchflußrate
Temperatur	Temperaturdifferenz	Wärmestrom

ELDO-FAS

ANACAD

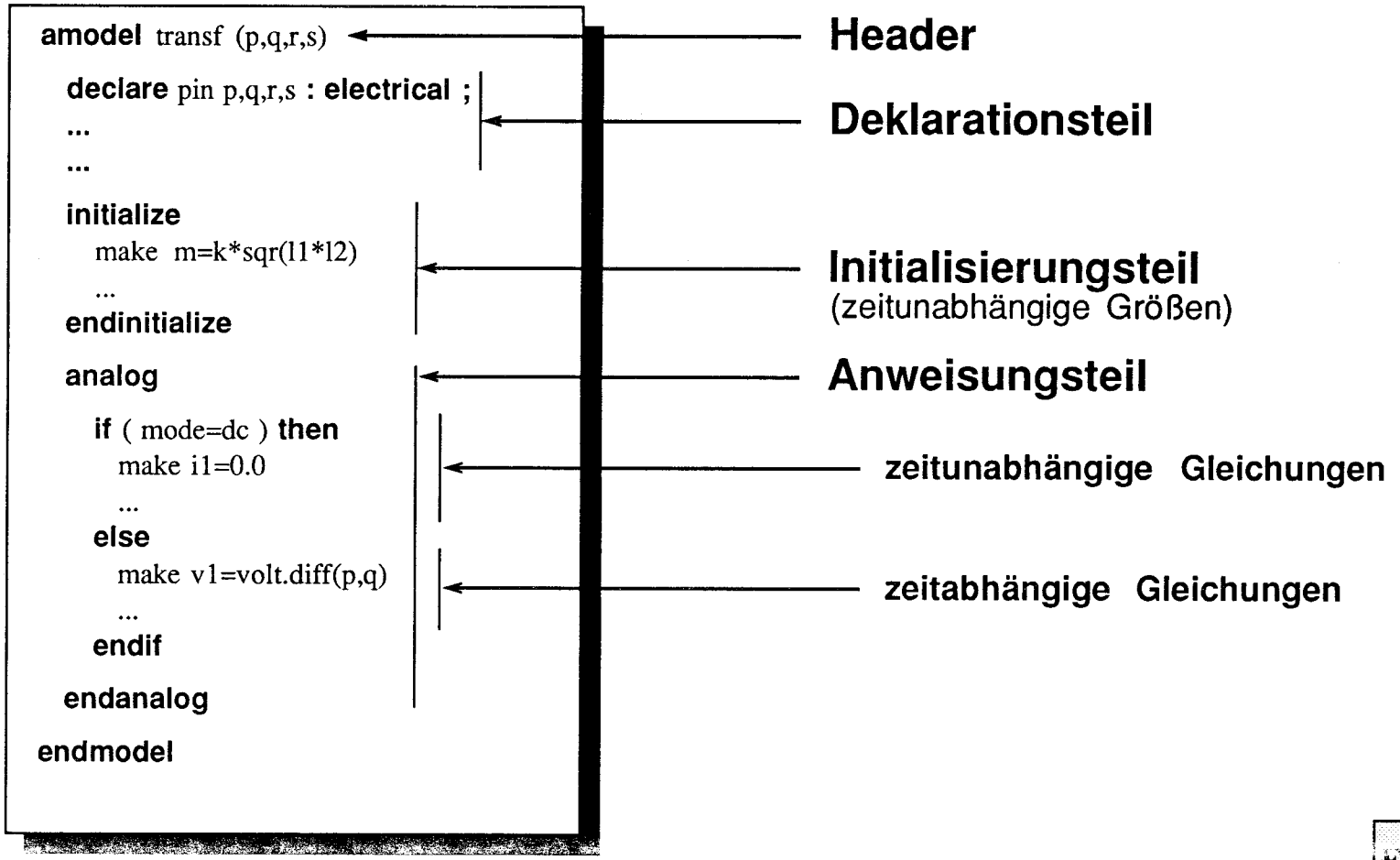
Across und through



ELDO-FAS

ANACAD

Aufbau des FAS-Modells



ELDO-FAS

ANACAD

Einfache FAS-Anweisungen

make curr.on(p1) = volt.diff(p1,p2)/R

make volt.across(p1,p2) = curr.on(p1) * R - volt.diff(p3,p4)

make speed.on (shaft) = K + L * curr.dt(p1)

if (mode = dc) then make cc = expo (- temp *K)

ELDO-FAS

ANACAD

Ableitung

Wenn 'u' eine Variable des Typs 'state ist, dann

make a = state.dt(u)

ist 'a' die zeitliche Ableitung von 'u'

Wenn 'u' ein Modell-Pin vom Typ 'electrical' ist, dann

make a = volt.dt(u)

make b = curr.dt(u)

ist 'a' bzw. 'b' die zeitliche Ableitung der Spannung bzw. des Stroms an Pin 'u'

Wenn 'u' ein Modell-Pin vom Typ 'mechanical' ist, dann

make a = vel.dt(u)

make b = force.dt(u)

ist 'a' bzw. 'b' die zeitliche Ableitung der Geschwindigkeit bzw. der Kraft an Pin 'u'

ELDO-FAS

ANACAD

Integral

Wenn 'u' eine Variable des Typs 'state ist, dann

make a = state.integ(u)

ist 'a' das zeitliche Integral von 'u'

Wenn 'u' ein Modell-Pin vom Typ 'electrical' ist, dann

make a = volt.integ(u)

make b = curr.integ(u)

ist 'a' bzw. 'b' das zeitliche Integral der Spannung bzw. des Stroms an Pin 'u'

Wenn 'u' ein Modell-Pin vom Typ 'mechanical' ist, dann

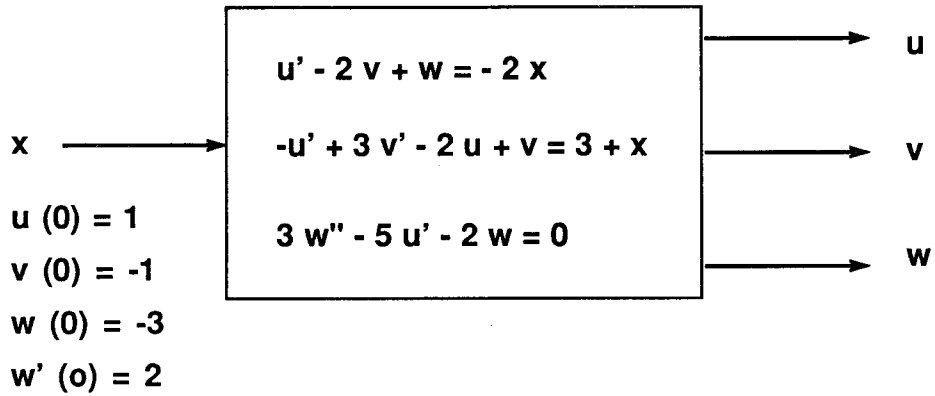
make a = vel.integ(u)

make b = force.integ(u)

ist 'a' bzw. 'b' das zeitliche Integral der Geschwindigkeit bzw. der Kraft an Pin 'u'

ELDO-FAS

Systemmodellierung mittels Differentialgleichungen

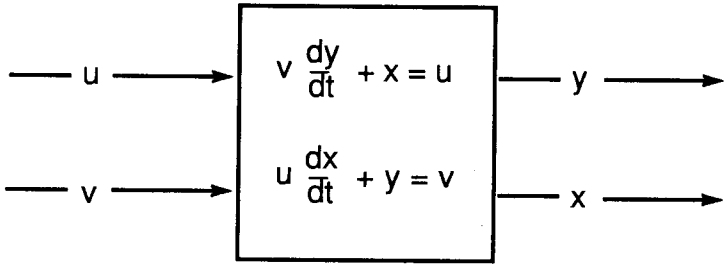



```
declare istate pu,pv,pw:          real;
.....
make xx = volt.value(x)
make uu = volt.value(u)
make vv = volt.value(v)
make ww= volt.value(u)
make uu1= volt.dt(u)
make vv1= volt.dt(v)
make ww1=volt.dt(w)
make ww2=state.dt(ww1)
solve(pu,uu1-2*vv-ww+2*xx)
solve(pv,-uu1+3*vv1-2*uu+vv-3-xx)
solve(pw,3*ww2-5*uu1-2*ww)
make volt.across(u) = pu
make volt.across(v) = pv
make volt.across(w) = pw
```

ELDO-FAS

ANACAD

Die SOLVING - Anweisung



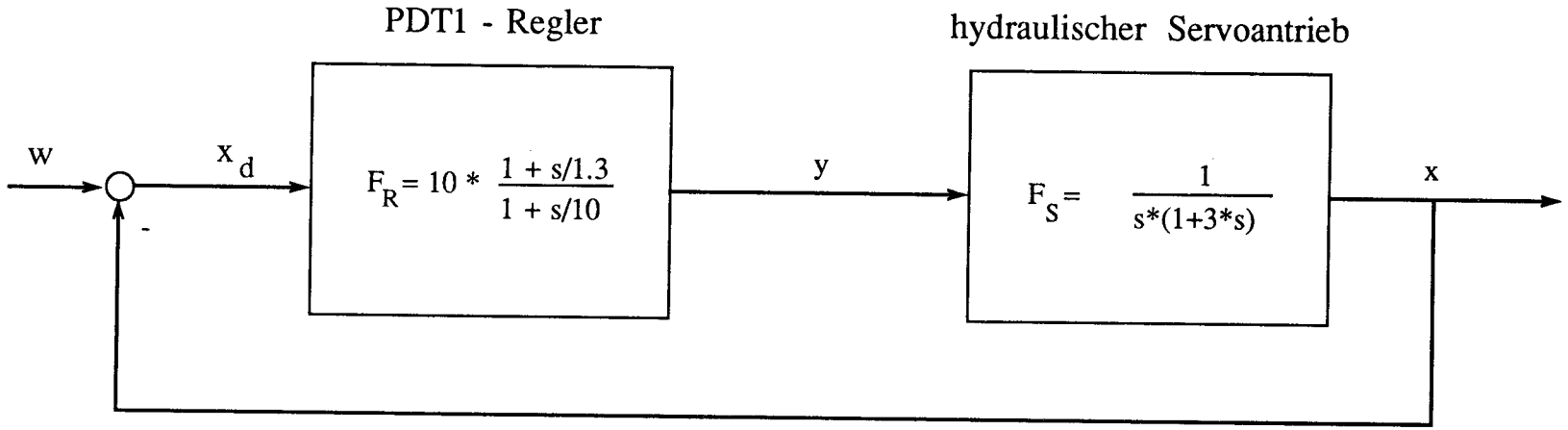

```
make xx = volt.value(x)
make yy = volt.value(x)
make vv = volt.value(v)
make uu = volt.value(u)

make yd = volt.dt(y)
make xd = volt.dt(x)
```

solve (yy, yd * vv + xx - uu) (yd * vv + xx - uu = 0)
solve (xx, xd * uu + yy - vv) (xd * uu + yy - vv = 0)

ELDO-FAS

ANACAD

Beispiel : PDT1 - oder phasenanhebender Regler



Vorgaben : $x_d <= 0.1\ w_0$ (Regeldifferenz bei Rampenanregung $w(t) = w_0 * f(t)$)

Phasenreserve >= 45^0

ELDO-FAS

ANACAD

PDT1 - Regler:

$$\frac{U_{out}}{U_{in}} = k * \frac{1 + 0.77 * s}{1 + 0.1 * s} \longrightarrow U_{out} = k * (U_{in} + 0.77 * U'_{in}) - 0.1 * U'_{out}$$

Hydraulischer Servoantrieb:

$$\frac{U_{out}}{U_{in}} = k * \frac{1}{s * (1 + 3*s)} \longrightarrow \text{solve} (U_{out} , U'_{out} + 3 * U''_{out} - k * U_{in})$$

ELDO-FAS

ANACAD

AMODEL PDT1-Regler:

```
amodel pdt1(in,out);
  declare pin in,out : electrical;
  declare state voutdt,vindt,vout,vin : real;
  declare param k,t1,t2 : real;
  initialize
    make k = 1.0
    make t1 = 0.77
    make t2 = 0.1
  endinitialize
  analog
    make vin = volt.value(in)
    make vout = volt.value(out)
    if(mode = dc)then
      make volt.across(out) = k * vin
    else
      make voutdt = state.dt(vout)
      make vindt = state.dt(vin)
      make vout = K * (vin + t1*vindt) - t2*voutdt
      make volt.across(out) = vout
    endif
  endanalog
endmodel
```

ELDO-FAS

ANACAD

AMODEL
Hydraulischer Servoantrieb:

```
amodel it1(in,out);
  declare pin in,out : electrical;
  declare state vdt1,vdt2,vvout : real;
  declare param k,t1 : real;
  declare istate vout : real;
  declare local vin : real;
  initialize
    make k = 1.0
    make t1 = 3.0
  endinitialize
  analog
    make vin = volt.value(in)
    make vvout = volt.value(out)
    if(mode = dc)then
      make volt.across(out) = k * vin
    else
      make vdt1 = state.dt(vvout)
      make vdt2 = state.dt(vdt1)
      solve(vout,vdt1 + t1 * vdt2 - K * vin)
      make volt.across(out) = vout
    endif
  endanalog
endmodel
```

ELDO-FAS

ANACAD

AMODEL
Subtrahierer

```
amodel sub(in1,in2,out);
  declare pin in1,in2,out : electrical;
  declare param k : real;
  declare local vin1,vin2 : real;
  analog
    make vin1 = volt.value(in1)
    make vin2 = k * volt.value(in2)
    make volt.across(out) = vin1 - vin2
  endanalog
endmodel
```

ELDO-FAS

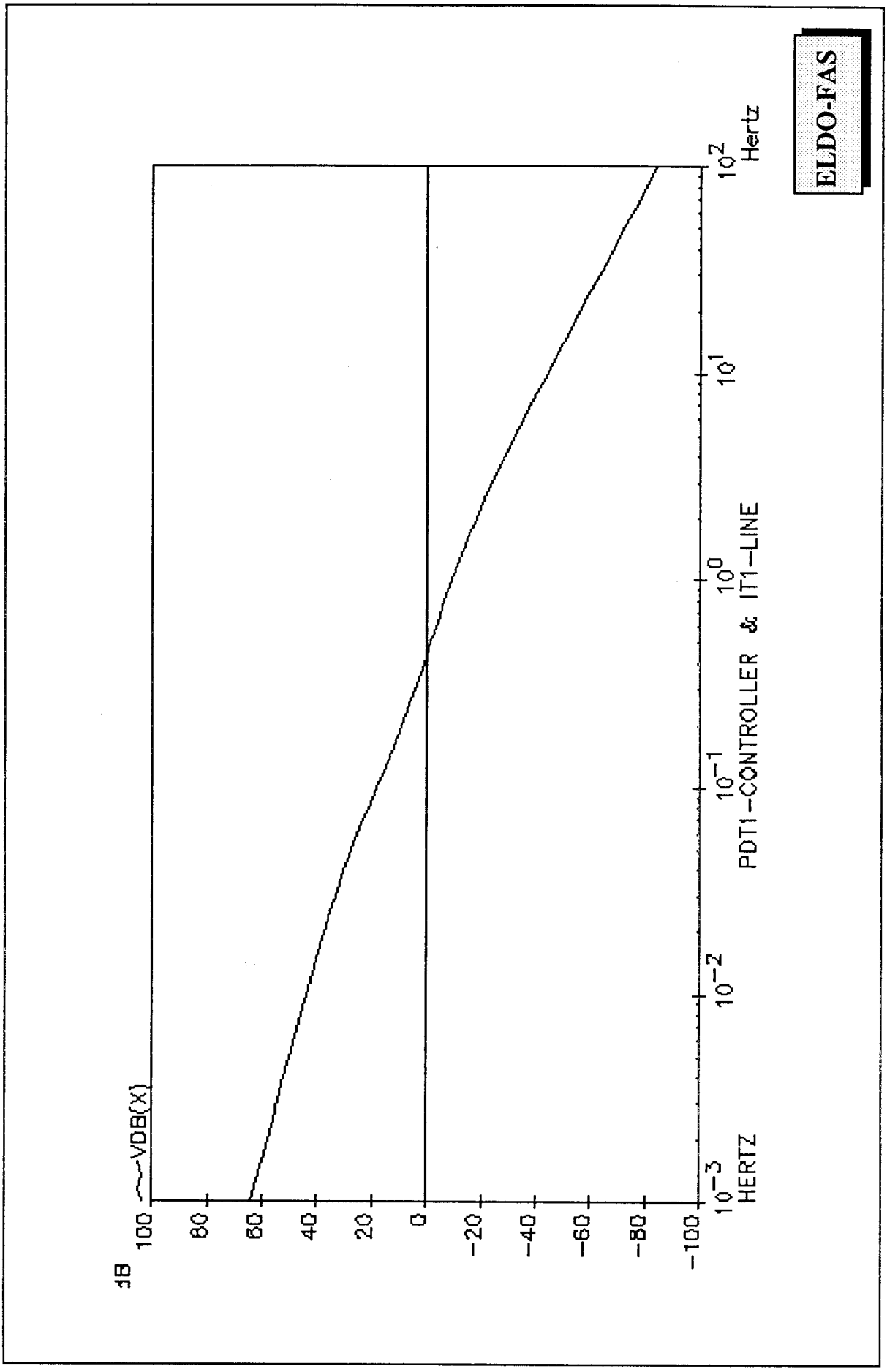
ANACAD

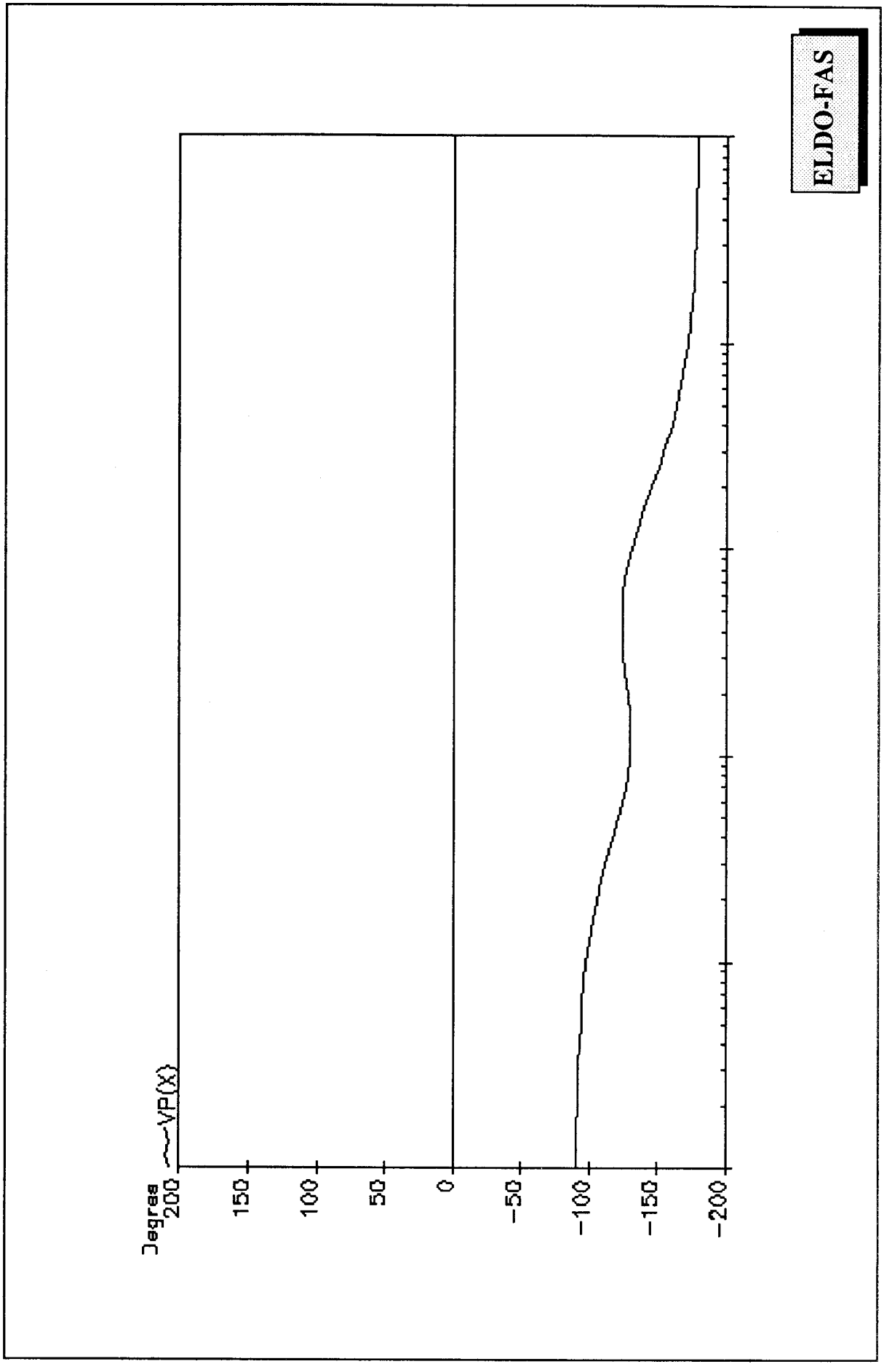
ELDO - Netzliste

```
PDT1-Controller & IT1-Line

amodel pdt1(in,out);
      . . .
endmodel

amodel it1(in,out);
      . . .
endmodel

amodel sub(in1,in2,out);
      . . .
endmodel

y1 pdt1 pin : xd y     param : k=10.0 t1=0.77 t2=0.1
y2 it1  pin : y  x     param : k=1.0 t1=3.0
y3 sub  pin : w  x  xd param : k=0.0

v1 w 0  ac
.ac dec 10 0.001 100
.option eps=1.0e-4
.plot ac vdb(x)
.plot ac vp(x)
.end
```

ELDO-FAS

ANACAD


dB
100
80
60
40
20
0
-20
-40
-60
-80
-100
VDB(X)
10^{-3}
HERTZ
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Hertz
PDT1-CONTROLLER & IT1-LINE
ELDO-FAS
ANACAD


ELDO-FAS
degres
VP(X)
200
150
100
50
0
-50
-100
-150
-200
ANACAD

Beschreibung
im S-Bereich:

S-Bereichsbeschreibung

```
v1 1 0 ac
fns1 1 2 1.0 , 0.0 1.0 3.0
fns2 2 3 10.0 7.7 , 1.0 0.1
r1 3 0 1g
.ac dec 10 0.001 100
.option eps=1.0e-4
.plot ac vdb(3)
.plot ac vp(3)
.end
```

ELDO-FAS

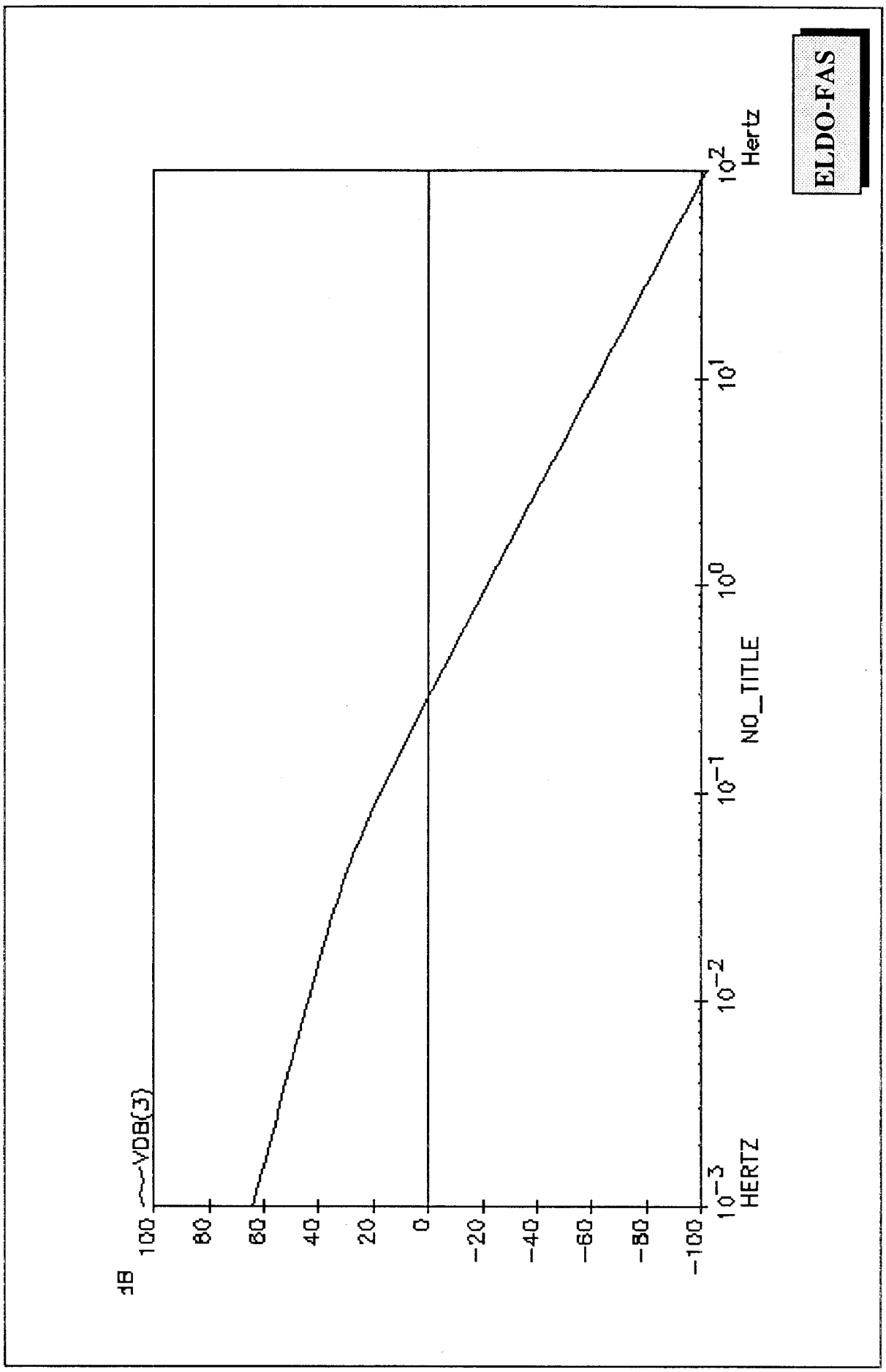
ANACAD

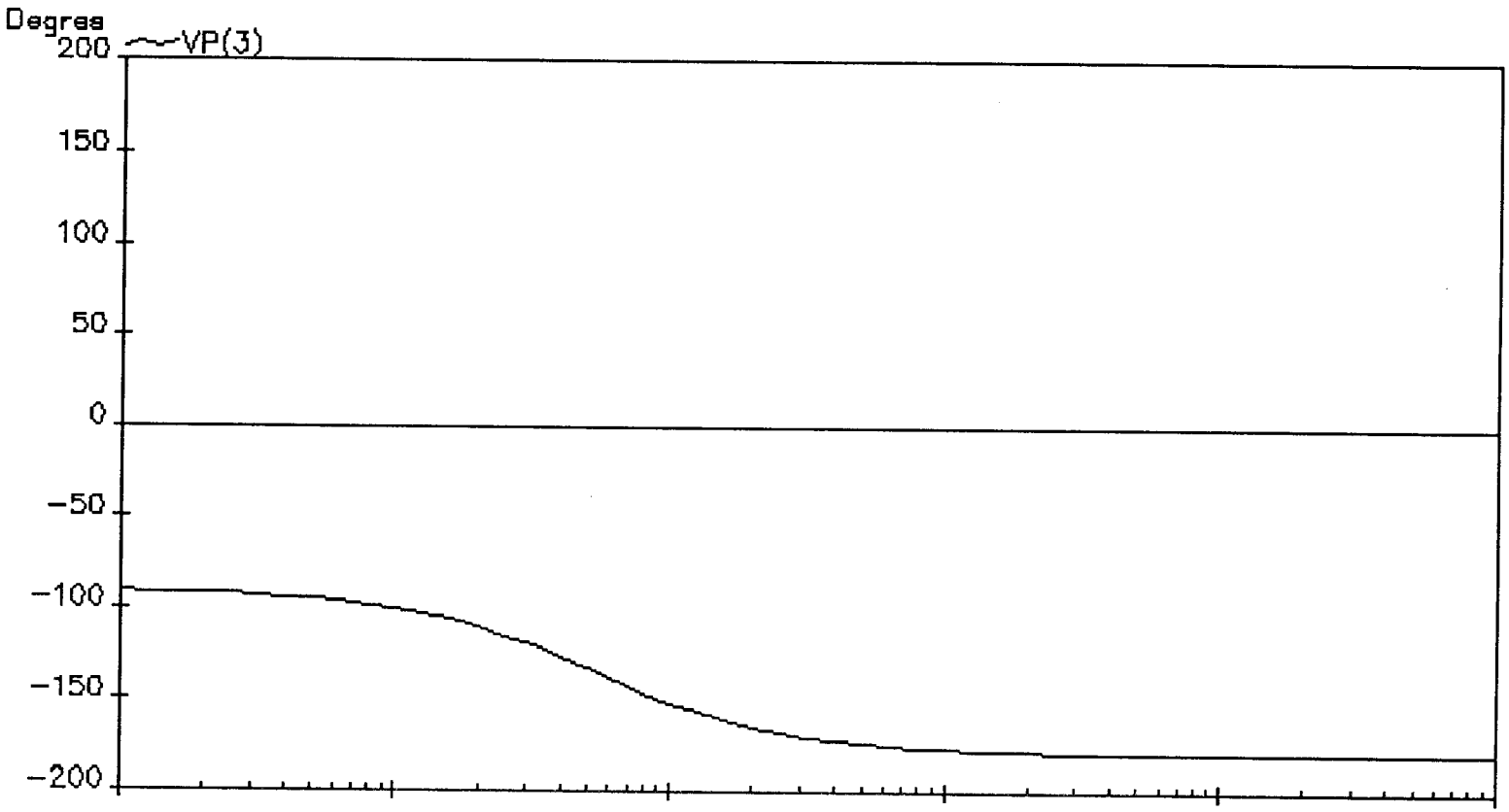
Beschreibung
mit P-Regler
im S-Bereich:

```
v1 1 0 ac
fns1 1 2 1.0 , 0.0 1.0 3.0
fns2 2 3 10 , 1
r1 3 0 1g
.ac dec 10 0.001 100
.option eps=1.0e-4
.plot ac vdb(3)
.plot ac vp(3)
.end
```

ELDO-FAS

ANACAD


ELDO-FAS
ANACAD
dB
VDB(3)
100
80
60
40
20
0
-20
-40
-60
-80
-100
10^{-3}
HERTZ
10^{-2}
10^{-1}
NO_TITLE
10^{0}
10^{1}
10^{2}
Hertz



ELDO-FAS

ANACAD

Beispiel: Integrator mit Begrenzung

amodel:

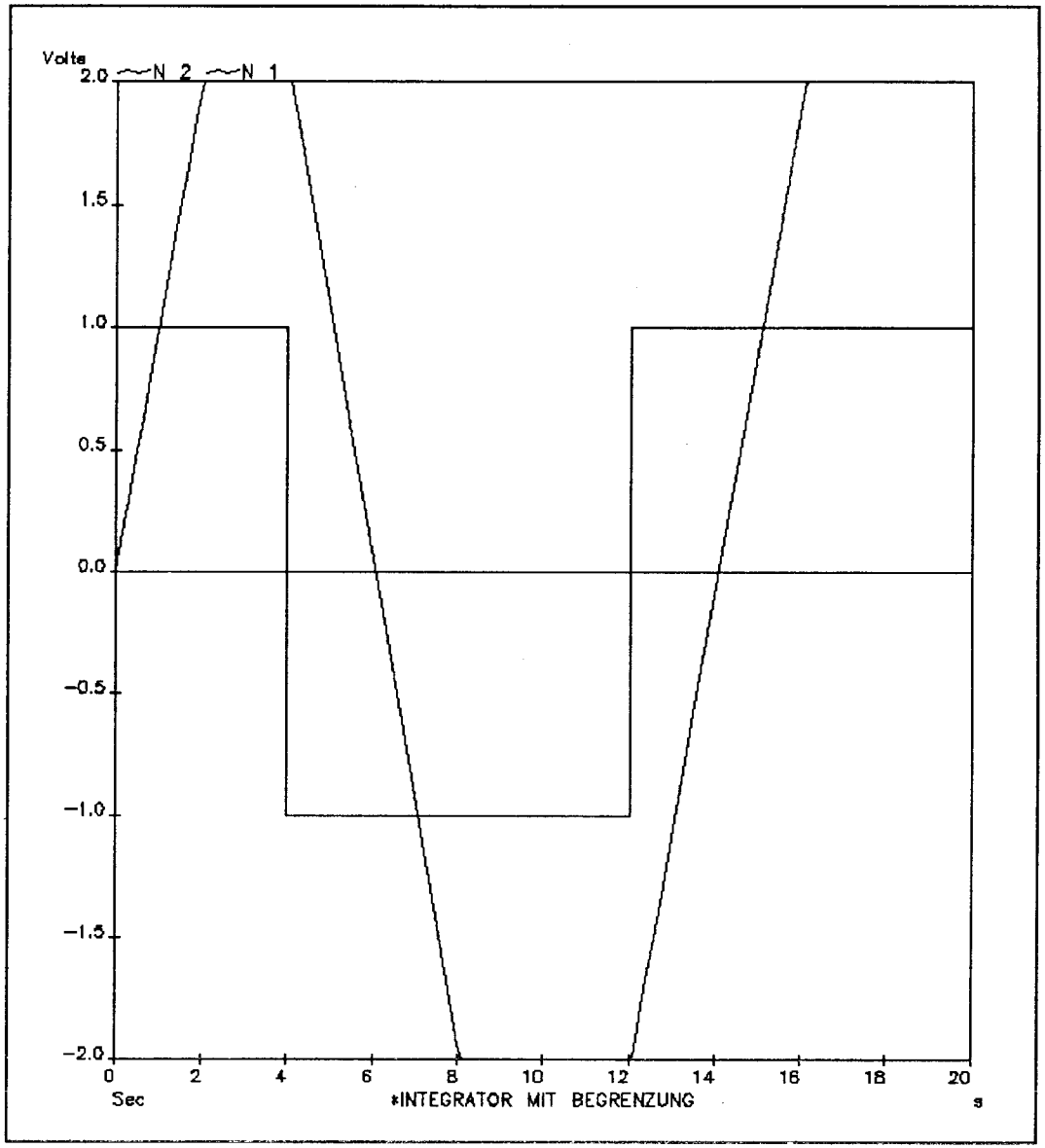
```
*Integrator mit Begrenzung
amodel inte(in,out);
  declare pin in:ELECTRICAL;
  declare pin out:ELECTRICAL;
  declare state r,ery,d: real;
  declare local k,t : real;

initialize
     make k = 1.0
     make t = 1.0
endinitialize
```

ANACAD

```
analog
  make ery = tstep
  make r = volt.value(in)
  if ((r >= 0) and (d < 2)) or ((r < 0) and  (d > -2)) then
    make d = state.integ(r)
  else
    make r = 0
  endif

  if (d >= 2)  then
    make d = 2
  endif
  if (d <= -2) then
    make d = -2
  endif
  make volt.across(out) = d
endanalog
endmodel
```

Eldo-Netzliste:

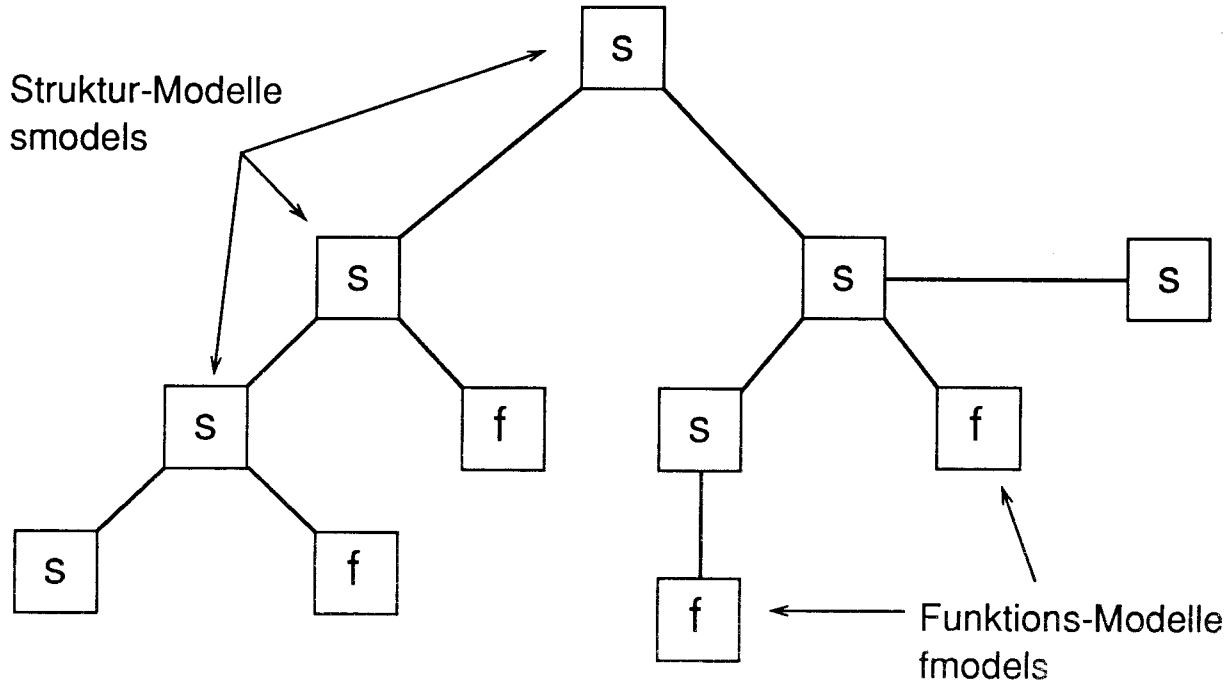
```
YINTEG inte PINS: N_1 N_2
r1 n_2 0 1g
v1 n_1 0 pulse (1 -1 4 0.0001 0.0001 8 20)
.option eps=1.0e-8

.tran 0.01 20
.plot tran V(N_2) V(N_1)
.end
```


Volts
N 2
N 1
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
0
2
4
6
8
10
12
14
16
18
20
Sec
*INTEGRATOR MIT BEGRENZUNG
s

Die Modell-Hierarchie

Struktur-Modelle können Struktur- oder Funktions-Modelle beinhalten

Funktions-Modelle dürfen nur in dem untersten Hierarchie-Leveln auftreten

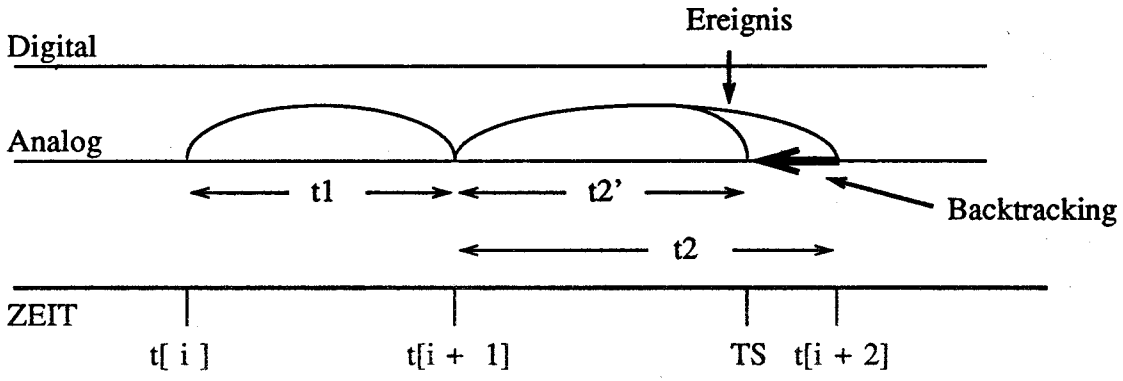


ELDO-FIDEL

ANACAD

Leap - Frog Synchronisation

Analog - Simulator	Dynamische Zeitschritt - Kontrolle
Digital - Simulator	Ereignis - Kontrolle



Leap - Frog Zeitschritt-Bestimmung

ELDO-FIDEL

ANACAD

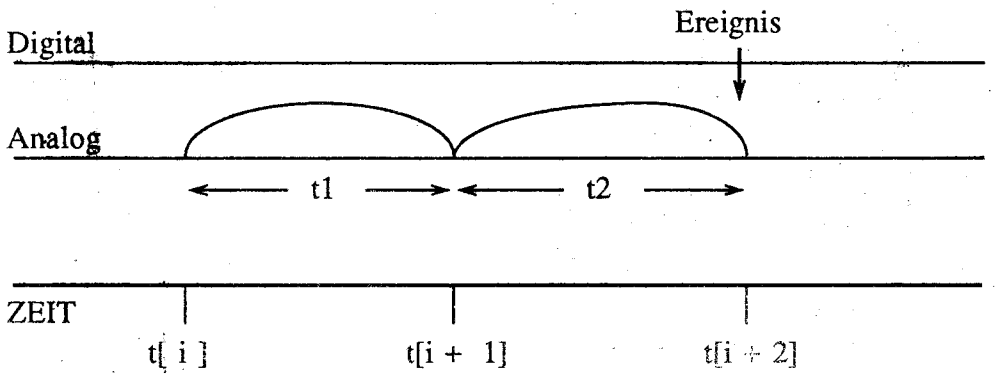
Lock - Step Synchronisation

Analog-Simulator & Digital-Simulator

arbeiten mit der gleichen Zeitschrittsteuerung

kein 'backtracking' notwendig

kleinster Zeitschritt wir gewählt



ELDO-FIDEL

ANACAD

Aufbau des S-Modells

```
smodel halfadd ( in1, in2, out1 ) ;

   option ns ;
   declare input in2, in2 : logic ;
   declare output out : logic ;
   declare component aa : xor(a,b,c);
  .....
  initialize aa.c to 0
  .....
  structural
    connect halfadd.in1,aa.a
    .....
    schedule halfadd.out1
  endstructural
endmodel
```

← header

← Attribute

← Deklaration der Pins

← Deklaration der Komponenten

← Initialisierungsteil

← Anfang des Anweisungsteils

← Plot-Anweisungen

← Ende des Anweisungsteil

← Ende der Modellbeschreibung

ELDO-FIDEL

ANACAD

Aufbau des F-Modells

```
fmodel fulladd (carin, in1, in2, ...) ;

      option ns ;
      declare input carin , ... : logic ;
      declare output out , ... : logic ;
      declare state aa , ... : real;
      .....
      initialize out to 0
      .....
      macro  .....
      endmacro
      .....
      plotall or plot out
      .....
      functional
         make out carin + in1 + in2
      endfunctional
endmodel
```

← header

← Attribute

← Deklaration der Pins

← Deklaration der Variablen

← Initialisierungsteil

← Makros

← Plot-Anweisungen

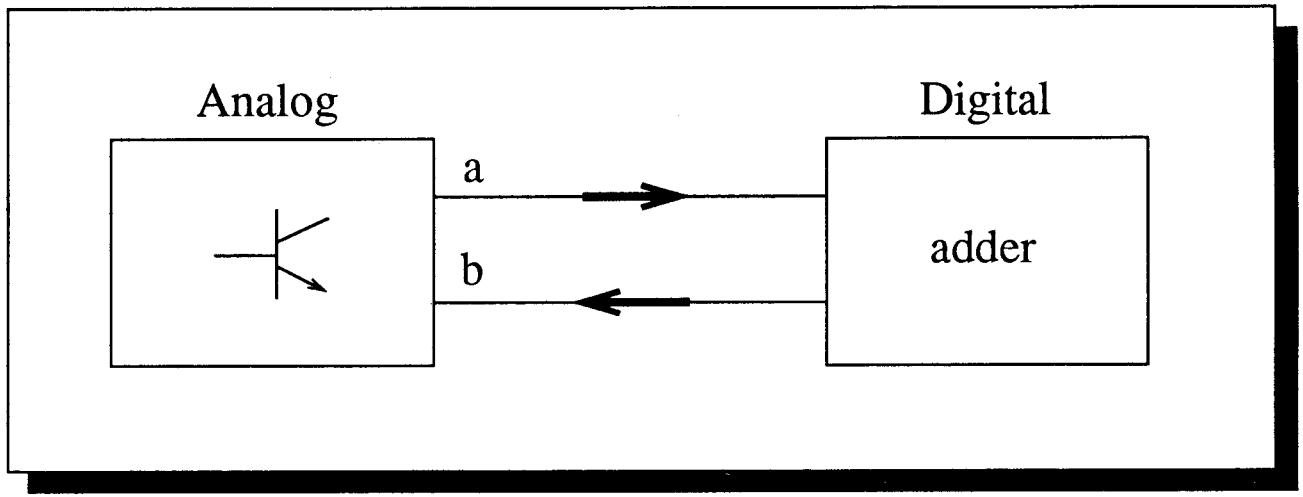
← Anweisungsteil

← Ende der Modellbeschreibung

ELDO-FIDEL

ANACAD

Verbindung zwischen analogen und digitalen Elementen



obitadd a:todigi b:toana **mod**=adder

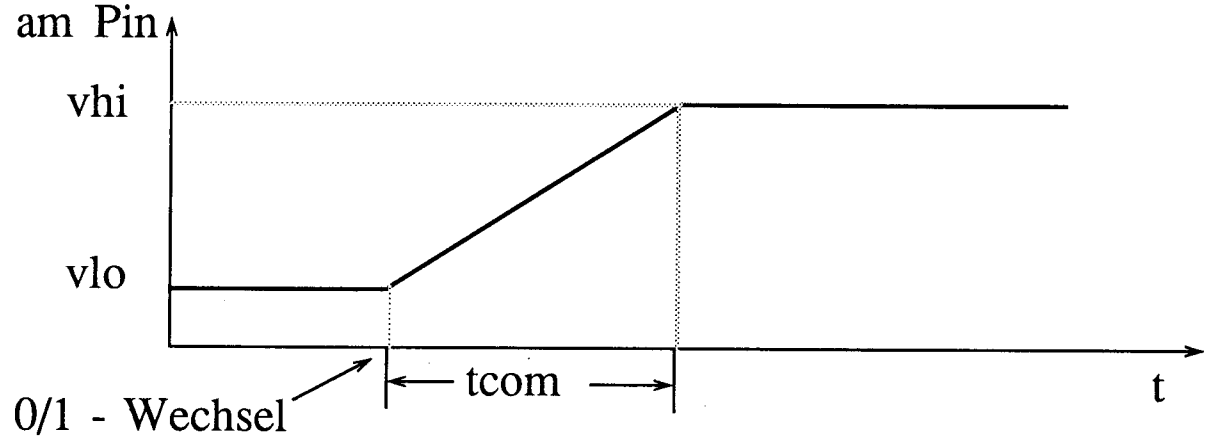
.model todigi **atod** **vth1**=val **vth2**=val
.model toana **dtoa** **vhi**=val **vlo**=val **tcom**=val

ELDO-FIDEL

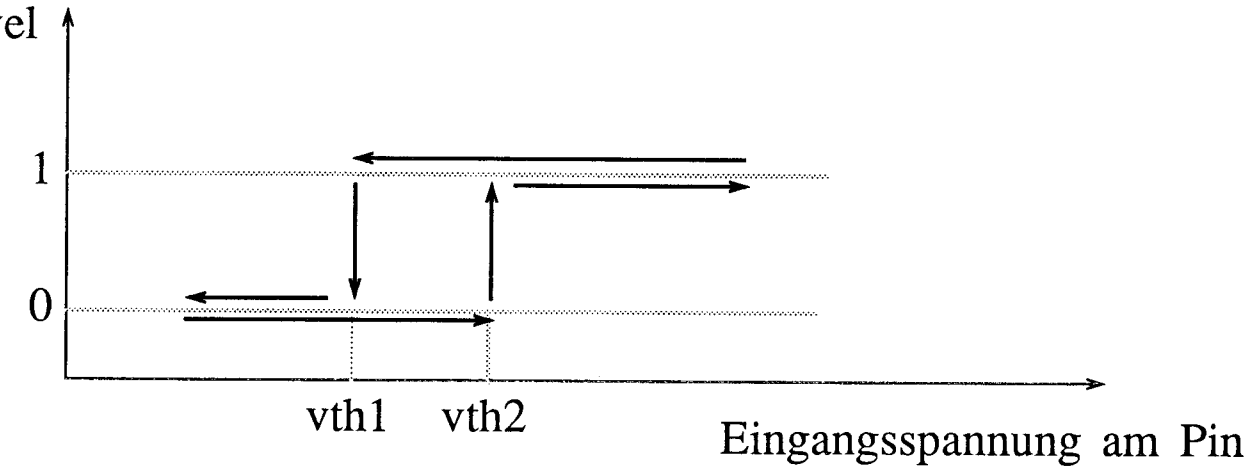
ANACAD

Übergabewerte

digital -> analog :

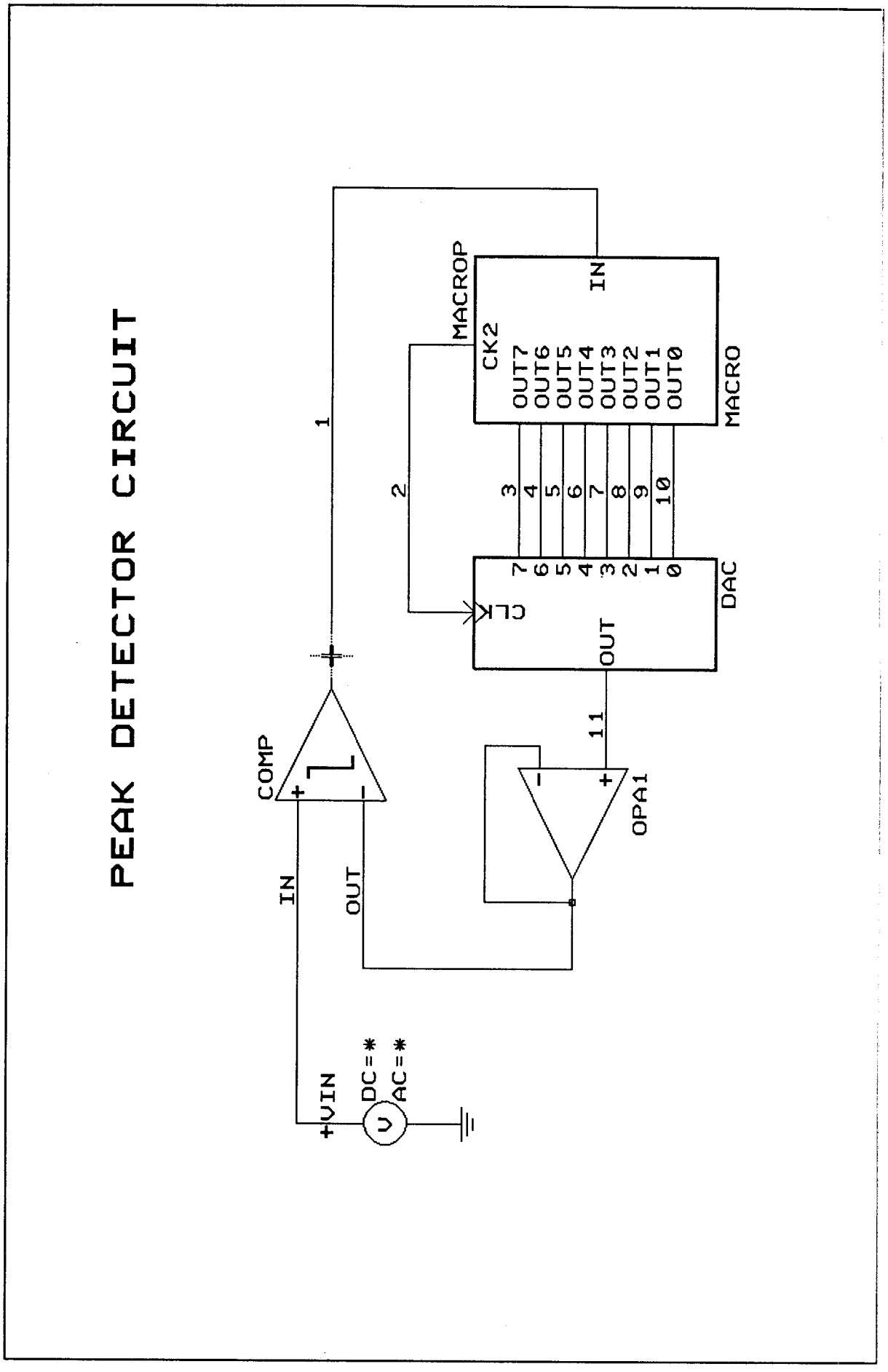


analog -> digital :



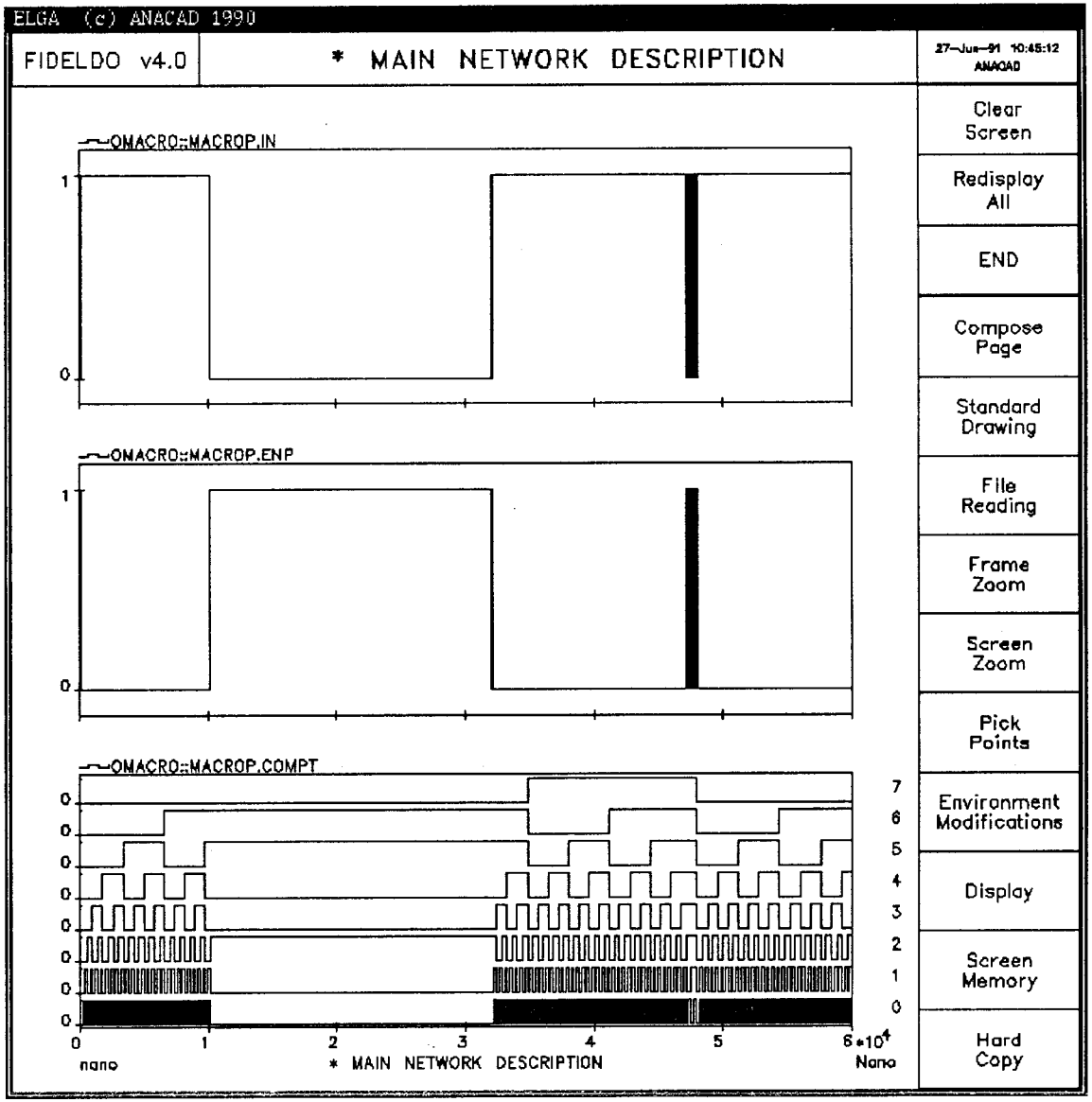
ELDO-FIDEL

ANACAD

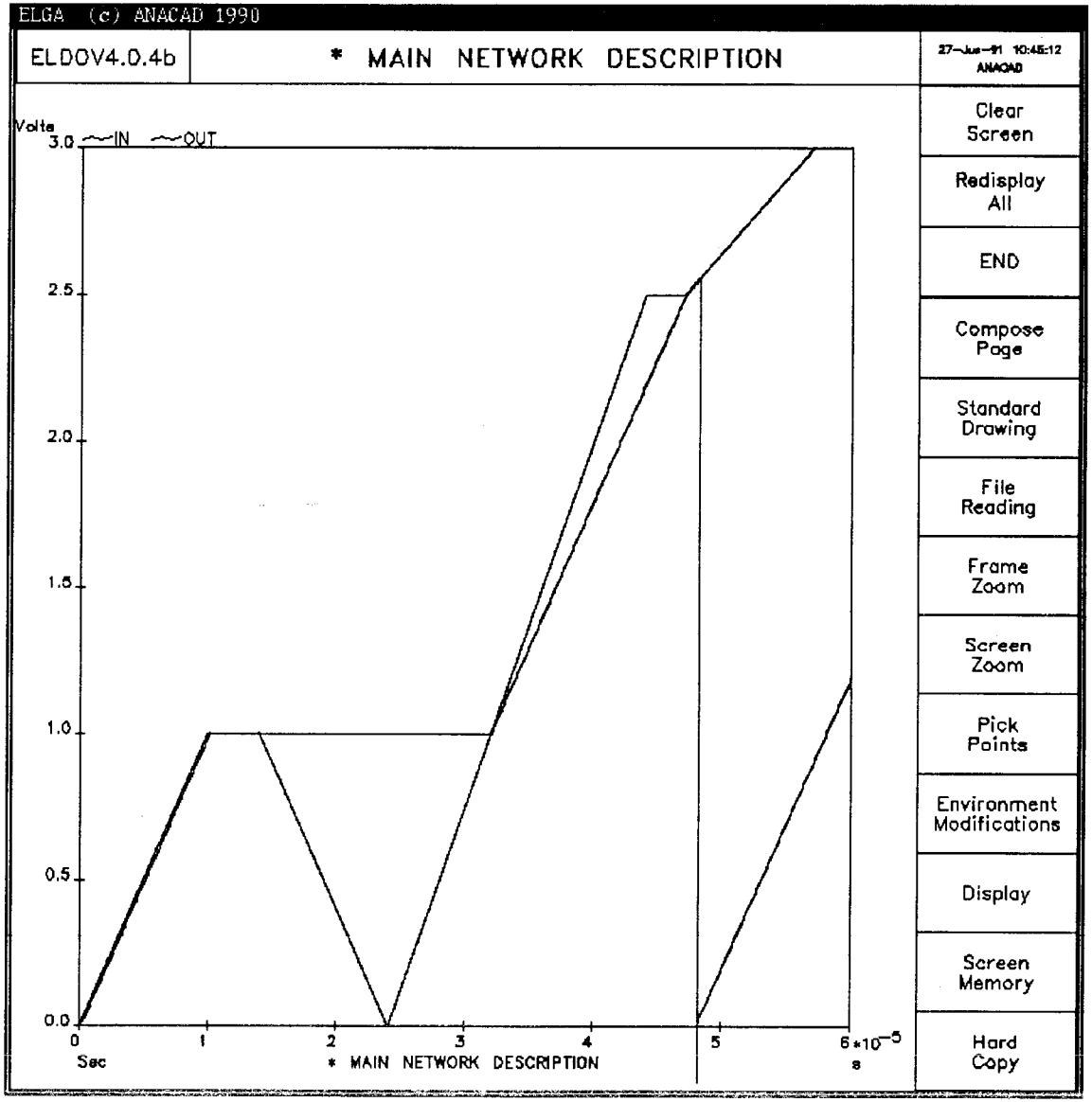

PEAK DETECTOR CIRCUIT
MACROP
IN
CK2
OUT7
OUT6
OUT5
OUT4
OUT3
OUT2
OUT1
OUT0
MACRO
1
2
3
4
5
6
7
8
9
10
7
6
5
4
3
2
1
0
CLK
OUT
DAC
11
COMP
+
-
IN
OUT
OPA1
+VIN
DC=*
AC=*
V
ANACAD

```
fmodel macrop(IN,OUT0,OUT1,OUT2,OUT3,OUT4,OUT5,OUT6,OUT7,CK2);
declare INPUT IN: LOGIC;
declare OUTPUT OUT0: LOGIC;
declare OUTPUT OUT1: LOGIC;
declare OUTPUT OUT2: LOGIC;
declare OUTPUT OUT3: LOGIC;
declare OUTPUT OUT4: LOGIC;
declare OUTPUT OUT5: LOGIC;
declare OUTPUT OUT6: LOGIC;
declare OUTPUT OUT7: LOGIC;
declare OUTPUT CK2: LOGIC;
declare STATE ck,enp,compt(0:7): logic;


initialize enp to 1 ;
initialize compt to 0 ;
initialize ck to falls ;

plot compt;
plot enp;
plot in;
```

```
functional

when ck changes
        within 50000 to 50000 make ck = not ck
        endwithin

        within 4000 to 4000 make ck2 = ck
        endwithin;
when in changes
        make enp = not in ;

when ck rises and enp = 0
        ns
        within 2000 to 2000
        make compt = compt + 1
        make out0 = compt(0)
        make out1 = compt(1)
        make out2 = compt(2)
        make out3 = compt(3)
        make out4 = compt(4)
        make out5 = compt(5)
        make out6 = compt(6)
        make out7 = compt(7)
        endwithin
        endmode ;
endfunctional
endmodel
```

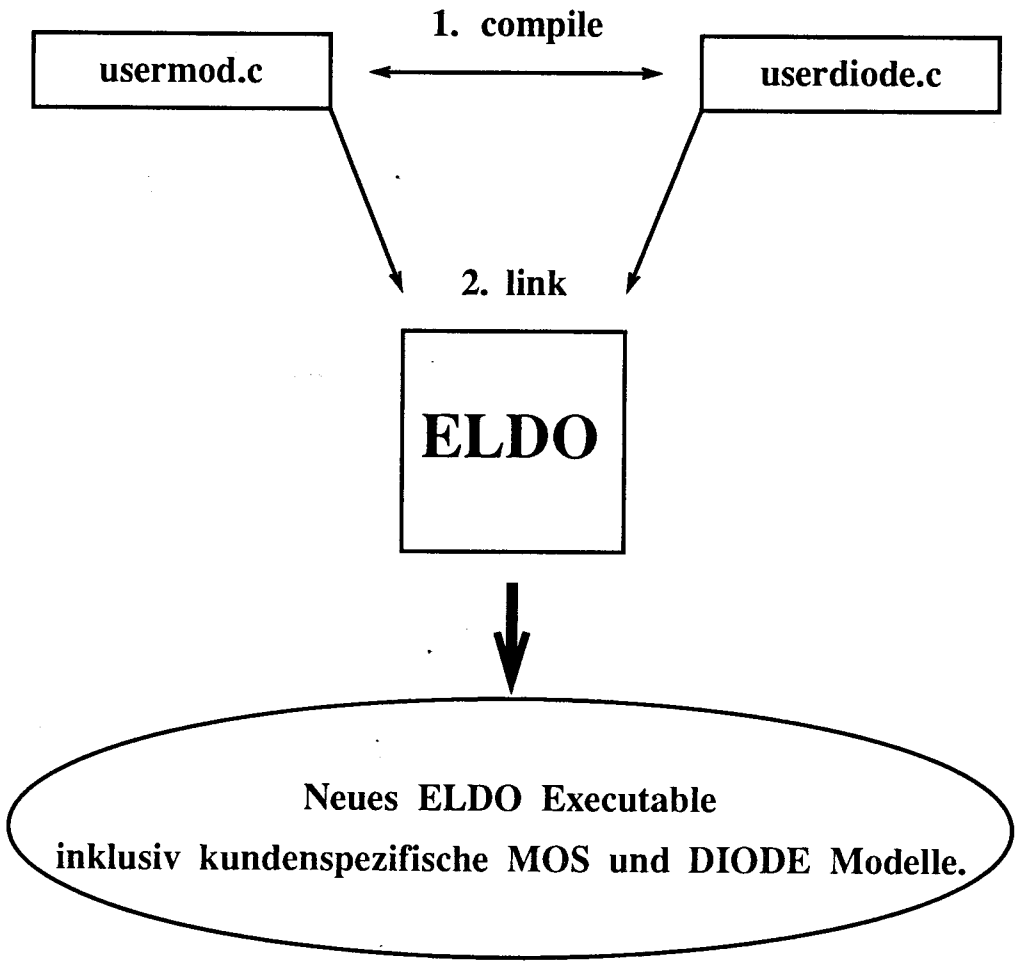

ELGA (c) ANACAD 1990
FIDELDO v4.0
* MAIN NETWORK DESCRIPTION
27-Jun-91 10:45:12
ANACAD
OMACRO::MACROP.IN
OMACRO::MACROP.ENP
OMACRO::MACROP.COMPT
Clear Screen
Redisplay All
END
Compose Page
Standard Drawing
File Reading
Frame Zoom
Screen Zoom
Pick Points
Environment Modifications
Display
Screen Memory
Hard Copy
nano
* MAIN NETWORK DESCRIPTION
Nano

ANACAD


ELGA (c) ANACAD 1990
ELDOV4.0.4b
* MAIN NETWORK DESCRIPTION
27-Jun-91 10:45:12
ANACAD
Volts
IN
OUT
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5
6*10-5
Sec
* MAIN NETWORK DESCRIPTION
s
Clear Screen
Redisplay All
END
Compose Page
Standard Drawing
File Reading
Frame Zoom
Screen Zoom
Pick Points
Environment Modifications
Display
Screen Memory
Hard Copy

ANACAD

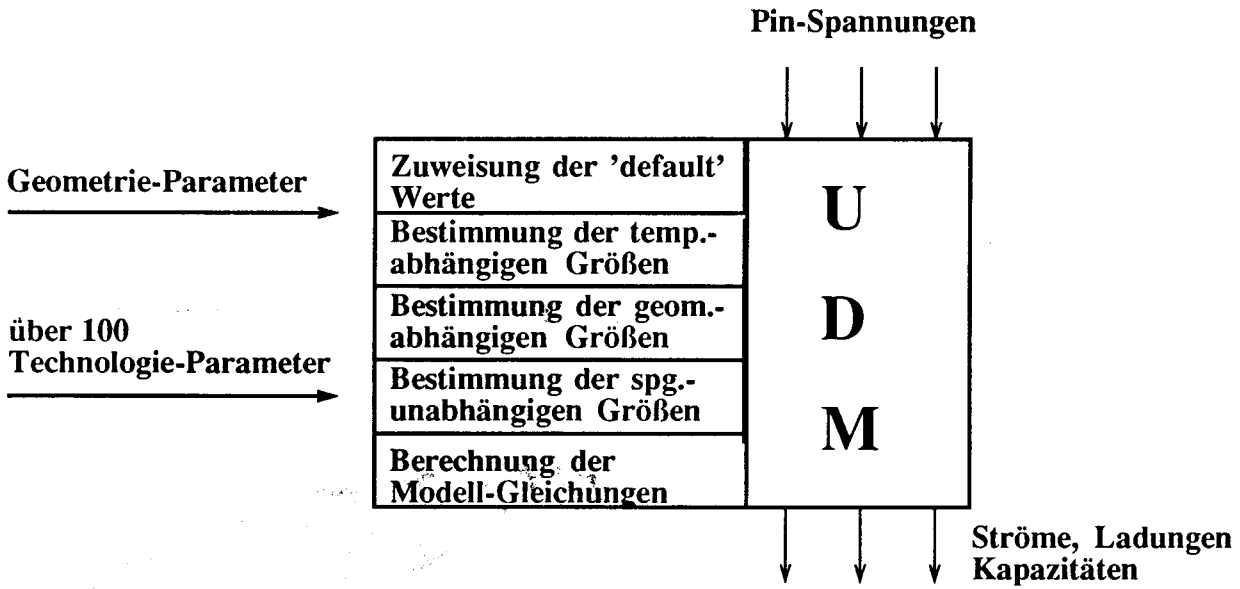
ELDO-UDM - User Definable Models



ELDO-UDM

ANACAD

Struktur eines Modells



ELDO-UDM

ANACAD

Anleitung

Vom programmtechnischen Gesichtspunkt ist das Erstellen der Modelle sehr einfach.

Der Anwender benötigt zur vereinfachten Erstellung der Modelle das Gerüst eines existierenden UDM's.

Die Modelle bestehen aus einer Anzahl gut dokumentierter Funktionen.

Das Berechnen von Ableitungen ist nicht notwendig.

Das Interface wird auch von ANACAD-Entwicklern zur Erstellung von MOS und Dioden-Modellen benutzt.

ELDO-UDM

2. ASIC unterstützt Sequenz-Suche, Frequenz- und Phasenmessung

H.Kreutzer, W.Ludescher

FH RT, FH RV_WG

Kurzfassung

Ein ASIC zur Unterstützung der Zeichensequenzsuche in Datenströmen und zur Frequenz- und Phasenmessung wird beschrieben. Die einzelnen Baugruppen werden auf einem Gate-Array in einer 1.5u-Technologie realisiert.
Das Gate-Array hat eine Komplexität von ca. 2000 Gattern, die Toggle-Rate der Sequenz-Flip-Flops (250MHz) bestimmt - neben der Zugriffszeit auf ein externes, schnelles Sequenz-RAM - die maximale Datenrate von ca. ½ GBit/sec, die ein Bitstrom aufweisen darf, der auf bestimmte Zeichensequenzen abgesucht werden soll.
Die Toggle-Rate des ersten Teiler-Flip-Flops bestimmt die obere Grenzfrequenz der Frequenzmessung (250MHz) und die Auflösung bei der Phasenmessung.

Das Gate-Array entstand in Zusammenarbeit der Fachhochschulen Reutlingen und Ravensburg-Weingarten im Rahmen eines Multi-Projekt-Chips (MPC). Wir danken dem Land Baden-Württemberg, das im Rahmen der MPC-Aktivitäten die Fertigung ermöglicht.
H.Kreutzer, W.Ludescher

Semicustom-IC unterstützt Sequenz-Suche, Frequenz -u. Phasenmessung

1) - Einleitung

Gelegentlich hört man am Arbeitsplatz den Spruch:

"Die Phase, sie springt hin und her,
der Ingenieur versteht's nicht mehr"

Um das Verständnis für die Grundlagen der Elektrotechnik und der Digitaltechnik im Rahmen der Lehre "durch Anfassen" zu vertiefen wurde ein ASIC entwickelt, der einfache digitale Schaltungskomponenten, als Baukastensystem ausgelegt, beschreibt, um meßtechnische Aufgaben zu lösen.

Meßaufgaben:

a) Suche einer binären Datensequenz
b) Frequenzmessung
c) Messung der Periodendauer
d) Messung der Phase

Die Komponenten sind auf einem Gate-Array realisiert, wichtige Teilschaltungen sind von außen einzeln zugänglich.
Das Gate-Array arbeitet als Peripherie-Baustein eines Single-Chip-Mikroprozessors, die Meßwerterfassung erfolgt vollständig in Hardware und erlaubt daher Signalfrequenzen von über 200 MHz.
Der Prozessor dient als Benutzer-Schnittstelle, nimmt Meßbefehle entgegen, erlaubt Messzyklen, liest Daten aus dem Gate-Array und stellt sie auf einer Anzeige dar.
Bild 1 gibt einen Überblick über die wesentlichen Baugruppen.

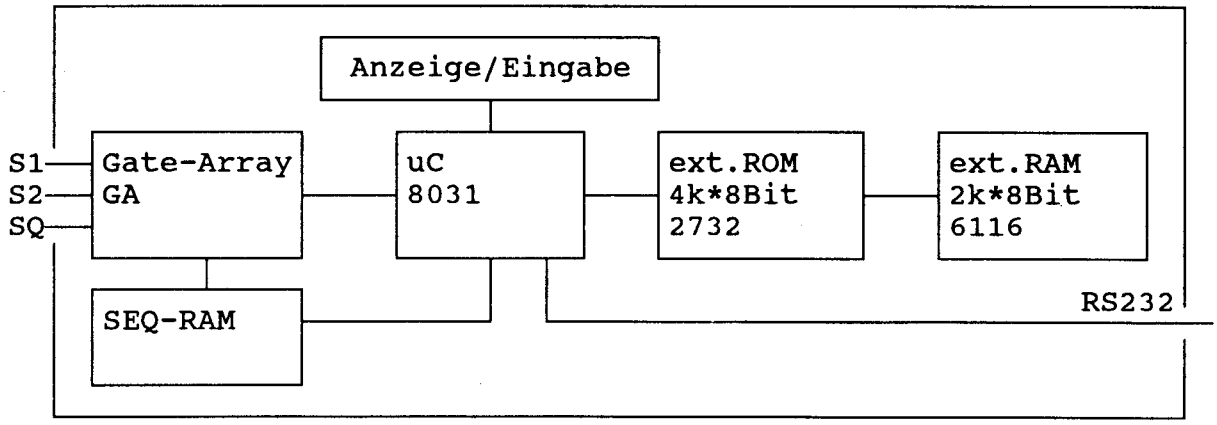


Bild 1 - Blockschaltbild im Überblick

2) Einzelbaugruppen

2.1) Sequenz-Suche

In modernen Datenkommunikationssystemen findet der Auf- u. Abbau von Verbindungen, sowie der Austausch der Daten mit Hilfe komplexer Protokolle statt. Die Steuer- sowie Adreßinformationen werden dabei zusammen mit den Daten auf der gleichen Leitung übertragen. Es ergibt sich somit die Notwendigkeit zur schnellen Analyse des Datenstromes um bestimmte Bit- und Zeichensequenzen zu detektieren, insbesonders dadurch, daß die Taktraten in den Datenkommunikationssystemen immer größer werden. Zur Lösung dieses Problems wird eine Schaltung, sowie deren Herleitung, vorgestellt. Diese Schaltung ermöglicht die Suche nach einer beliebig vorgebbaren Zeichensequenz bis zu hohen Taktraten.

Aufgabe der Sequenzsuchschaltung

Es soll nach einer Zeichensequenz gesucht werden können, wobei für die einzelnen Zeichen auch eine Menge von Zeichen aus einem Zeichenvorrat zugelassen sind. Die Menge der Zeichen bestehen aus dem Zeichenvorrat, der z.B. durch 8 Bit gebildet werden kann. Die Länge der zu suchenden Zeichensequenz soll einstellbar sein.

Beispiel für eine Suchsequenz mit der Länge von 3 Zeichen:

Menge der Zeichen die als 1. zu suchendes Zeichen zugelassen sind: {G}

Menge der Zeichen die als 2. zu suchendes Zeichen zugelassen sind: {F,H,I}

Menge der Zeichen die als 3. zu suchendes Zeichen zugelassen sind: {O,F,H,I,Z}

Mit {G,F,I,O,H,Z} ϵ {Menge z.B. der EBCDIC-Zeichen}.

Modell der Sequenzsuchstrategie

Die Herleitung der Schaltungsstruktur wird mit Hilfe von Zustands-Übergangsdiagrammen durchgeführt. Dabei sind folgende Mengenbezeichnungen notwendig:

E_i = {Zeichen, die in der i-ten Position der Zeichensuchsequenz vorkommen dürfen}

N_i = {Zeichen, die in der i-ten Position der Zeichensuchsequenz nicht vorkommen dürfen}

Um den Algorithmus anschaulich darstellen zu können wird ein Zustandsdiagramm verwendet, das in Ebenen unterteilt ist. Die Ebenen entsprechen der jeweiligen Zeichenposition in der Suchsequenz, wobei die oberste (erste) Ebene der 1.Zeichenposition in der Suchsequenz, die zweite Ebene der 2.Zeichenposition in der Suchsequenz, usw. entsprechen (Bild 2).

Die Ebenen selbst sind in Zustände unterteilt, die sich in der Berücksichtigung der jeweiligen relevanten Position der Suchsequenz unterscheiden. In den einzelnen Zuständen werden Mengenunterscheidungen vorgenommen. So wird z.B. in der 2.Ebene in dem dort vorkommenden Zustand zwischen den Schnittmengen $E_1 \cap E_2$, $N_1 \cap E_2$, $E_1 \cap N_2$, $N_1 \cap N_2$ unterschieden. Für diesen Zustand in der 2.Ebene sind also das 1. und 2. Zeichen der Suchsequenz relevant. Ein Element aus $E_1 \cap E_2$ ist in der 1. und der 2. Zeichenposition der zu suchenden Sequenz zugelassen. Je nachdem zu welcher Schnittmenge in diesem Zustand das momentan untersuchte Datenstromzeichen gehört, wird zu einem entsprechenden Zustand weitergesprungen.

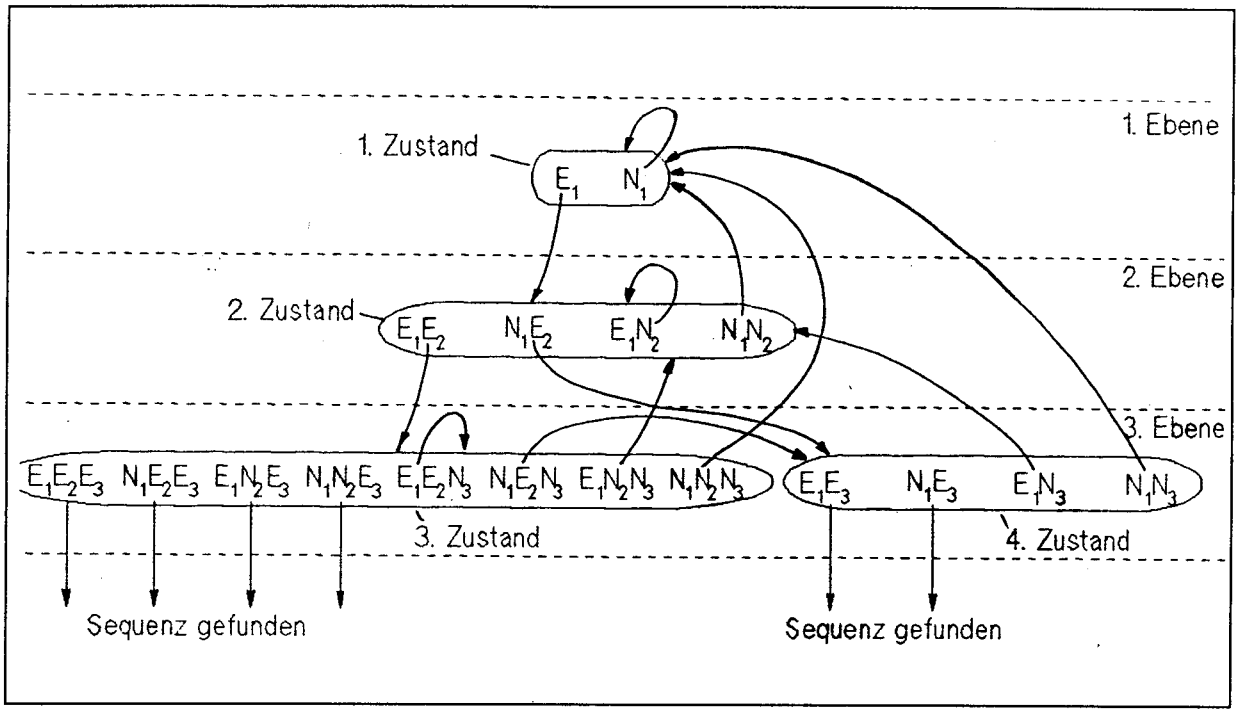


Bild 2: Zustandsdiagramm für eine Suchsequenz mit einer Länge von 3 Zeichen.
Dabei wird z.B. $X \in E_1 \cap E_2 \cap N_3$ durch $E_1E_2N_3$ abgekürzt.

Eine Ablauferläuterung der Sequenzsuche verdeutlicht den Aufbau des Modells (s.a.Bild 2). Falls Zeichen im Datenstrom auftreten, die nicht in der 1. Zeichenposition der Suchsequenz zugelassen sind (d.h. $x \epsilon N_1$ in der 1. Ebene), soll in der 1. Ebene verblieben werden. Sobald ein Zeichen im Datenstrom auftaucht, das in der ersten Zeichenposition der Suchsequenz zugelassen ist, wird in die zweite Ebene gesprungen. Das im Datenstrom folgende Zeichen muß nun daraufhin untersucht werden, ob es in der zweiten Zeichenposition und darüber hinaus, ob es auch in der ersten Zeichenposition der Suchsequenz zugelassen ist.
Damit gibt es folgende vier Unterscheidungsmöglichkeiten:

1) Ist $x \epsilon E_1 \cap E_2$, so wird in die 3.Ebene in einen Zustand weitergesprungen, in dem untersucht wird ob das nächste im Datenstrom zu untersuchende Zeichen in der 1., 2. oder 3. Zeichenposition der Suchsequenz zugelassen ist.

2) Ist $x \epsilon N_1 \cap E_2$, so wird ebenfalls in die 3. Ebene in einen Zustand weitergesprungen, in dem untersucht wird ob das im Datenstrom nächste zu untersuchende Zeichen in der 1. oder 3. Zeichenposition der Suchsequenz zugelassen ist. Im Gegensatz zu 1) muß nun nicht untersucht werden, ob dieses im Datenstrom zu untersuchende 3. Zeichen auch in der 2. Zeichenposition der Suchsequenz zugelassen ist, da das vorherige im Datenstrom untersuchte Zeichen nicht an der 1. Zeichenposition zugelassen war.

3) Ist $x \epsilon E_1 \cap N_2$, so ist das vorliegende Datenstromzeichen in der zweiten Zeichenposition nicht zugelassen. Die Sequenzsuche muß hier abgebrochen werden. Da das Datenstromzeichen aber in der ersten Zeichenposition der Suchsequenz zugelassen ist ($x \epsilon E_1$), kann direkt in die zweite Ebene gesprungen werden, um zu entscheiden, ob das nächste zu untersuchende Zeichen in der 2.Zeichenposition der zu suchenden Zeichensequenz zugelassen ist.

4) Ist $x \epsilon N_1 \cap N_2$, so ist das vorliegende Datenstromzeichen weder in der ersten noch der zweiten Suchsequenzposition zugelassen. Für das nächste zu untersuchende Datenstromzeichen muß die Sequenzsuche wieder in der 1.Ebene beginnen.

Diese Überlegungen sind sinngemäß für die weiteren Ebenen durchzuführen.

Systematik der Zustände

Aufgrund obiger Überlegungen kann eine Systematik der Zustände angegeben werden.
Die Kennzeichnung der Zustände ist eindeutig durch die in den jeweiligen Zuständen zu untersuchenden Suchsequenzpositionen möglich.
Dies wird für die Suchsequenzfolgen für 3, 4 und 5 Zeichen in Bild 3 dargestellt.
Zur Zustandskennzeichnung werden hier nur die Indizes, die die zu untersuchenden Positionen angeben, dargestellt.

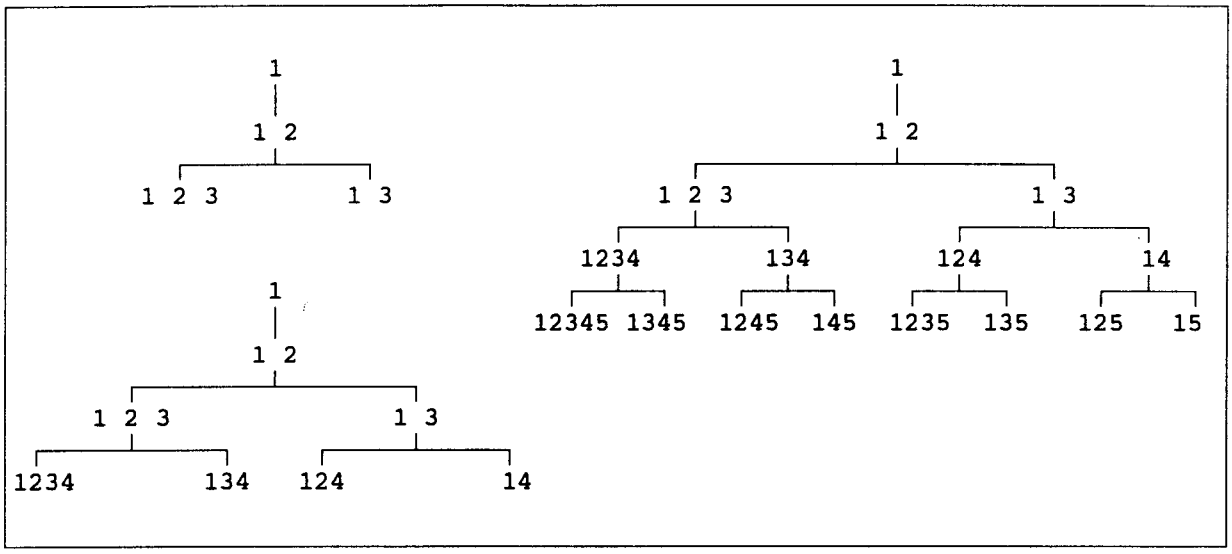


Bild 3 - Systematik der Zustände für Suchsequenzfolgen mit 3, 4 und 5 Zeichen

Die Bildungsregel der Zustandsanordnung

Die Anzahl der Zustände verdoppelt sich mit jeder tieferen Ebene. Jede in der Verzeigungs-systematik hinzukommende Ebene bedeutet die Erweiterung der Sequenzsuchfolge um ein Zeichen.
Jeder Zustand weist zwei Verzweigungen in die tiefere Ebene auf, die jeweils auf einen Zustand führt. Die zu untersuchenden Positionen werden aus dem abgeleiteten Zustand gebildet.

Bildungsregeln

Die erste relevante Position taucht in allen Zuständen auf und ist in den folgenden Bildungsregeln nicht enthalten.

In beiden abgeleiteten Zuständen tauchen alle im darüberliegenden Zustand enthaltenen Positionen um 1 inkrementiert auf.

Im ersten abgeleiteten Zustand taucht zusätzlich noch die relevante Position 2 auf.

Systematik der Verzweigungen

Die Anzahl der Zustände im Zustandsdiagramm ist exponentiell abhängig von der Anzahl der in der Suchsequenz zu suchenden Zeichen (Zustandsanzahl = $2^{\text{Zeichenanzahl}}$).
Unter Beachtung der Systematik der Verzweigungen kann die Anzahl der Zustände reduziert werden.
Die Sprungziele aus verschiedenen Zuständen heraus sind identisch, wenn nur die in diesen Zuständen mit E_i bezeichneten Positionen betrachtet werden. Die Zustände können zusammengefaßt werden, wenn man jedem Zustand eine Maske zuordnet, die angibt welche Positionen zur Untersuchung relevant sind. Diese Maske stellt eine Zustandskennzeichnung dar. Das Zusammenfassen der Zustände hat zum Ziel Speicherplatz zu sparen.
Aufgrund obiger Überlegungen kann die Sequenzsuche daher folgendermaßen durchgeführt werden:

1) Jedem Zeichen des Zeichenvorrates wird eine Kennung zugeordnet, aus der hervorgeht an welcher Position der Suchsequenz dieses Zeichen zugelassen ist.
2) Es wird eine Maske verwendet, die nur diejenigen Positionen zur Untersuchung freigibt, die aufgrund des Zustandes untersucht werden dürfen.
3) Es wird ein Decodierer benötigt, der entscheidet, in welchen Zustand gesprungen werden soll, d.h. welche Positionen im nächsten Schritt zur Untersuchung freigegeben werden.

Das Zustandsdiagramm (siehe Bild 2) ist so konstruiert, daß die um 1 inkrementierten, in den Sprungquellen mit E_i gekennzeichneten Positionen, in dem Sprungzielzustand untersucht werden müssen. Zusätzlich ist immer die erste Position zu untersuchen.

Realisierung der Sequenzsuchstrategie

Ordnet man den Zeichen des Zeichenvorrates eine logische 1 zu, wenn an einer bestimmten Position der Suchsequenz dieses Zeichen zugelassen ist, so muß jedem Zeichen des Zeichenvorrates ein Binärwort zugewiesen werden, das so lang wie die gesuchte Zeichensequenz ist. Dieser Teil der Schaltung wird aufgrund der sich ändernden Suchsequenzen durch ein RAM realisiert.
Ordnet man weiterhin der in einem Zustand mit E_i gekennzeichneten Position eine logische 1 zu, kann die Maske mit AND-Gatter realisiert werden. Der Maskenzustand wird für jede Position durch ein D-FF abgespeichert.

Damit ergibt sich eine Schaltung nach Bild 4 zur Suche einer beliebig langen Zeichensequenz.

Eigenschaften der Schaltung

Unter Verwendung schneller RAM-Bausteine ist es für die realisierte Testschaltung möglich, in einem Datenstrom der Datenrate von ½ GBit/s nach einer 8 Byte langen Zeichensequenz zu suchen. Man sieht, daß die Verarbeitungsgeschwindigkeit durch eine Erhöhung der Zeichensequenzlänge prinzipiell nicht vermindert wird. In der zu suchenden Zeichensequenz können an jeder Stelle mehrere oder alle Zeichen zugelassen sein. Die Suchsequenzlänge ist in der Testschaltung durch einen Multiplexer von 1 bis 8 einstellbar.

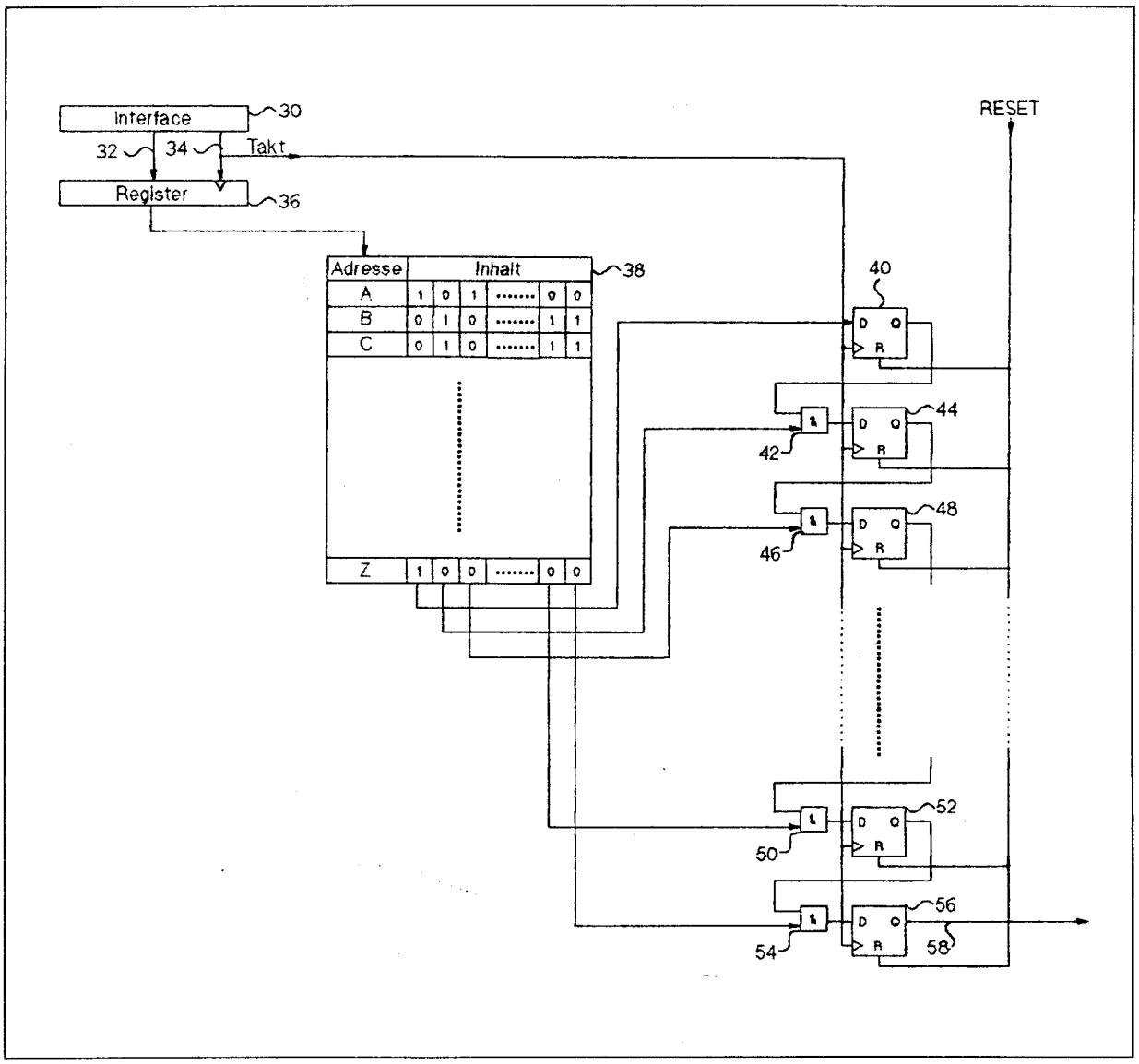


Bild 4 - Schaltung zur Suche einer beliebig langen Zeichensequenz

2.2) Frequenzmessung

An der Frequenzmessung sind drei Baugruppen des Gate-Arrays beteiligt. Es handelt sich dabei um die Zeitbasis ZB, die Zählerkette Zä und die Steuerung St.
Möchte man ein Signal S1 z.B. auf 1 Hertz genau messen, so muß man S1 mindestens eine Sekunde lang beobachten und dabei z.B. alle "steigenden" Nulldurchgänge von S1 zählen. Diese Beobachtungszeit wird üblicherweise "Torzeit" genannt, denn das Signal S1 durchläuft ein Gatter (Tor), ehe seine steigenden Flanken einen Zähler inkrementieren.
Bild 5 zeigt den Impulsplan, Bild 6 das Blockschaltbild dieser Betriebsart.

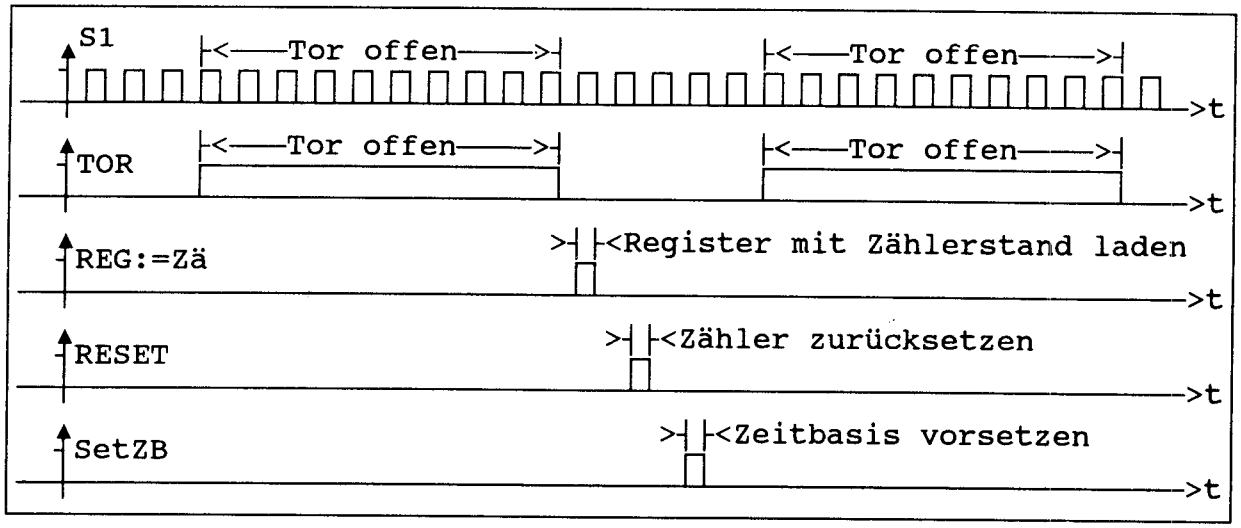


Bild 5 - Impulsplan Frequenzmessung

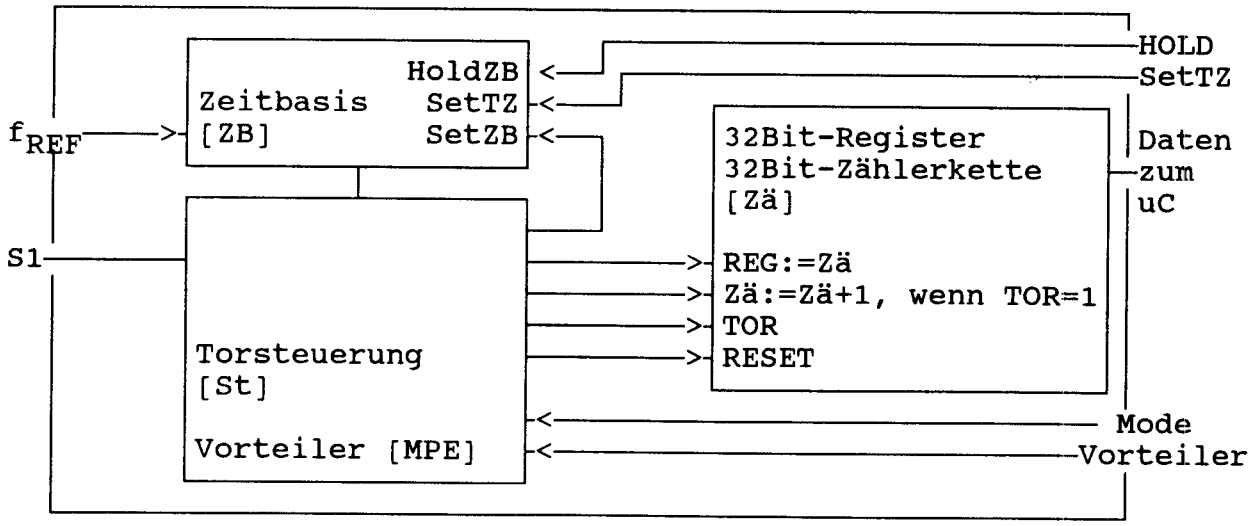


Bild 6 - Blockschaltbild Frequenzmessung

Um Frequenzen überhaupt genau zu messen wird ein hochgenaues Vergleichs-Signal benötigt, aus dem man die Torzeit ableitet.
Man nimmt dazu einen Quarz-Oszillator, packt ihn in einen kleinen Ofen, der eine hohe Wärmekapazität aufweist, sehr gut gegen die Umgebung isoliert ist und betreibt ihn bei jener Temperatur, bei welcher der TK des Oszillators minimal wird.
Damit lassen sich Referenz-Signale mit einer Genauigkeit von < 0.1ppm/°C erreichen.
Eine weitere Verbesserung der Genauigkeit ist durch die Anbindung dieses Quarz-Oszillators an einen Normal-Frequenz-Sender (z.B. DCF77, Cäsium-Normal,...) möglich.

Schaltungstechnische Realisierung

Die Zeitbasis [ZB] wird durch ein Referenz-Signal f_{REF}=1MHz gespeist. Mittels SetTZ läßt sich durch dekadische Teilerketten in ZB eine Torzeit auswählen, die an den Block [St] übergeben wird.
Wie der Impulsplan zeigt, realisiert St die Torzeit, während der die Impulse von S1 zum Zähler gelangen.
Nach dem Schließen des Tores wird der Zähler-Eingang von S1 getrennt und der Zählerstand von Zä an ein Register übergeben.
Anschließend wird der 32-Bit-Zähler Zä wieder auf HEX 0000:0000 gesetzt.
Jetzt könnte der Mikroprozessor Daten aus dem Register auslesen, was jedoch etwas Zeit benötigt. Mit dem Signal HOLD kann ein langsamer Prozessor die Steuerung anhalten bzw. die Totzeit verlängern, so daß alle Daten der Messung sicher aus dem Gate-Array übertragen werden.
Wird HOLD wieder freigegeben, generiert SetZB ein Signal an die Zeitbasis, das deren Zähler auf 999xxx vorsetzt, so daß eine aktive Flanke für die Torschaltung unmittelbar bevorsteht.
Diese Maßnahme verkürzt die Totzeit zwischen einzelnen Messungen, was besonders bei Torzeiten von 1 oder 10 Sekunden von Nutzen ist.

```
Kenndaten:  Signal  S1 : 0Hz < S1 < 250MHz
            Torzeit TZ : 1ms < TZ < 0.1s
            Zähler  ZÄ : 32 Bit, Binär
            Totzeit    : <10ms ohne HOLD
```

2.3) Messung der Periodendauer

Bei der Messung der Periodendauer eines Signals S1 wird die Torzeit nicht aus dem Referenz-Signal der Zeitbasis, sondern aus dem Signal S1 abgeleitet.
Die Torzeit ist dann z.B. die Zeit zwischen 2 steigenden Flanken von S1.
Während dieser Zeit laufen Referenz-Impulse der Zeitbasis auf die Zähler und inkrementieren diese.
Nach dem Schließen des Tores wird der Zählerstand im Register REG gespeichert, das durch den Mikroprozessor wieder ausgelesen und zur Anzeige gebracht werden kann.
Um die Meßgenauigkeit zu erhöhen kann ein mehrstufiger, binärer Vorteiler (Block MPE) mit dem Signal S1 gespeist werden. Man spricht dann von einer Multi-Periodendauer-Messung.

2.4) Phasenmessung

Die Phasenverschiebung von S1 und S2 soll gemessen werden.
Dazu bilden wir eine EXOR-Verknüpfung von S1 und S2.
Der Zeitraum t_P, während dem (S1)EXOR(S2) wahr ist, entspricht der Phasenverschiebung (Bild 6).
Der Periodendauer T entspricht ein Winkel von 360°.
Durch den Mikroprozessor wird t_P in den Phasenwinkel umgerechnet.

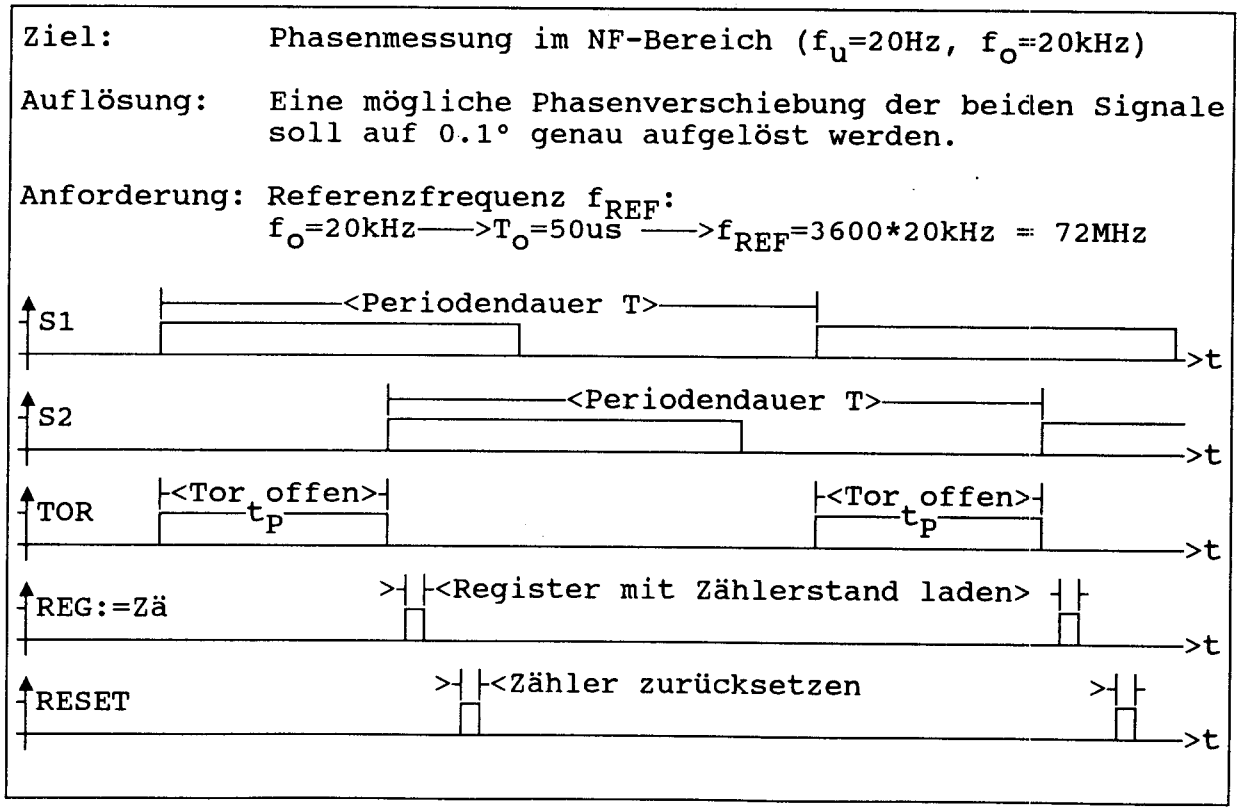


Bild 7 - Impulsplan Phasenmessung

3) Analyse der Testbarkeit

Naturgemäß enthält der Schaltungsentwurf einen großen Anteil an sequenzieller Logik.

- Sequenzsuche: div. Flip-Flops
- Zeitbasis: 8-stufiger, dekadischer Frequenzteiler
- Zähler-Gruppe: 32-Bit-Zähler mit Ausgabe-Register
- Steuerung: div. Flip-Flops

Alle 4 Blöcke arbeiten im Normalbetrieb voneinander unabhängig und sind über eigene Eingangs u.- Ausgangspins erreichbar.

Sequenz-Sucher

Durch die große Anzahl der Ausgangspins zum Zugriff auf das Sequenz-RAM ist die Kontrollierbarkeit und Beobachtbarkeit der Schaltungsknoten innerhalb des Sequenz-Suchers ohne zusätzlichen Aufwand bereits gegeben.

Zeitbasis

Die Zeitbasis wird durch einen Eingangs-Pin vom Normal-Betrieb in einen Testmodus umgeschaltet.
Im Testmodus sind alle Verbindungen zwischen den einzelnen Teilerdekaden aufgetrennt, so daß jede einzelne Dekade direkt erreichbar ist und mit einer kurzen Testsequenz durchgeprüft werden kann.

Zähler-Gruppe

Der 32-Bit-Zähler wird mittels eines Eingangs-Pins vom Normal-Betrieb in einen Testmodus umgeschaltet. Im Testmodus sind die Zählerstufen zu 8 parallelen 4-Bit-Binärzählern verschaltet. Die Gruppen werden parallel inkrementiert und über eine Parallel-Schnittstelle ausgelesen.

Steuerung

Die Steuersignale werden aus einer Schieberegister-Kette abgeleitet, deren Ausgangs-Signale an externen Pins zur Verfügung stehen und damit gut beobachtbar sind.

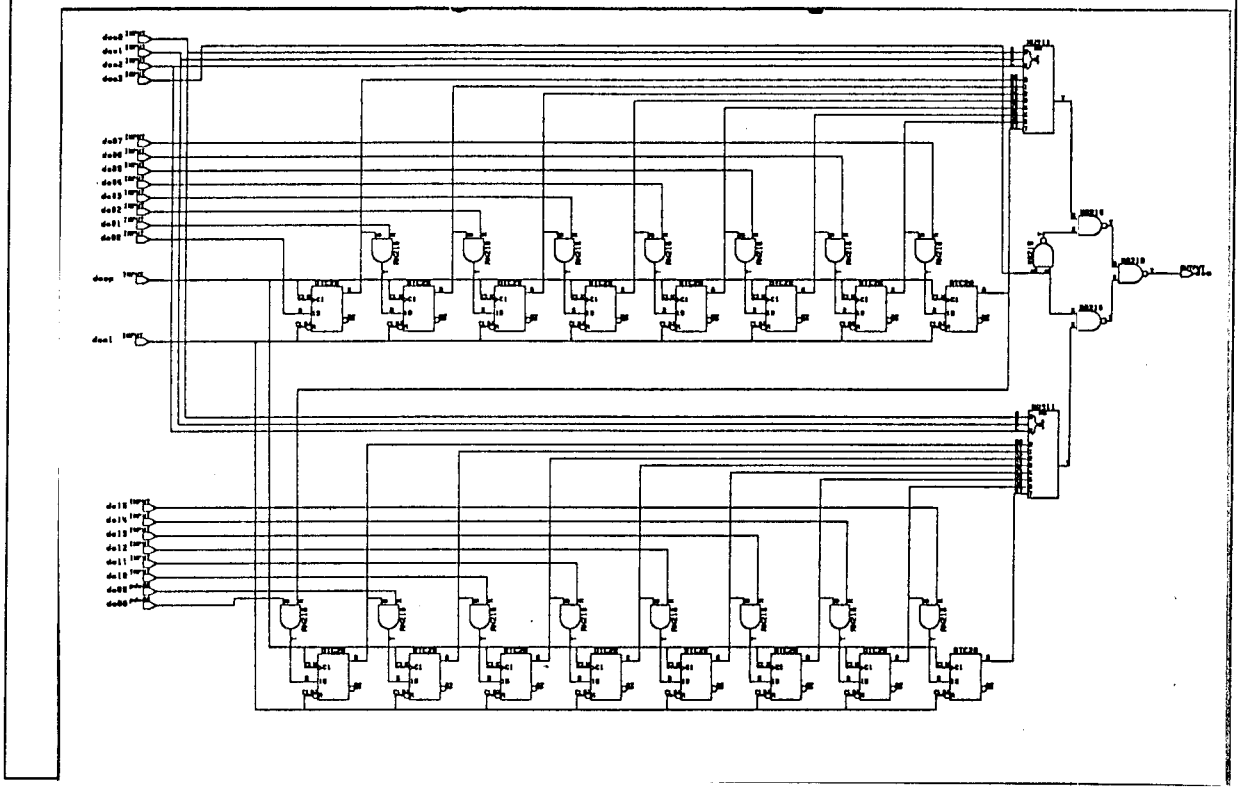
4) Ausblick

Der ASIC wird in den Bereichen "Grundlagen der Elektrotechnik" und "Digitaltechnik" für Laborprojekte eingesetzt werden.

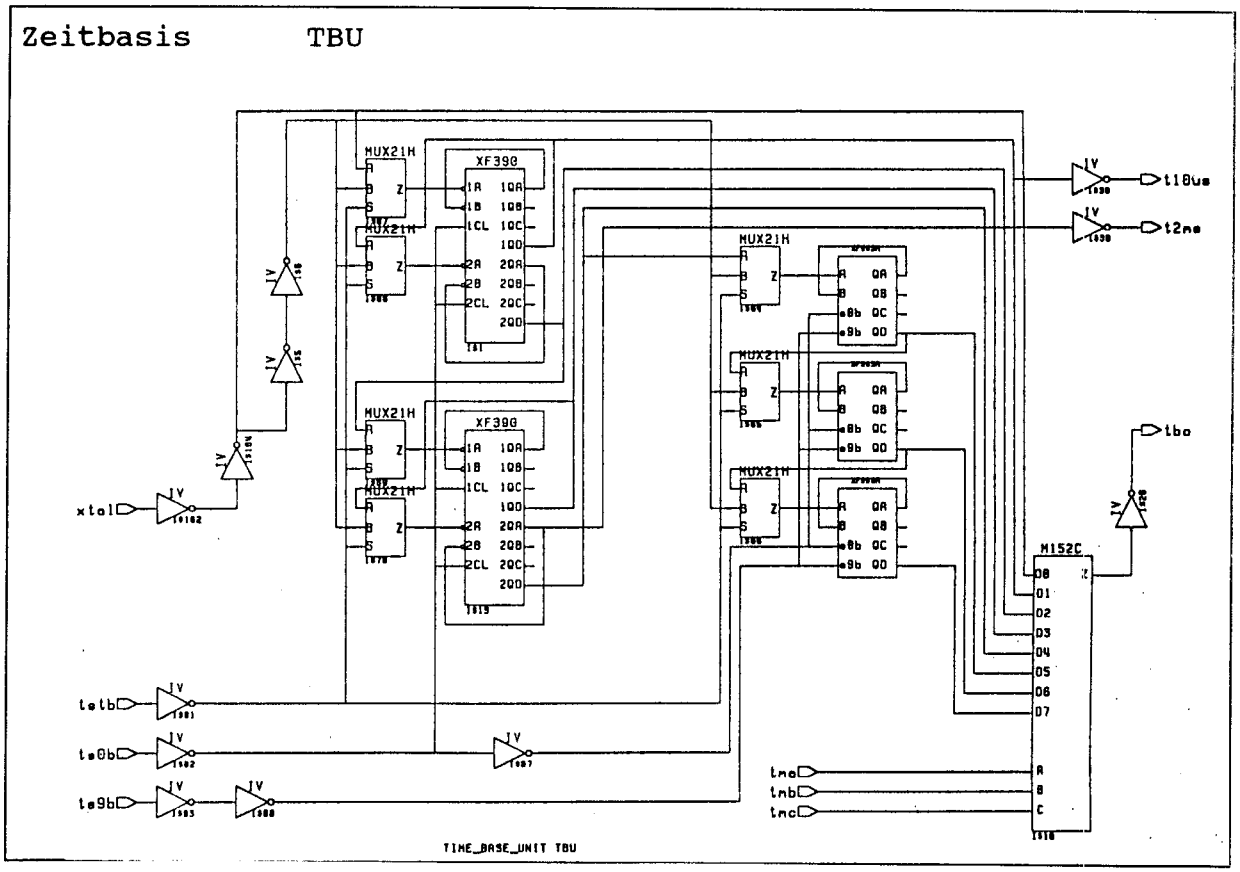
5) Anhang

Schaltungsauszüge

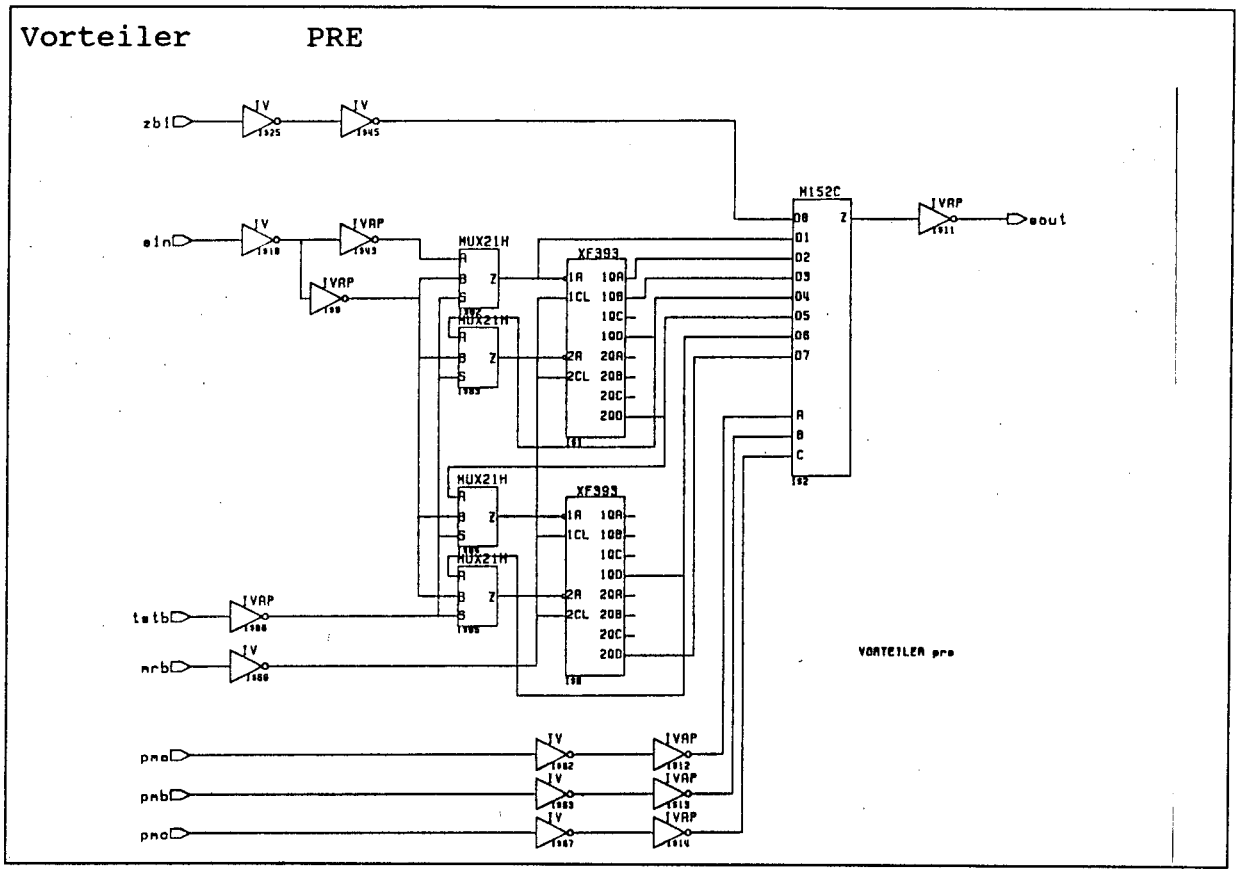
Sequenz-Sucher SEQ

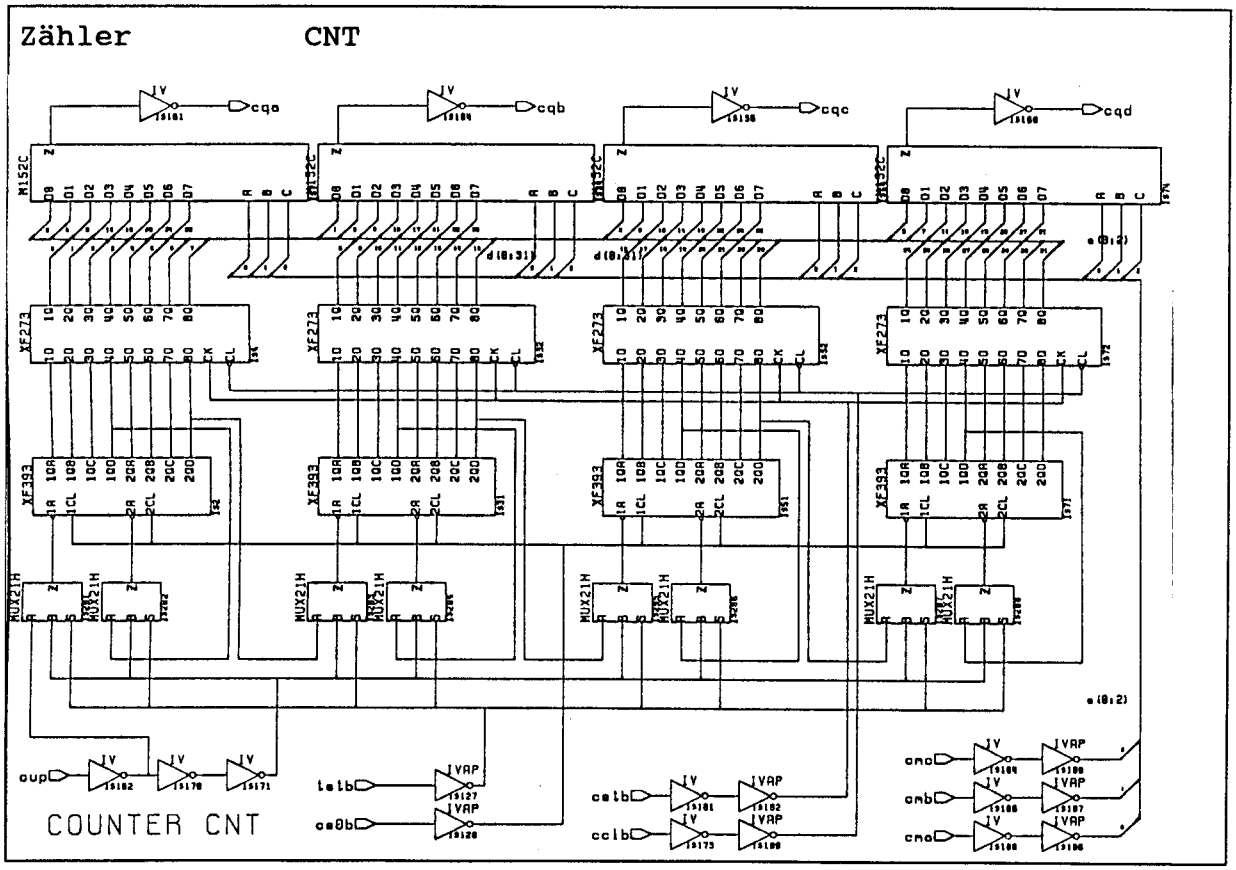


Zeitbasis TBU



Schaltungsauszüge





3. DIGITAL (IIR) FILTER BIQUAD SECTION SIMULATED WITH PSPICE

H. NIELINGER

ABSTRACT:

This paper briefly describes a simple PSPICE model for an IIR direct form 2 biquad section. Structures consisting of up to four biquad sections can be analysed with the evaluation version of PSPICE. Two applications of the model in undergraduate electrical engineering education are given.

1. INTRODUCTION

Due to the importance of Digital Signal Processing concepts in modern electrical engineering an undergraduate course of electrical filters should also include basic information about digital filters. As the student will already have learned during the discussion of analog filters that more sophisticated systems are formed by cascading systems 1st or 2nd order he or she should not be surprised to find the same principle in the world of digital filters again. The step from the analog to the digital world is simply done by introducing the bilinear transformation [1] which maps the infinite analog frequency range to the finite digital frequency range (up to half of the sampling frequency). Provided that during a course on filters, PSPICE has been extensively used for exercises and simulated laboratory experiments for analog filters, a similar simulation tool for

digital filters in continuation of that valuable pseudo-experimental work would probably be highly appreciated by both the students and the instructor. Filter design programs like DISPRO [2] or FDAS [3] are rather costly and much too sophisticated for the DSP beginner. As the analog world is strongly related to the digital world by the bilinear transform why not use the analog simulator PSPICE, well known to the students to simulate digital filters?

2. IIR BIQUAD SECTION MODEL

Figure 1 shows the well known block diagram of a IIR biquad section in direct form 2 [1]. This form was chosen because only two additions were needed compared to direct form 1 which needs three. This decision not only saves adders but solves problems with the minus sign as adders are formed with inverting operational amplifiers. Though PSPICE provides a very simple adding facility by means of polynomial voltage controlled voltage sources this solution was not considered as coefficients of those sources cannot be parameterised.

Figure 2 shows the analog equivalent of Figure 1 in which the weighted additions are done with ideal operational amplifiers and the delays are simulated with matched transmission lines.

The controlled voltage sources E1 and E2 are introduced in order not to load the transmission lines. Input X, Output Y and the intermediate values W_o, W_1, W_2 which represent pure numbers in Figure 1 are signified as voltages in Figure 2 so the correspondence is easily registered.

Figure 3 shows the program of the model in Figure 2 as a parameterised PSPICE subcircuit. The parameters have, according to PSPICE rules to be given default values, these were selected as equal to one for reasons of simplicity. The actual parameters of the circuit will be taken from the information in the call statement which will be shown in the examples. For the following examples the subcircuit was placed in the library NOM.LIB.

3. Example 1:

COMPARISON OF AN ANALOG AND A DIGITAL SYSTEM 2ND ORDER

Figure 4 shows a passive analog system 2nd order which can be considered as low pass filter (LP), highpass filter (HP), bandpass filter (BP), bandstop filter (BS) or lowpass notch filter (LPN) dependent on which voltage is referred to the excitation voltage V1.

For further discussion the following definitions and relations apply:

f = frequency in the analog range

f' = frequency in the digital range

f_o = resonant frequency

f_p = passband frequency

f_{max} = frequency for maximum in Chebyshev characteristic.

f_z = frequency of the transfer function zero for LPN

f_s = sampling frequency

$\Omega = \frac{f}{f_o}$ = normalised frequency in the analog range

For the circuit in Figure 4 one finds

$$\omega_0 = 2\pi f_o = \frac{1}{\sqrt{(L_1 + L_2)C_1}}$$

$$\omega_z = 2\pi f_z = \frac{1}{\sqrt{L_2 C_1}}$$

$$Q = \frac{\omega_o(L_1+L_2)}{R_1} = \frac{1}{\omega_o C_1 R_1}$$

Table 1 shows the analog transfer functions for the different filters mentioned above and the corresponding digital transfer functions derived by applying the bilinear transform

$$S = 1\,\frac{z - 1}{z + 1}$$

where $S = \frac{s}{\omega_o}$ = normalised Laplace variable

and $1 = \cot(\pi\frac{f_o'}{f_s}) \; ! \; = 1$

The last equation signifies a special (simplest) case and is equivalent to

$$f_o' = \frac{f_s}{4} \tag{1}$$

The following abbreviations were used in Table 1.

$$K = \frac{1}{2 + \frac{1}{Q}} \tag{2}$$

$$A_2 = \frac{2 - \frac{1}{Q}}{2 + \frac{1}{Q}} \tag{3}$$

The numerical values of the circuit elements shown in Figure 4 are found by choosing a resonant frequency of

$$f_o = 1\text{kHz}$$

and a passband attenuation for the LP (Chebyshev characteristic) of $a_p = 3.01\ \text{db} \approx 3\text{db}$

which leads to [4]

$$\varepsilon = \sqrt{10^{\,a_p/10} - 1} = 1$$

With that the Q-factor can be calculated

$$Q = \frac{1}{\sqrt{2 - \frac{2\varepsilon}{\sqrt{1+\varepsilon^2}}}} \quad \frac{1}{\sqrt{2 - \sqrt{2}}} = 1.307 \qquad (4)$$

Q and f_o determine all filter characteristics except for LPN, for that filter

$$\Omega_z = 3$$

was chosen so all circuit elements in Figure 4 can be determined.

Figure 5 shows the frequency response of the different filters analysed by PSPICE where BS was left out because it is a special case of LPN ($\Omega_z = 1$). In Table 2 some special frequencies in the analog frequency range are listed. These are the passband frequencies f_p (for BP: 3db- frequencies!) and the frequencies f_{max} (for BP $f_{max} = f_o$!). In the first column general relations are listed, in the second numerical values for the example in Figure 4 and in the third the numerical values of the corresponding

frequencies in the digital frequency range (calculated by means of the bilinear transform formula for frequencies for $l = 1!$):

$$f' = \frac{f_s}{\pi} \arctan \Omega$$

Students should be asked to measure those frequencies in the analog frequency range by means of the CURSOR function of PSPICE and to calculate the corresponding frequencies in the digital frequency range using the formula given above in order to experience the nonlinear mapping of frequencies by the bilinear transform.

Figure 6 shows the frequency response of the corresponding digital filters to Figure 4 where the PSPICE program given in Figure 7 was used. It was demanded that

$f_o' = f_o = 1\text{kHz}$

leading according to equation (1) to

$f_s = 4f_o' = 4\ \text{kHz}$

With equation (4) inserted in equation (2) and (3) one yields

$K = 0.3616$

$A_2 = 0.4466$

With the general digital transfer function given in Figure 1 and the information given in Table 1 the parameter lists shown in Figure 7 can easily be determined.

The simulation range for the frequency response in Figure 6 was chosen up to f_s = 4kHz in order to show the students the periodicity of the responses of digital filters. The compression of the frequency range is seen with one glance e.g. considering f_z = 3kHz and f_z' ≈ 1.6kHz! Measurements of the special frequencies given in Table 2 by means of the CURSOR function of PSPICE will show the compression quantitatively.

4. EXAMPLE II: SIMULATION OF A DIGITAL CAUER - LP 6TH ORDER

To discuss the theoretical background of Cauer filter design in undergraduate electrical engineering education is impossible! But even experienced theoretically orientated filter designers will rely nowadays on sophisticated filter design programs like those mentioned above [2], [3] and not design Cauer filters from scratch! It is therefore considered legitimate to discuss even with undergraduate students Cauer's optimal solution of the filter problem by equal ripple design both in passband and stopband. Though the filter design programs provide a lot of valuable information about the filters designed there are still questions unanswered which justify an additional simulation with the proposed PSPICE model. It is mainly the problem of ordering the systems 2nd order which are cascaded in order to form a higher order filter. Where the filter design programs deliver only an overall frequency response of a complicated system it is quite helpful and enlightening for

an engineer to see how the single systems 2nd order behave and how one can check that, at no point in the system, the restricted number range is exceeded. This will be demonstrated in the following example.

Figure 8 shows the result of a design of a digital Cauer-LP 6th order done with DISPRO [2]. Using the given coefficients of the three transfer functions 2nd order the PSPICE program in Figure 9 for this filter was written where the sub-systems with the lowest pole magnitude (lowest value for A_2 equivalent to lowest Q system) was taken as first, then the subsystem with the next lowest value of A_2 and so on. Figure 10 shows the overall frequency response and the responses of the subsystems or combination of the subsystems (node 3!).

With these different frequency responses, valuable insights into the problems, which engineers face all the time in DSP applications, namely danger of saturation on the one hand and numerical round-off noise on the other hand, can be experienced by undergraduate students to advantage.

5. CONCLUSION

A simple PSPICE model for the digital IIR biquad section has been presented which was written in the form of a parameterised subcircuit so it can be placed in a library file of PSPICE. The use of the model was demonstrated with two examples which run with the evaluation versions of PSPICE. It therefore opens possibilities of low-cost exercises and simulated laboratory experiments in undergraduate DSP education.

FIGURE CAPTIONS

Figure 1 IIR biquad section and its transfer function.

Figure 2 PSPICE model for the IIR biquad section.

Figure 3 Parameterised SUBCKT definition for the model in Figure 2.

Figure 4 Analog system 2nd order to be transformed into related digital systems.

Figure 5 Frequency responses of different filters formed by the circuit in Figure 4.

Figure 6 Frequency responses of the digital equivalent filters.

Figure 7 PSPICE program used for Figure 6.

Figure 8 DISPRO - design result of a Cauer-LP 6th order.

Figure 9 PSPICE program for the simulation of Cauer-LP 6th order.

Figure 10 Frequency response of the different stages of the Cauer-LP 6th order.

Table 1 Analog and correspondong digital transfer functions ($fo' = f_S/4$).

Table 2 Special frequencies in the analog and digital frequency range.

FOOTNOTE

Affiliation of author:

The author was with

School of Electrical and Computer Engineering,
Curtin University of Technology,
Perth, Western Australia, 6001

He is now with

Fachhochschule Furtwangen,
7743 Furtwangen,
Germany

REFERENCES

[1] A V Oppenheim, R W Schafer, Digital Signal Processing Prentice Hall, Englewood Cliffs, NY, 1975

[2] DISPRO (Digital filter design program, details will be given, when back in Germany!)

[3] FDAS Filter Design and Analysis System Version 2.2 (June 1990)
Momentum Data Systems 1520 Nutmeg Place, Suite # 108, Costa Mesa, CA 92626.

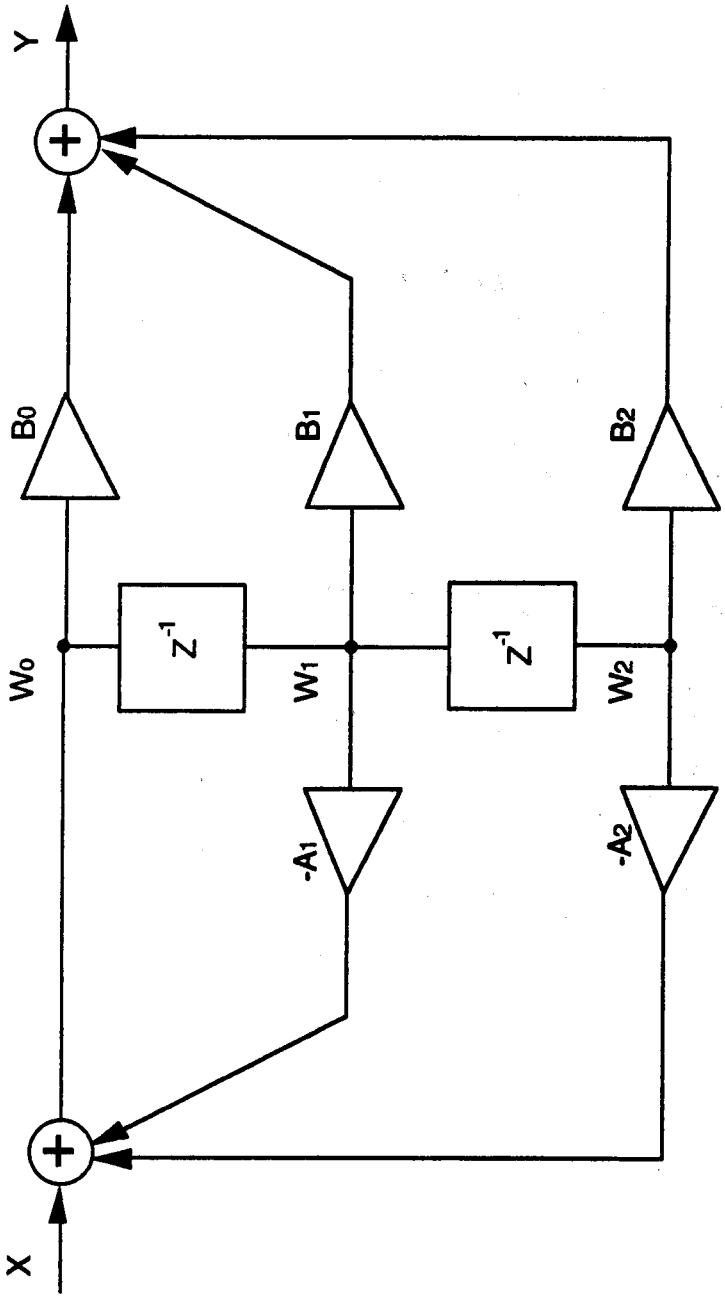
[4] M E van Valkenburg, Analog Filter Design, Holt, Rinehart and Winston, New York, 1982.

TABLE 1

	H(S)	H(z)
LP	$\frac{V_4}{V_1} = \frac{1}{S^2 + \frac{1}{Q}S + 1}$	$K \frac{(z+1)^2}{z^2 + A_2}$
HP	$\frac{V_{2,3}}{V_1} = \frac{S^2}{S^2 + \frac{1}{Q}S + 1}$	$K \frac{(z-1)^2}{z^2 + A_2}$
BP	$\frac{V_{1,2}}{V_1} = \frac{\frac{1}{Q}S}{S^2 + \frac{1}{Q}S + 1}$	$\frac{K}{Q} \frac{z^2 - 1}{z^2 + A_2}$
BS	$\frac{V_2}{V_1} = \frac{S^2 + 1}{S^2 + \frac{1}{Q}S + 1}$	$2K \frac{z^2 + 1}{z^2 + A_2}$
LPN	$\frac{V_3}{V_1} = \frac{1}{\Omega_z^2} \frac{S^2 + \Omega_z^2}{S^2 + \frac{1}{Q}S + 1}$	$\frac{\Omega_z^2 + 1}{\Omega_z^2} K \frac{z^2 + 2\frac{\Omega_z^2 - 1}{\Omega_z^2 + 1} z + 1}{z^2 + A_2}$

TABLE 2

	Ω	f	f'
LP	$\Omega_p = \sqrt{2\ (1 - \frac{1}{2Q^2})}$ $\Omega_{max} = \frac{\Omega_p}{\sqrt{2}}$	f_p = 1.189kHz f_{max} = 0.8409kHz	f_p' = 1.110kHz f'_{max} = 0.8902kHz
HP	$\Omega_p = \frac{1}{\sqrt{2(1 - \frac{1}{2Q^2})}}$ $\Omega_{max} = \Omega_p \sqrt{2}$	f_p = 0.8409kHz f_{max} = 1.189kHz	f_p' = 0.8902kHz f'_{max} = 1.110kHz
BP	$\Omega_{p2} = \frac{1}{\Omega_{p1}} = \frac{1}{2Q} + \sqrt{1 + \frac{1}{4Q^2}}$	f_{p1} = 0.6881kHz f_{p2} = 1.453kHz	f'_{p1} = 0.7674kHz f'_{p2} = 1.233kHz
LPN	$\Omega_p = \Omega_z\sqrt{2}\sqrt{\frac{\Omega_z^2(1 - \frac{1}{2Q^2}) - 1}{\Omega_z^4 - 1}}$ $\Omega_{max} = \sqrt{\frac{\Omega_z^2\ (1 - \frac{1}{2Q^2}) - 1}{\Omega_z^2 - (1 - \frac{1}{2Q^2})}}$	f_p = 1.099kHz f_{max} = 0.8042kHz f_z = 3kHz	f_p' = 1.060kHz f'_{max} = 0.8624kHz f_z' = 1.590kHz



$$H(z)=\frac{Y}{X}=\frac{B_0z^2+B_1z+B_2}{z^2+A_1z+A_2}$$

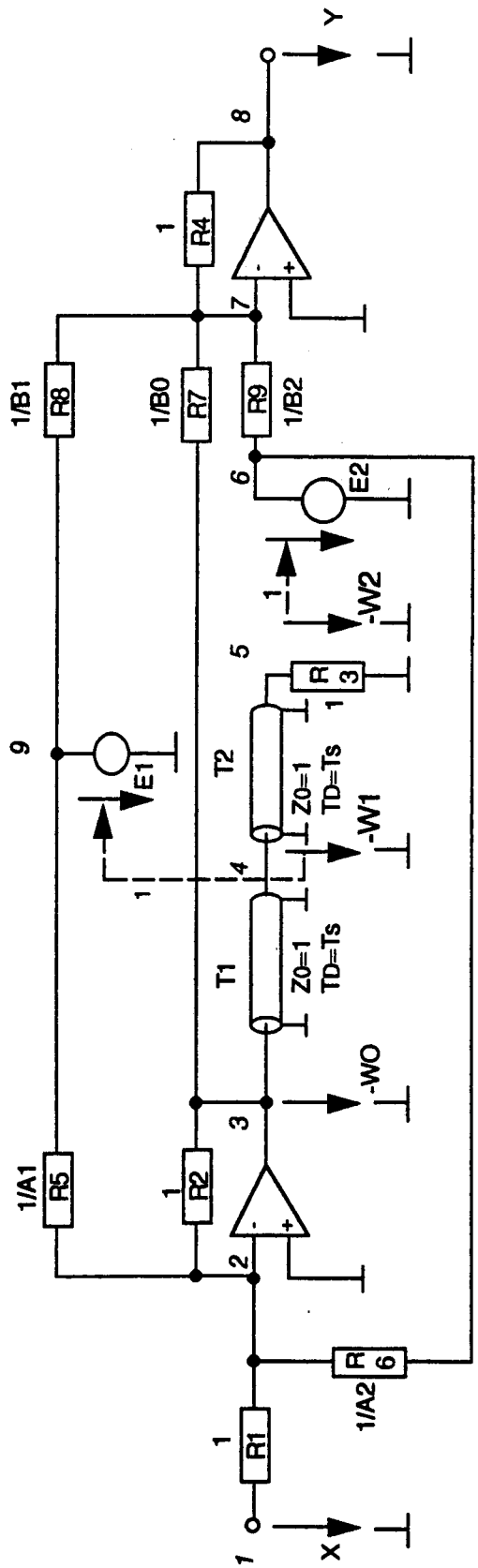

X
Y
R1
R2
R3
R4
R5
R6
R7
R8
R9
1/A1
1/A2
1/B0
1/B1
1/B2
E1
E2
T1
T2
Z0=1
TD=Ts
-WO
-W1
-W2
1
2
3
4
5
6
7
8
9

Fig. 2

```
.SUBCKT BQDIG 1 8 PARAMS: B0=1 B1=1 B2=1 A1=1 A2=1 TS=1
R1 1 2 1
R2 2 3 1
R3 5 0 1
R4 7 8 1
R5 9 2 {1/A1}
R6 6 2 {1/A2}
R7 3 7 {1/B0}
R8 9 7 {1/B1}
R9 6 7 {1/B2}
T1 3 0 4 0 Z0=1 TD={TS}
T2 4 0 5 0 Z0=1 TD={TS}
E1 9 0 4 0 1
E2 6 0 5 0 1
EOP1 3 0 0 2 1E6
EOP2 8 0 0 7 1E6
.ENDS
```

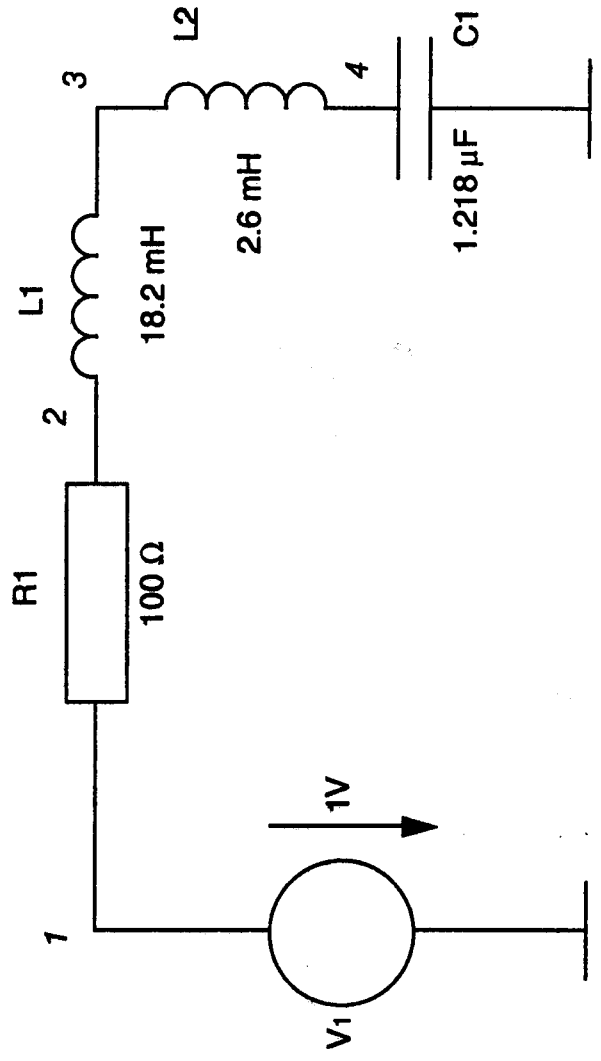
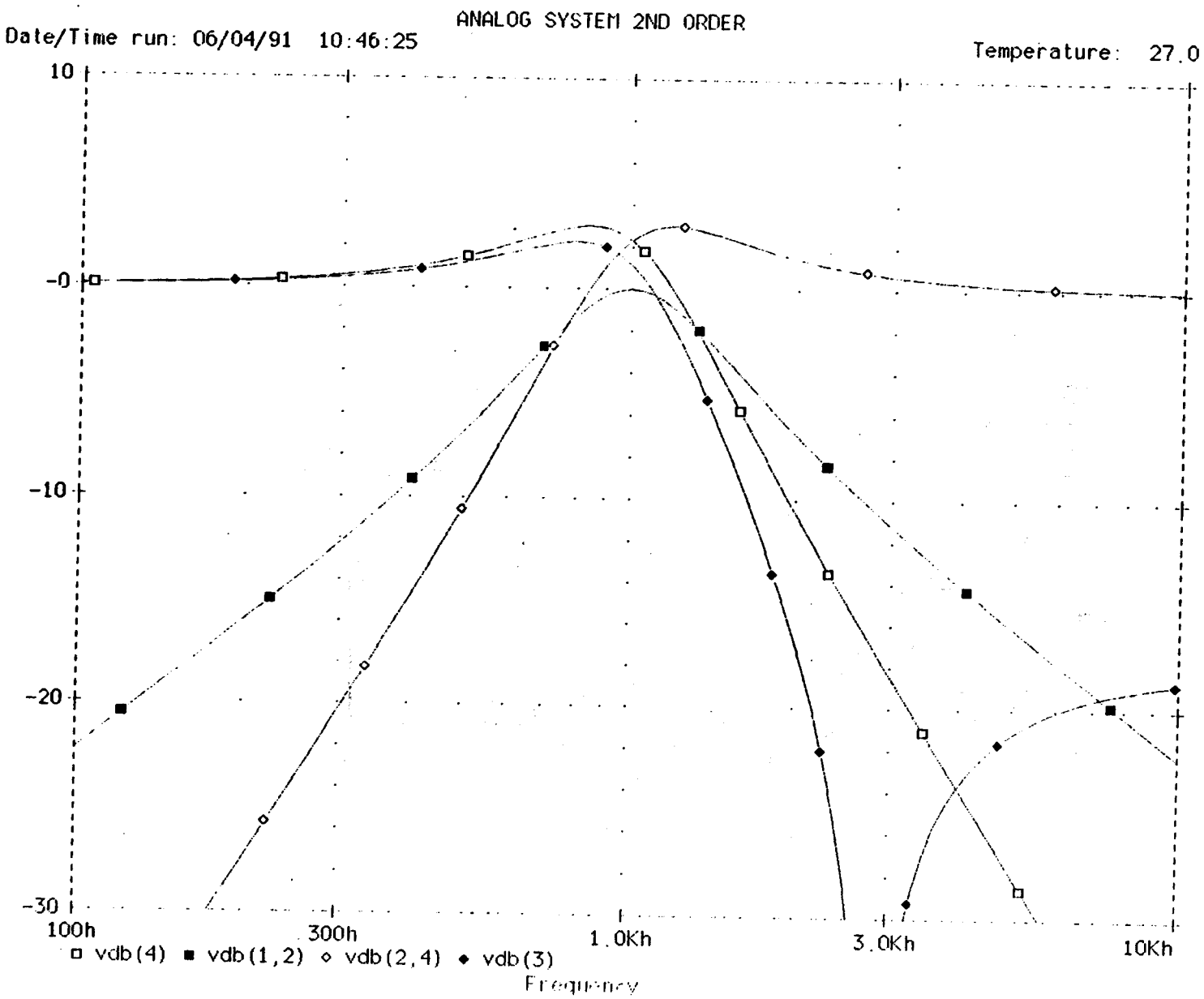


Fig 4


ANALOG SYSTEM 2ND ORDER
Date/Time run: 06/04/91 10:46:25
Temperature: 27.0
10
-0
-10
-20
-30
100h
300h
1.0Kh
3.0Kh
10Kh
vdb(4) vdb(1,2) vdb(2,4) vdb(3)
Frequency

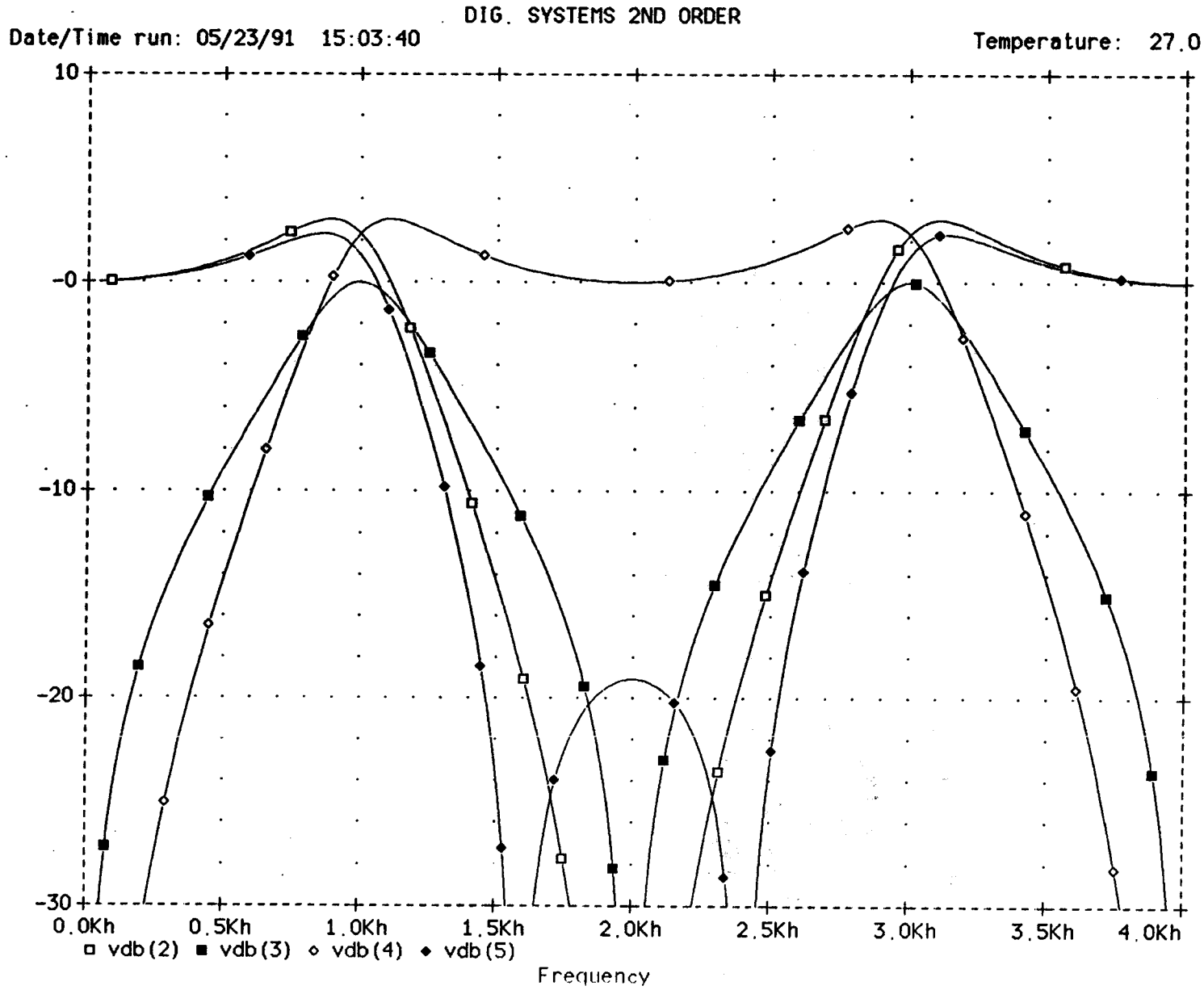

DIG. SYSTEMS 2ND ORDER
Date/Time run: 05/23/91 15:03:40
Temperature: 27.0
10
-0
-10
-20
-30
0.0Kh
0.5Kh
1.0Kh
1.5Kh
2.0Kh
2.5Kh
3.0Kh
3.5Kh
4.0Kh
vdb(2) vdb(3) vdb(4) vdb(5)
Frequency

Fig 6

```
DIG. SYSTEMS 2ND ORDER
.LIB
V1 1 0 AC 1
X1 1 2 BQDIG PARAMS: B0=.3616 B1=.7232 B2=.3616 A1=1U A2=.4466 TS=250U
X2 1 3 BQDIG PARAMS: B0=-.2767 B1=1E-6 B2=.2767 A1=1U A2=.4466 TS=250U
X3 1 4 BQDIG PARAMS: B0=.3616 B1=-.7232 B2=.3616 A1=1U A2=.4466 TS=250U
X4 1 5 BQDIG PARAMS: B0=.4018 B1=.6428 B2=.4018 A1=1U A2=.4466 TS=250U
.AC LIN 200 40 4K
.PROBE
.END
```

ELLIPTIC LOWPASS FILTER OF ORDER 6 (FILE B:ELLIP06A.DAT)

Sampling frequency (Hz) 10000
End of passband (Hz) 3023.9
Beginning of stopband (Hz) 3385.525
Maximum passband attenuation (dB) 2
Minimum stopband attenuation (dB) 60
Coefficient wordlength: floating point

COEFFICIENTS OF SECOND-ORDER SECTIONS

$$A\,\frac{Z^2 + D\,Z + E}{Z^2 + B\,Z + C}$$

A	D	E	B	C
0.6577277	1.0870550	1.0000000	0.6112135	0.9449614
0.5875868	1.3627340	1.0000000	0.2141723	0.7617255
0.1625832	1.8798521	1.0000000	-0.7485645	0.3793633

Fig 8

```
DIG.LP 6TH ORDER (CAUER)
.LIB
V1 1 0 AC 1
X1 1 2 BQDIG PARAMS: B0=.1626 B1=.3057 B2=.1626 A1=-.7486 A2=.3794 TS=100U
X2 2 3 BQDIG PARAMS: B0=.5876 B1=.8009 B2=.5876 A1=.2142 A2=.7617 TS=100U
X3 3 4 BQDIG PARAMS: B0=.6577 B1=.7149 B2=.6577 A1=.6112 A2=.9450 TS=100U
.AC LIN 200 50 5K
.PROBE
.END
```

Fig 9

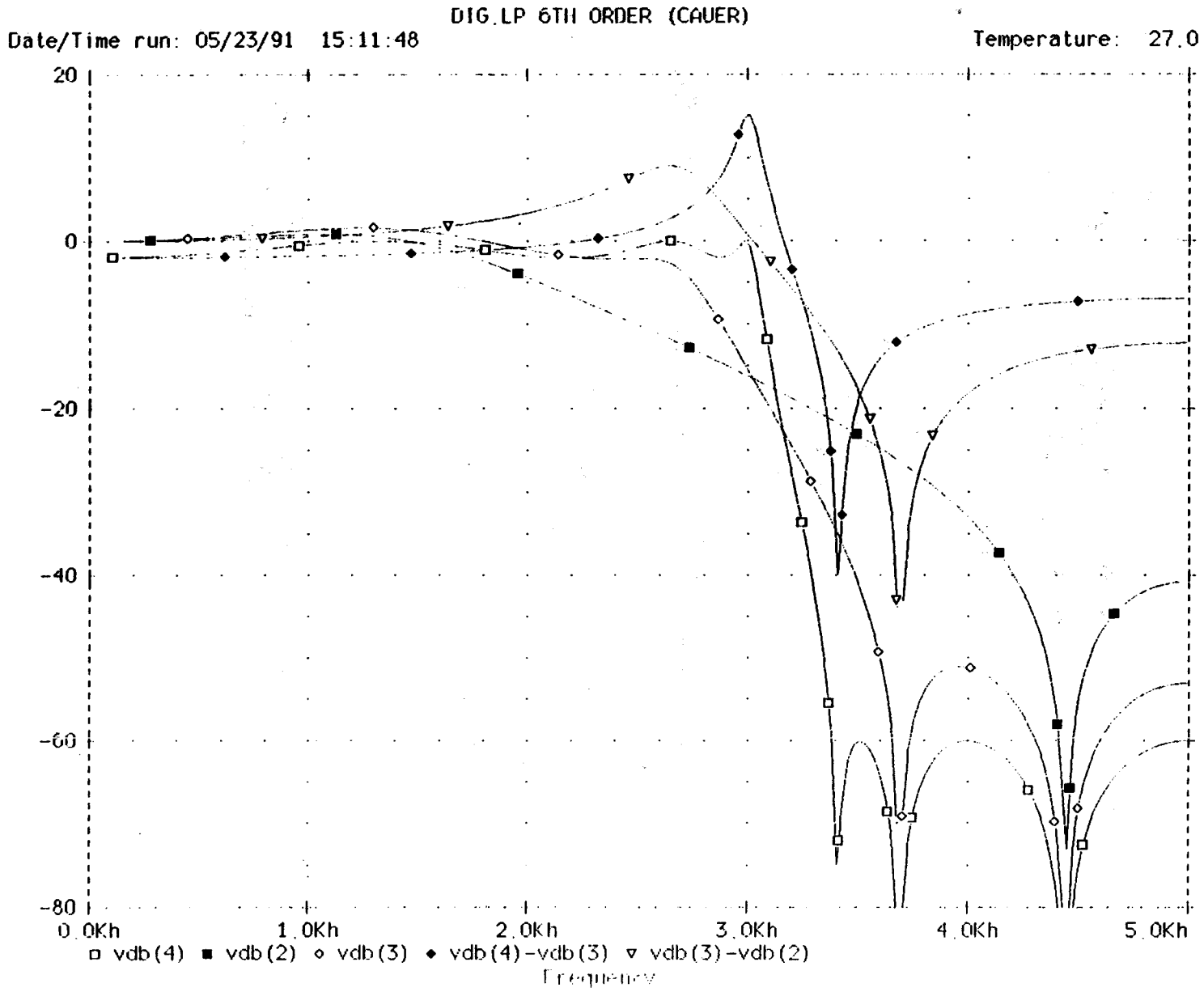

DIG.LP 6TH ORDER (CAUER)
Date/Time run: 05/23/91 15:11:48
Temperature: 27.0
20
0
-20
-40
-60
-80
0.0Khz
1.0Khz
2.0Khz
3.0Khz
4.0Khz
5.0Khz
vdb(4) vdb(2) vdb(3) vdb(4)-vdb(3) vdb(3)-vdb(2)
Frequency

Fig 10

ELLIPTIC LOWPASS FILTER OF ORDER 6 (FILE B:ELLIP06A.DAT)

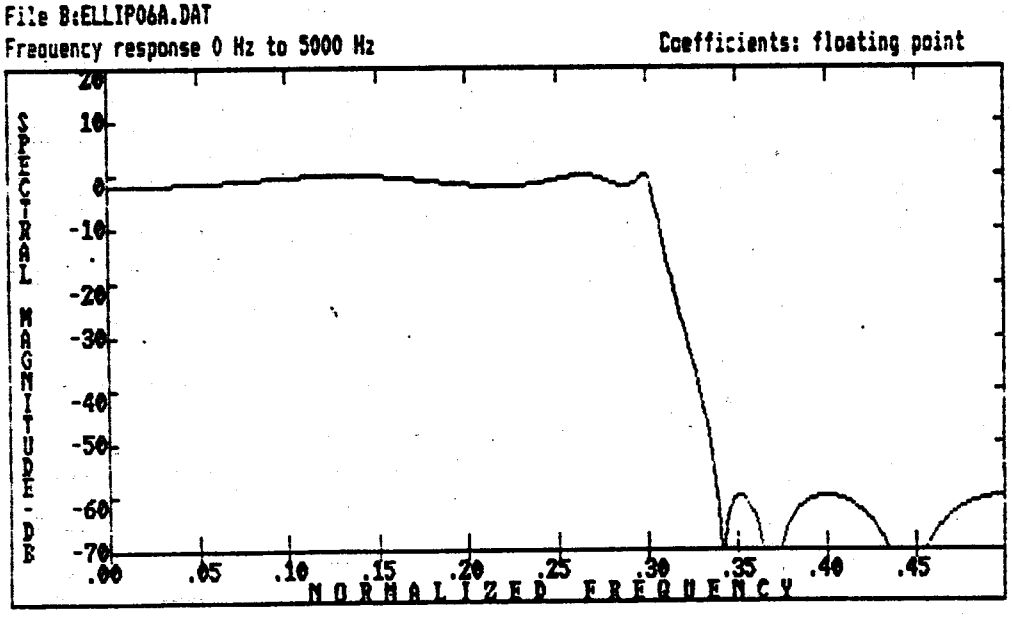
Sampling frequency (Hz) 10000
End of passband (Hz) 3023.9
Beginning of stopband (Hz) 3385.525
Maximum passband attenuation (dB) 2
Minimum stopband attenuation (dB) 60
Coefficient wordlength: floating point

COEFFICIENTS OF SECOND-ORDER SECTIONS

$$A\,\frac{Z^2 + D\,Z + E}{Z^2 + B\,Z + C}$$

A	D	E	B	C
0.6577277	1.0870550	1.0000000	0.6112135	0.9449614
0.5875868	1.3627340	1.0000000	0.2141723	0.7617255
0.1625832	1.8798521	1.0000000	-0.7485645	0.3793633

Fig 8



Fachhochschule Aalen
Fachbereich Elektronik

4. *Entwurf testbarer Schaltkreise / Einführung in das Programm QUICKFAULT*

Verfasser: F. Guffler, U. Epple, H. Graf

Betreuer: Prof. Dr. B. Kohlhammer

Präsentation: MPC-Workshop am 03.02.92 an der Fachhochschule Karlsruhe

Inhaltsverzeichnis

1 Testbarkeit von ASICs

1.1 Allgemeine Betrachtung

Zu den herkömmlichen Randbedingungen beim Entwurf von integrierten Schaltkreisen (Fläche, Taktfrequenz, Verlustleistung) kommt, durch die zunehmenden Integrationsdichten, immer stärker die Forderung nach einer möglichst vollständigen Testbarkeit der Schaltkreise dazu.
In den folgenden Abschnitten wird gezeigt, wie man die Testbarkeit der Schaltung ermittelt, und mit welchen Maßnahmen beim Schaltungsentwurf ("Design for Testability") die Testbarkeit verbessert werden kann.
Besteht ein System aus n Bauteilen, die alle mit der Wahrscheinlichkeit K korrekt funktionieren, so ergibt sich daraus die Wahrscheinlichkeit P, mit der das Gesamtsystem einwandfrei arbeitet.

$$P = K^n$$

1.2 Testarten

Das ASIC muß mehrere Tests bestehen. Die folgende Liste zeigt die wichtigsten Testarten auf.

Test auf Waferebene
Parametrische Tests
Funktions Tests

Bei einem **Test auf der Waferebene** stellt eine Nadelkarte (Probecard) die Verbindung der integrierten Schaltkreise auf dem Wafer mit der Testumgebung her. Diese Nadelkarte unterscheidet sich bei den verschiedenen Testsystemen durch die Anordnung und Anzahl der Kontaktnadeln. Nachdem die Nadeln der Nadelkarte exakt auf die Schaltkreisanschlüsse justiert sind, erfolgt eine Überprüfung der Kontaktierung aller Chips auf dem Wafer. Danach werden verschiedene Testmessungen nacheinander durchgeführt. Der Test beginnt immer an dem Startchip, der auf der rechten unteren Seite des Wafers liegt. Durch die Benutzung einer beheizbaren Waferauflage (Thermochuck) können die Messungen bei verschiedenen Wafertemperaturen durchgeführt werden. Defekte Chips werden direkt auf einem Tintenpunkt markiert (inken). Dies geschieht durch das Absenken der Spitze einer Tintenpatrone auf den Wafer.

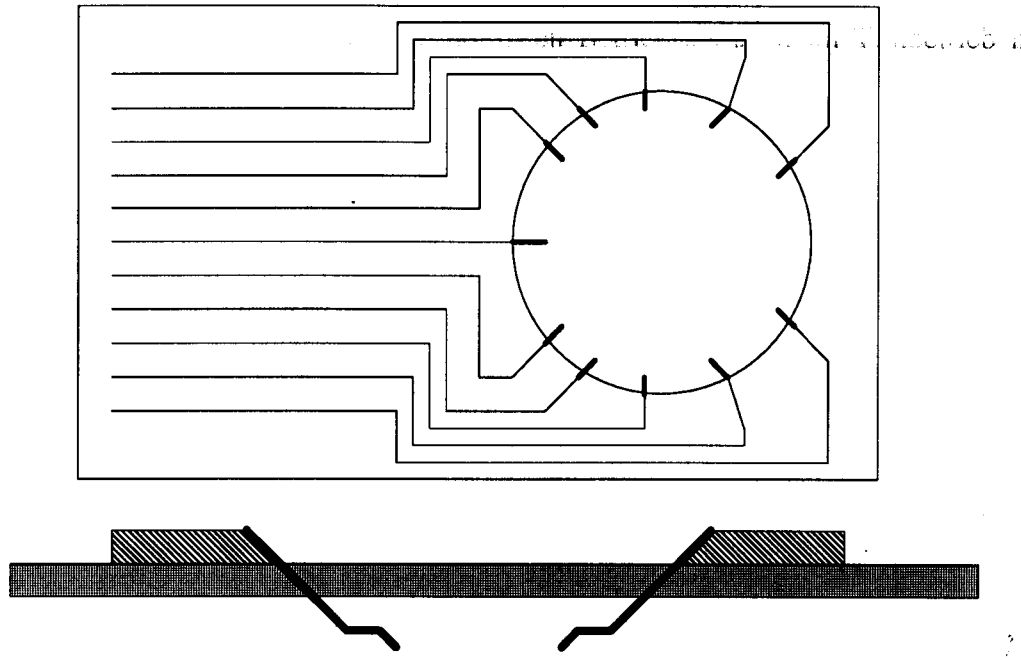


Abb. 1.1: Nadelkarte

Abbildung 1.1 zeigt eine Nadelkarte von oben und eine Ausschnittvergrößerung von der Seite. Die Abbildung 1.2 zeigt die Position des Startchips an.

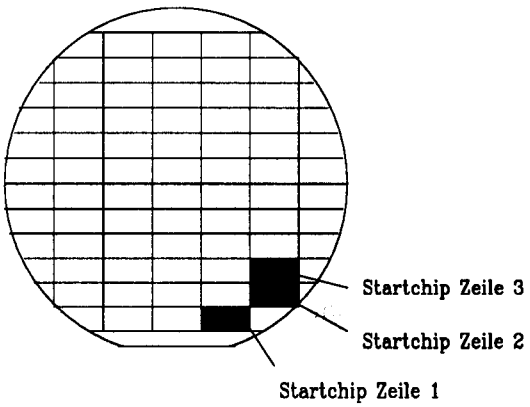


Abb. 1.2: Startchip eines Wafers beim Test

Beim **parametrischen Test** prüft ein Testautomat am fertigen ASIC im Gehäuse die Spannungspegel an den Ausgängen, die Ströme an den Eingängen und den

Gesamtstromverbrauch. So können grobe Fehler wie Kurzschlüsse und abgerissene Bond-Drähte erkannt werden.
Nachdem das ASIC durch einen Reset- oder Testeingang auf einen definierten Anfangszustand gebracht wurde, untersucht ein Testautomat Pegel und Ströme auf ihre spezifizierten Werte. Mit Hilfe parametrischer Tests lassen sich mehr als 95% aller fehlerhaften Chips herausfinden.

Beim **Funktionstest** wird der vollständig gefertigte Schaltkreis am Ausgang auf Signalvariationen, die an den Eingängen angelegt werden, geprüft. Ein aussagekräftiger Funktionstest sollte möglichst alle Gatter auf ihre ordnungsgemäße Funktion überprüfen. Eine Mindestanforderung dafür ist, daß jedes Gatter wenigstens einen Wechsel von logisch 0 auf logisch 1 oder umgekehrt durchläuft. Man spricht dabei von der Kontrollierbarkeit ("Controllability") der Schaltung. Zeigt ein Signal ein fehlerhaftes Verhalten und ist dies an den Ausgangsanschlüssen erkennbar, spricht man von der Beobachtbarkeit ("Observability") von Signalen [4].

1.3 Fehlermodelle

Im funktionellen Betrieb einer digitalen Schaltung kann man grundsätzlich drei Arten von Fehlern unterscheiden.

Statische Fehler
Dynamische Fehler
Intermittierende Fehler

Die **statischen Fehler** lassen das ASIC im normalen Betrieb als auch im Testbetrieb in der gleichen Weise defekt erscheinen. Diese Fehler sind vor allem Isolationsfehler. Sie entstehen durch Fehler im Dielektrikum. Dies sind:

Stuck-At-Open- (Leiterbahnunterbrechungen)
Stuck-At-Short- (Kurzschlüsse)
Stuck-At-High-
Stuck-At-Low-Fehler

Diese statischen Fehler sind ständig vorhanden und gleichbleibend. Das Potential an den Schaltungsknoten bzw. Stecker-Pins nimmt ständig den Pegel der logischen 0 oder den Pegel der logischen 1 ein. Abbildung 1.3 zeigt einen Stuck-At-Fehler an einem NAND-Gatter

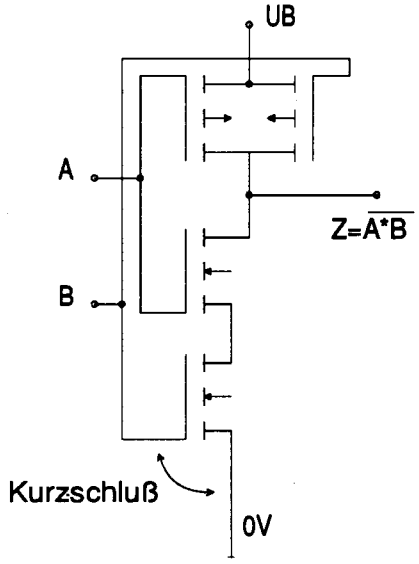


Abb. 1.3: Stuck-At-Fehler an einem NAND-Gatter

Dynamische Fehler werden vorwiegend durch Komponenten bzw. durch Schaltungsanordnungen, welche an den Grenzen ihrer Spezifikation liegen, hervorgerufen. Man unterscheidet hierbei:

Versorgungsfehler
Timing-Fehler
Parametrische Fehler.

Versorgungsfehler werden dadurch verursacht, daß z.B. der Pegel der Versorgungsspannungen während des Normalbetriebs nicht innerhalb der spezifizierten Grenzwerte liegt. Timing-Fehler werden dadurch verursacht, daß zeitliche Zusammenhänge unabgestimmt ablaufen (z.B. ungenügende Flankensteilheit, Toleranz der Bauelemente). Generell treten parametrische Fehler auf, wenn ein oder mehr Parameter einer Komponente oder Schaltung bei Normalbetrieb außerhalb ihrer spezifizierten Grenzwerte liegen.

Intermittierende Fehler sind gelegentlich auftretende Fehler. Diese Gattung von Fehlern ist sehr schwierig zu erkennen und zu lokalisieren.

1.4 Fehlerabdeckung und Defektrate

Unabhängig vom zugrundegelegten Fehlermodell charakterisiert man die Güte eines Funktionstests durch dessen Fehlerabdeckung ("Fault Coverage"). Die Fehlerabdeckung gibt an, welcher Anteil der innerhalb eines Fehlermodells möglichen Einzelfehler vom Test entdeckt wird. Ein vollständiger Test hat demnach eine Fehlerabdeckung von 100% [4].
Die Wahrscheinlichkeit, daß ein IC trotz bestandener Tests defekt ist, ist

$$D = 1 - Y^{(1-F)} \qquad (3.4.1)$$

Y = korrekt arbeitende ICs
F = Fehlerabdeckung

Man kann die Gleichung 3.4.1 herleiten, indem man die Wahrscheinlichkeit dafür berechnet, daß der vom Funktionstest nicht abgedeckte Teil des Chips nicht funktioniert, während im getesteten Teil kein Fehler festgestellt wird [4]..
Das Auflösen der Gleichung 3.4.1 nach der Fehlerabdeckung F ergibt:

$$F = 1 - \log (1-D)/\log Y \qquad (3.4.2)$$

Abbildung 1.4 zeigt die Defektrate D in Abhängigkeit von der Fehlerabdeckung F und der Ausbeute.

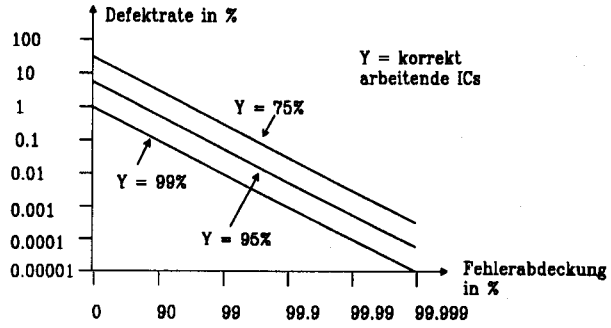


Abb. 1.4: Defektrate D in Abhängigkeit von Fehlerabdeckung F und Ausbeute

Durch eine geeignete Wahl von Testvektoren sollte eine Fehlerabdeckung von mindestens 90% erreicht werden. Eine Aussage über die Fehlerabdeckung der verwendeten Testvektoren kann mit dem Programm QUICKFAULT gemacht werden.
Bessere Werte sind oft aus kostentechnischen Gründen nicht realisierbar, da sich die Testzeit mit steigender Anzahl von Testvektoren vergrößert. Eine kostengünstige Fehlerabdeckung ist nur mit einem testfreundlichen Schaltungsentwurf zu erlangen.

1.4.1 Entwurf testbarer Schaltungen (Design for testability)

1.4.2 Allgemeine Betrachtung

Bei MSI-Schaltkreisen mit bis zu 100 Gattern kann ein 100% Test noch durchgeführt werden. Bei der Anwendung von VLSI-Schaltkreisen würde die Anzahl der Testvektoren, welche für einen allumfassenden Test erforderlich sind, derart eskalieren, daß der benötigte Zeitaufwand nicht mehr gerechtfertigt wäre. Selbst wenn es möglich wäre, diese Testzeiten zu investieren, wären die Testkosten unerträglich hoch. Abbildung 1.5 zeigt den Vergleich zwischen der Gatteranzahl und der relativen Testkosten bei einem testfreundlichen und einem testunfreundlichen Entwurf.

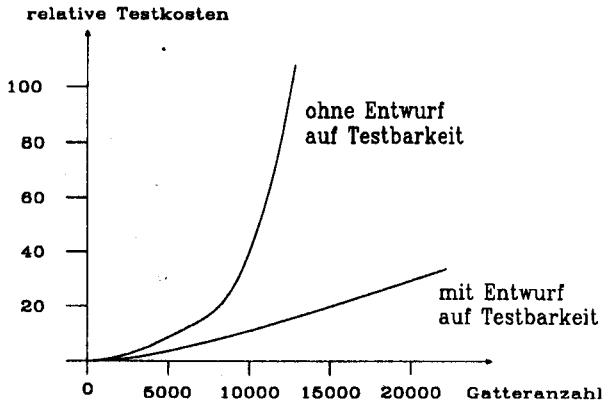


Abb. 1.5: Vergleich zwischen Gatteranzahl und relativen Kosten.

Nimmt man die relativ einfache sequentielle Schaltung nach Abbildung 1.6 mit X Eingangsfunktionen sowie Y Rückkopplungen unter die Lupe, so ergibt sich für die Mindestzahl der Testvektoren Q nach Moore und McCluskey:

$$Q = 2^{(X+Y)}$$

Werden beispielsweise X = 20 Eingangsfunktionen und Y = 40 Speicherelemente verwendet, so ergibt sich die Summe der erforderlichen Testvektoren zu ca. $1{,}15 \cdot 10^{18}$

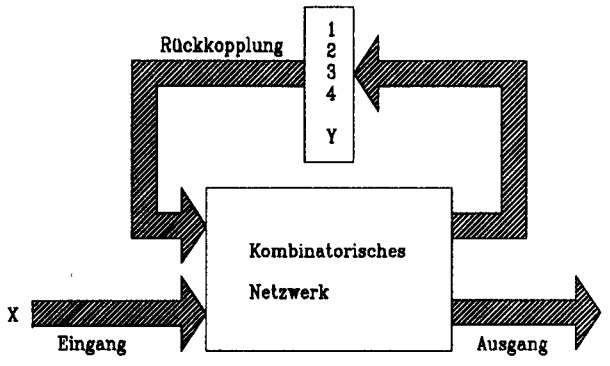


Abb. 1.6: Schematische Darstellung eines Schaltwerks

1.4.3 Die Schaltungsaufteilung

Aufteilen oder Partitionieren einer Schaltung bedeutet, daß die Gesamtschaltung in Teilschaltungen zerlegt wird. Dadurch wird der Aufwand für die Generierung der Testmuster und für die Durchführung des Tests verringert [8]. Die beiden Vorgänge beziehen sich dann auf Schaltungsteile reduzierter Komplexität.
Eine Aufteilung des Gesamtschaltwerks erfolgt meistens nach der Funktion der Teilschaltwerke. Ein weiterer Vorteil der Schaltungsaufteilung ist, daß interne Knoten zu primären Ein- bzw. Ausgängen werden. Abbildung 1.7 zeigt den Schaltungsaufbau des Treppenhausautomaten, der in verschiedene Teilschaltwerke partitioniert wurde.

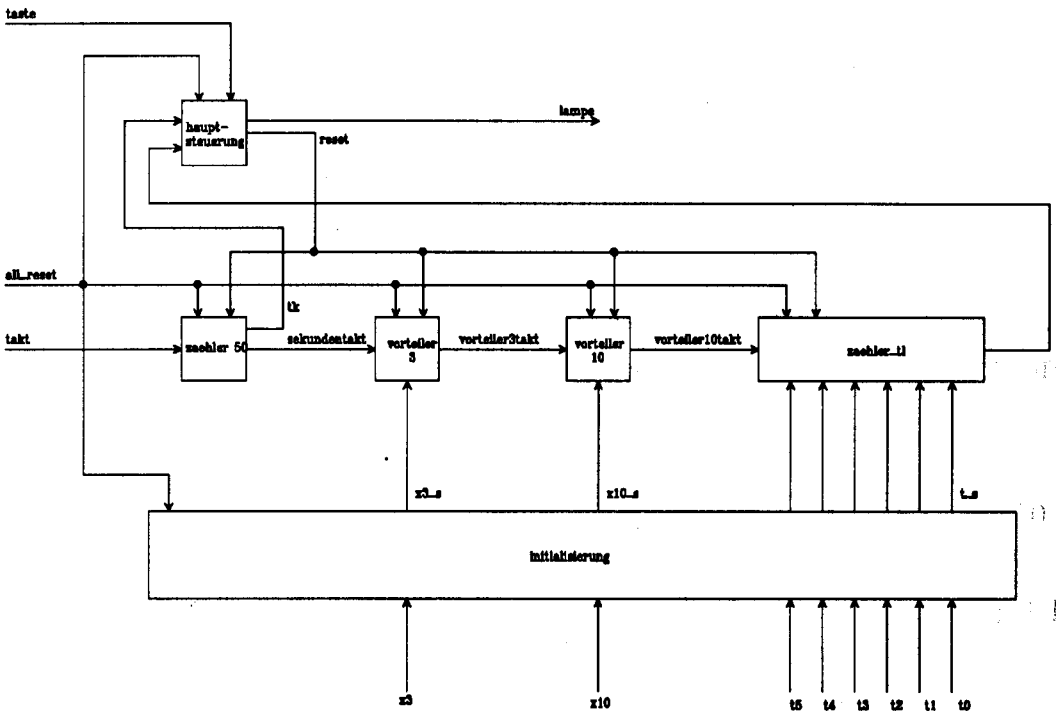


Abb. 1.7: Blockschaltbild des Treppenhausautomaten

1.4.4 Testpunkte

Durch die Einführung von Testpunkten können praktisch alle Knoten einer Schaltung erfaßt werden. Man unterscheidet:

passive Testpunkte
aktive Testpunkte

Passive Testpunkte geben während des Prüfablaufs den momentanen Zustand bestimmter wichtiger schaltungsinterner Knoten an. Sie werden ausschließlich zur Beobachtung interner Vorgänge verwendet und nehmen dadurch keinen Einfluß auf das Verhalten der Schaltung.
Die Verzweigung eines Signals gleichzeitig auf mehrere funktionelle Schaltungsblöcke (Fan-Out Knoten), sowie die Zusammenfassung mehrerer Signale gleichzeitig zu einem resultierenden Ausgangssignal (Fan-In Knoten), sind von besonderer Bedeutung.
Abbildung 1.8 zeigt passive Testpunkte an einem Fan-In Knoten.
Abbildung 1.9 zeigt einen passiven Testpunkt an einem Fan-Out Knoten

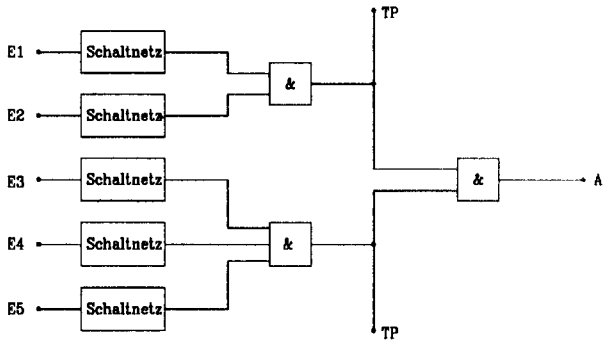


Abb. 1.8: Passive Testpunkte an einem Fan-In Knoten

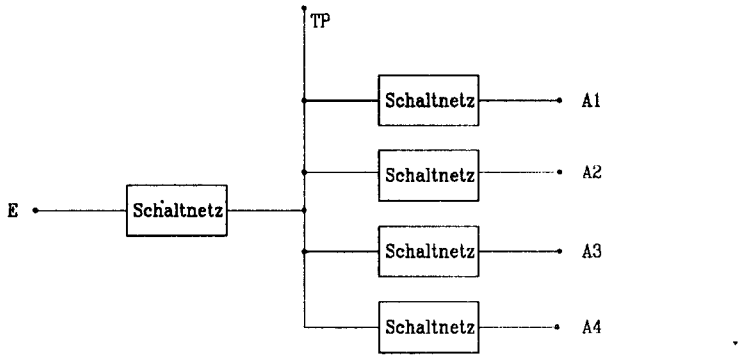


Abb. 1.9: Passiver Testpunkt an einem Fan-Out Knoten

Mit **aktiven Testpunkten** können schaltungsinterne Vorgänge während des Prüfablaufs von außen gesteuert und beeinflußt werden. Aktive Testpunkte werden vorwiegend zur

Zuführung von Testsignalen,
Herstellung eines definierten Anfangszustandes,
Unterbrechung von Rückkopplungsschleifen,
Unterbrechung interner Taktgeneratoren,
Isolation von Schaltnetzen und
Entkopplung redundanter Logik

angewandt. Abbildung 1.10 zeigt einen aktiven Testpunkt zur Auftrennung der Rückkopplung in digitalen Schaltungen.

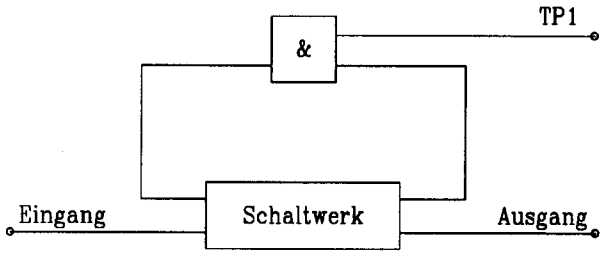


Abb. 1.10: Einsatz eines UND-Gatters zur Auftrennung der Rückkopplungsschleife

Wird TP1 auf logisch 1 gesetzt, so ist die Rückkopplung wirksam. Bei logich 0 ist Sie aufgetrennt.
Zum Test einer redundanten Logik müssen die Zweige auftrennbar sein. Abbildung 1.11 zeigt eine Entkopplung einer redundanten Logik mit einem aktiven Testpunkt.

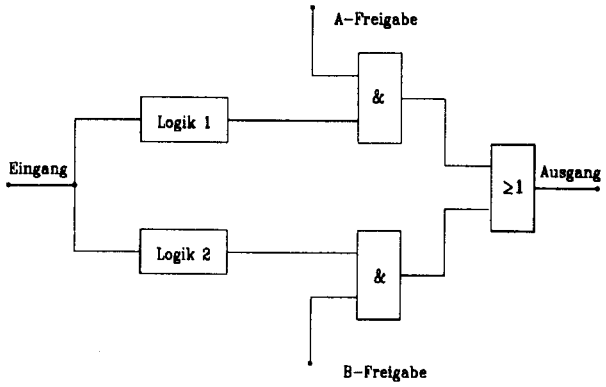


Abb. 1.11: Entkopplung einer redundanten Logik

Die Erhöhung der Beobachtbarkeit und Kontrollierbarkeit durch die Verwendung unbenutzter Pins als Testeingänge bzw. -ausgänge ist ein Vorteil von passiven und aktiven Testpunkten.
Ein Nachteil dagegen ist, daß sich in der Gesamtschaltung die Verzögerungszeiten der zusätzlichen Logik bemerkbar machen können.

1.4.4.1 Strukturierte Verfahren

1.4.4.2 Scan-Path Methode

Ein in der Praxis häufig eingesetztes Verfahren zur Verbesserung der Testbarkeit sequentieller Schaltungen, das auch für Partitionierungszwecke verwendet werden kann, ist die Scan-Path Methode.
Bei diesem Verfahren wird eine Methode angewandt, die praktisch unabhängig von der Schaltungsgröße mit sehr wenigen, typischerweise ein bis vier zusätzlichen externen Verbindungen auskommt [9]. Es wird eine Schaltung entworfen, die in zwei Betriebsarten arbeiten kann. Im Normalmodus führt sie ihre Sollfunktion aus, im Testmodus werden alle Doppelspeicherflipflops der Schaltung als Schieberegister betrieben.
Das Schaltwerk in Abbildung 1.12 kann nach Abbildung 1.13 mit der Scan-Path Methode modifiziert werden.

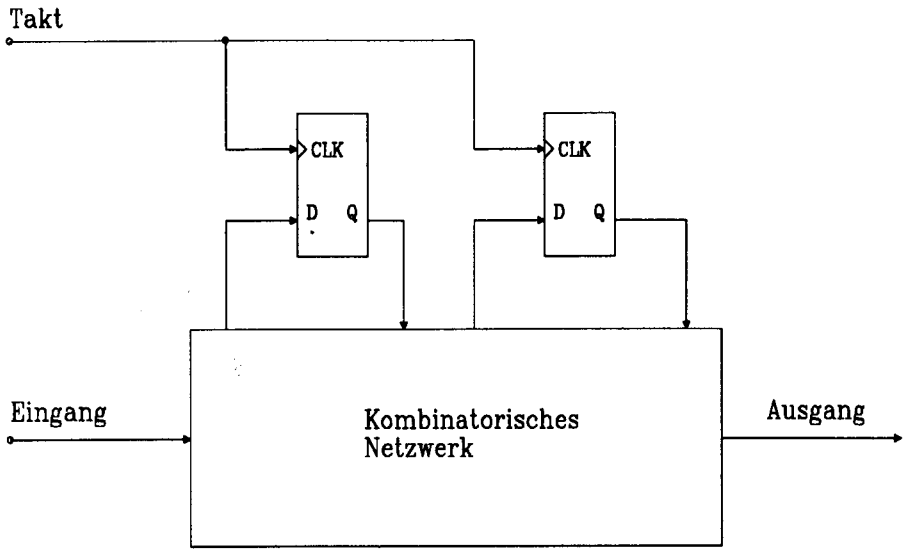


Abb. 1.12: Aufbau eines Schaltwerks

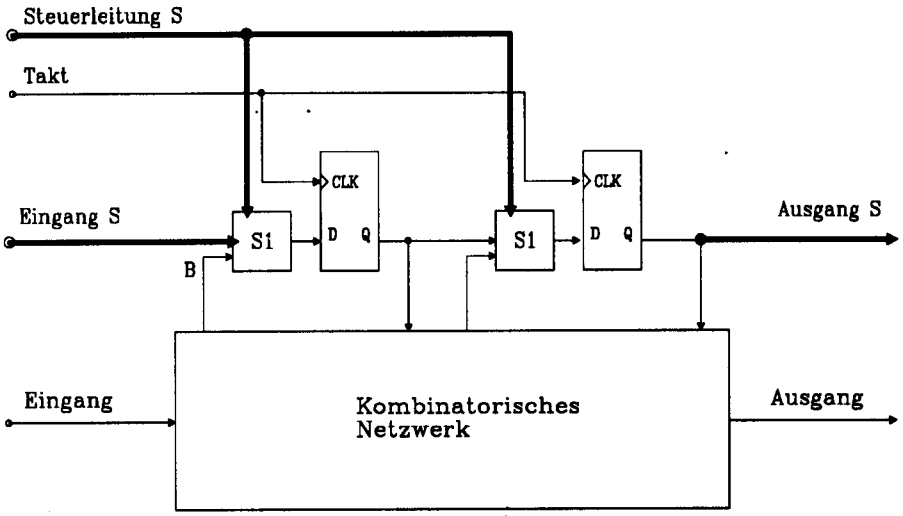


Abb. 1.13: Aufbau eines Schaltwerks mit Scan-Path Methode

Abbildung 1.14 zeigt den Aufbau des Teilschaltwerks S1.

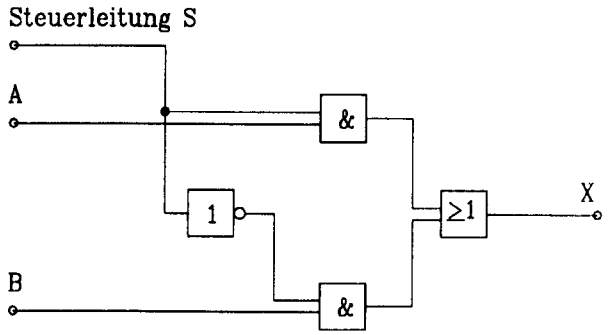


Abb. 1.14: Aufbau des Teilschaltwerks S1

Ist die Steuerleitung S auf logisch 1 gesetzt, so befindet sich die Schaltung im Testbetrieb. In dieser Betriebsart bilden die Doppelspeicherflipflops ein Schieberegister. Die Information des letzten Doppelspeicherflipflops soll am Ausgang zur Verfügung stehen. Während des Testbetriebs kann die gesamte Schaltung in jeden beliebigen Initialzustand versetzt werden, indem die Zustandsvariablen am Eingang A verändert werden. Die Funktionsfähigkeit der Speicherelemente ist sichergestellt, wenn die Zustandsvariablen am Ausgang mit den Zustandsvariablen am Eingang übereinstimmen.
Ist die Steuerleitung S auf logisch 0, so befindet sich die Schaltung im normalen Betriebszustand.

1.4.4.3 LSSD Methode

LSSD (Level-Sensitive-Scan-Design) ist eine Scan-Path-Technik unter Einsatz von Auffangflipflops (Latch) anstelle von flankengesteuerten Doppelspeicherflipflops. Während des Normalbetriebs wird ein System mit zwei nichtüberlappenden Taktsignalen CK1 und CK2 getaktet. Solange CK1 auf logisch 1 liegt, sind die Auffangflipflops L_{1X} transparent und die Auffangflipflops L_{2X} halten den vorherigen Logikpegel ihrer Dateneingänge.
Beim Testbetrieb werden die Daten seriell am Eingang SDI eingelesen. Durch abwechselndes Takten des Test-Taktes TCK und des Systemtaktes CK2 werden diese Daten am Ausgang SDO seriell ausgegeben. Abbildung 1.15 zeigt die allgemeine Struktur einer Schaltung mit LSSD-Methode

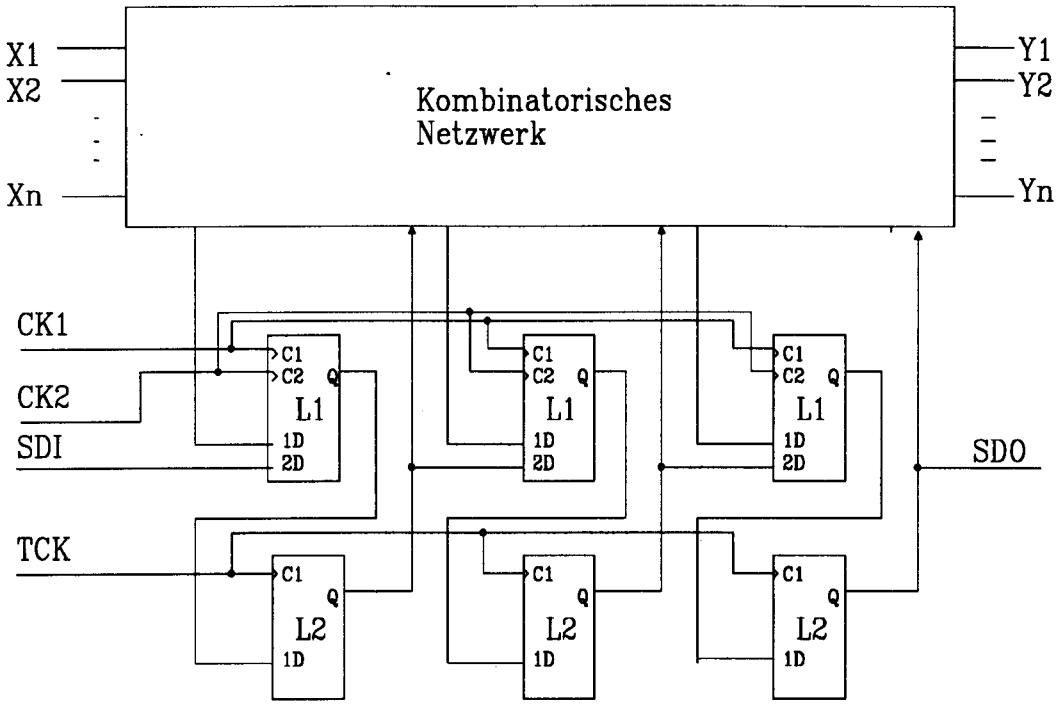


Abb. 1.15: Allgemeine Struktur einer Schaltung mit LSSD-Methode

1.4.4.4 Multiplexer-Scan Methode

Bei der Multiplexer-Scan Methode (siehe Abbildung 1.16) werden die Registerinhalte seriell an dem Ausgangssignal sichtbar gemacht. Dieses Strukturprinzip erhöht die Beobachtbarkeit aber nicht die Kontrollierbarkeit einer Schaltung. Die Multiplexer-Scan Methode wird seltener angewandt.

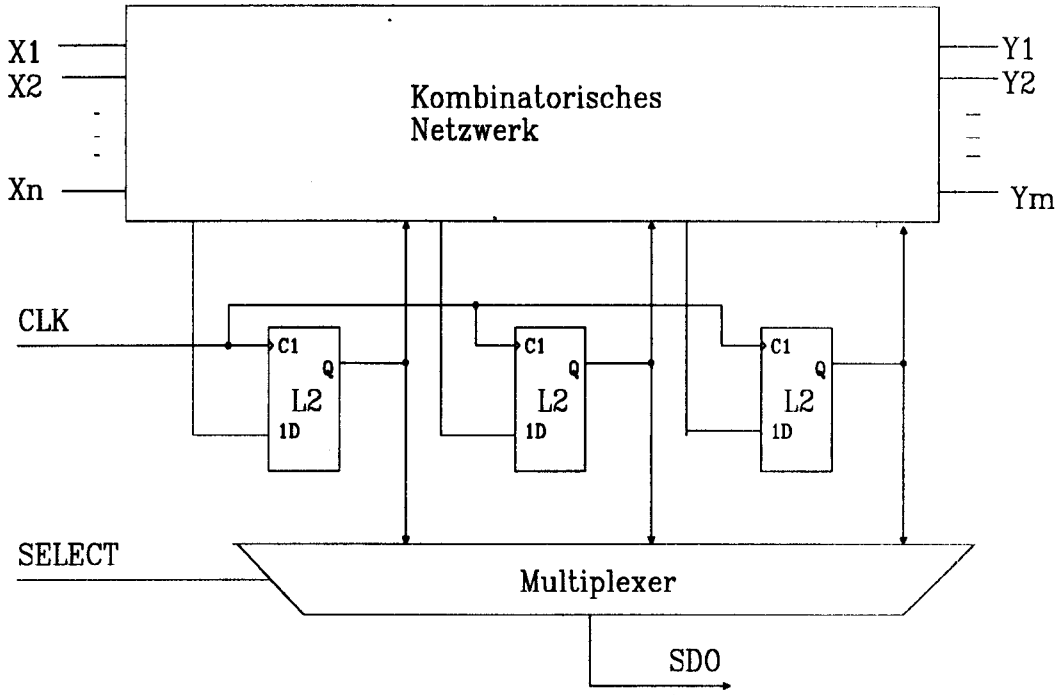


Abb. 1.16: Allgemeine Struktur einer Schaltung mit Multiplexer_Scan Methode

1.4.4.5 Boundary-Scan-Methode

Mit den Boundery-Scan-Techniken wird die Testbarkeit dadurch gesteigert, daß die Anschlußerfordernisse an die Testsysteme gemildert werden. Die Boundary-Scan-Technik wird hier speziell für das JTAG-Testsystem beschrieben, welches im Institut für Mikroelektronik Stuttgart auch verwendet wird. JTAG-Testsysteme (Joint Test Action Group) können integrierte Schaltungen, die dem JTAG Boundary Scan Standard entsprechen mittels eines PC-basierenden Systems testen [11]. JTAG definiert eine Testarchitektur, die aus einem vierdrahtigen Testbus

TDI Test Data In
TDO Test Data Out

TCK Test Clock
Tms Test Mode Select

besteht.
Abbildung 1.17 zeigt die Anwendung dieser Architektur. An den Ein- und Ausgängen des eigentlichen Schaltwerks werden die Boundary-Scan-Zellen (BSZ) angebracht. Bei Normalbetrieb gelangen die Eingangssignale durch die BSZ zu den Ausgängen. Im Boundary-Scan-Test-Modus können beispielsweise Systemdaten an den ASIC-Eingängen oder/und interne Testdaten aufgesammelt und seriell über TDO zur Auswertung ausgegeben werden (siehe Abbildung 1.18). Auf ähnliche Weise können dem Schaltwerk Testvektoren seriell zugeführt werden (siehe Abbildung 1.19). Der Aufbau einer Boundary-Scan-Zelle ist aus Abbildung 1.20 ersichtlich.

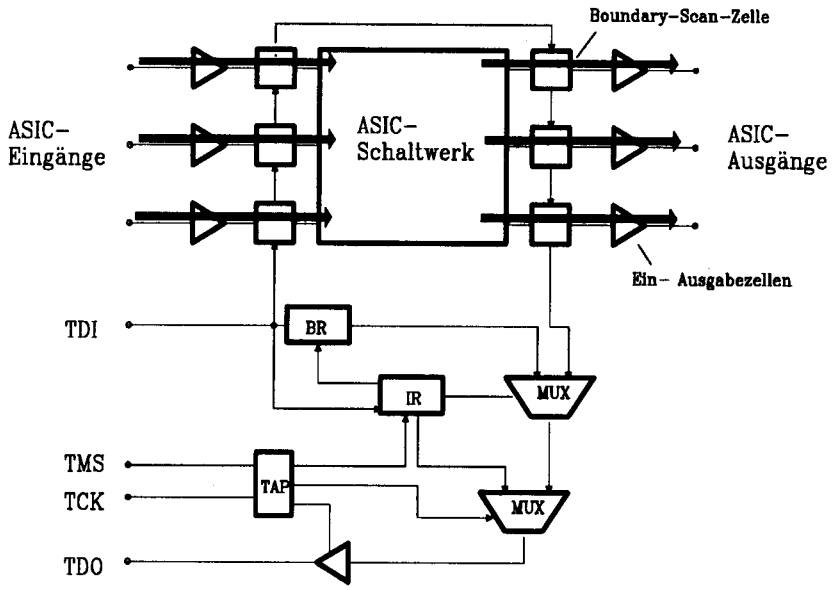


Abb. 1.17: Boundary-Scan-Test Prinzip unter Anwendung des JTAG Testbusses

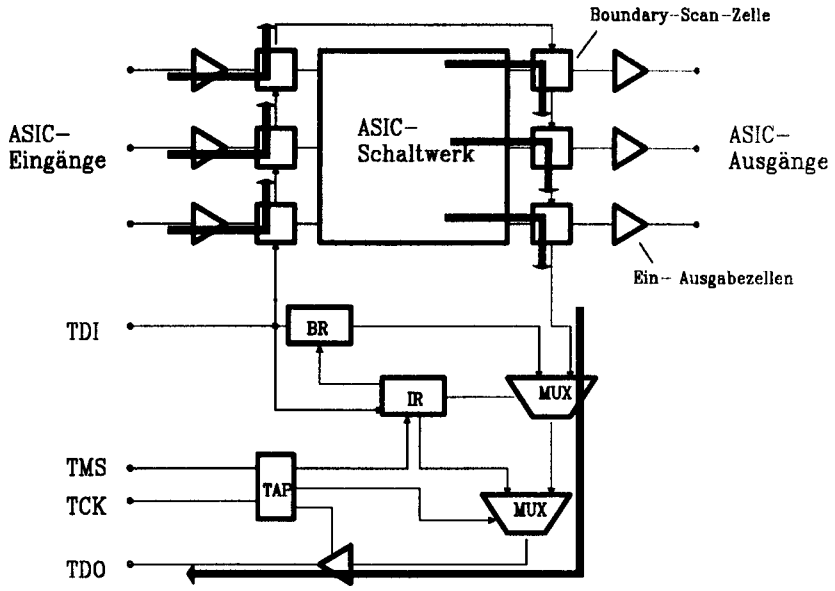


Abb. 1.18: Testdatengewinnung und serielle Ausgabe auf dem Testbus mit der Boundary-Scan-Technik

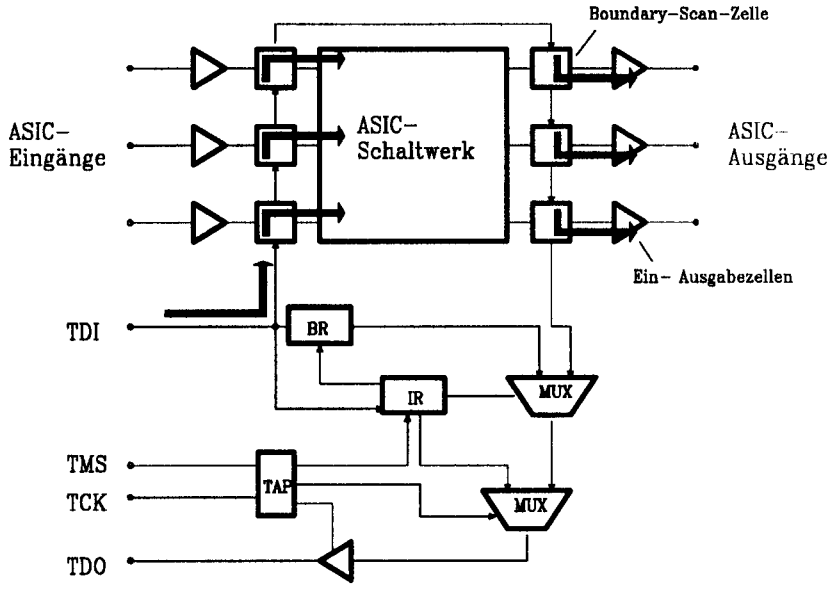


Abb. 1.19: Serielles Zuführen von Testdaten mit der Boundary-Scan-Technik

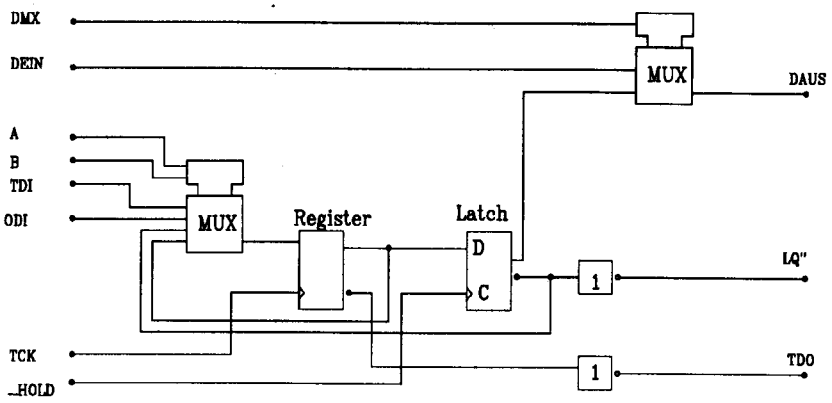


Abb. 1.20: Aufbau einer Boundary-Scan-Zelle

2 Fehlersimulation mit QUICKFAULT

2.1 Einführung in QUICKFAULT

QUICKFAULT ist ein Fehlersimulator, der die Effektivität der verwendeten Testmuster ermittelt. Bei der Fehlersimulation werden an allen Knoten (Gatterein- bzw. Gatterausgänge) Fehler (stuck-at-Fehler) impliziert, die Mittels der Testmuster herausgefunden werden sollen. Die verwendete Stuck-at-Fehlermethode geht davon aus, daß sich die in der Praxis vorkommenden Fehler so auswirken, als würden bestimmte Knotenpunkte innerhalb einer Schaltung auf einem festen Potential gehalten werden (Siehe Kapitel 3.3). Nach einem Simulationsdurchlauf gibt QUICKFAULT eine Aussage über den Fehlerabdeckungsgrad des verwendeten Testmusters an. Diese Fehlerabdeckung zeigt an, wieviele der eingesetzten Fehler prozentuell mit diesem Testmuster gefunden werden können. Ein Fehler kann nur gefunden werden, wenn beim Vergleich der Zustandszeittabellen zwischen der fehlerfreien und der fehlerbehafteten Simulation der eingesetzte Fehler sichtbar wird (Siehe Kapitel 8.2). Angestrebt werden sollte eine Fehlerabdeckung von mindestens 90%.

2.2 Ablauf der Fehlersimulation

Abbildung 2.1 zeigt den Ablauf einer Fehlersimulation auf. Nach dem Einlesen der Testvektoren durchläuft das Programm QUICKFAULT eine fehlerfreie Simulation. Gleichzeitig legt es eine Zustandszeittabelle der fehlerfreien Simulation an. Danach werden Fehler (stuck-at-1- und stuck-at-0-Fehler) an allen Knotenpunkten impliziert. Bei einer zweiten Simulation (Fehlerbehaftete Simulation) wird eine Zustandszeittabelle für die fehlerbehaftete Simulation angelegt. Anschließend wird die Zustandszeittabelle der fehlerfreien Simulation mit der fehlerbehafteten Simulation verglichen. Werden Unterschiede erkannt, so ist der eingesetzte Fehler als erkannt zu bezeichnen.

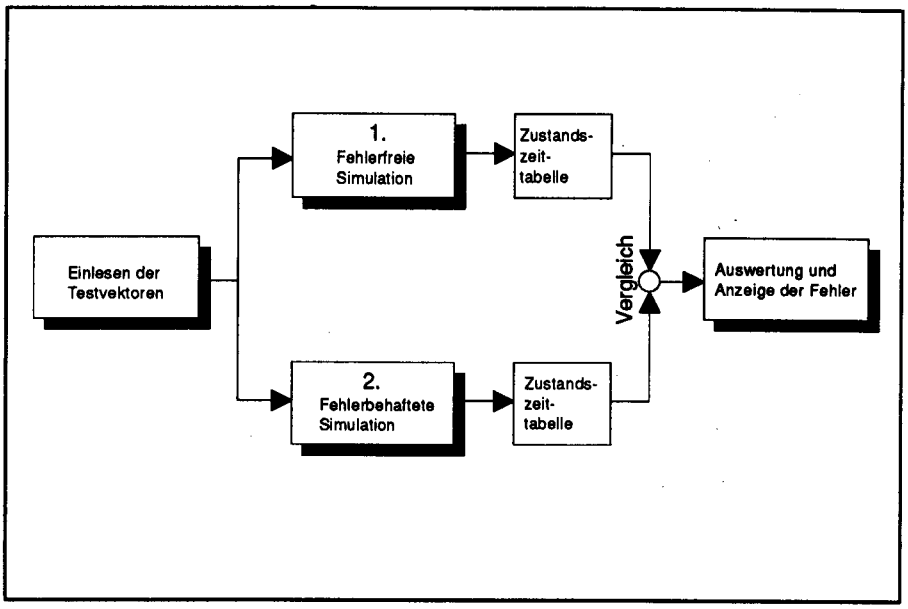


Abb. 2.1: Ablauf einer Fehlersimulation

Es gibt 3 verschiedene Fehlersimulationsarten, die alle nach dem obenbenannten Verfahren arbeiten.

serieller Fehlersimulator
paralleler Fehlersimulator
konkurrierender Fehlersimulator

Beim **seriellen Fehlersimulator** wird nur ein Fehler je Simulationslauf analysiert. Diese Art ist sehr langsam.
Beim **parallelen Fehlersimulator** werden in Abhängigkeit von der Breite des Datenbusses des Rechners eine entsprechend Anzahl von Fehlern gleichzeitig simuliert.
Beim **konkurrierenden Fehlersimulator** wird die Schaltung einige Male kopiert und die Fehler zufällig über die Kopien verteilt. Nach der Simulation werden die Ergebnisse zusammengefaßt und interpretiert. Diese Art der Fehlersimulation ist sehr speicherplatzintensiv.

2.3 Aufruf von QUICKFAULT

Der Start von QUICKFAULT erfolgt aus der Betriebssystemebene (Shell) mit

$ quickfault Name <RETURN>

Als "Name" ist derselbe Name einzugeben, der bei NETED und SYMED benutzt wurde.
Das auf dem Bildschirm (siehe Abbildung 2.2) entstandene Bild weist folgende voreingestellte Fenster und Menüs auf.

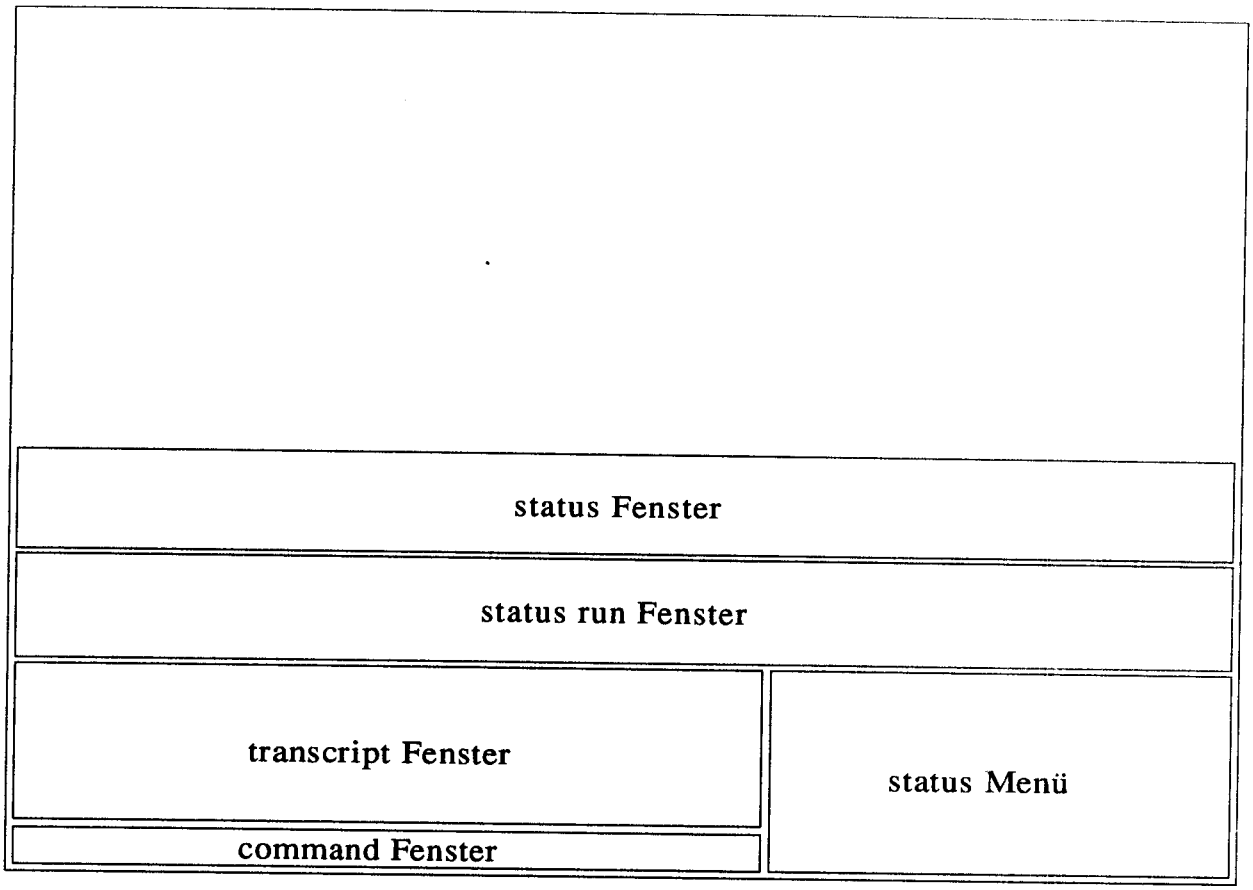
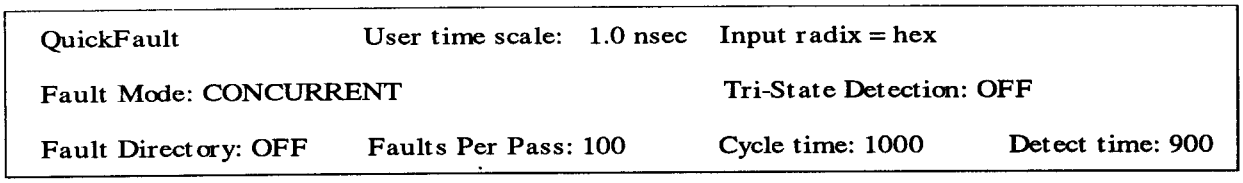


Abb. 2.2: Voreingestellte Fenster und Menüs in QUICKFAULT

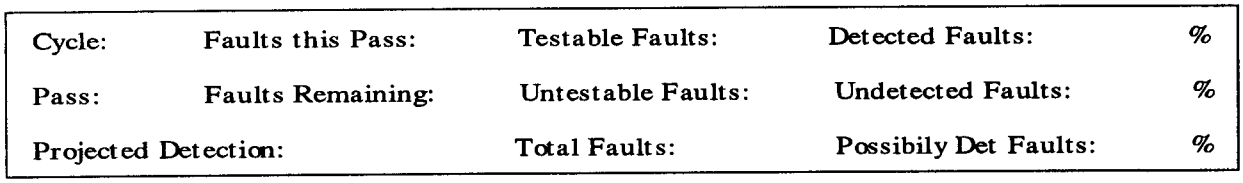
Das **status Fenster** (Statisches Fenster) enthält Befehle (z.B. Zeitskalierung und Zahlenbasis), die durch Befehlseingaben in das **command Fenster** eingestellt werden können.

Das **status run Fenster** (Statisches Ergebnis Fenster) zeigt die Fehlerabdeckung, die Anzahl der Durchgänge, die überprüften und nicht überprüften Fehler an.
Das **transcript Fenster** (Wiedergabefenster) zeigt alle Eingabebefehle und Antworten des Rechners, z.B. Anforderungen zu Tastatureingaben, sowie Fehlermeldungen an.
In das **command Fenster** (Kommandofenster) können Kommandos über die Tastatur eingegeben werden.
Das **status Menü** (Statisches Menü) enthält Funktionen und Befehle (z.B. Hilfefunktion), die mit der linken Maustaste abrufbar sind.

2.4 Die Funktionen im status und status run Fenster



status Fenster



status run Fenster

Abb. 2.3: Inhalt des status und des status run Fensters

Abbildung 2.3 zeigt den Inhalt des status und des status run Fensters an.
User time scale gibt die Zeitskalierung an, auf welche die zeitbezogenen Kommandos basieren. Sie kann durch das Kommando

QuickFault> **SCALe USER Time Zahlenwert <RETURN>**

verändert werden. Dieser Befehl gilt nur im NONE Modus.
Input radix zeigt die Zahlenbasis an. Das Kommando zum Ändern der Zahlenbasis lautet

QuickFault> **Radix Basis** **<Return>**

Als "Basis" kann zwischen **Hex, Octal, Binary** und **Decimal** ausgewählt werden.
Fault Mode legt fest, ob QUICKFAULT als Fehlersimulator oder als Logiksimulator arbeiten soll.
Mit dem Kommando

QuickFault> **FAUlts MOde Zusatz** **<RETURN>**

wird die Simulationsart festgelegt. Als "Zusatz" muß **Concurrent** (Fehlersimulator) oder **NONE** (Logiksimulator) folgen. Dieser Befehl muß vor dem Simulationsstart eingegeben werden. Leider kann dadurch kein Simulationsartwechsel nach einem Simulationsdurchlauf durchgeführt werden.
Tri-State Detection stellt eine Tri-State Fehlererkennung ein.
Fault Dictionary legt fest, ob eine Fehlerdatei angelegt oder nicht angelegt werden soll.
Durch das Kommando

QuickFault> **FAULTS DICTIONARY** **<RETURN>**

wird die Datei angelegt.
Faults Per Pass legt die maximal mögliche Fehleranzahl pro Simulationsdurchgang fest. Die Grundeinstellung der Fehler pro Simulation liegt bei 100. Durch das Kommando

QuickFault> **FAULts Per Pass Zahlenwert** **<RETURN>**

kann die mögliche Fehleranzahl verändert werden. Als "Zahlenwert" muß eine Ganzzahl <= 400 eingegeben werden.
Cycle Time legt die Meßperiode und
Detect Time legt den Zeitpunkt der Meßung fest. Mit dem Kommando

QuickFault> **CYCLE Zahl1 Zahl2** **<RETURN>**

wird die Meßperiode und der Meßzeitpunkt verändert. Bei "Zahl1" muß eine Ganzzahl für die gewünschte Zeit der Meßperiode eingegeben werden. Die Ganzzahl "Zahl2" wird legt den Meßzeitpunkt fest. Diese Zahl muß kleiner sein als die Zahl1.

Im status run Fenster zeigt
Cycle die Anzahl der Periodendurchgänge an.
Pass zeigt die Anzahl der Simulationsdurchgänge für eine komplette Fehlersimulation an. Sie hängt von dem eingestellten Faults Per Pass-Zahlenwert ab.
Faults this Pass zeigt die Anzahl der Fehler an, die in diesem Durchgang erkannt werden können.
Faults Remaining zeigt die Anzahl der Fehler an, die im nächsten Durchgang simuliert werden.
Total Faults zeigt die Anzahl aller im Schaltkreis eingefügten Fehler auf.
Projected Detection zeigt die mit einem bestimmten Testmuster aufgefundene Fehlerabdeckung auf.

Cycle:	22	Faults this Pass:	0	Testable Faults:	536	Detected Faults:	512	95.52 %
Pass:	4	Faults Remaining:	0	Untestable Faults:	2	Undetected Faults:	17	3.17%
Projected Detection:		95%		Total Faults:	538	Possibily Det Faults:	7	1.31 %

status run Fenster

Abb. 2.4: Inhalt des status run Fensters nach einer Fehlersimulation

Abbildung 2.4 zeigt das **status run Fensters** nach einem Simulationsdurchgang. In diesem Fall wurden 4 Simulationsdurchgänge (Pass) benötigt. Die Testmusterdatei besteht aus 22 Periodendurchgängen (Cycle). Es wurden von QUICKFAULT 538 Fehler impliziert (Total Faults), davon sind 536 testbar (Testtable Faults). Ein untestbarer Fehler (Untestable Faults) ist ein auf Betriebsspannung oder Masse gelegter Eingang einer Schaltung. Das Testmuster deckt 512 Fehler auf (Detected Faults), was eine Fehlerabdeckung (Projected Detection) von 95% bedeutet. Mögliche Fehler (Possibily Det Faults) treten bei einem Übergang von einem unbestimmten Wert auf logisch 1 bzw. logisch 0 oder umgekehrt auf.

2.5 Fehlersimulationsumgebung aktivieren

QUICKFAULT besitzt zusätzlich noch das **view Fenster** (Sichtfenster). Dieses zeigt das Schaltbild der zu simulierenden Schaltung.

2.5.1 Erzeugen des view Fenstes

Zum Erzeugen des view Fensters muß das Kommando

Quickfault> **VIEw Sheet <RETURN>**

eingegeben werden. Abbildung 2.8 zeigt den Aufbau des Bildschirmes nach der Kommandoeingabe. In diesem Fenster besteht wie bei QUICKSIM die Möglichkeit einer Auschnittvergrößerung. Durch Drücken der Funktionstaste F8 (VIEW AREA) und Bewegen der Maus kann ein bestimmter Bereich vergrößert werden. Mit der Tastenkombination Shift F8 (VIEW ALL) kann die Ausschnittvergrößerung rückgängig gemacht werden.

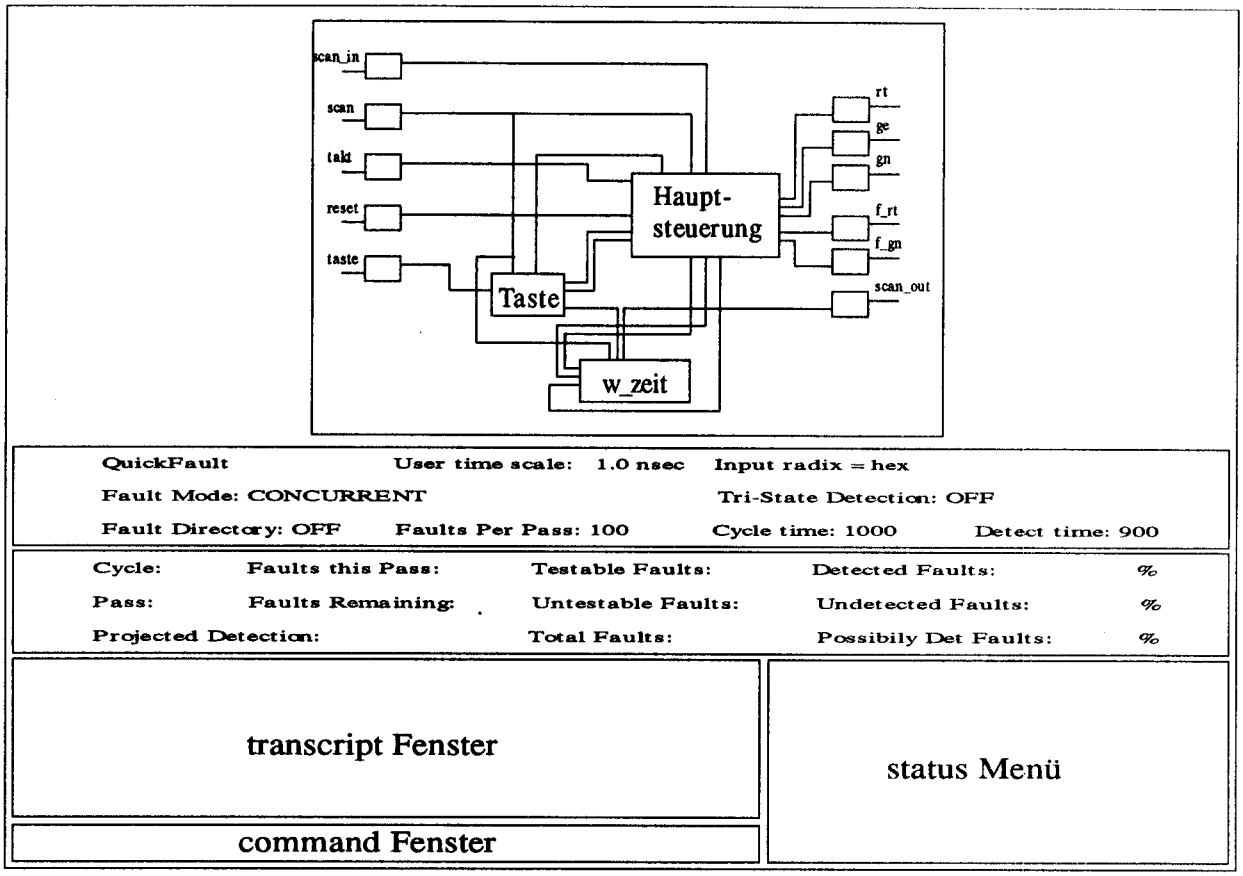


Abb. 2.5: Bildschirmaufbau mit view Fenster

2.6 Einlesen der Testvektoren

Es gibt drei verschiedene Möglichkeiten zum Einlesen der Testvektoren.

Force Kommandos,
MISL Dateien (Mentor Interaktive Stimulus Language) oder
Log Dateien.

Force Kommandos können wie bei QUICKSIM einzeln in das command Fenster oder als Do-Datei verwendet werden. Einzelheiten über Force Kommandos ist in Kapitel QUICKSIM beschrieben.

MISL Dateien besitzen eine größere Kapazität wie Do-Dateien. Nach dem Aufruf des Kommandos

QuickFault> **READ FORCe Misl** **<RETURN>**

wird diese ASCII-Datei compiliert. Das Ergebnis dieses Vorganges ist eine LOG-Datei. Darin sind die Signalnamen und der Zeitpunkt der Vektorzuweisung bzw. -aenderungen explizit dargestellt. Bei diesem Compilierungsvorgang erfolgt ein Abgleich mit der Schaltung. Ist er fehlerfrei abgeschlossen, kann der Simulationslauf beginnen.
Das direkte Einlesen einer **LOG Datei** ist ebenfalls möglich.

2.7 Simulationsstart

Die Simulation kann durch das Kommando

QuickFault> **RUN Zeit** **<RETURN>**

gestartet werden.

2.8 Auswertung des Simulationsergebnisses

2.8.1 Das Anzeigen der nicht überprüften Fehler (view faults undetected)

Durch das Drücken der Funktionstaste <F7> oder durch Eingabe des Kommandos

QuickFault>...**VIEw FAUlts Undetected** **<RETURN>**

erscheinen im view Fenster die mit diesem Testmuster nicht gefundenen Fehler. Stuck-at-1 Fehler werden oberhalb der entsprechenden Ein- bzw. Ausgangspins mit einem grünen Rechteck, stuck-at-0 Fehler werden unterhalb der Pins mit einem roten Rechteck angezeigt. Hierarische Fehler werden auf den Teilschaltwerken angezeigt. Die Anzahl der nicht gefundenen stuck-at-1 und stuck-at-0 Fehler wird bei hierarischen Fehlern in den Rechtecken angezeigt. Abbildung 2.8 zeigt Beispiele der Kennzeichnung nicht gefundener Fehler.

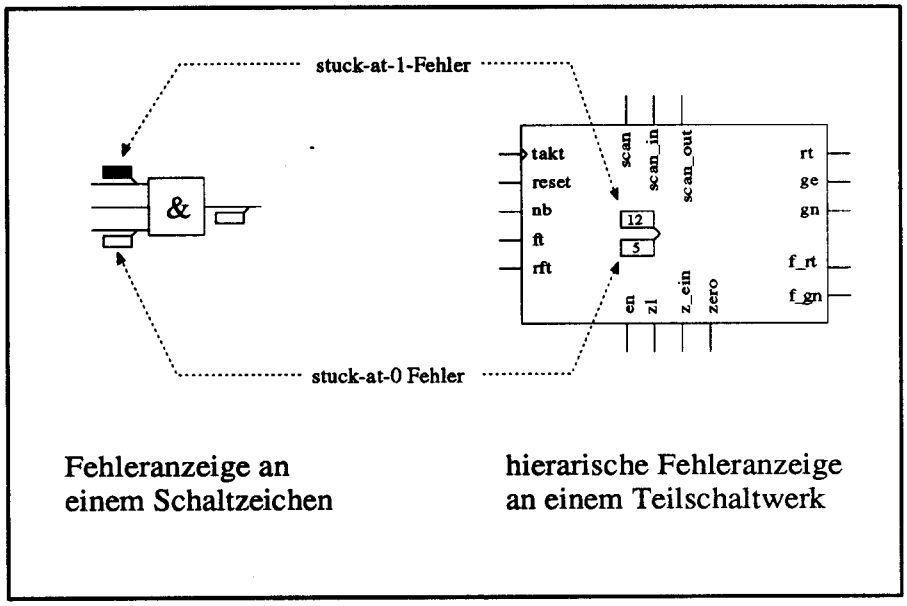


Abb. 2.6: Kennzeichnung nicht überprüfter Fehler

2.8.2 Das Anzeigen der Fehlerblockaden (view faults blockages), der gefundenen Fehler (view faults detected) und der möglichen Fehler (view faults possible)

Durch das Drücken der Taste <SHIFT> und der Funktionstaste <F7> oder durch Eingabe des Kommandos

QuickFault>...**VIEw FAUlts Blockages** **<RETURN>**

erscheinen im view Fenster die Schaltzeichen, die eine Fehlerblockade bilden. Eine Zahl zeigt an, wieviele Fehler bei dem verwendeten Testmuster durch dieses Bauelement nicht gefunden werden konnten.
Durch das Drücken der Funktionstäste <F6> (view faults detected) werden die gefundenen Fehler und durch das Drücken der Taste <SHIFT> und der Funktionstaste <F6> (view faults possible) werden die möglichen Fehler im view Fenster angezeigt

2.8.3 Überprüfung tieferer Ebenen (view down)

Eine Überprüfung hierarchischer Fehler ist durch das Setzen des Cursors auf das Teilschaltwerk und Drücken der Taste <CTRL> und der Funktionstaste <F8> (view down) möglich. Bei diesem Befehl erscheint der Stromlaufplan des Teilschaltwerkes im view Fenster. In allen Ebenen können die gleichen Funktionen wie auf der Hauptebene abgerufen werden (z.B. eine Ausschnittvergrößerung, eine Anzeige der nicht gefundenen Fehler usw.) Siehe Kapitel 8.5.1 view Fenster.
Durch Drücken der Taste <CTRL> und der Funktionstaste <F7> (view up) kann dieser Befehl rückgängig gemacht werden.

2.8.4 Fehlererkennungsdiagramm (detection charts)

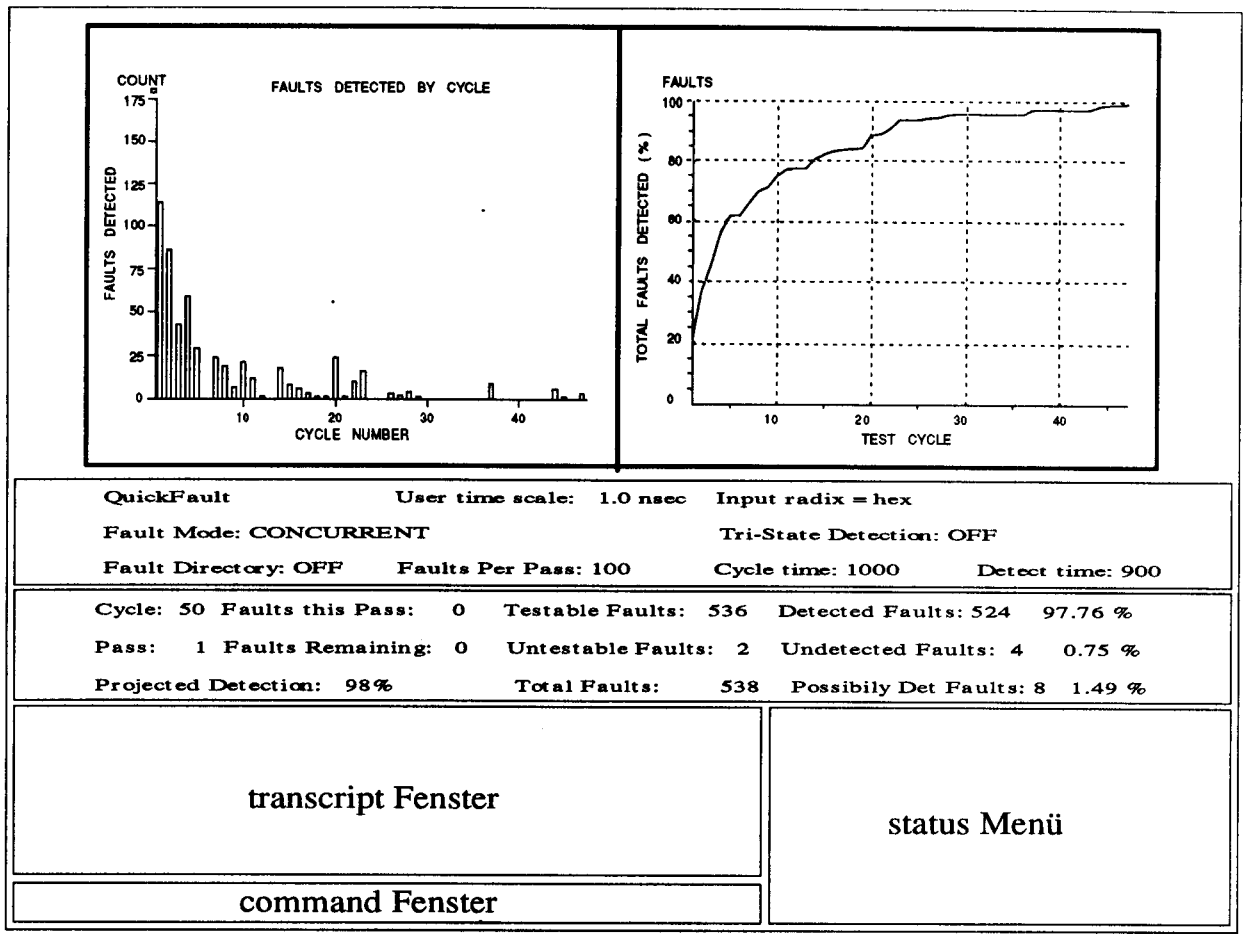


Abb. 2.7: Bildschirmaufbau mit Fehlererkennungsdiagramm

Durch das Drücken der Funktionstaste <F5> (detection charts) baut sich das in Abbildung 2.7 dargestellte Bildschirmbild auf.
Die linke Grafik in Abbildung 2.8 gibt die Anzahl der gefundenen Fehler je Meßperiode an. Aus diesem Diagramm ist ersichtlich, daß in einigen Meßperioden keine Fehler entdeckt wurden. Durch das Entfernen dieser Zyklen kann ein kürzeres Testmuster mit gleicher Fehlerabdeckung entwickelt werden. Die rechte Grafik in Abbildung 2.8 zeigt die prozentuale Fehlerabdeckung der Meßperioden auf.

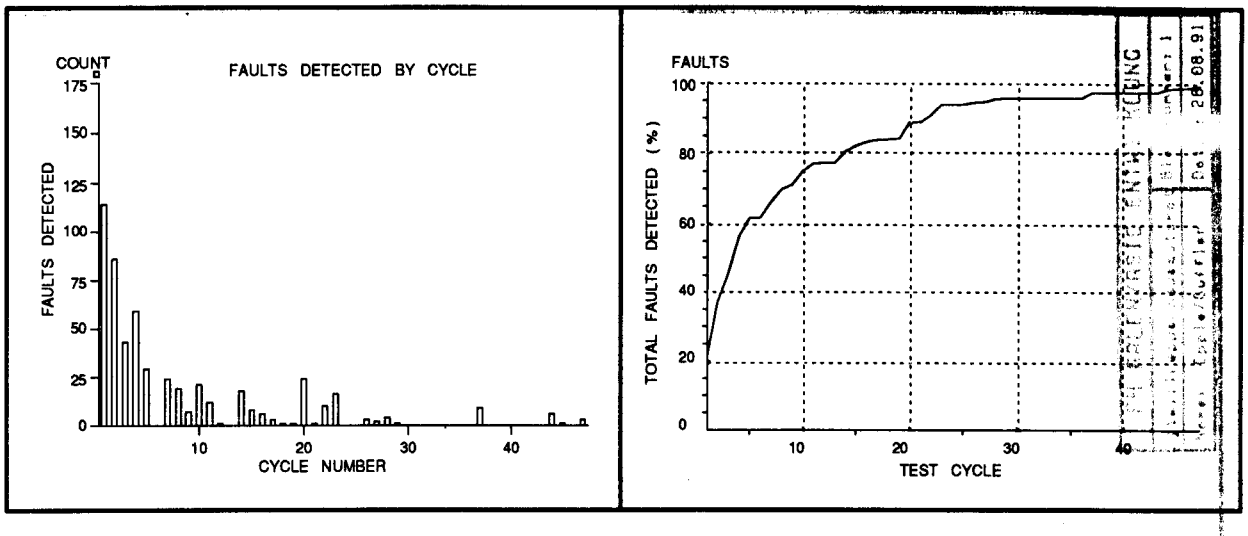


Abb. 2.8: Fehlererkennungsdiagramm

Durch Drücken der Taste <POP> kann zwischen dem Schaltbild und dem Fehlererkennungsdiagramm umgeschaltet werden. QUICKFAULT besitzt 16 Diagrammfenster. Da bei jedem Aufruf zwei Diagrammfenster belegt werden, müssen nach dem achten Aufruf die Fenster geschlossen werden. Durch die Eingabe des Kommandos

QuickFault> **FORGet CHART** **<RETURN>**

wird das letzte Diagrammfenster und durch die Eingabe des Kommandos

QuickFault> **FORGet CHART -All** **<RETURN>**

werden alle Diagrammfenster gelöscht.

2.8.5 zusätzliche Fehlerausgänge hinzufügen (primary outputs)

Da die Fehlererkennung an offenen Ausgängen eines Bauelements nicht kontrolliert werden können, besteht die Möglichkeit, zusätzliche Fehlerausgänge hinzuzufügen. Durch die Eingabe des Kommandos

QuickFault> **PRIMary Outputs Netzname** **<RETURN>**

oder durch Setzen des Cursors an die gewünschte Stelle und Drücken der Tasten <SHIFT> und <F3> werden die Ausgänge gesetzt. Als Netzname muß der Name des Netzes z.B. N$121 eingegeben werden.

2.9 Beenden und Unterbrechen der Simulation

Es ist jederzeit möglich das Simulationsprogramm zu unterbrechen, um auf die Shell-Ebene zurückzukehren. Durch Drücken der Taste **CTRL-E** oder durch Eingabe des Kommandos

Quickfault> **EXIT** **<RETURN>**

wird diese Unterbrechung ausgelöst.
Durch das Kommando

Quickfault> **<RETURN>**

wird dieser Befehl rückgängig gemacht.
Das Kommando **CTRL-N** oder die Eingabe von

Quickfault> **QUIT** **<RETURN>**

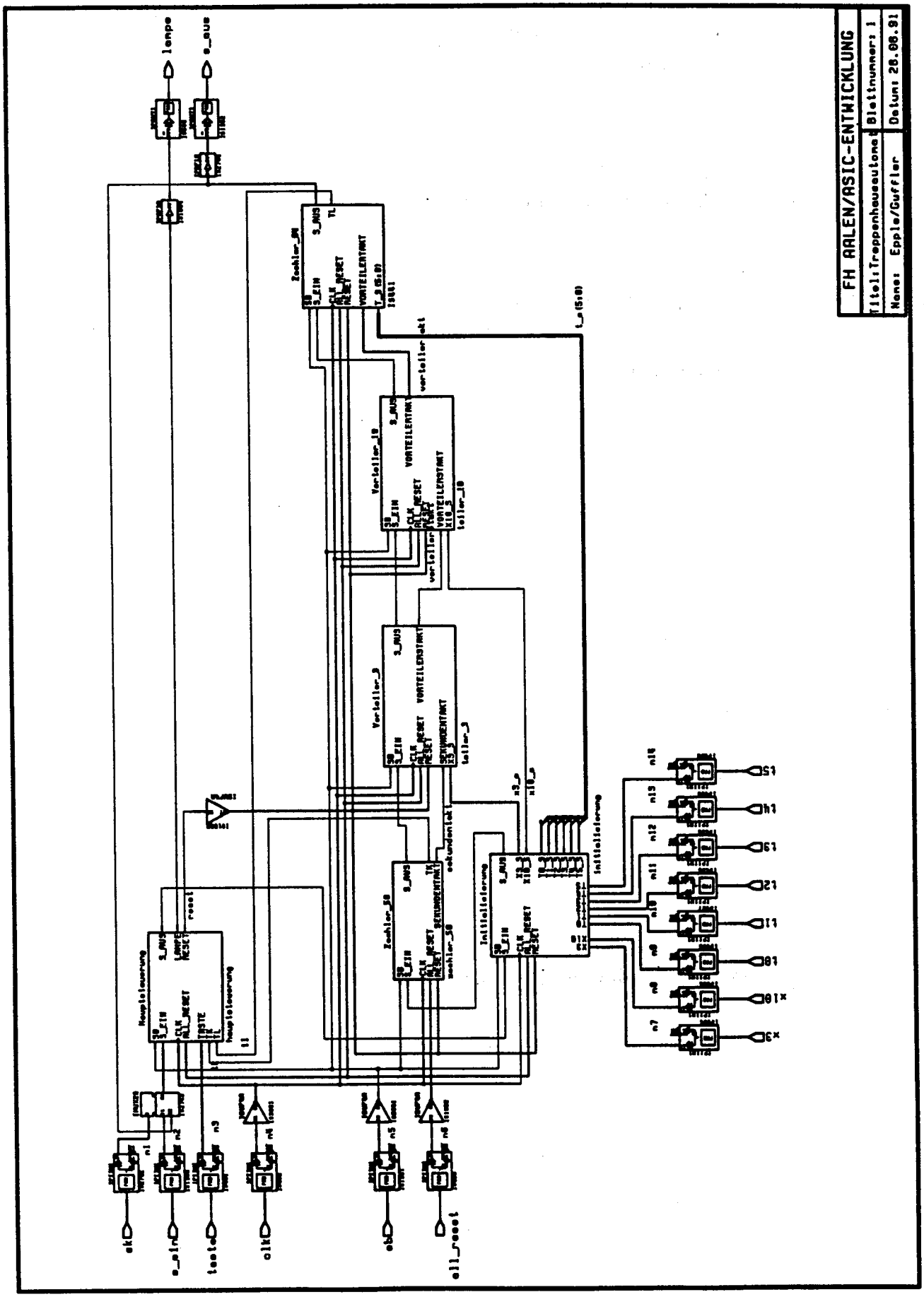
beendet das Programm QUICKFAULT.
Dagegen wird bei dem Kommando CTRL-Y oder bei der Eingabe von

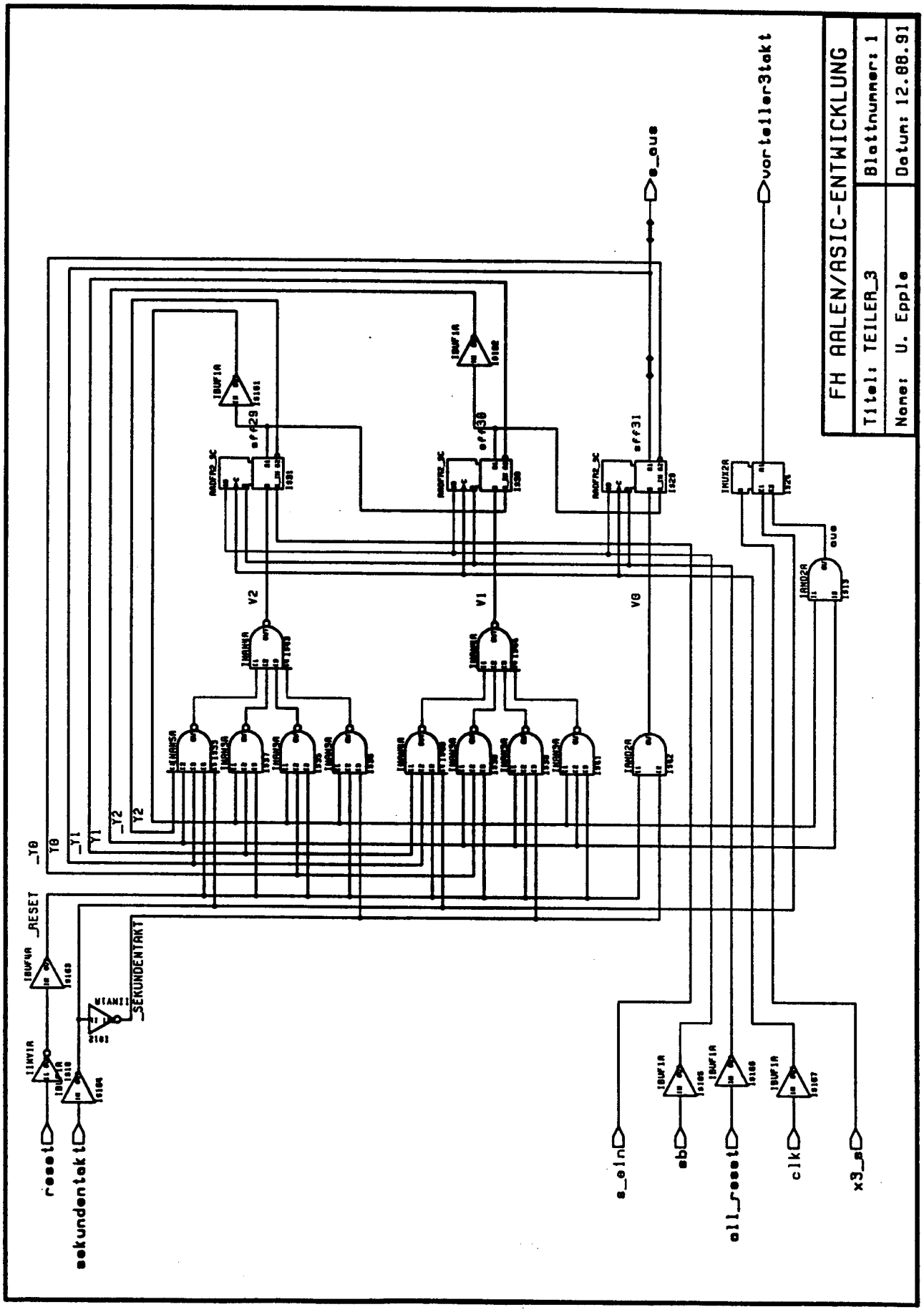
Quickfault> **BYE** **<RETURN>**

das Programm beendet und abgespeichert.

2.10 Stimuli für das Programm Quickfault anhand eines Beispiels

Auf den folgenden Seiten ist das Quickfault-Stimuli, für eine ebenfalls auszugsweise dargestellten Schaltungsbeispiels, zu sehen.


FH AALEN/ASIC-ENTWICKLUNG
Titel:Treppenhaussteuerung
Blattnummer: 1
Name: Epple/Guffler
Datum: 26.06.91
lampe
s_aus
taste
clk
all_reset
Zaehler_MN
Verteiler_10
Verteiler_3
Zaehler_60
Hauptsteuerung
Initialisierung


FH AALEN/ASIC-ENTWICKLUNG
Titel: TEILER_3
Blattnummer: 1
Name: U. Epple
Datum: 12.08.91
vorteiler3takt
s_aus
aus
V2
V1
V0
_Y0
Y0
_Y1
Y1
_Y2
Y2
_RESET
SEKUNDENTAKT
reset
sekundentakt
s_eln
sb
all_reset
clk
x3_s

Stimuli zur Initialisierung von Quickfault:

Simulationsdatei trepenhausautomat fuer Quickfault

```
#    Datei : qf.ini
#    Datum : 11.09.91
#     view sheet
     transcript on
     faults per pass 400

     cycle 200 180

     primary output /am/I$1/fa1
     primary output /am/I$1/fa2
     primary output /am/I$1/fa3
     primary output /am/I$81/fa4
     primary output /am/I$81/fa5
     primary output /am/I$21/fa6
     primary output /am/ba1
     primary output /am/ba2
     primary output /am/ba3
     primary output /am/ba4
     primary output /am/ba5
     primary output /am/ba6
     primary output /am/ba7
     primary output /tha/hauptsteuerung/sff1
     primary output /tha/hauptsteuerung/sff2
     primary output /tha/hauptsteuerung/sff3
     primary output /tha/hauptsteuerung/sff4
     primary output /tha/initialisierung/sff5
     primary output /tha/initialisierung/sff6
     primary output /tha/initialisierung/sff7
     primary output /tha/initialisierung/sff8
     primary output /tha/initialisierung/sff9
     primary output /tha/initialisierung/sff10
     primary output /tha/initialisierung/sff11
     primary output /tha/initialisierung/sff12
     primary output /tha/initialisierung/sff13
     primary output /tha/initialisierung/sff14
     primary output /tha/initialisierung/sff15
     primary output /tha/initialisierung/sff16
     primary output /tha/initialisierung/sff17
     primary output /tha/initialisierung/sff18
     primary output /tha/initialisierung/sff19
     primary output /tha/initialisierung/sff20
     primary output /tha/zaehler_50/teiler_50/sff21
     primary output /tha/zaehler_50/teiler_50/sff22
     primary output /tha/zaehler_50/teiler_50/sff23
     primary output /tha/zaehler_50/teiler_50/sff24
     primary output /tha/zaehler_50/teiler_50/sff25
     primary output /tha/zaehler_50/teiler_50/sff26
     primary output /tha/zaehler_50/tk/sff27
     primary output /tha/zaehler_50/tk/sff28
     primary output /tha/teiler_3/sff29
     primary output /tha/teiler_3/sff30
     primary output /tha/teiler_3/sff31
     primary output /tha/teiler_10/sff32
```

```
    primary output /tha/teiler_10/sff33
    primary output /tha/teiler_10/sff34
    primary output /tha/teiler_10/sff35
    primary output /tha/teiler_10/sff36
    primary output /tha/zaehler_64/sff37
    primary output /tha/zaehler_64/sff38
    primary output /tha/zaehler_64/sff39
    primary output /tha/zaehler_64/sff40
    primary output /tha/zaehler_64/sff41
    primary output /tha/zaehler_64/sff42
    primary output /tha/zaehler_64/sff43
    primary output /tha/zaehler_64/sff44
    primary output /tha/zaehler_64/vergleicher/sff45
    primary output /tha/n1
    primary output /tha/n2
    primary output /tha/n3
    primary output /tha/n4
    primary output /tha/n5
    primary output /tha/n6
    primary output /tha/n7
    primary output /tha/n8
    primary output /tha/n9
    primary output /tha/n10
    primary output /tha/n11
    primary output /tha/n12
    primary output /tha/n13
    primary output /tha/n14

# Simulation starten
    do qf.do
#------------------------------ENDE-------------------------------
```

Stimuli zur Fehlersimulation mit Quickfault:

```
write line ^$time
# Dateiname: qf.do
# Schaltungsname: asic_ims_ta
# Funktion: Fehlersimulation
# Datum : 23.10.91

    transcript on
#    view sh
 #  DEFine Bus t_t t5_t t4_t t3_t t2_t t1_t t0_t -Combine

    CLOck Period 200
    FORCe takt_i 0S 50 -Repeat
    FORCe takt_i 1S 100 -Repeat

    CLOck Period 200
    FORCe takt_a 0S 50 -Repeat
    FORCe takt_a 1S 100 -Repeat

    CLOck Period 200
    FORCe takt_t 0S 50 -Repeat
    FORCe takt_t 1S 100 -Repeat

#Impulsformer

    force td_i    0   0
    force td_i    1   700
    force td_i    0   800
    force td_i    1   900
    force td_i    0   1000



#Ampelsteuerung
# Anfangswerte setzen

    force   nb_a      0   0
    force   taste_a   0 . 0
    force   s_ein_a   0   0
    force   sb_a      0   0
    force   sk_a      1   0

# Reset

    force   res_a     1   400
    force   res_a     0   800

# In Normalbetrieb uebergehen

    force   nb_a      1   2000

# Fussgaengertaste_t betaetigt

    force   taste_a   1   2200
```

```
    force   taste_a   0   4000
    force   taste_a   1   5000

# In Blinkbetrieb uebergehen

    force   nb_a      0   5400

# Scanbtrieb

    force   sb_a      1   7600
    force   s_ein_a   1   8200
    force   s_ein_a   0   8400
    force   s_ein_a   1   8600

#zuruecksetzen

    force   sb_a      0   9400
    force   res_a     1   9600

# Scanbetrieb

    force   sk_a      0   10000
    force   sb_a      1   10000
    force   res_a     0   10200
    force   s_ein_a   0   10200
    force   s_ein_a   1   10400
    force   s_ein_a   0   10400
    force   s_ein_a   1   11400

# Normalbetrieb

    force   sb_a      0   11400
    force   nb_a      1   12000
    force   taste_a   0   12400

# Ampel zuruecksetzen

    force res_a       1   12600
    force res_a       0   13000

# Scanbetrieb

    force   taste_a   1   13400
    force   sb_a      1   13800
    force   s_ein_a   1   13800
    force   s_ein_a   0   14400
    force   s_ein_a   1   14800
    force   sb_a      0   15000
    force   res_a     1   15000

# In Blinkbetrieb uebergehen

    force   nb_a      0   15400

# Scanb_aetrieb
```

```
    force   sb_a        1   15800
    force   s_ein_a     0   15800
    force   s_ein_a     1   16000
    force   s_ein_a     0   16400
    force   s_ein_a     1   16600
    force   s_ein_a     0   16800
    force   sb_a        0   16800

# Normalbetrieb

    force   nb_a        1   20000
    force   taste_a     1   24000



# Treppenhausautomat
# Voreinstellung
    force   all_reset_t   1   0
    force   taste_t       0   0
    force   sb_t          1   0
    force   s_ein_t       0   0
    force   t_t           15  0
    force   sk_t          0   0
    force   x3_t          0   0
    force   x10_t         0   0

#Flipflops zuruecksetzen
    force   all_reset_t   0   600

# Scanbetrieb
    force   s_ein_t       1   1000
    force   s_ein_t       0   1200
    force   s_ein_t       1   1400
    force   s_ein_t       0   1600
    force   s_ein_t       1   1800
    force   s_ein_t       0   2000
    force   s_ein_t       1   2200
    force   s_ein_t       0   2400
    force   s_ein_t       1   2600
    force   s_ein_t       0   2800
    force   s_ein_t       1   3000
    force   s_ein_t       0   3200
    force   s_ein_t       1   3400
    force   s_ein_t       0   3600
    force   s_ein_t       1   3800
    force   s_ein_t       0   4000
    force   s_ein_t       1   4200
    force   s_ein_t       0   4400
    force   s_ein_t       1   4600
    force   s_ein_t       0   4800
    force   s_ein_t       1   5000
    force   s_ein_t       0   5200
    force   s_ein_t       1   5400
    force   s_ein_t       0   5600
    force   s_ein_t       1   5800
```

```
    force   s_ein_t       0   6000
    force   s_ein_t       1   6200
    force   s_ein_t       0   6400
    force   s_ein_t       1   6600
    force   s_ein_t       0   6800
    force   s_ein_t       1   7000
    force   s_ein_t       0   7200
    force   s_ein_t       1   7400
    force   s_ein_t       0   7600
    force   s_ein_t       1   7800
    force   s_ein_t       0   8000
    force   s_ein_t       1   8200
    force   s_ein_t       0   8400
    force   s_ein_t       1   8600
    force   s_ein_t       0   8800
    force   s_ein_t       1   9000
    force   s_ein_t       0   9200
    force   s_ein_t       1   9400
    force   s_ein_t       0   9600
    force   s_ein_t       1   9800
    force   s_ein_t       0   10000
    force   s_ein_t       1   10200
    force   s_ein_t       0   10400
    force   s_ein_t       1   10600
    force   s_ein_t       0   10800
    force   s_ein_t       1   11000
    force   s_ein_t       0   11200
    force   s_ein_t       1   11400

#Normalbetrieb
    force   sb_t          0   11600
    force   sk_t          1   11600
#Flipflops zuruecksetzen
    force   all_reset_t   1   11800
    force   all_reset_t   0   12000

#lampe 21s an obwohl taste_t gedrueckt
    force   taste_t       1   12200
    force   taste_t       0   223200

#zaehlen nach 9s
    force   x3_t          1   223600
    force   x10_t         0   223600
    force   t_t           4   223600
    force   taste_t       1   223800
    force   taste_t       0   224000

#nachtriggerung
    force   x3_t          0   361000
    force   x10_t         0   361000
    force   t_t           19  361000
    force   taste_t       1   361800
    force   taste_t       0   362000
    force   taste_t       1   373000
    force   taste_t       0   373200
```

```
#dauerlicht
    force   taste_t       1   634400
    force   taste_t       0   634600
    force   taste_t       1   635400
    force   taste_t       0   635600
    force   taste_t       1   786000
    force   taste_t       0   786200

#30s zaehlen mit x10
    force   x10_t         1   78700
    force   x3_t          0   78800
    force   t_t           3   78800
    force   taste_t       1   78820
    force   taste_t       0   78840

#umschalten waehrend lampe brennt
    force   x3_t          1   100560
    force   x10_t         0   100560
    force   taste_t       1   100580
    force   taste_t       0   100600
    force   x10_t         0   100640
    force   x3_t          0   100640
    force   t_t           3F  100640

#umschasltung wirkt sich erst beim naechsten aus
#Zaehler 64 durchzaehlen lassen
    force   taste_t       1   1036000
    force   taste_t       0   1036200

# Teiler 3 und Teiler 10 durchgeschalten
    force   x3_t          1   1700000
    force   x10_t         1   1700000
    force   t_t           27  1700200
    force   taste_t       1   1700400
    force   taste_t       0   2110000

#s_aus wieder einlesen
    force   sk_t          1   2349400
    force   sb_t          1   2349400

#Normalbetrieb
    force   sb_t          0   2449400
    force   sk_t          0   2449400

# Alle Flipflops auf 1 setzen
    force   s_ein_t       1   2650000
    force   sb_t          1   2651000
    force   sb_t          0   2651000

#Simulation starten
    run     2700000

 write line ^$time
#------------------------------ENDE-----------------------------
```

Literaturverzeichnis

[1] Jutta Oppold, Thomas Schneider, Jürgen Maier, Markus Kreis
Diplomarbeiten: Arnulf Heilig Studienarbeit:
Mentor Graphics
FH Aalen 1989/1990

[2] Mentor Graphics: QUICKSIM Family Referenz Manual
Software Version 7.0, April 1989

[3] Mentor Graphics: QUICKSIM User's Manual
Software Version 7.0, April 1989

[4] Axel Kemper, Manfred Meyer: Entwurf von
Semicustrom-Schaltungen. Springer-Verlag 1989

[5] Manfred Weyerer, Gerald Goldemund: Prüfbarkeit
elektronischer Schaltungen. Carl Hanser Verlag 1988

[6] J. Mavor, M. A. Jack, P. B. Denyer: Indroduction to
MOS LSI Design. Addison-Wesley Publishing Company 1983

[7] Jon Turino: Design to test. Logical Solutions Technologie
1989

[8] Hans Wojtkowiak: Test und Testbarkeit digitaler
Schaltungen. B. G. Teubner Stuttgart 1988

[9] Johann Siegl, Herbert Eichele: Hardwareentwicklung mit
ASIC. Hüthing Buch Verlag GmbH Heidelberg 1990

[10] H. Reichl, M. Bleicher: SMT/ASIC Systemintegration.
Hüthing Buch Verlag GmbH Heidelberg 1988

[11] Institut für Mikroelektronik Stuttgart: Testen Digitaler
Integrierter Schaltungen am IMS. Technischer Bericht 1990

[12] Mentor Graphics: QUICKFAULT User's Manual
Software Version 7.0, April 1989

[13] Peter Ammon: Gate Arrays Hüthig-Verlag 1986

[14] Hans-Ulrich Post: Entwurf und Technologie
hochintegrierter Schaltungen

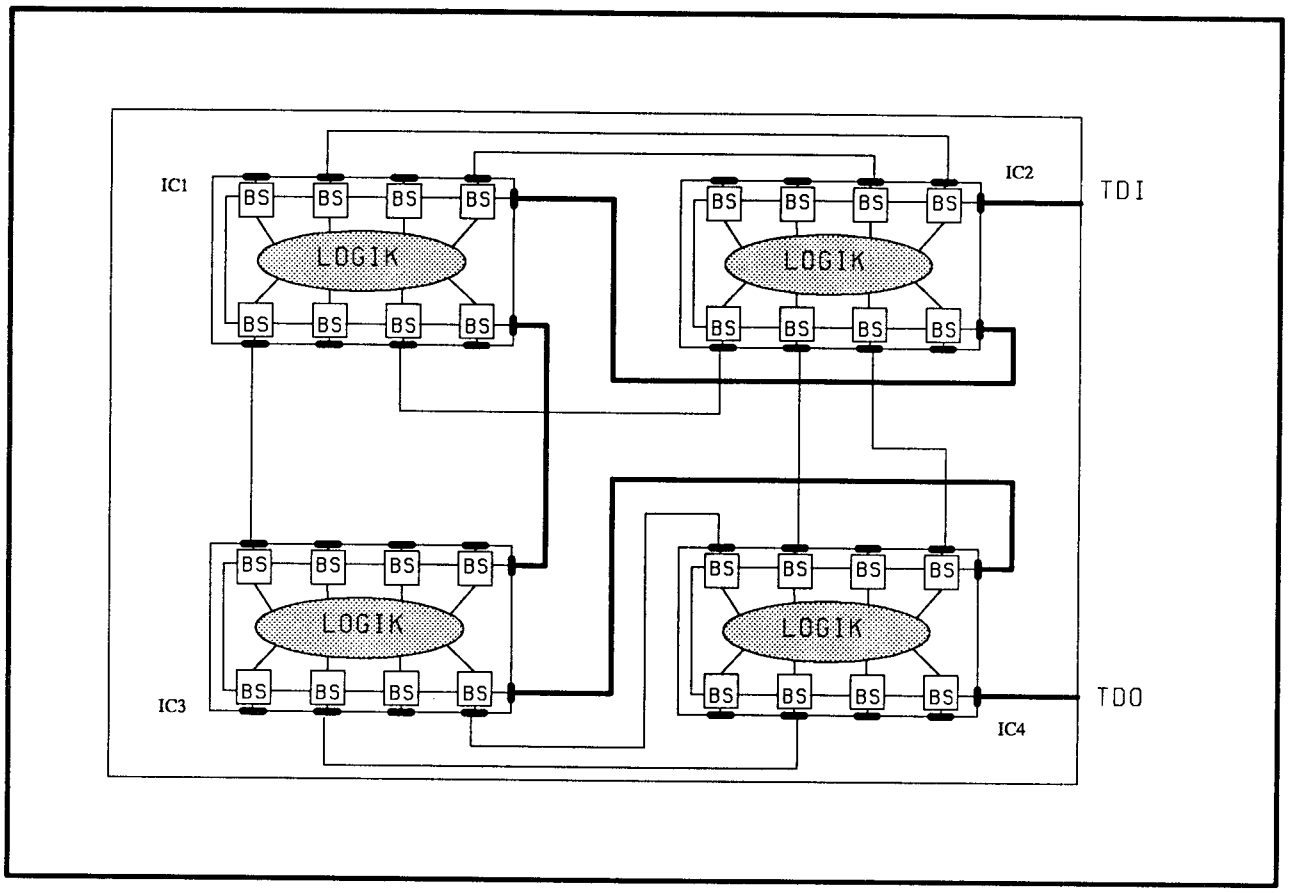
[15] Institut für Mikroelektronik Stuttgart: Gate Forrest
2-icron CMOS Standard Cell Library Version 1.1.
Technischer Bericht 1990

[16] Institut für Mikroelektronik Stuttgart: Gate Forrest
Pad Library GFPADLIB1C2A. Technischer Bericht 1990

[17] Institut für Mikroelektronik Stuttgart: Die Sizes, Qualitätssicherung,
Packages und Test-Services. Technischer Bericht 1990

[18] Mentor Graphics: CHIPGRAPH User's Manual
Software Version 7.0, April 1989

5) Entwurf eines JTAG- Testpfads für GATEFOREST- Gate Arrays



Bearbeiter : Reinhard Gaber
Fachbereich : Technische Informatik
Zeitraum : WS 91/92
Betreuer : Prof. A. Führer
Fachhochschule : Ulm

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

LIST OF FIGURES

1. Allgemeines zu Pruefverfahren

Die Funktionsüberprüfung einer gefertigten Schaltung wird mit Testautomaten (englisch *"Automatic Test Equipment"*, ATE) durchgeführt. Ziel ist es mit Hilfe der Testautomaten eine sehr hohe Prüfqualität zu erreichen. Die Weiterentwicklung der Schaltungsintegration in Bezug auf Komplexität und Integrationsdichte führte dazu, daß Testautomaten immer höhere Anforderungen erfüllen müssen. Heute sind für einige Prüfverfahren bereits die technischen Grenzen erreicht.

Die vollständige Prüfung einer gefertigten Schaltung läßt sicht in drei Testphasen unterteilen:

1) Die Prüfung der benutzten Bausteine.
2) Die Prüfung der Bausteinverbindungen.
3) Die Prüfung des Echzeitverhaltens der gesamten Schaltung.

In der Praxis setzten sich zwei Verfahren durch :

1) Der In- Circuit- Test
2) Der Funktionstest

Beide Verfahren weisen Vor- und Nachteile auf, und für eine hohe Prüfqualität kann auf keines verzichtet werden.

Der **In- Circuit- Test** wird eingesetzt, die logische Funktion der gesamten Schaltung und die der einzelnen Bausteine zu prüfen. Die Zugänglichkeit der Bausteinanschlüsse ist eine wichtige Voraussetzung, da ein Nagelbrett die Verbindung zwischen ATE und Schaltung herstellt. Mit den Testnadeln können definierte Signalfolgen der Schaltung zugeführt oder Schaltungszustände abgefragt werden. Dieses kontrollierte Anlegen und Abfragen der Signalzustände ermöglicht es, neben der Bausteinfunktion auch die Bausteinverbindungen zu prüfen. Ein großer Vorteil liegt in der flexiblen Handhabung der Testsoftware. Jede bausteinspezifische Testsoftware muß nur einmal erstellt werden und steht sämtlichen Schaltungen zur Verfügung, die den Baustein benutzen. Die Hauptkosten entstehen durch die erforderliche Testhardware, weil sie meistens individuell hergestellt werden muß. (Nagelbrett)

Beim **Funktionstest** wird das Verhalten der Schaltung unter Echtzeitbedingungen geprüft. Sehr schnelle Schaltungen erfordern diese Prüfung, da der In- Circuit- Test nicht beliebig schnell ablaufen kann. In der Regel werden die gleichen Anschlüsse benutzt, die später im Betrieb zur Verfügung stehen. In Sonderfällen werden schwerzugängliche Signale während der Testphase abgegriffen und protokolliert. Neben der Echzeittauglichkeit wird zusätzlich die Funktionsfähigkeit der Bausteine und deren korrekte Leitungsverbindungen bestätigt. Ein großes Problem ist die vollständige Testbarkeit einer Schaltung und die Zeit, die dazu benötigt wird. Dieser Faktor darf nicht unterbewertet werden, da er sehr kostenträchtig ist. Ein großer Nachteil ist die geringe Wiederverwendbarkeit existierender Testsoftware.

Moderne ATE's sind so konzipiert, daß beide Verfahren verwendet werden. Der In-Circuit- Test wird eingesetzt, übliche Produktionsfehler schnell zu lokalisieren, und nach deren Behebung wird der Funktionstest benutzt, das Echtzeitverhalten der Schaltung zu prüfen. (Siehe Abbildung 1-1)

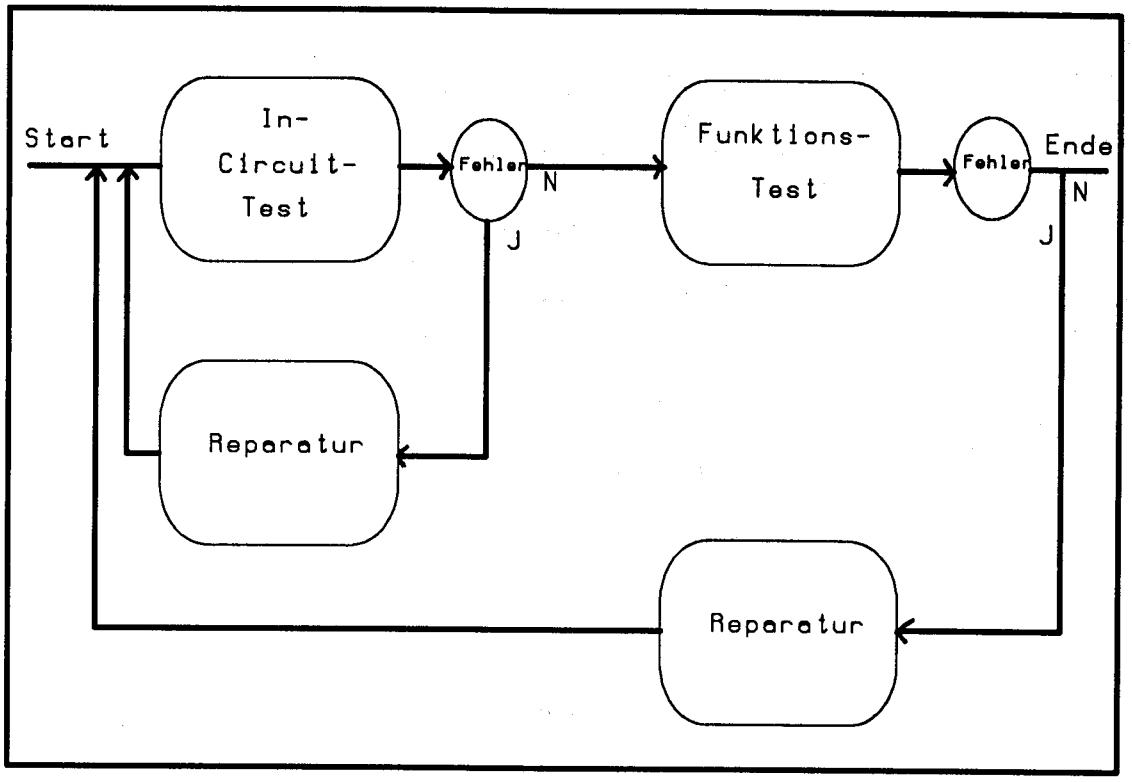


Figure 1-1 Pruefalgorithmus

Wichtige Parameter eines Testautomaten sind die Testgeschwindigkeit, die Zahl der testbaren Anschlüsse, Durchsatz, Genauigkeit und die Auflösung. Die Kosten eines Testautomaten richten sich nach den gestellten Anforderungen und können die Kosten der Schaltung erheblich erhöhen.

Zunehmend werden neue Technologien für die Herstellung gedruckter Schaltungen verwendet, welche die Zugänglichkeit der Bausteinanschlüsse nicht mehr gewährleisten. Ein Beispiel wäre die **SMT** Technologie (englisch *"Surface- mount Technology")* und das *"Silicon- on Silicon"*- Verfahren. Werden diese Verfahren benutzt, müssen zusätzliche Testpunkte vorgesehen werden, damit herkömmliche Testautomaten eingesetzt werden können. Diese Vorgehensweise ist teuer, und mit zunehmender Verfeinerung der SMT- Technik wird die Benutzung eines Testautomaten immer aufwendiger.

Die Entwicklung einer neuen Testmethode, die den In- Circiut- Test ähnelt und ausschließlich die Anschlüsse des Funktionstest benutzt, wäre den neuen Herstellungstechnologien gewachsen. Viele Firmen erkannten dieses Problem und entwickelten gemeinsam die Boundary- Scan- Architektur, die international normiert wurde. Effektiv ist diese neue Technik nur dann, wenn die eingesetzten Bausteine sie auch tatsächlich enthalten.

2. Die Entwicklung des Boundary- Scan- Verfahrens

Das Boundary- Scan- Verfahren ist eine Weiterentwicklung der *Scan- Design-Technik*. Diese Technik wird benutzt, eine möglichst vollständige Prüfbarkeit komplexer Schaltungen auf IC- Basis zu erzielen. Die Funktionsweise basiert auf der Zusammenschaltung vorhandener Flipflops zu einem Schieberegister, mit dessen Hilfe ein Binärmuster von außen geladen oder ein Testergebnis zwischengespeichert werden kann. Wichtig ist die Verwendung spezieller Scan- Design- FFs, die über ein Steuersignal verfügen, das zwischen Normal- und Testbetrieb umschaltet. Ein geladenes Binärmuster beeinflußt die Bausteinlogik, und das Ergebnis wird in ein Schieberegister zwischengespeichert, um später analysiert zu werden. Die eindeutige Beziehung zwischen Eingangs- und Ausgangssignal ermöglicht die vollständige Funktionsprüfung einer Schaltung. Stellt man Eingang und Ausgang des Schieberegisters nach außen zur Verfügung und verwendet ausschließlich Bausteine die beide Anschlüsse bereitstellen, so läßt sich die gesamte Schaltung ähnlich wie eine integrierte Schaltung prüfen. (Siehe Abbildung 2-1)

Sämtliche Schieberegister müssen zu einem gemeinsamen Schiebepfad verbunden werden, damit Testmuster geladen und Ergebnisse abgefragt werden können.

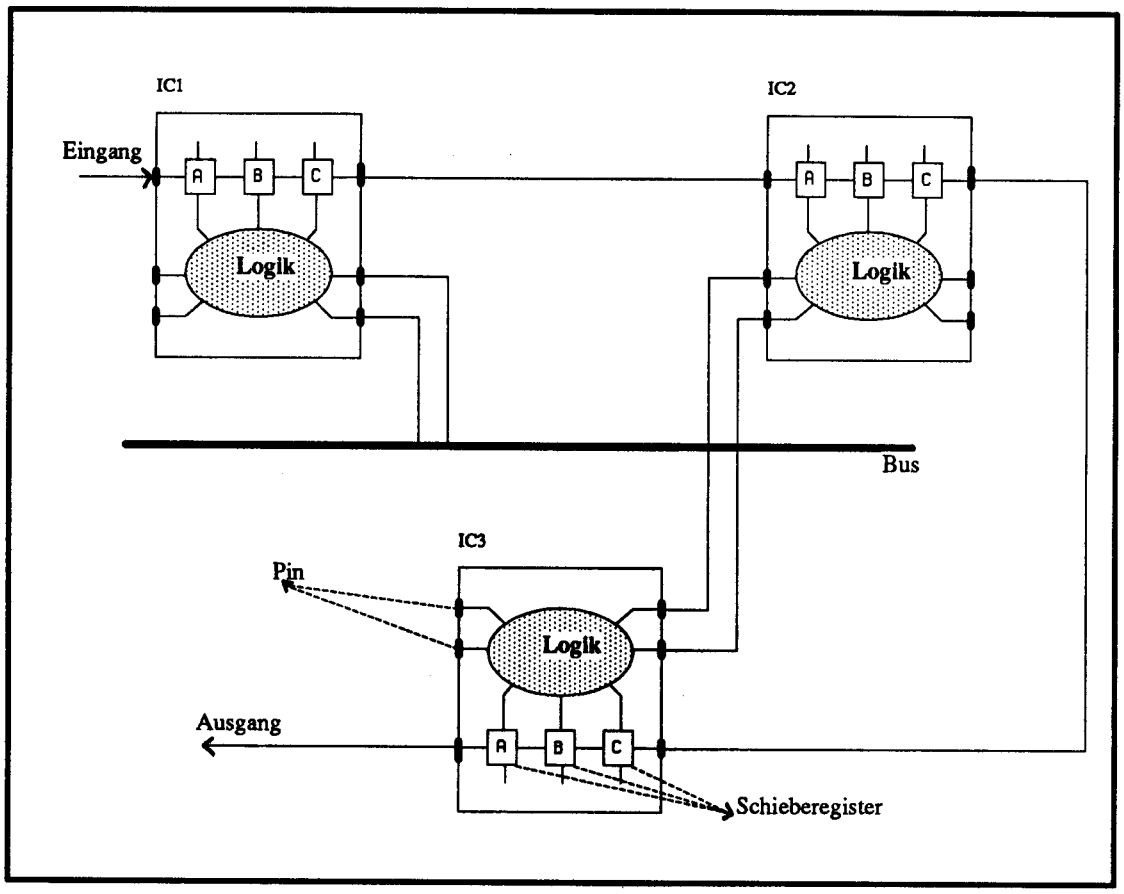


Figure 2-1 Die erweiterte Scan- Design- Technik

Betrachtet man die erweiterte Scan Design- Technik näher, so sind erhebliche Mängel festzustellen (Siehe Abbildung 2-1) :

1) Ein Bit muß sämtliche Bausteine durchlaufen bevor es an der gewünschten Position ist. (Beispiel: IC3 [C]=1 --> Eingang auf 1 setzen und 7 mal schieben.

2) Beim Schieben bleiben einzelne Bausteine nicht immer in einem stabilen Zustand. Kritisch wird dieses Verhalten, wenn mehrere Bausteine über bidirektionale Anschlüsse an einen Bus gekoppelt sind.

3) Fehler müssen häufig indirekt aus mehreren Ergebnissen ermittelt werden. Das Resultat sind große Datenströme und eine lange Testdauer.

Die zuvor erwähnten Mängel lassen sich beheben, indem jeder Ein- /Ausgang eines Bausteines mit einer speziellen FF- Schaltung versehen wird. Dieses Verfahren wird *Boundary- Scan* und die FF- Schaltungen *BS- Zellen* (englisch "Boundary- Scan Cells", BSC) genannt. Alle BS- Zellen werden zu einem Schiebepfad verbunden, der es ermöglicht Zelleninhalte zu prüfen oder auf definierte Zustände zu setzen (Siehe Abbildung 2-2).

Zusätzlich zur Bausteinfunktion muß eine Logik vorhanden sein, welche die Steuerung der BS- Zellen vornimmt.

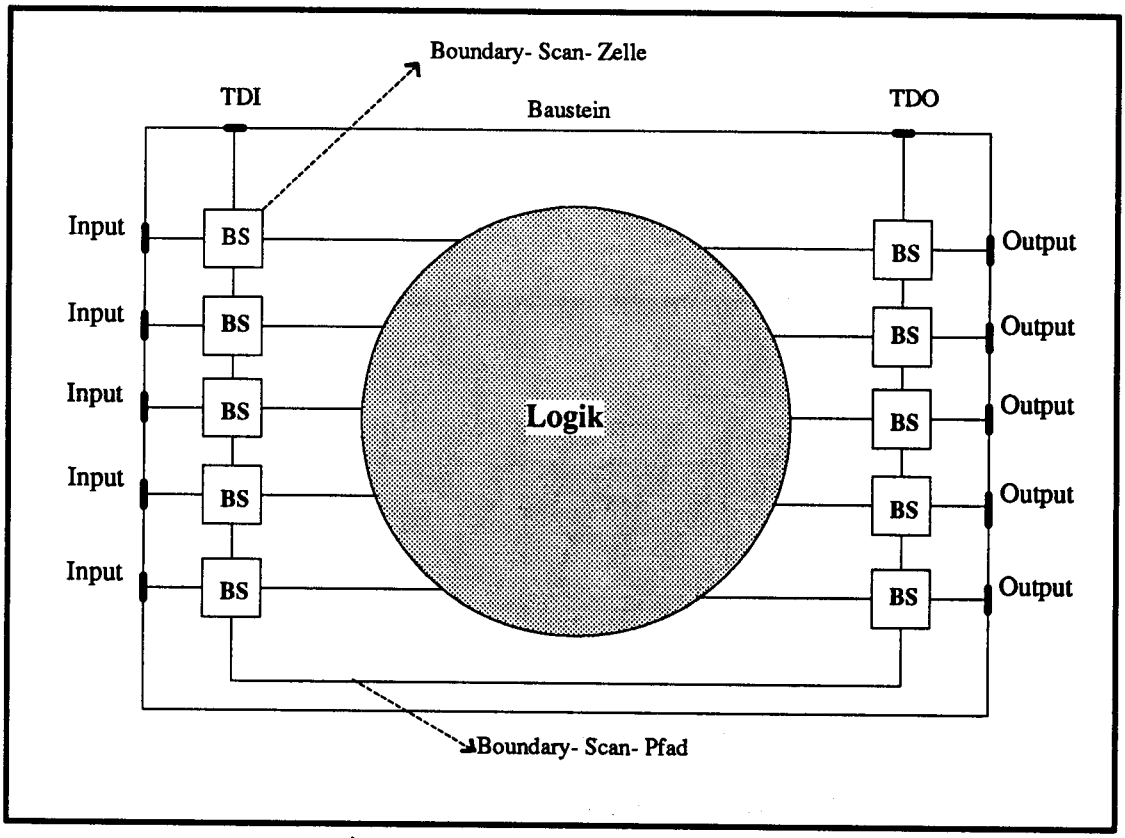


Figure 2-2 Anordnung der Boundary- Scan- Zellen

Folgende Eigenschaften müssen von jeder BS- Zelle erfüllt werden :

a) Im Normalbetrieb müssen die BS- Zellen Ein- und Ausgangssignale direkt durchschalten. BS- Eingangszellen : Pin --> Bausteinlogik.
BS- Ausgangszelle : Bausteinlogik --> Pin.

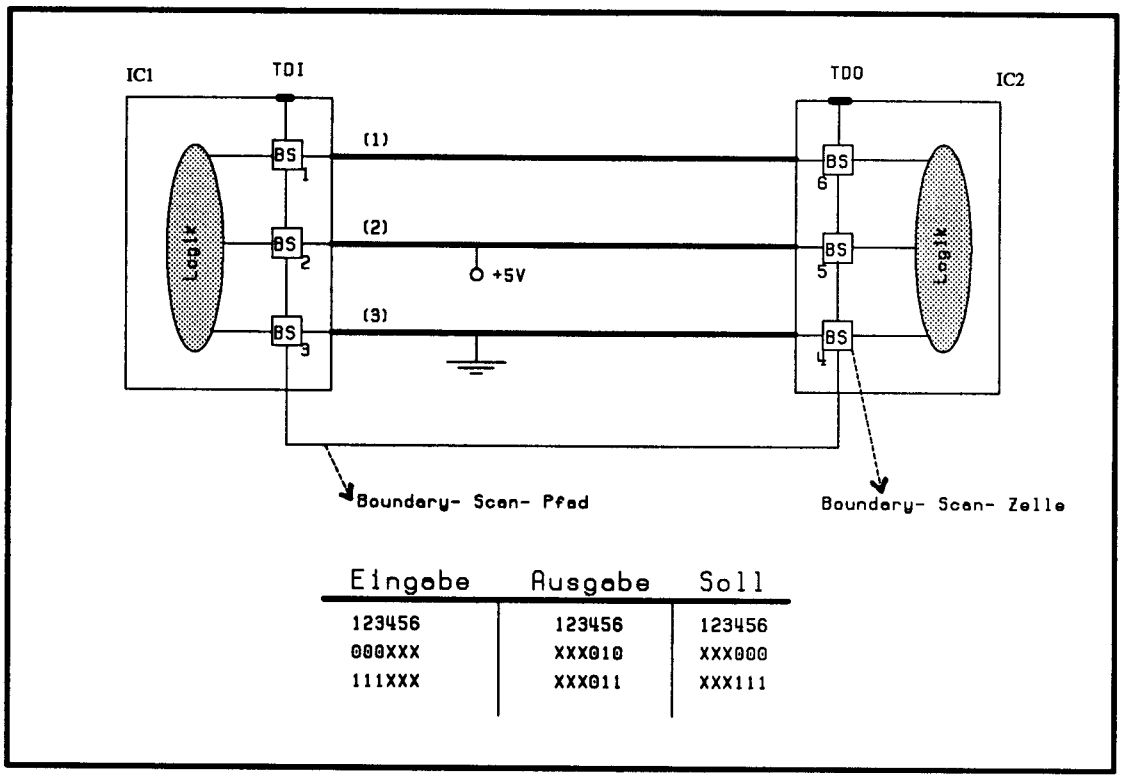
b) Im Testbetrieb werden zwei Phasen unterschieden, in denen das Verhalten der BS- Eingangszellen und BS- Ausgangszellen aufeinander abgestimmt sein müssen.

1) **Extest- Modus** : Dieser Modus wird benutzt externe Verbindungen zu prüfen. Die BS- Ausgangszellen geben definierte Signale nach außen, und die BS- Eingangszellen speichern diese externen Signale zwischen.

2) **Intest- Modus** : Dieser Modus dient der Prüfung verschiedener Bausteinfunktionen. Die BS- Eingangszellen stellen Testdaten bereit, und die BS- Ausgangszellen speichern die Ergebnisse zwischen.

Beispiel : *Extest- Modus*

Beim Prüfen der Leitungsverbindungen werden die BS- Ausgabezellen benutzt, definierte Pegel auszugeben, die von den BS- Eingabezellen anderer Bausteine gespeichert werden. Durch vergleichen der Zelleninhalte miteinander verbundener BS- Eingabe- /BS- Ausgabezellen, läßt sich bestimmen welche Leitungen defekt sind. Fehlerursachen wie Kurzschlüsse mit Vcc oder Vdd können schnell und eindeutig ermittelt werden. (Siehe Abbildung 2-3)



Eingabe	Ausgabe	Soll
123456	123456	123456
000XXX	XXX010	XXX000
111XXX	XXX011	XXX111

Figure 2-3 Extest- Modus

Ergebnis : Leitung 2 kann keine 0 übertragen.
BS- Ausgabezelle 2 (0) --> BS- Eingabezelle 5 (1)
BS- Ausgabezelle 2 (1) --> BS- Eingabezelle 5 (1)

Leitung 3 kann keine 1 übertragen.
BS- Ausgabezelle 3 (0) --> BS- Eingabezelle 4 (0)
BS- Ausgabezelle 3 (1) --> BS- Eingabezelle 4 (0)

Beispiel : *Intest- Modus*

Beim Prüfen der Bausteinfunktion dienen die BS- Eingabezellen dazu, Testmuster aufzunehmen und diese der Bausteinlogik bereitzustellen. Die BS- Ausgabezellen halten die Testergebnisse fest, und mit der Überprüfung kann gleichzeitig ein neues Testmuster geladen werden (Siehe Abbildung 2-4). Das Und- Gatter ist defekt, wenn die BS- Zelle 5 nicht die Bits 0 0 0 1 als Ergebnis liefert, wie im Beispiel dargestellt.

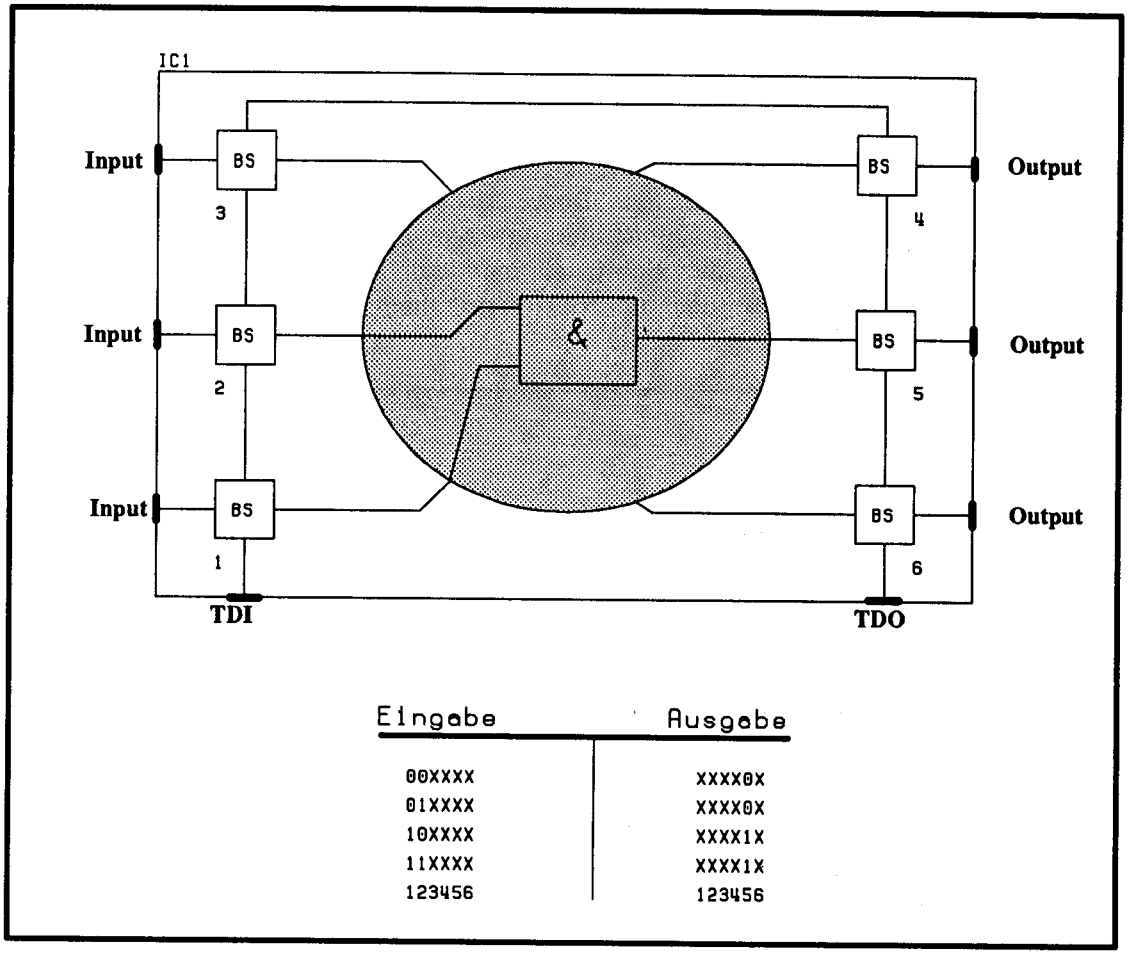


Figure 2-4 Intest- Modus

Erst nach Anwendung beider Testphasen kann die fehlerfreie Funktion der Bausteine und deren korrekten Leitungsverbindungen angenommen werden.

3. Die Boundary- Scan- Architektur

Die gesamte Testlogik setzt sich aus 4 Einheiten zusammen.

1) TAP- Schnittstelle (englisch: "Test Access Port", TAP)
2) TAP- Controller
3) Befehlsregister (englisch: "Instruction Register", IR)
4) Testdatenregister

Abbildung 3-1 zeigt die Funktionsanordnung der 4 Einheiten.

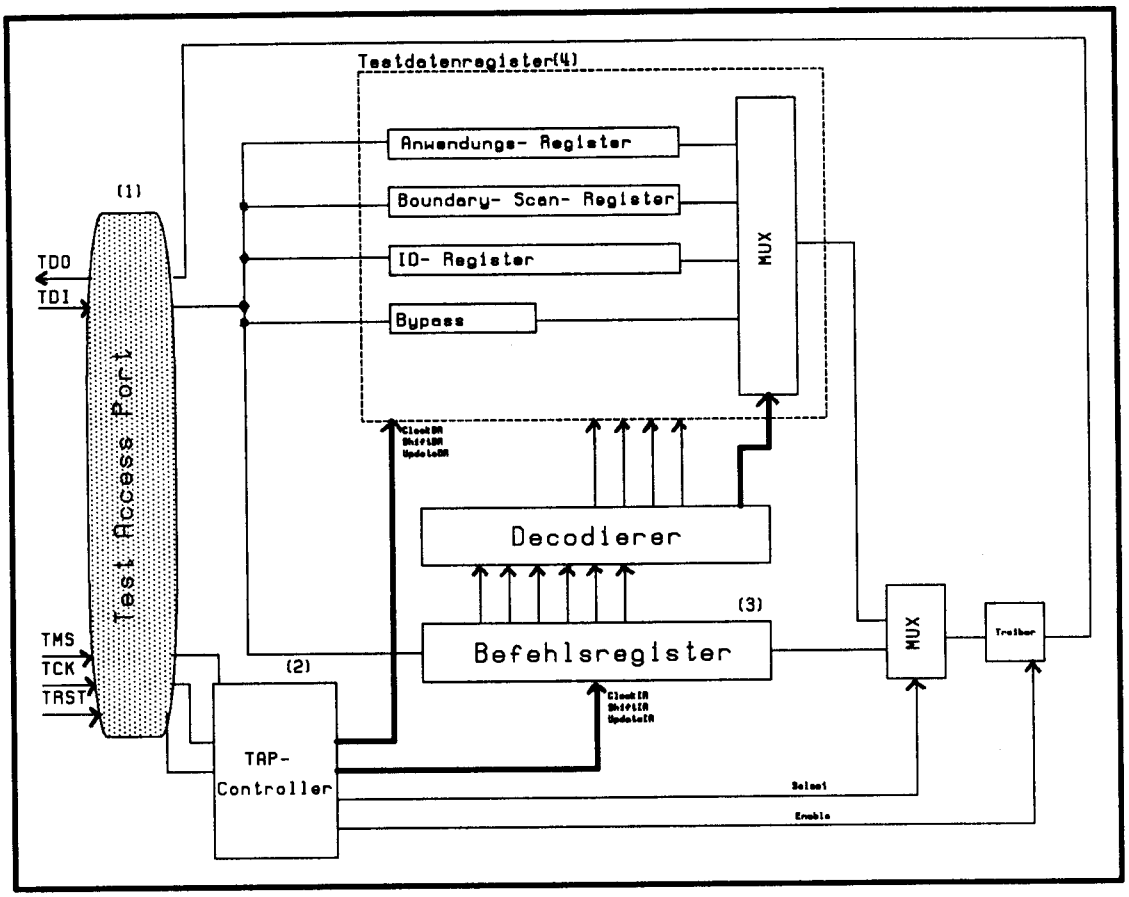


Figure 3-1 Die Boundary- Scan- Architektur

Befehle, Funktionsweise und Eigenschaften jeder Einheit sind im IEEE 1149.1 Standard genau festgelegt.

3.1 Die TAP- Schnittstelle (Test Access Port)

Bausteine, die mit der BS- Architektur ausgestattet sind, müssen über eine TAP-Schnittstelle verfügen. Aufgabe der normierten Schnittstelle ist eine synchrone Kommunikation zwischen mehreren Bausteinen und deren Testeinheiten herzustellen. Der Standard schreibt eine Mindestzahl von 3 Eingängen und einem Ausgang vor. Optional darf ein vierter Eingang vorgesehen werden, mit dem die Testlogik asynchron zurückgesetzt wird.

Vorgeschriebene Eingänge :

TDI --> Test Data Input
TMS --> Test Mode Select Input
TCK --> Test Clock Input
TRST --> Test Reset Input (Optional)

Vorgeschreibener Ausgang :

TDO --> Test Data Output

Die vorgeschriebenen Signale müssen an einem Bausteinanschluß liegen, der nicht anderweitig benutzt werden darf. Diese Vorschrift ermöglicht die volle Bausteinfunktion während gleichzeitig Testfunktionen parallel ablaufen.

Der TCK- Eingang (Test Clock Input)

Über den TCK- Eingang wird die Testlogik, speziell der TAP- Controller, mit einem eigenen Takt versehen. Der Takt synchronisiert den Ablauf der Testfunktionen zwischen mehreren Bausteinen. Der Systemtakt darf zu diesem Zweck nicht benutzt werden, da er häufig zu schnell oder mehrfach vorhanden ist. Ausgenutzt werden nur die Flanken des Taktsignals. Mit der steigenden Flanke werden die Eingangssignale TDI und TMS geladen bzw. ausgewertet. Die fallende Flanke wird benutzt, Daten am TDO- Ausgang bereitzustellen.

Der TMS- Eingang (Test Mode Select Input)

Daten, die am TMS- Eingang angelegt werden, beeinflussen den Testablauf. Testoptionen wie z.B. Daten schieben, neue Daten bereitstellen, Ein- /Ausgangszustände zwischenspeichern usw. lassen sich durch entsprechende 0/1- Folgen aktivieren.
Der Tap- Controller nimmt die Daten auf und generiert weitere Takt- und Steuersignale, die anderweitig benötigt werden. Die fallende TCK- Flanke ist ein günstiger Zeitpunkt ein neues Datenbit anzulegen.

Der TDI- Eingang (Test Data Input)

Über den TDI- Eingang werden die Daten, Befehle und Testdaten seriell eingelesen. Abhängig vom Testablauf wird mit jeder steigenden TCK- Flanke ein Bit ins Befehlsregister oder eines der Testdatenregister geladen. Ein günstiger Zeitpunkt ein neues Datenbit anzulegen ist mit der fallenden TCK- Flanke.

Der TRST- Eingang (Test Reset Input)

Eine 0 am TRST- Eingang setzt die gesamte Testlogik in den Urzustand zurück. Das TRST- Signal kann auch als Enable- Signal benutzt werden, da eine anliegende 0 die Testlogik außer Betrieb setzt. Erreicht wird der Urzustand auch, indem das
TMS- Signal für 5 TCK- Takte auf 1 gehalten wird.

Der TDO- Ausgang (Test Data Output)

Beim Schieben werden Daten vom Befehlsregister oder eines der Testdatenregister zum TDO- Ausgang geschoben. Mit jeder fallenden TCK- Flanke erscheint ein neues Bit am Ausgang. Nur im Schiebebetrieb wird dieser Ausgang vom TAP- Controller freigegeben, ansonsten ist er hochohmig geschaltet.

3.2 Die Funktionsweise der BS- Architektur

Die Boundary- Scan- Architektur stellt dem Benutzer zahlreiche Testoptionen zur Verfügung, die durch entsprechendes Laden des Befehlsregisters aktiviert werden. Zum Laden eines Befehls muß der TAP- Controller zuvor in den Lade- Zustand gesetzt werden. Benutzt wird zu diesem Zweck das TMS- Signal. Im Lade- Zustand generiert der TAP- Controller Takt- und Steuersignale, die das Befehlsregister benutzt, Bits vom TDI- Eingang zu laden. Am TDO- Ausgang erscheint der alte Inhalt des Befehlsregisters. Beendet wird der Ladevorgang, indem der TAP- Controller im Update_IR- Zustand gesetzt wird. Der Decodierer bearbeitet den neuen Befehl und gibt ein bestimmtes Testdatenregister frei. Jeder Befehl darf sich zu diesem Zeitpunkt nur auf ein Testdatenregister beziehen, und eine ungültige Angabe bezieht sich immer auf das Bypass- Register. Angewandte Testfunktionen wirken sich nur auf den Inhalt des selektierten Testdatenregisters aus. Die Selektion ist solange gültig bis ein neuer Befehl geladen wird.

3.3 Befehle, die der IEEE 1149.1 Standard vorsieht

Der IEEE 1149.1 Standard sieht ein Minimum an Befehlen vor, die jede Implementierung vorweisen muß. Optional können, unter Beachtung folgender Regeln, weitere Befehle realisiert werden :

1) Jeder Befehl darf sich zu einem bestimmten Zeitpunkt nur auf ein Testdatenregister beziehen. Bei Bedarf ist eine Zusammenarbeit zwischen Register und Bausteinlogik gestattet.

2) Nicht selektierte Testdatenregister dürfen die Bausteinlogik und das selektierte Register nicht beeinflussen.

3) Jeder Befehl darf nur einen Schiebepfad zwischen TDI und TDO im SHIFT_DR- Controllerzustand ermöglichen.

Befehle die jede Implementierung vorweisen muß :

1) Bypass
2) Sample/Preload
3) Extest

Weitere nützliche Befehle sind :

1) Idcode
2) Usercode
3) Runbist
4) Intest

Zu internen Testzwecken wird empfohlen, daß entweder der Befehl **INTEST** oder **RUNBIST** vorhanden sein sollte. Länge und Binärcode der Befehle sind vom Standard nicht festgelegt, ausgenommen die Binärcodierung der Befehle **BYPASS** und **EXTEST**. Der Bypass- Befehl muß die Codierung 11...11 und der Extest- Befehl die Codierung 00...00 aufweisen.

Der Bypass- Befehl

Nur der Bypass- Befehl darf das Bypass- Register selektieren. Aufgebaut ist das Register aus einer einzigen Registerstufe, die den kürzesten Weg zwischen TDI und TDO herstellt. Benutzt wird dieser Befehl, Testdaten schnell zum Zielbaustein zu schieben.

Der Sample/Preload- Befehl

Der Sample/Preload- Befehl gestattet Signalproben an Ein- und Ausgängen im Normalbetrieb zu nehmen. Die Analyse der einzelnen BS- Zellen liefert die Zustände, die zu einem bestimmten Zeitpunkt vorlagen. Weiterhin dient dieser Befehl die, BS- Zellen mit Daten vorzubelegen. (Siehe INTEST,EXTEST)

Der Extest- Befehl

Mit dem Extest- Befehl werden die Bausteinverbindungen geprüft. Die Schattenspeicher der BS- Ausgabezellen werden zuvor mit dem Sample/Preload- Befehl auf definierte Werte gesetzt. Diese Daten werden ausgegeben, sobald der EXTEST-Befehl geladen wurde.

Der Intest- Befehl

Der Intest- Befehl wird benutzt, die Bausteinlogik zu prüfen. Die BS- Eingabezellen stellen Testdaten bereit, und die BS- Ausgabezellen speichern die Ergebnisse. Während die Ergebnisse analysiert werden, können bereits neue Testdaten geladen werden. Bevor der Intest- Befehl benutzt wird, sind die Schattenspeicher der BS- Zellen auf definierte Zustände zu setzen.

Der Runbist- Befehl

Der Runbist- Befehl stellt eine weitere Möglichkeit bereit, die Bausteinlogik zu prüfen. Im Gegensatz zum Intest- Befehl wird eine schnelle Hardware (Selbsttest) eingesetzt. Der Selbsttest reduziert die benötigte Datenmenge sehr stark, wobei auf den Single- Step- Modus versichtet wird. Die BS- Eingabezellen stellen die Anfangsdaten bereit, und ein Testdatenregister speichert das Endergebnis. Das Endergebnis gibt nur Auskunft darüber, ob der Baustein funktionsfähig ist oder nicht.

Der Idcode- Befehl

Der Idcode- Befehl ermöglicht es die bausteinspezifischen Daten herauszulesen. Bei den Daten handelt es sich um die Herstellernummer, Bausteinnummer und Versionsnummer. Abgelegt sind die Daten in einem speziellen Register- dem Identifikationsregister (englisch : " Device Indentification Register", ID- Register)

Der Usercode- Befehl

Der Usercode- Befehl erlaubt es dem Anwender, seine Bausteindaten ins ID- Register zu laden und herauszuschieben. Sämtliche Bausteine, bei denen die Bausteindaten nachträglich programmiert werden können, brauchen den Usercode- Befehl.

Die Implementierung für die Gate Forest 2.0 Familie sieht außer dem USERCODE-Befehl alle erwähnten Befehle vor. Die Befehle INTEST, SAMPLE/PRELOAD und RUNBIST sind mehrfach vorgesehen, und maximal können 7 weitere anwendungsspezifische Testdatenregister angesprochen werden.

Die implementierten Befehle sind :

1) Extest	--> *XXX000*	--> *Befehlscodes.*
2) Bypass	--> *XXX111*	
3) Idcode	--> *XXX110*	
4) Intest0 .. Intest7	--> *XXX001*	
5) Sample/Preload0 .. Sample/Preload7	--> *XXX010*	
6) Runbist0 .. Runbist7	--> *XXX011*	

4. Die Hardware der verschiedenen Einheiten

4.1 Der TAP- Controller

Der TAP- Controller ist ein Netzwerk, das 16 Zustände vorsieht und auf Veränderungen am TMS- und TCK- Eingang reagiert. Sämtliche Sequenzen einer Testoption werden vom TAP- Controller gesteuert. Im Standard sind 9 Ausgangssignale und maximal 3 Eingangssignale definiert.

Vorgeschriebene Eingangssignale :

1) TMS	-->	*Steuersignal*
2) TCK	-->	*Taktsignal*
3) TRST	-->	*Reset- Signal (Optional)*

Vorgeschriebene Ausgangssignale :

1) RESET	-->	*setzt das Befehlsregister zurück*
2) SELECT	-->	*trifft die Auswahl zwischen Befehlsregister oder Testdatenregister*
3) ENABLE	-->	*gibt den TDO- Ausgang frei*
4) SHIFTIR	-->	*Schiebesignal für das Befehlsregister*
5) CLOCKIR	-->	*Taktsignal für das Befehlsregister*
6) UPDATEIR	-->	*lädt den Schattenspeicher des Befehlsregisters*
7) SHIFTDR	-->	*Schiebesignal für das selektierte Testdatenregister*
8) CLOCKDR	-->	*Taktsignal für das selektierte Testdatenregister*
9) UPDATEDR	-->	*lädt den Schattenspeicher des selektierten Testdatenregisters*
10) RT_I	-->	*kennzeichnet den RUN_ TEST/IDLE- Zustand*

Ergänzt wurde der TAP- Controller um ein weiteres Signal (RT_I), das den RUN- TEST/IDLE- Zustand kennzeichnet. Das RT_I- Signal wird von der Runbist-Hardware benötigt, und trotzdem wird in sämtlichen Unterlagen auf die Erzeugung des Signals nicht eingegangen.
Die möglichen Zustände, in denen sich der TAP- Controller befinden kann, sind in Abbildung 4-1 dargestellt. Mit jeder steigenden TCK- Flanke wird das TMS- Signal ausgewertet und je nach Wert ein Zustandswechsel vorgenommen. Neue Ausgangssignale werden entweder mit der fallenden oder steigenden TCK- Flanke ausgegeben.

Aufgaben der einzelnen Zustände

Der Test- Logic- Reset- Zustand (Binärcodierung F)

Im Test- Logic- Reset- Zustand wird die Testlogik inaktiv geschaltet und der Normalbetrieb des Bausteines vorgesehen. Erreicht wird dieser Zustand, indem das TRST- Signal auf 0 gesetzt wird oder das TMS- Signal für 5 Takteperioden auf 1 gehalten wird - egal in welchem Zustand der TAP- Controller sich befindet. Das Reset- Signal ist das einzige aktive Ausgangssignal und setzt das Befehlsregister asynchron zurück. Verlassen wird dieser Zustand sobald eine 0 am TMS- Eingang erkannt wird. Der nachfolgende Zustand, Run- Test/Idle, beeinflußt die Bausteinfunktion nicht, da entweder das Bypass- oder ID- Register selektiert ist.

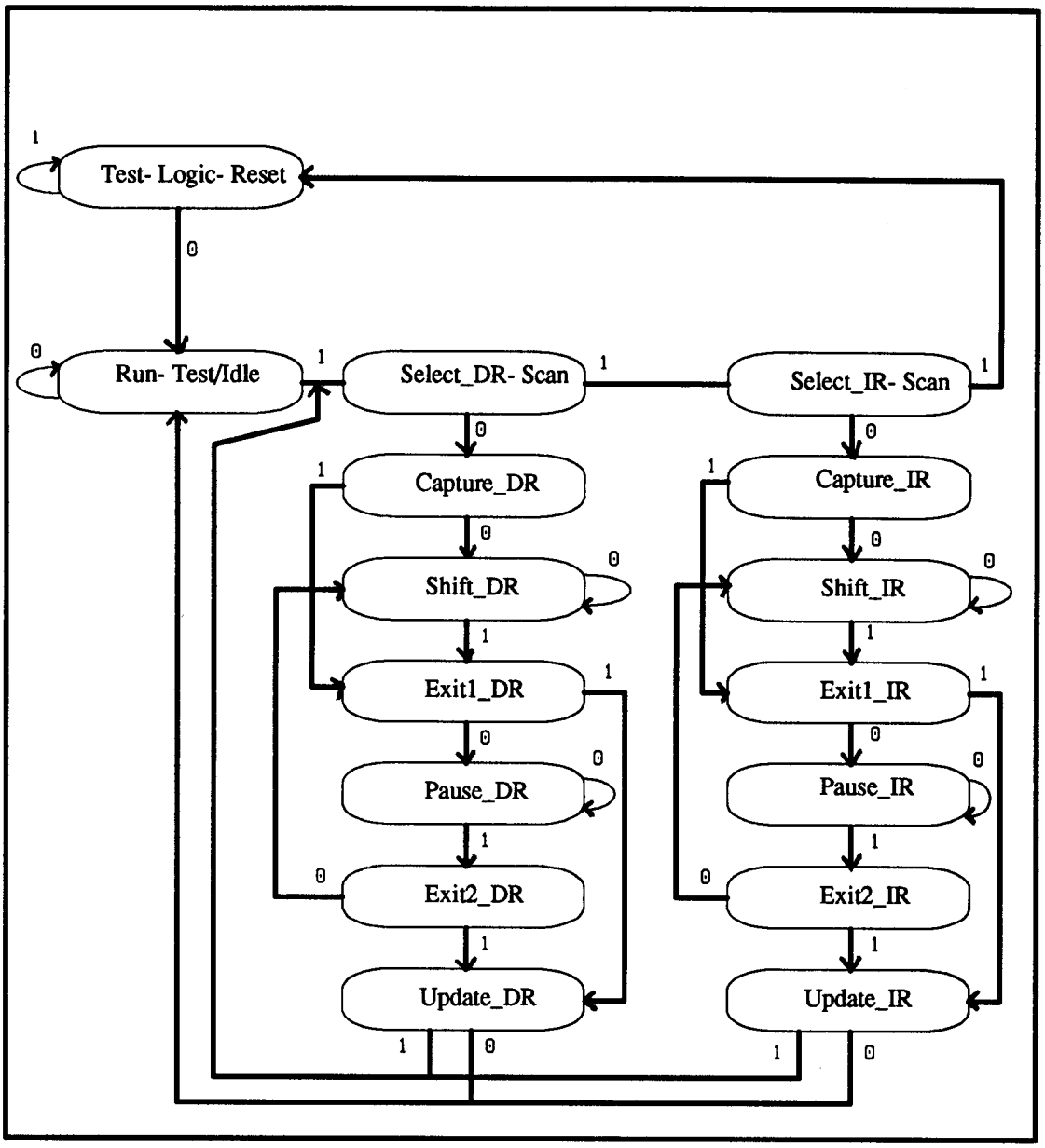


Figure 4-1 TAP- Controller- Zustaende

Der Run- Test/Idle- Zustand (Binärcodierung C)

Der Run- Test/Idle- Zustand zeigt ausschließlich dann eine Wirkung, wenn eines der Befehle Runbist1..7 oder Intest0 geladen wurde. Beim Intest0- Befehl wird nur in diesem Zustand der Systemtakt freigegeben (Das "single stepping" wird hierdurch ermöglicht). Das RT_I- Signal ist das einzige aktive Ausgangssignal, deshalb können die Befehle Runbist1..7 nur hier ablaufen. Für die restlichen Befehle dient dieser Zustand als Haltezustand. Verlassen wird er, sobald eine 1 am TMS- Eingang erkannt wird. Gewechselt wird zum Select_DR- Scan- Zustand.

Der Select_DR- Scan- Zustand (Binärcodierung 7)

Im Select_DR- Scan- Zustand wird entschieden, ob neue Testdaten geladen werden sollen. Liegt eine 0 am TMS- Eingang wird zum Capture_DR- Zustand gewechselt, der die erforderlichen Schiebesequenzen initialisiert. Eine 1 am TMS- Eingang veranlaßt einen Wechsel zum Select_IR- Scan- Zustand.

Der Select_IR- Scan- Zustand (Binärcodierung 4)

Im Select_IR- Scan- Zustand wird entschieden, ob ein neuer Befehl geladen werden soll. Liegt eine 0 am TMS- Eingang, wird zum Capture_IR- Zustand gewechselt, der die erforderlichen Schiebesequenzen initialisiert. Eine 1 am TMS- Eingang veranlaßt einen Wechsel zum Test- Logic- Reset- Zustand.

Der Capture_DR- Zustand (Binärcodierung 6)

Im Capture_DR- Zustand werden die Daten an den Paralleleingängen des selektierten Testdatenregisters geladen. Zu diesem Zweck wird ein Taktimpuls auf die ClockDR-Ausgangsleitung gelegt. Das anschließende Auswerten des Registerinhalts gibt genau darüber Auskunft, welche Zustände innerhalb der Bausteinlogik zu einem bestimmten Zeitpunkt vorlagen (BS- Register). Der Zustand dauert eine Taktperiode und abhängig vom TMS- Eingang (1/0) wird entweder zum Exit1_DR- Zustand oder Shift_DR-Zustand gewechselt.

Der Shift_DR- Zustand (Binärcodierung 2)

Der Shift_DR- Zustand stellt über ein bestimmtes Testdatenregister eine Verbindung zwischen den Anschlüssen TDI und TDO her. Mit jeder steigenden TCK- Flanke wird ein neues Bit vom TDI- Eingang ins selektierte Testdatenregister geladen und der alte Inhalt erscheint bitweise am TDO- Ausgang. Aktive Ausgangssignale sind ShiftDR und ClockDR. Verlassen wird dieser Zustand, sobald eine 1 am
TMS- Eingang erkannt wird. Nachfolgezustand ist der Exit1_DR- Zustand.

Der Exit1_DR- Zustand (Binärcodierung 1)

Im Exit1_DR- Zustand muß entschieden werden, ob die Ladeoption beendet werden soll. Liegt eine 1 am TMS- Eingang vor, wird die Ladeoption beendet, ansonsten eine Pause eingelegt. Alle Ausgangssignale des TAP- Controllers sind inaktiv. Der Zustand dauert genau eine Taktperiode.

Der Pause_DR- Zustand (Binärcodierung 3)

Während der Schiebephase kann eine Pause beliebiger Länge eingelegt werden. Der Inhalt des selektierten Testdatenregisters bleibt unverändert. Beendet wird dieser Zustand, sobald eine 1 am TMS- Eingang erkannt wird. Der Nachfolgezustand ist der Exit2_DR- Zustand. Alle Ausgangssignale des TAP- Controllers sind inaktiv.

Der Exit2_DR- Zustand (Binärcodierung 8)

Im Exit2_DR- Zustand wird entschieden, ob die Ladeoption beendet werden soll. Liegt eine 1 am TMS- Eingang, wird die Ladeoption beendet, ansonsten zum Shift_DR- Zustand gewechselt. Alle Ausgangssignale des TAP- Controllers sind inaktiv. Der Zustand dauert genau eine Taktperiode.

Der Update_DR- Zustand (Binärcodierung 5)

Falls ein Testdatenregister über Schattenspeicher verfügt werden sie mit der nächsten fallenden TCK- Flanke im Update_DR- Zustand parallel geladen. Während des Schiebens müssen die Ausgänge der Schattenspeicher konstant bleiben, damit die Bausteinlogik nicht ungewollt beeinflußt wird. Das UpdateDR- Signal ist das einzige aktive Ausgangssignal. Der Zustand dauert genau eine Taktperiode, und liegt eine 1 am TMS- Eingang, wird zum Select_DR- Scan- Zustand gewechselt, ansonsten zum Run- Test/Idle- Zustand.

Der Capture_IR- Zustand (Binärcodierung E)

Im Capture_IR- Zustand wird das Befehlsregister mit einer festgelegten ....0101- Folge geladen. Zu diesem Zweck wird ein Taktimpuls auf die ClockIR- Ausgangsleitung gelegt. Der Sinn dieser ..0101- Folge liegt darin, eine Möglichkeit zu bieten, den gesamten Schiebepfad im Shift_IR- Zustand prüfen zu können. Der alte Befehl ist noch immer gültig, und der Inhalt des noch selektierten Testdatenregisters bleibt unverändert. Der Zustand dauert genau eine Taktperiode, und liegt eine 1 am TMS- Eingang wird zum Exit1_IR- Zustand gewechselt, ansonsten zum Shift_IR- Zustand.

Der Shift_IR- Zustand (Binärcodierung A)

Der Shift_IR- Zustand stellt über das Befehlsregister eine Verbindung zwischen den Anschlüssen TDI und TDO her. Mit jeder steigenden TCK- Flanke wird ein neues Bit vom TDI- Eingang ins Befehlsregister geladen, und der alte Inhalt erscheint bitweise am TDO- Ausgang. Aktive Ausgangssignale sind ShiftIR und ClockIR. Verlassen wird dieser Zustand, sobald eine 1 am TMS- Eingang erkannt wird. Nachfolgezustand ist der Exit1_IR- Zustand.

Der Exit1_IR- Zustand (Binärcodierung 9)

Im Exit1_IR- Zustand muß entschieden werden, ob die Ladeoption beendet werden soll. Liegt eine 1 am TMS- Eingang wird die Ladeoption beendet, ansonsten eine Pause eingelegt. Alle Ausgangssignale des TAP- Controllers sind inaktiv. Der Zustand dauert genau eine Taktperiode.

Der Pause_IR- Zustand (Binärcodierung B)

Während der Schiebephase kann eine Pause beliebiger Länge eingelegt werden. Der Inhalt des Befehlsregisters bleibt unverändert. Beendet wird dieser Zustand, sobald eine 1 am TMS- Eingang erkannt wird. Der Nachfolgezustand ist der Exit2_IR- Zustand. Alle Ausgangssignale des TAP- Controllers sind inaktiv.

Der Exit2_IR- Zustand (Binärcodierung 8)

Im Exit2_IR- Zustand muß entschieden werden, ob die Ladeoption beendet werden soll. Liegt eine 1 am TMS- Eingang wird die Ladeoption beendet, ansonsten wird zum Shift_IR- Zustand gewechselt. Alle Ausgangssignale des TAP- Controllers sind inaktiv. Der Zustand dauert genau eine Taktperiode.

Der Update_IR- Zustand (Binärcodierung D)

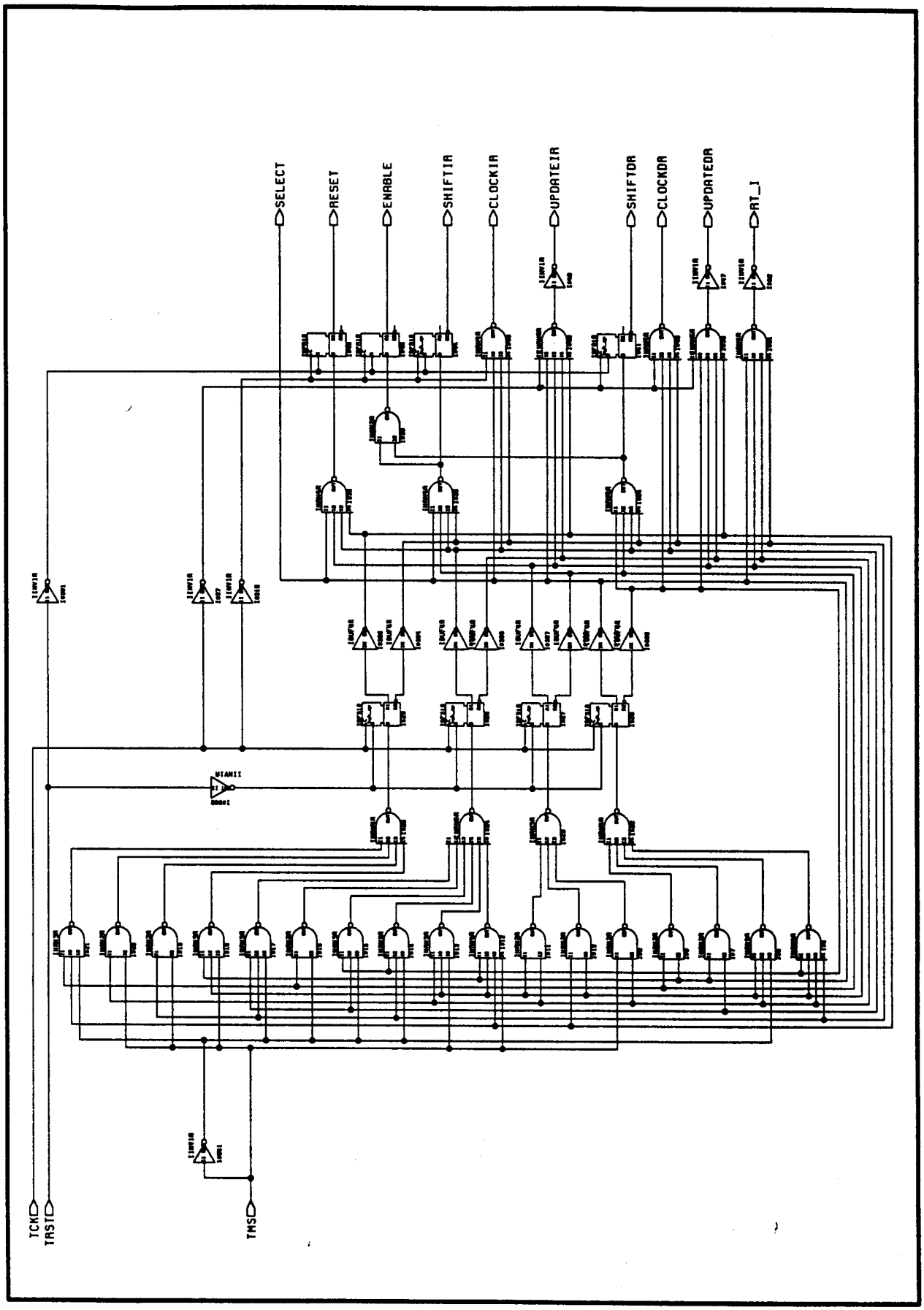
Die neuen Daten werden mit der nächsten fallenden TCK- Flanke in den Schattenspeicher des Befehlsregisters übernommen. Der Schattenspeicher wird während der Schiebephase benötigt, stabile Eingangssignale dem Decodierer bereitzustellen. Das Update_IR- Signal ist das einzige aktive Ausgangssignal. Der Zustand dauert eine Taktperiode, und liegt eine 1 am TMS- Eingang wird zum Select_DR- Scan- Zustand gewechselt, ansonsten zum Run- Test/Idle- Zustand.

Die Zustände Pause_DR und Pause_IR geben einem Testautomaten die Möglichkeit neue Daten hereinzuladen. Sehr komplexe Schaltungen können bis zu 100* 10E3 Testmuster benötigen.

Schaltungsbeschreibung (Siehe Schaltung 4-2)

Implementiert wurde die vom Standard vorgeschlagene Lösung mit der Ergänzung des RT_I- Signals. Die 16 Zustände erfordern 4 Flipflops. Folgende Codierung ist vorgesehen :

Zustand		*Hex- Wert*
Exit2_DR	-->	*0*
Exit1_DR	-->	*1*
Shift_DR	-->	*2*
Pause_DR	-->	*3*
Select_IR- Scan	-->	*4*
Update_DR	-->	*5*
Capture_DR	-->	*6*
Select_DR- Scan	-->	*7*
Exit2_IR	-->	*8*
Exit1_IR	-->	*9*
Shift_IR	-->	*A*
Pause_IR	-->	*B*
Run- Test/Idle	-->	*C*
Update_IR	-->	*D*
Capture_IR	-->	*E*
Test- Logic- Reset	-->	*F*


SELECT
RESET
ENABLE
SHIFTIR
CLOCKIR
UPDATEIR
SHIFTDR
CLOCKDR
UPDATEDR
AT_I
TCK
TAST
TMS

4.2 Das Befehlsregister (englisch "Instruction Register", IR)

Das Befehlsregister stellt dem Decodierer einen Befehl über eine längere Zeitdauer bereit. Der Decodierer selektiert anhand des Befehls ein bestimmtes Testdatenregister, auf dessen Inhalt sich die angewandten Testfunktionen beziehen. Die Befehle müssen so ausgelegt sein, daß jeweils nur ein Testdatenregister selektierbar ist.

Spezifikationen :

- *Die Größe des Befehlsregisters muß mindestens 2 Bit sein.*
- *Während ein neuer Befehl geladen wird, müssen die Parallelausgänge des Befehlsregisters stabil bleiben.*
- *Daten dürfen zwischen Ein- und Ausgang des Registers nicht negiert werden.*
- *Im Capture_IR- Zustand müssen mindestens die zwei niederwertigsten Zellen mit 01 geladen werden. (Die 1 in der niederwertigsten Zelle)*
- *Wird die Testlogik zurückgesetzt, muß entweder der Befehl Idcode oder Bypass automatisch geladen werden. Vorrang hat der Idcode- Befehl, falls das Identifikations- Register vorhanden ist.*

Das Befehlsregister wird im Capture_IR- Zustand mit 010101 geladen, damit anschließend im SHIFT_IR- Zustand der gesamte Schiebepfad auf Unterbrechungen und sonstige Fehler geprüft werden kann.

Vorgeschriebene Eingangssignale :

TDI	*--> Schiebeeingang des Registers*
SHIFTIR	*--> aktiviert mit einer 1 den Schiebebetrieb des Registers*
CLOCKIR	*--> synchrone Taktsteuerung*
UPDATEIR	*--> mit einer 1 wird der neue Registerinhalt übernommen*
TRST	*--> externes Reset- Signal*
RESET	*--> internes Reset- Signal, erzeugt vom TAP- Controller*

Vorgeschriebene Ausgangssignale :

IR_OUT	*--> Ausgang des Registers*
IR1..IR6	*--> Parallelausgänge des Registers*

Schaltungsbeschreibung

Das Befehlsregister gibt es in 2 Versionen. Die Version IR_ID wird automatisch mit dem IDCODE- Befehl und die Version IR_BY mit dem BYPASS- Befehl geladen, sobald die Testlogik zurückgesetzt wird. Aufgrund der Befehlscodierung unterscheiden sich die beiden Registerversionen nur im ersten Bit (Bef. IDCODE XXX110 Bef. BYPASS XXX111). Vorgesehen ist eine Registergröße von 6 Bit. Die 3 niederwertigsten Zellen nehmen den Befehl und die restlichen 3 Zellen die Registeradresse auf. Diese Aufteilung ermöglicht es, maximal 8 Register und 8 Befehle zu unterscheiden. Für die Praxis reichen 8 Testdatenregister, die eine beliebige Länge annehmen dürfen, aus. An den CAP_BIT- Eingängen wird jeweils eine 0 oder 1 fest angelegt, damit das Register im Capture_IR- Zustand mit 010101 geladen wird.

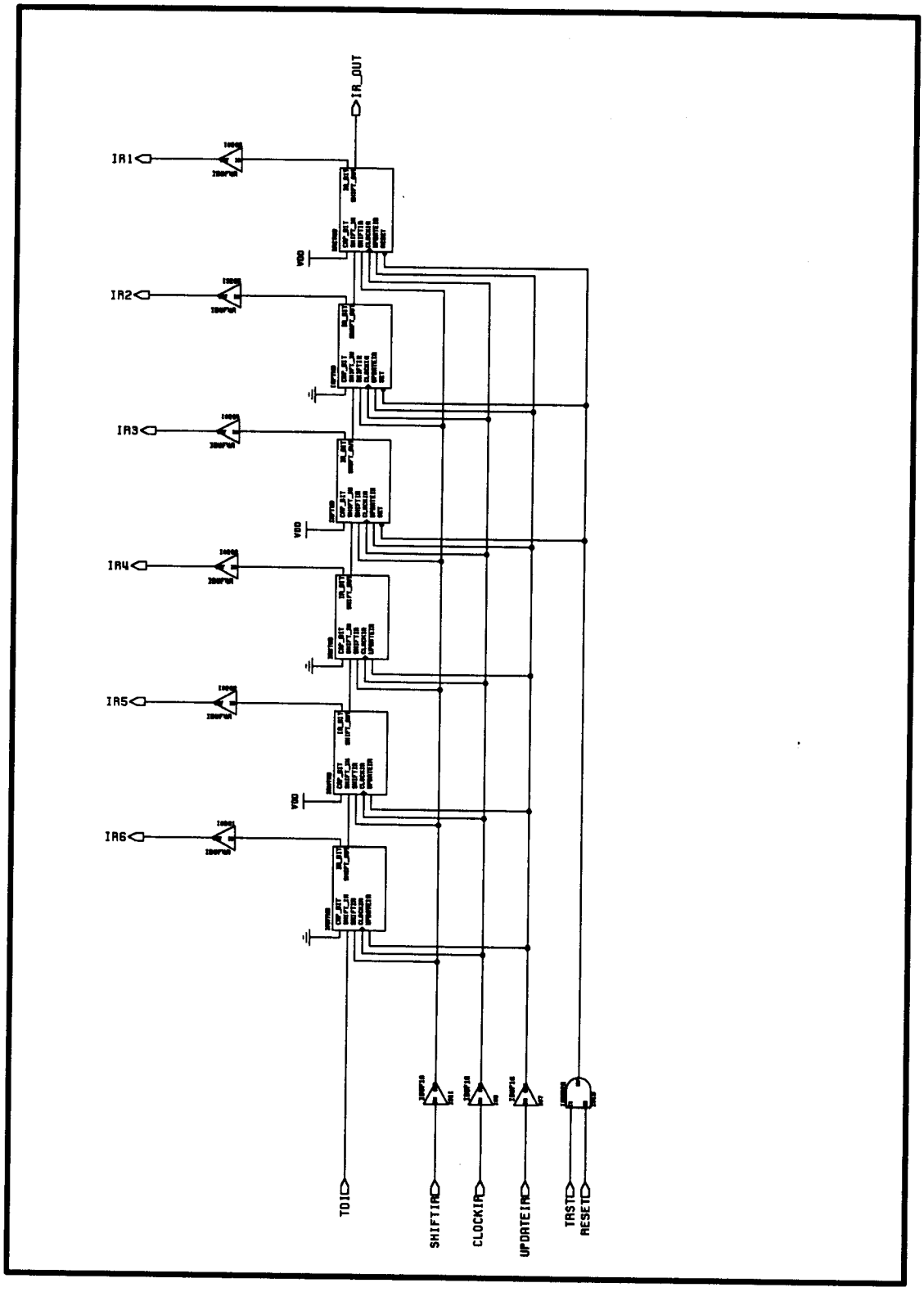


Figure 4-3 IR_ID

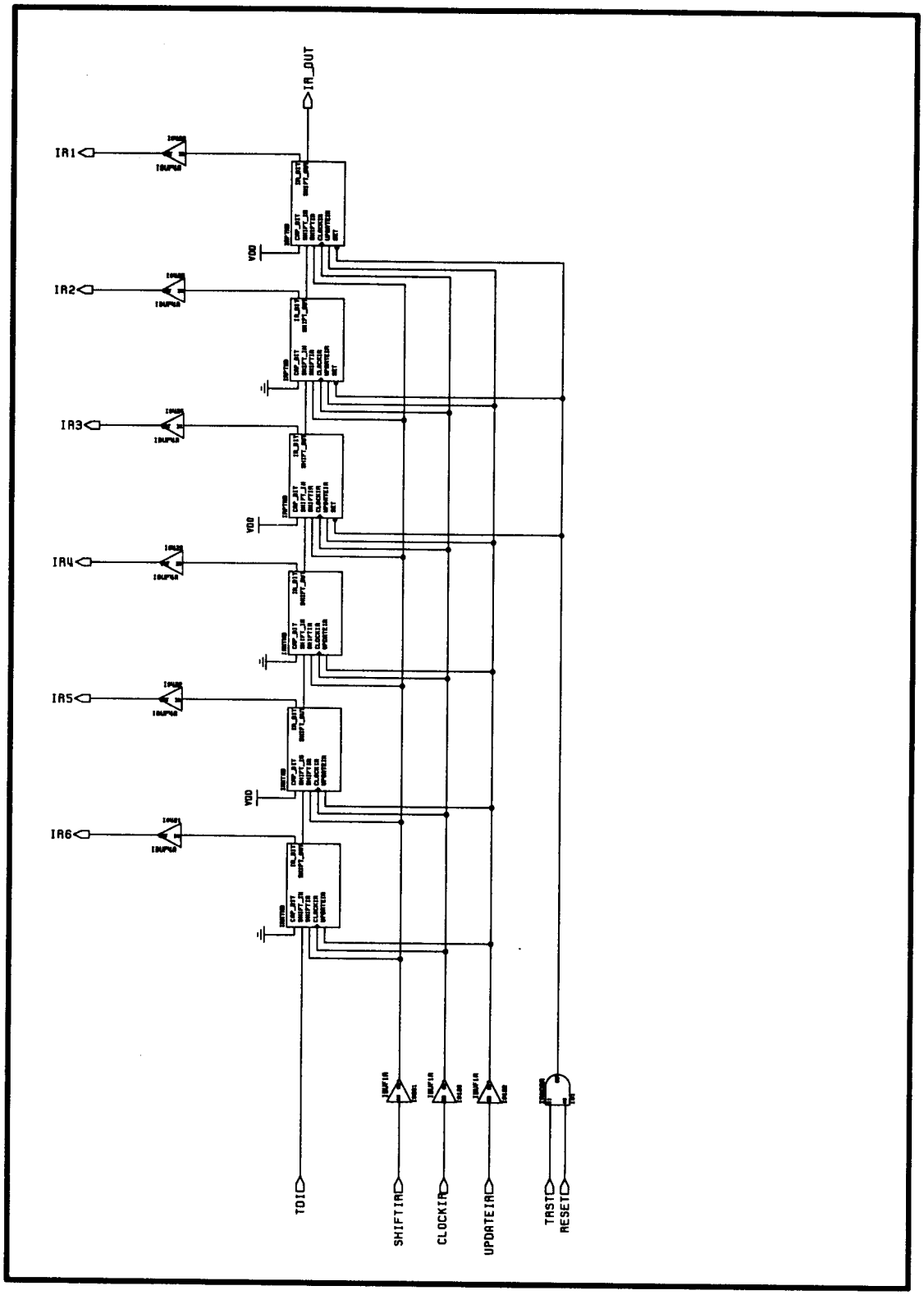


Figure 4-4 IR_BY

4.3 Der Decodierer (CS_Logic)

Je nach geladenem Befehl erzeugt der Decodierer Steuersignale für das Testdatenregister. Der Tap- Controller erzeugt ausschließlich das Taktsignal (CLOCKDR), Schiebesignal (SHIFTDR) und das Übernahmesignal (UPDATEDR). Der Decodierer ist, im Gegensatz zum TAP- Controller, nicht standardisiert und muß auf die Befehlscodierung abgestimmt sein, d.h. je nach Befehl ein bestimmtes Testdatenregister selektieren. Vorgesehen sind 3 Decodiererversionen, die eine unterschiedliche Anzahl Testdatenregister verwalten können.

Die Bezeichnung Decodierer wurde für die Schaltung nicht gewählt, da zusätzlich ein Demultiplexer integriert ist, der mit dem Taktsignal (CLOCKDR) ein bestimmtes Testdatenregister selektiert. (CS_LOGIC = Control Select Logic)

4.3.1 Die CS_Logic1- Schaltung

Die CS_Logic1- Schaltung kann 4 Register und eine RBIST- Schaltung selektieren.

Die Register sind :

1) Bypass- Register
2) ID- Register
3) Boundary- Scan- Register
4) 1 anw. -spezifisches Register

Die Schaltungsstruktur wird in Abbildung 4-5 dargestellt.

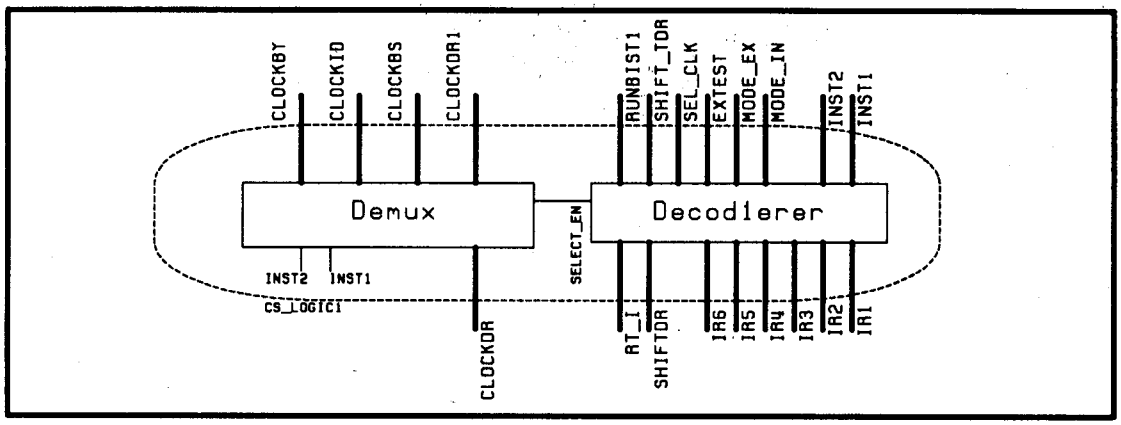


Figure 4-5 CS_Logic1

Die Eingangssignale lassen sich in 3 Gruppen (Befehl, Takt und Steuersignale) und die Ausgangssignale in 4 Gruppen (Taktsignale, Runbistsignale, Steuersignale und Mux- Steuersignale) einteilen.

Die Eingangssignale

IR1..IR6 : Diese Signale werden vom Befehlsregister gestellt, und abhängig von deren Zustände erzeugt der Decodierer entsprechende Ausgangssignale. Während des Ladens eines neuen Befehls, ist es wichtig, daß diese Signale stabil bleiben, da

sonst ständig ein anderes Testdatenregister selektiert würde. Das Bypass- Register wird selektiert, wenn kein definierter Befehl vorliegt.

RT_I : Das RT_I- Signal, das vom TAP- Controller gestellt wird, dient dazu das RUNBIST1 und SEL_CLK- Signal zu generieren. Diese beiden Ausgangssignale dürfen nur im RUN- TEST/IDLE- Zustand aktiviert sein.

SHIFTDR : Das SHIFTDR- Signal wird benötigt das SHIFT_TDR- Signal zu erzeugen. (Siehe SHIFT_TDR- Beschreibung)

CLOCKDR : Der TAP- Controller sieht ein einziges Taktsignal für alle Testdatenregister vor. Der Demultiplexer ordnet dieses Taktsignal, abhängig von den Mux-Steuerleitungen, einem bestimmten Testdatenregister zu.

Die Ausgangssignale

INST1..INST2 : Diese Signale steuern den Multiplexer, der ein bestimmtes Testdatenregister zum TDO- Ausgang durchschaltet. Welches Register durchgeschaltet wird, hängt direkt von den Eingängen IR1..IR6 ab. Weiterhin werden diese Signale intern benötigt, den CLOCKDR- Takt zu verteilen.

Die Registerzuordnung :

INST2	INST1	Register
0	0	Bypass- Register
0	1	ID- Register
1	0	BS- Register
1	1	anw. -spezifisches Register

MODE_IN : Das MODE_IN- Signal kennzeichnet den internen Testbetrieb für die BS- Eingabezellen. Das Signal muß auf 1 gesetzt sein, sobald einer der Befehle (INTEST0..7 oder RUNBIST1..7) geladen ist. Die BS- Eingabezellen entkoppeln die Bausteinlogik von der Außenwelt.

MODE_EX : Das MODE_EX- Signal kennzeichnet den internen und externen Testbetrieb für die BS- Ausgabezellen. Das Signal muß auf 1 gesetzt sein sobald einer der Befehle EXTEST, INTEST0..7 oder RUNBIST1..7 geladen ist. Die BS-Ausgabezellen entkopplen die Bausteinlogik von der Außenwelt.

EXTEST : Das EXTEST- Signal kennzeichnet den EXTEST- Befehl und bezieht sich nur auf die BS- Ausgabezelle. Benötigt wird dieses Signal, damit die BS-Ausgabezelle in der Lage ist, den Signalzustand am Bausteinanschluß einzulesen.

SHIFT_TDR : Das SHIFT_TDR- Signal wird benötigt die, Scan- Design- Flipflops in den Schiebebetrieb zu schalten. Das SHIFTDR- Signal vom TAP- Controller darf nicht benutzt werden, da im Normalbetrieb der SAMPLE/PRELOAD- Befehl das SHIFTDR- Signal benutzt und nicht die Scan- Design- Flipflops beeinflussen darf.

RUNBIST1 : Dieses Signal gibt die Runbist- Schaltung im RUN- TEST/IDLE-Zustand frei.

CLOCKxxx : Jedes Testdatenregister wird mit einem eigenen Takt versehen, um eine gegenseitige Beeinflussung beim Schieben zu vermeiden.

SEL_CLK : Das SEL_CLK- Signal wird benutzt, den Systemtakt zu steuern. Benötigt wird für die Steuerung ein 2- zu- 1- Multiplexer, und der Systemtakt muß so angeschaltet sein, daß mit einer 1 am Steuereingang die Freigabe erfolgt. Am übrigen Multiplexereingang wird eines der Taktsignale CLOCKDR1.. CLOCKDR7 angeschaltet (Siehe Abbildung 4-6). Auf die benutzung des SEL_CLK- Signals kann verzichtet werden, falls ein Testgerät diese Steuerung übernimmt.

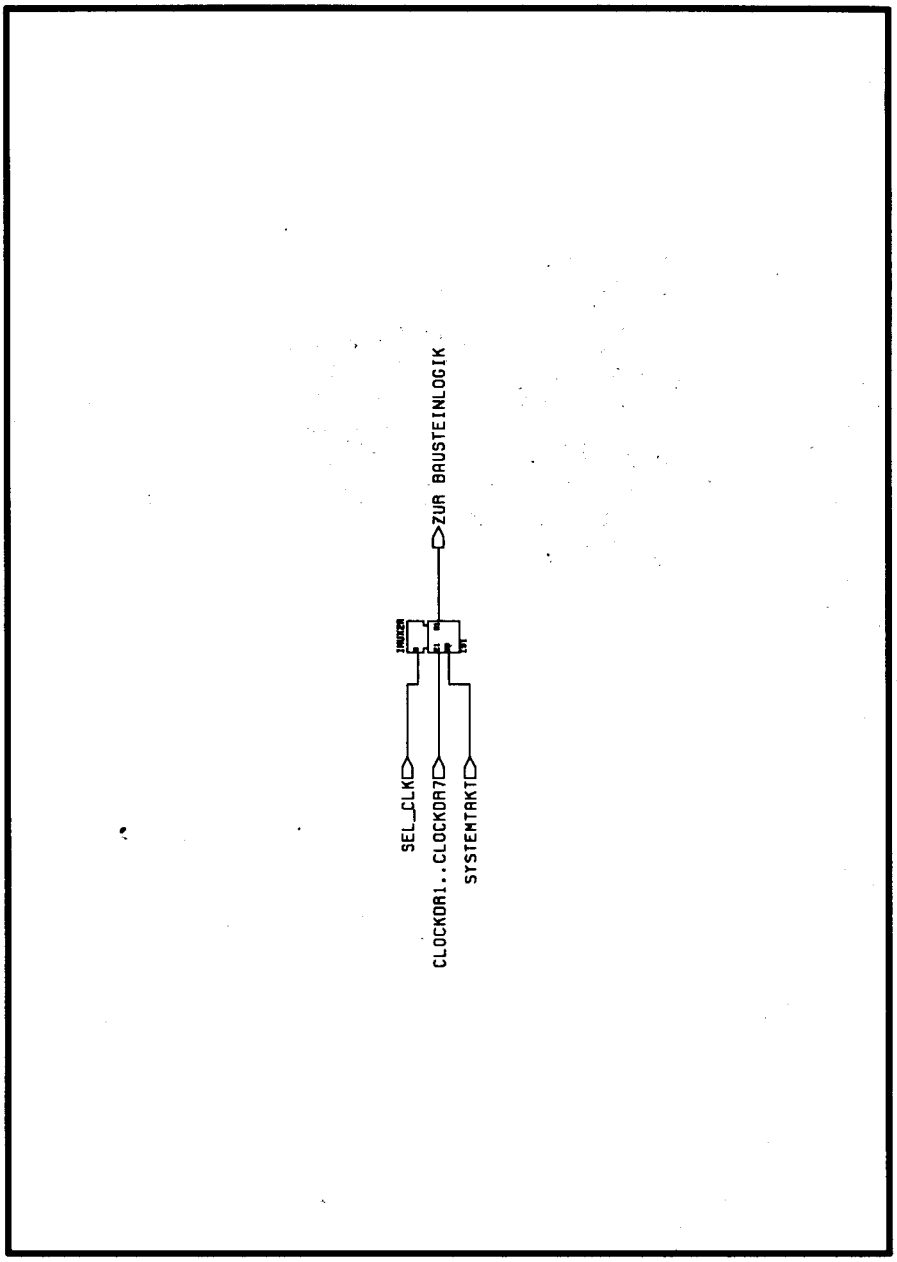


Figure 4-6 SEL_CLK- Anschaltung

Das interne SELECT_EN- Signal

Das SELECT_EN- Signal wird benötigt, den RUNBIST0- Befehl vom INTEST0, SAMPLE/PRELOAD0 und EXTEST- Befehl zu unterscheiden. Wichtig ist diese Unterscheidung für den CLOCKBS- Takt, der beim RUNBIST0- Befehl nicht aktiviert werden darf (unzulässiger Befehl).

4.3.2 Die CS_Logic2- & CS_Logic3- Schaltung

Diese beiden Schaltungen unterscheiden sich von der ersten Schaltung durch die Zahl der Register die jeweils selektiert werden können. Bei beiden Schaltungen liegt die gleiche Funktionsweise wie bei der CS_Logic1- Schaltung vor.

Die CS_Logic2- Schaltung kann 8 Register und 5 RBIST- Schaltungen und die CS_Logic3- Schaltung kann 10 Register und 7 RBIST- Schaltungen selektieren.

Selektierbare Register (CS_Logic2) :

1) Bypass- Register
2) ID- Register
3) Boundary- Scan- Register
4) 5 anw. -spezifische Register

Selektierbare Register (CS_Logic3) :

1) Bypass- Register
2) ID- Register
3) Boundary- Scan- Register
4) 7 anw. -spezifische Register

Bemerkung : Bei der CS_Logic1- und CS_Logic2- Schaltung beziehen sich die Befehle, für die das notwendige Anwendungsregister nicht vorgesehen ist, stetz auf das Anwendungsregister mit der höchsten Nummer. Erreicht wird mit dieser Festlegung, daß alle Befehle, unabhängig von der eingesetzten CS_Logic- Version, immer gültig sind und auch benutzt werden können.

Beispiel : CS_Logic1- Schaltung
Intest2..Intest7 --> anw. -spez. Register1

CS_Logic2- Schaltung
Intest6..Intest7 --> anw. -spez. Register5

4.4 Das Testdatenregister

Mit dem Begriff Testdatenregister sind die Register gemeint, die ausschließlich durch ein Befehl selektierbar sind. Vorgeschrieben sind mindestens 2 Register - Bypass und Boundary- Scan. Takt- und Steuersignale erhalten die Register von TAP- Controller und Decodierer. Je nach Schaltung können weitere anwendungsspezifische Register implementiert werden. Die Implementierung für die Gateforest 2.0 Familie sieht maximal 7 anwendungsspezifische Register vor. Die Zahl der Register wird beschränkt durch die Größe des Befehlsregisters und die Größe des Multiplexers, der eines der Testdatenregister zum TDO- Ausgang durchschaltet.

Die Register im Überblick

1) Bypass- Register
2) IDentifikations- Regsiter
3) Boundary- Scan- Register
4) 7 anwendungsspezifische Register

4.4.1 Das Bypass- Register (BYNTNB)

Das Bypass- Register stellt die kürzeste Verbindung zwischen den Anschlüssen TDI und TDO her. Benutzt wird dieses Register, irrelevante Daten schnell durchzuschieben. Bei sehr großen Schaltungen kann die Testdauer hierdurch erheblich verkürzt werden.

Spezifikationen :

- Das Register muß aus einer einzigen Registerzelle bestehen.
- Im CaptureDR- Zustand muß eine 0 ins Register geladen werden.
- Das Register darf nur durch den BYPASS- Befehl selektiert werden.

Vorgeschriebene Eingangssignale :

1) TDI --> Schiebeeingang des Registers
2) SHIFTDR --> aktiviert mit einer 1 den Schiebebetrieb des Registers
3) CLOCKBY --> synchrone Taktsteuerung

Vorgeschriebenes Ausgangssignal :

1) BY_OUT --> Schiebeausgang des Registers

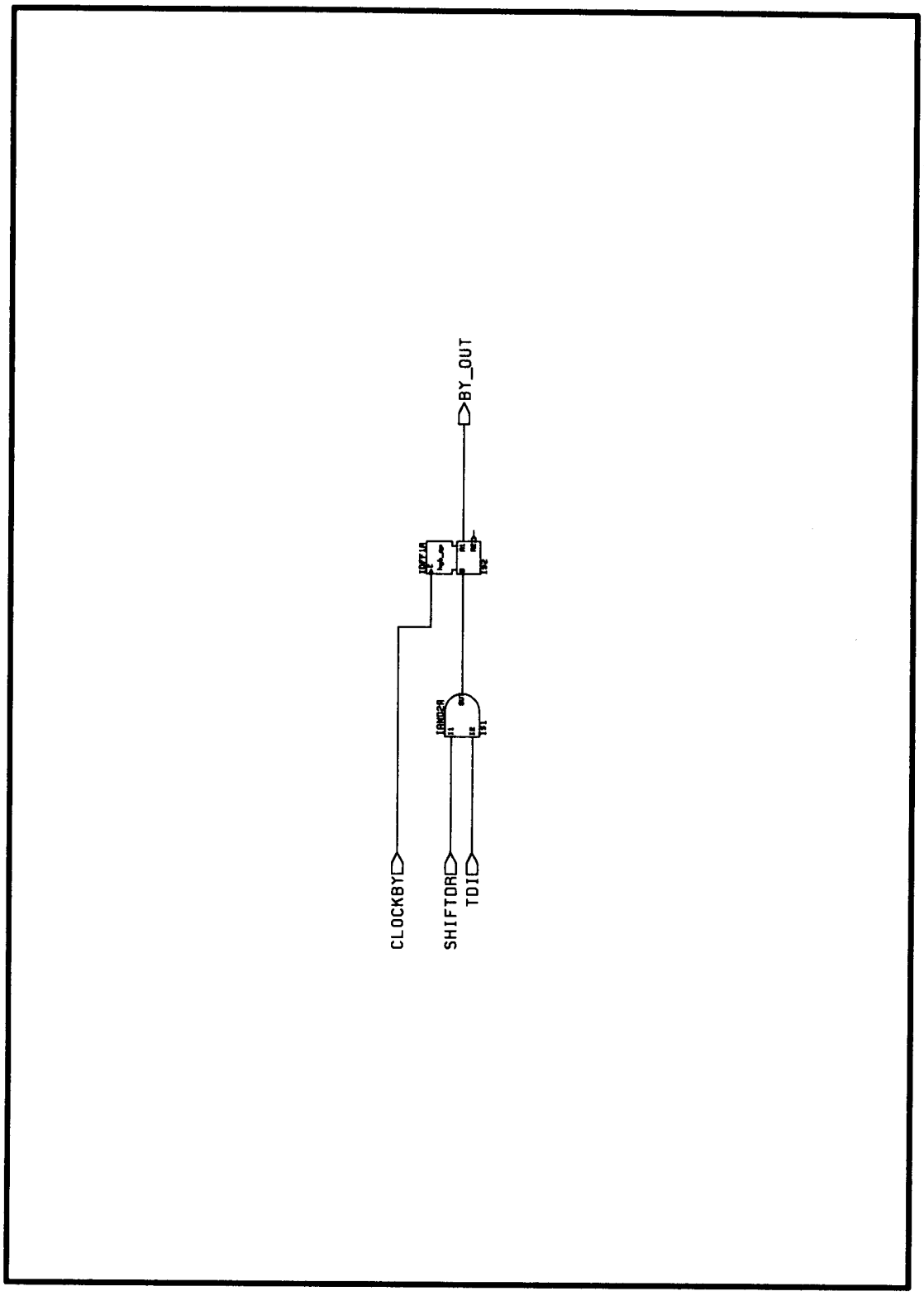


Figure 4-7 BYNTNB

4.4.2 Das Identifikations- Register (ID_REG)

Das Identifikation- Register (ID- Register) bietet dem Benutzer die Möglichkeit, den Aufbau einer Schaltung schnell herauszufinden. Neben der Herstellernummer stehen zusätzlich eine Bausteinnummer und eine Versionsnummer zur Verfügung. Diese bausteinspezifischen Daten können benützt werden, die entsprechende Testsoftware zu laden oder zu prüfen, ob die richtigen Bausteine eingesetzt sind.

Spezifikationen :

- *Das Register muß über Paralleleingänge verfügen und 32 Bit groß sein.*
- *Die Daten müssen mit der steigenden TCK- Flanke im CaptureDR- Zustand geladen werden.*
- *Die Bitfolge 00001111111 darf nicht als Herstellercode verwendet werden.*
- *Im Capture_DR- Zustand muß die niederwertigste Zelle mit einer 1 geladen werden.*

Vorgeschriebene Eingangssignale :

1) TDI --> serieller Eingang des Registers
2) SHIFTDR --> aktiviert mit einer 1 den Schiebebetrieb des Registers
3) CLOCKID --> synchrone Taktsteuerung
4) ID2..ID32 --> Paralleleingänge des Registers

Vorgeschriebenes Ausgangssignal :

1) ID_OUT --> serieller Ausgang des Registers

Registeraufbau

Die Registerlänge ist auf 32 Bit festgelegt. Das niederwertigste Bit ist fest auf 1 gelegt und kennzeichnet die Anwesenheit des Registers. Es stehen somit 31 Bits für die Daten zur Verfügung. Bits 1..11 kennzeichnen den Hersteller, Bits 12..27 den Bausteintyp und Bits 28..31 die Bausteinversion. Durch eine Restriktion, Hexcode 7F darf nicht im Bit- Bereich 1..11 benutzt werden, stehen insgesamt 2032 Herstellernummern, 65536 Bausteinnummern und 4 Versionsnummern zur Verfügung.

Den Aufbau einer Schaltung bestimmen (Siehe Abbildung 4-8)

Ausgegangen wird vom TEST- LOGIC- RESET- Zustand, der das ID- Register bzw. Bypass- Register selektiert. Im CAPTURE_DR- Zustand werden die Herstellerdaten ins ID- Register bzw. eine 0 ins Bypass- Register geladen. Im SHIFT_DR- Zustand werden die bausteinspezifischen Daten herausgeschoben. Als Endekennung wird die Bitfolge 00001111111 am TDI- Eingang des ersten Bausteines im Schiebepfad angelegt. Zu Beginn müssen solange Daten gelesen werden bis eine 1 auftritt. Eine 0 kennzeichnet die Anwesenheit eines Bypass- Registers, und der entsprechende Baustein verfügen nicht über ein ID- Register. Wird die erste 1 gelesen kennzeichnen die nachfolgenden 11 Bits den Hersteller, falls die Bitkombination 0000111111 nicht vorliegt. Sollte es nicht der Fall sein, können die nächsten 20 Bits, Bausteintyp und Versionsnummer, gelesen werden. Anschließend muß wieder solange gelesen werden bis die nächste 1 auftritt.

Sollte als Herstellernummer die Bitkombination 00001111111 auftreten, kann die Schiebeoption beendet werden (0000111111 = Endekennung).

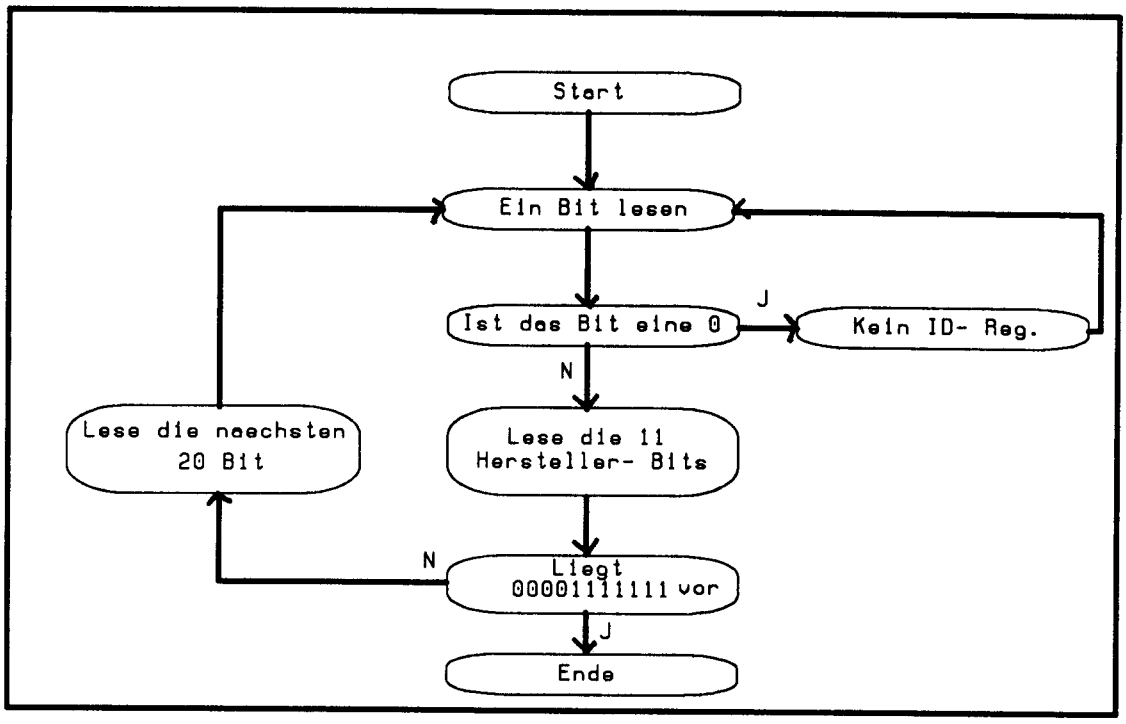


Figure 4-8 ID- Algorithmus

Beispiel :

Bit	Daten	Bedeutung
0	0	> Bypass- Register (Baustein 1)
1	1	> ID- Register (Baustein 2)
2	C	
.	O	Bits 2..32 Herstellercode, Bausteinnummer, Versionsnummer
.	D	
.	E	
32		
33	1	> ID- Register (Baustein 3)
34	C	
.	O	Bits 33..64 Herstellercode, Bausteinnummer, Versionsnummer
.	D	
.	E	
64		
65	0	> Bypass- Register (Baustein 4)
66	1	> ID_ Register (Baustein 5)
.	C	
.	O	Bits 65..98 Herstellercode, Bausteinnummer, Versionsnummer
.	D	
98	E	
99	1	> Endekennung 00001111111 folgt (Bits 99..111)
.	1	
.	.	
110	.	
111	0	

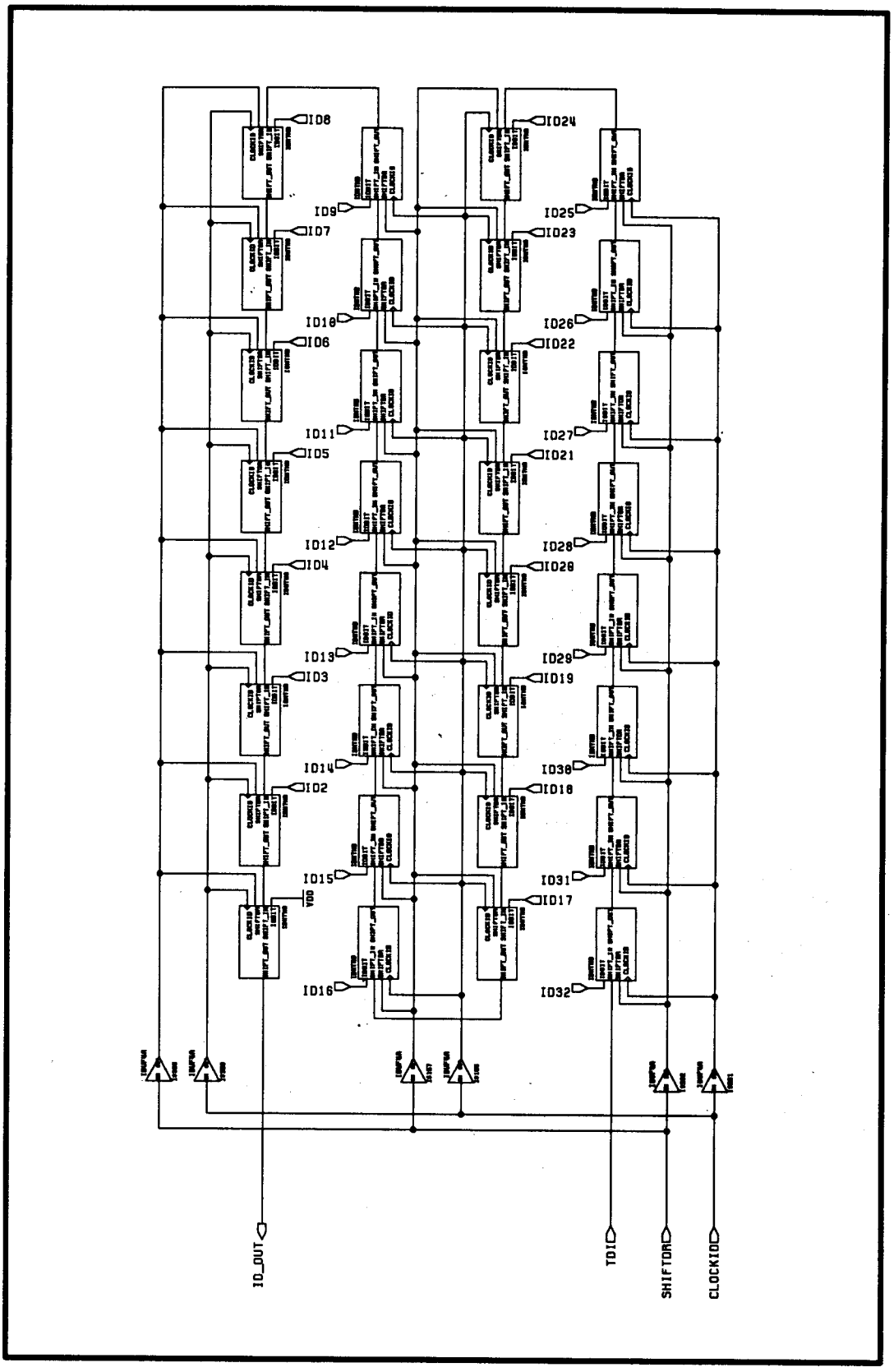


Figure 4-9 ID_REG

4.4.3 Das Boundary- Scan- Register (BS- Register)

Das Boundary Scan- Register ist das wichtigste Testdatenregister. Es ist das einzige Register, mit dem die Bausteinverbindungen geprüft werden können. Neben den Bausteinverbindungen läßt sich die Bausteinlogik auch gut prüfen. Eine weitere nützliche Anwendung, im Normalbetrieb, ist die Bestimmung von Signalzuständen zu einem definierten Zeitpunkt. Das Register setzt sich aus mehreren Zelltypen zusammen, die bestimmte Eigenschaften vorweisen müssen.

Spezifikationen :

- *Zwischen Bausteinlogik und jedem Bausteinanschluß muß eine BS- Zelle vorhanden sein.*
- *Bausteinanschlüsse, die hochohmig schaltbar sind, müssen über spezielle BS- Zellen verfügen, die einen Steuer- und einen Dateneingang vorweisen.*
- *Während neue Daten in das Register geschoben werden, müssen die Zustände auf den externen Verbindungsleitungen stabil bleiben.*
- *Die BS- Zellen an Takteingängen dürfen keine zusätzliche Verzögerung verursachen.*
- *Zwischen BS- Zelle und Bausteinanschluß dürfen keine Verstärker, Flipflops und Gatter geschaltet sein.*

Die Implementierung für die GateForest 2.0 Familie sieht 6 unterschiedliche BS-Zellen vor. Einige Vorschläge vom Standard wurden nicht übernommen, da sie erhebliche Mängel aufweisen.

Implementierte Eingabezellen :

1) BSINP1 --> BS- Zelle für Takteingänge
2) BSINP2 --> BS- Zelle für taktgesteuerte Eingänge
3) BSINP3 --> BS- Zelle für nicht taktgesteuerte Eingänge

Implementierte Ausgabezellen :

1) BSOUT1 --> BS- Zelle für normale Ausgänge (0/1)
2) BSOUT2 --> BS- Zelle für hochohmigschaltbare Ausgänge
3) BSOUT3 --> BS- Zelle für bidirektionale Ausgänge

Die 3 BS- Ausgabezellen sind mit Zusatzschaltungen versehen, die im Extest- Betrieb die Zellen gegen Kurzschlüsse schützt. Die Schutzschaltung ist nur wirksam, wenn ein spezieller Prüfalgorithmus durchlaufen wird. Alle BS- Ausgabezellen sind bereits mit einem Pad- Anschluß versehen, der verhindert, daß weitere Komponenten zwischen BS- Zelle und Bausteinanschluß geschaltet werden. Sollte es der Fall sein meldet der "Design Checker" an dieser Stelle einen Fehler.

4.4.3.1 Die BSINP1- Zelle

Die BSINP1- Zelle (Siehe Schaltung 4-10) ist die einfachste Zelle und wird für Takteingänge vorgesehen. Das Taktsignal wird nicht über Multiplexer und Flipflops weitergeschaltet, sondern nur abgegriffen, damit der Taktzustand zu jedem Zeitpunkt bestimmbar ist.

Eingangssignale :

1) CLOCKBS --> *synchrone Taktsteuerung (Decodierer)*
2) SHIFTDR --> *aktiviert mit einer 1 den Schiebebetrieb (TAP- Controller)*
3) SIGNAL_IN --> *Abgriff des Takts*
4) SHIFT_IN --> *Schiebeeingang der Zelle*

Ausgangssignal :

1) SHIFT_OUT --> *Schiebeausgang der Zelle*

4.4.3.2 Die BSINP2- Zelle

Die BSINP2- Zelle (Siehe Schaltung 4-11) ist für takt- oder flankengesteuerte Eingänge vorgesehen. Verglichen mit der BSINP1- Zelle wurde diese Zelle um einen weiteren Multiplexer ergänzt.

Eingangssignale :

1) MODE_IN --> *schaltet die Zelle zwischen Test- und Normalbetrieb um (Decodierer)*
2) SIGNAL_IN --> *Signalanschluß (Vom Pin)*
3) CLOCKBS --> *synchrone Taksteuerung (Decodierer)*
4) SHIFTDR --> *aktiviert mit einer 1 den Schiebebetrieb der Zelle (Decodierer)*
5) SHIFT_IN --> *Schiebeeingang der Zelle*

Ausgangssignale :

1) SIGNAL_OUT --> *Signalausgang*
2) SHIFT_OUT --> *Schiebeausgang der Zelle .*

4.4.3.3 Die BSINP3- Zelle

Die BSINP3- Zelle (Siehe Schaltung 4-12) verfügt über einen Schattenspeicher, der beim Laden neuer Daten ein stabiles Eingangssignal (SIGNAL_OUT) bereitstellt. Vorgesehen ist diese Zelle für Bausteineingänge, die sofort auf Signaländerungen reagieren. Das zusätzliche UPDATEDR- Signal wird benötigt, den Schattenspeicher zu laden. Das UPDATEDR- Signal wird nur im Update_DR- Zustand auf 1 gesetzt. Die restlichen Signale funktionieren wie bei der BSINP2- Zelle beschrieben.

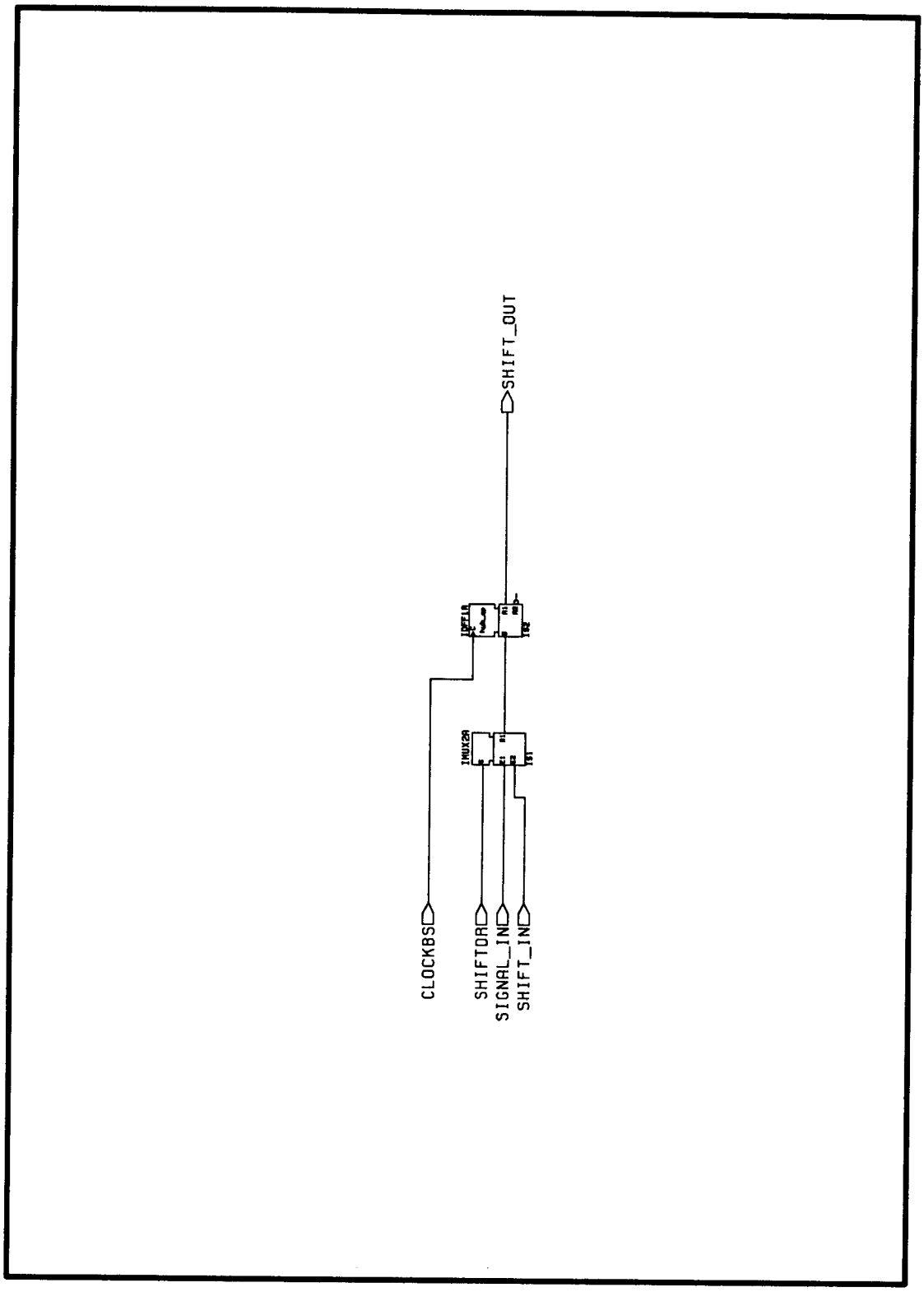


Figure 4-10 BSINP1

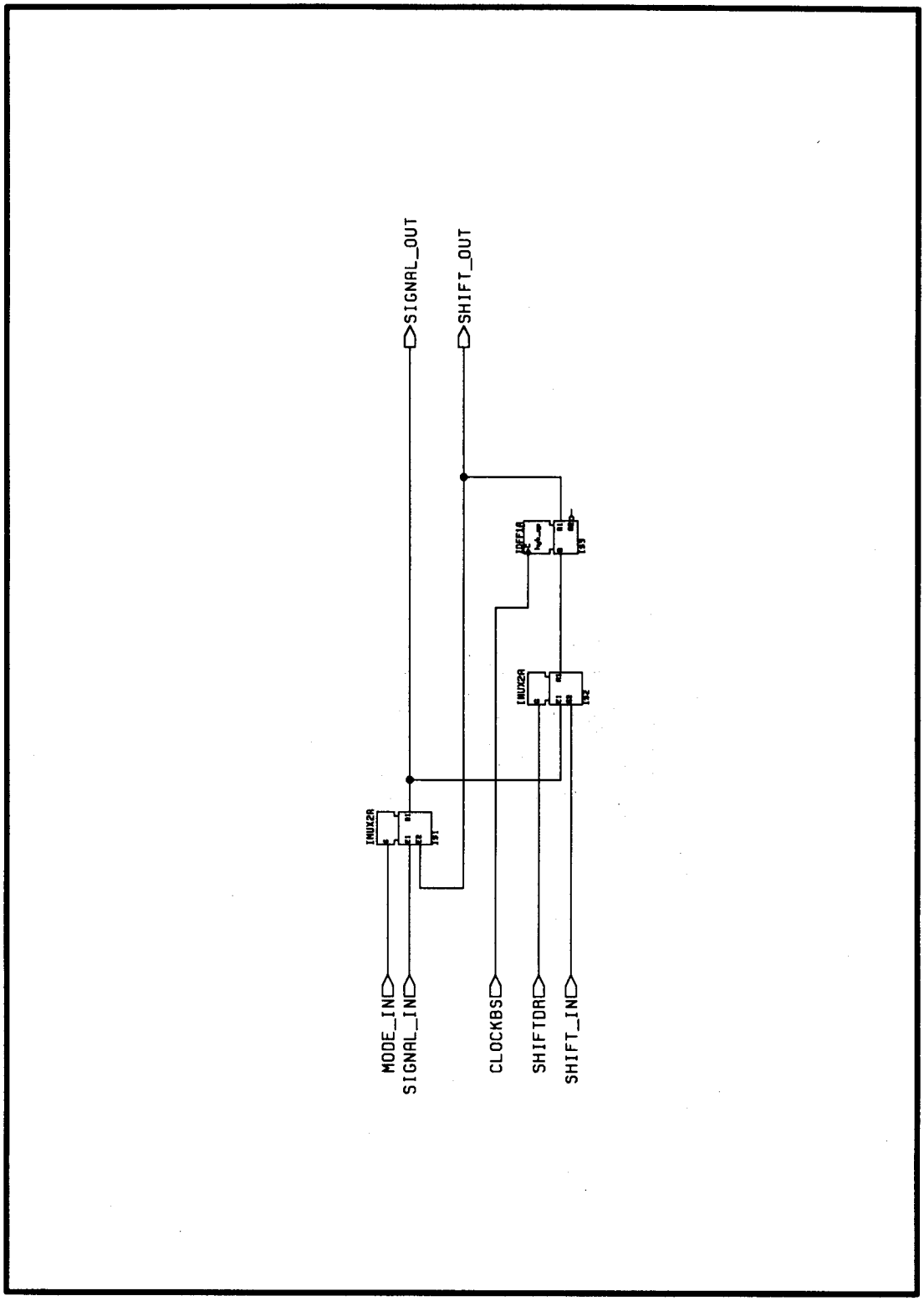


Figure 4-11 BSINP2

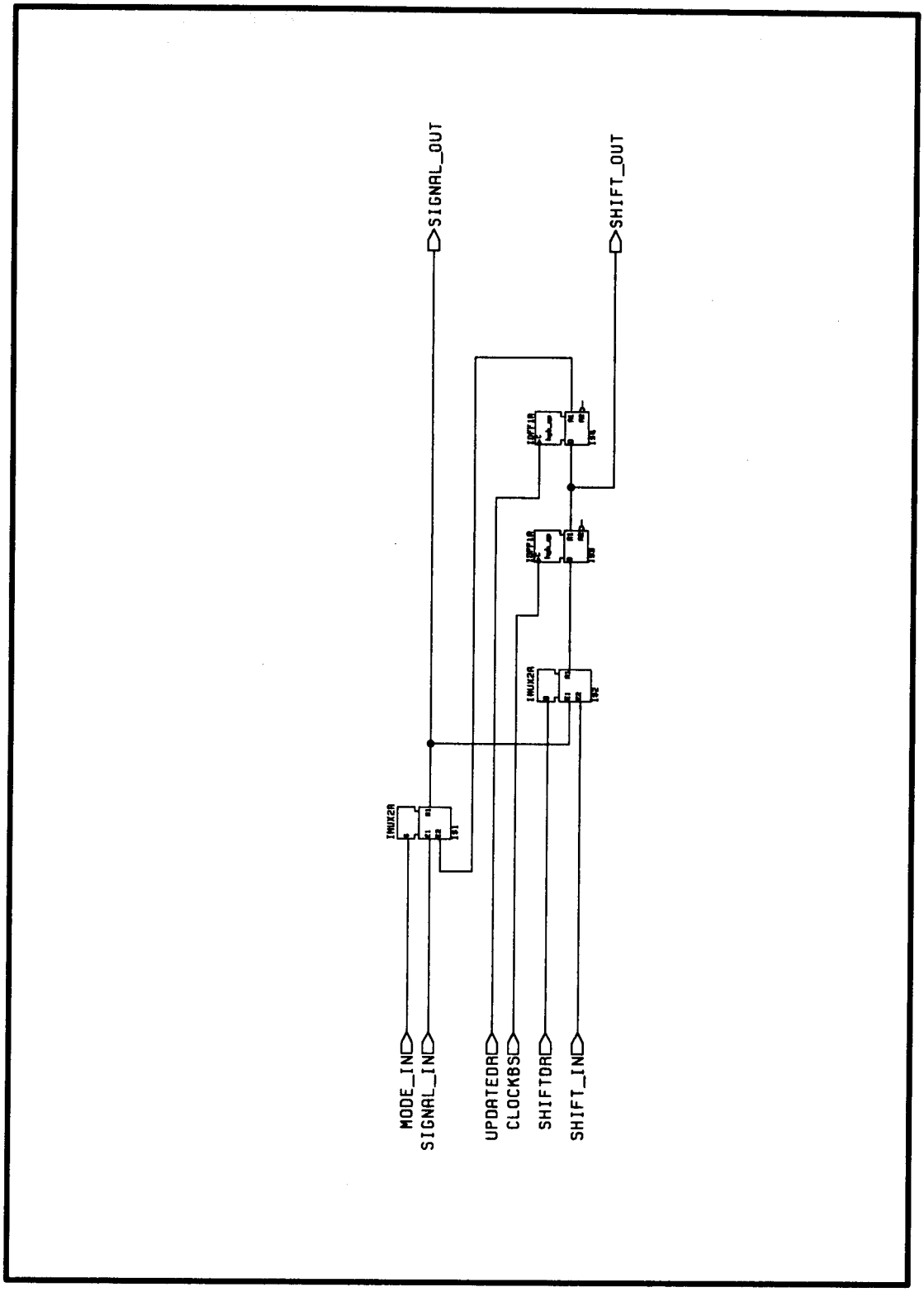


Figure 4-12 BSINP3

Die BS- Ausgabezellen

Keiner der Vorschläge aus der Literatur wurden realisiert, sondern spezielle Zellen entwickelt, die unempfindlich gegen Kurzschlüsse sind. Wichtig ist die Aktivierung der entsprechenden TAP- Zustände, damit der Schutzmechanismus wirkt.

Der Prüfvorgang

Folgende Beschreibung gilt für alle 3 BS- Ausgabezellen. Ausgegangen wird vom UPDATE_DR- Zustand, in dem die Schattenspeicher neu geladen werden. Die neuen Daten werden sofort nach außen geschaltet, und bei einem Kurzschluß fließt ein zu hoher Strom. Je länger dieser Zustand verbleibt, desto kritischer wird es für die Ausgabezelle. Eine Zerstörung der Zelle kann verhindert werden, indem der kritische Zeitabschnitt kurz gehalten wird. Aufgehoben wird der Kurzschluß, indem der negierte Schattenspeicherinhalt ausgegeben wird. Sehr große Schaltungen weisen den Nachteil auf, daß es zu lange dauert Kurzschlüsse zu lokalisieren und ein entsprechendes Testmuster zu laden.
Aus diesem Grunde sind die BS- Ausgabezellen mit bidirektionalen Pad- Zellen versehen, die es ermöglichen, Signalzustände an den Bausteinausgängen zu lesen. Tritt ein Kurzschluß mit Vdd oder Vcc auf wird immer der negierte Zustand des Schattenspeichers gelesen. Beendet wird der Kurzschluß, sobald der Schattenspeicher mit dem gelesenen Wert überschrieben wird. Hierzu müssen folgende TAP- Zustände aktiviert werden :

1) UPDATE_DR --> neuer Schattenspeicherinhalt wird ausgegeben
2) SELECT_DR- SCAN
3) CAPTURE_DR --> Zustand am Bausteinanschluß lesen
4) EXIT1_DR
5) UPDATE_DR --> Schattenspeicher überschreiben

4.4.3.4 Die BSOUT1- Zelle

Die BSOUT1- Zelle (Siehe Schaltung 4-13) wird für normale Bausteinausgänge vorgesehen.

Eingangssignale :

1) MODE_EX --> schaltet die Zelle zwischen Test- und Normalbetrieb um (Decodierer)
2) SIGNAL_IN --> Signaleingang (Bausteinlogik)
3) UPDATEDR --> Schattenspeicher wird neu geladen
4) CLOCKBS --> synchrone Taksteuerung (Decodierer)
5) SHIFTDR --> aktiviert mit einer 1 den Schiebebetrieb (Decodierer)
6) EXTEST --> eine 1 kennzeichnet den EXTEST- Betrieb (Decodierer)
7) SHIFT_IN --> Schiebeeingang der Zelle

Ausgangssignale :

1) SIGNAL_OUT --> Ausgang des durchgeschalteten Signals
2) SHIFT_OUT --> Schiebeausgang der Zelle

4.4.3.5 Die BSOUT2- Zelle

Diese Zelle (Siehe Schaltung 4-14) ist eine Erweiterung der BSOUT1- Zelle, und mit dem EN_IN- Signal wird der Ausgangstreiber freigegeben oder hochohmig geschaltet. Ein Bit benötigt zwei Takte, um vom SHIFT_IN- Eingang zum SHIFT_OUT- Ausgang zu gelangen. Diese Zelle funktioniert genau wie die BSOUT1- Zelle, mit dem Unterschied, daß der Ausgangstreiber vom EN_IN- Signal oder dem Schattenspeicher (I$5) gesteuert wird. Die Bausteinlogik muß das EN_IN- Signal erzeugen. (2 state output)

4.4.3.6 Die BSOUT3- Zelle

Die BSOUT3- Zelle (Siehe Schaltung 4-15) ist für bidirektionale Ausgänge vorgesehen. Der SYS_INP- Anschluß stellt die Verbindung zwischen BS- Zelle und Bausteinlogik her. Mit dem MODE_IN- Eingang wird zwischen Normalbetrieb und Intest- Betrieb geschaltet. Im Intest- Betrieb wird der Schattenspeicherinhalt (I$9) am SYS_INP- Ausgang bereitgestellt ansonsten der SIGNAL_OUT- Ausgang/Eingang. Vom SHIFT_IN- Eingang zum SHIFT_OUT- Ausgang benötigt ein Bit 2 Takte.

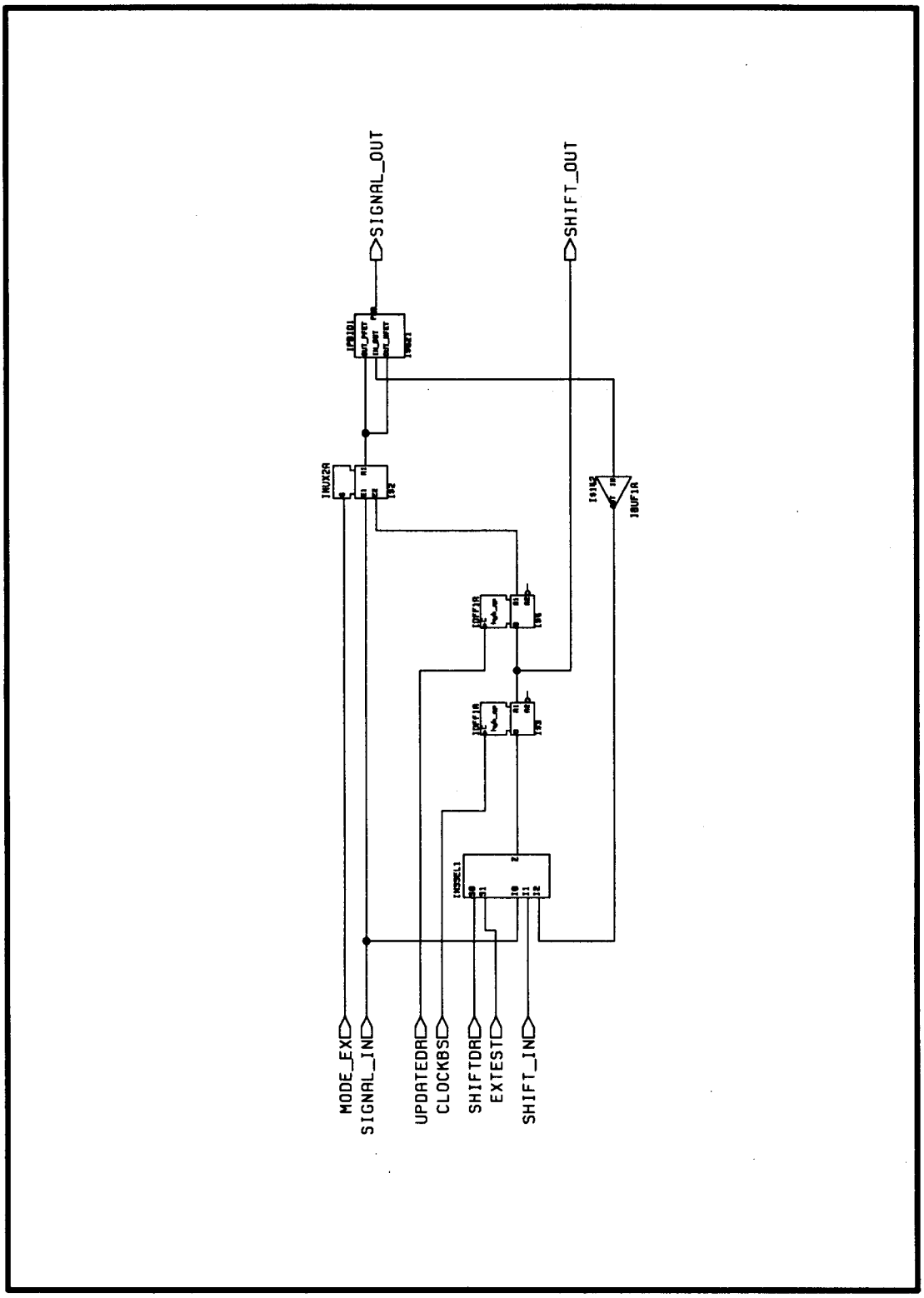


Figure 4-13 BSOUT1

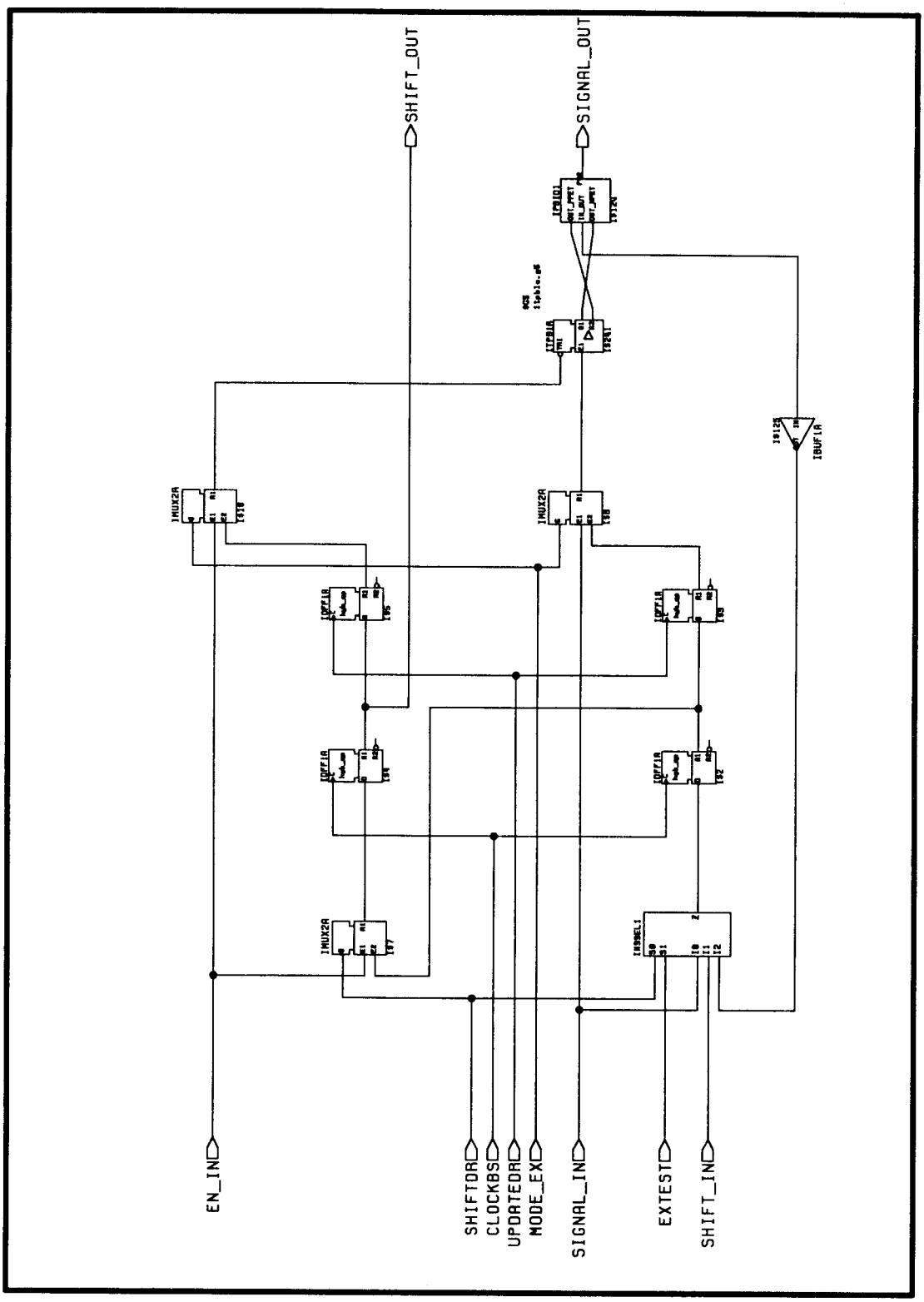


Figure 4-14 BSOUT2

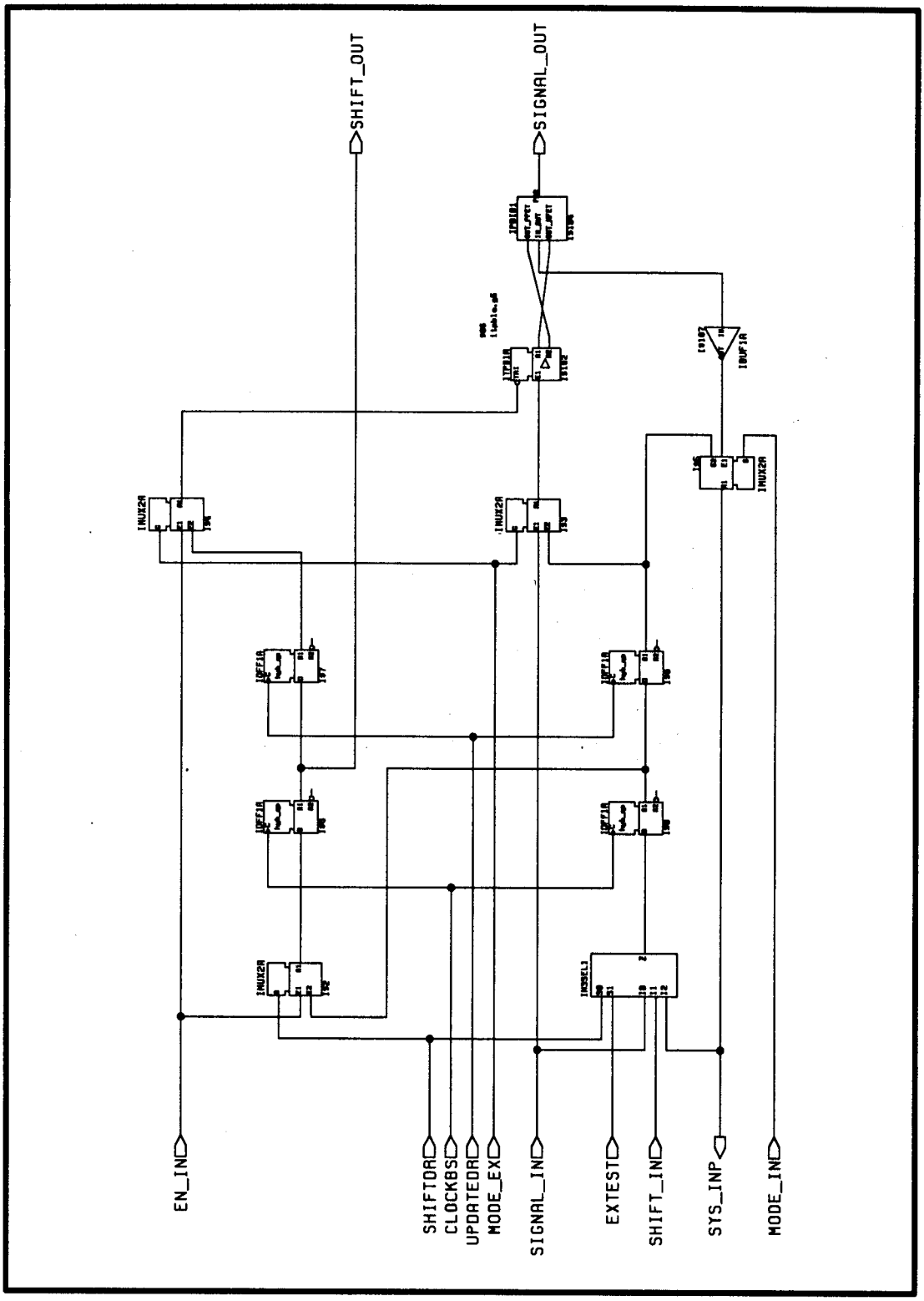


Figure 4-15 BSOUT3

5. Die Boundary- Scan- Module

Die Vielzahl der Testlogik- Einheiten verursachen, bei der Ergänzung einer Schaltung mit der BS- Architektur, einen erheblichen Verdrahtungsaufwand. Das Ziel ist es mit einem modularen Aufbau diesen Aufwand stark zu reduzieren und dem Benutzer eine gute Übersicht zu verschaffen. Jede Schaltung sollte, ohne eine ausführliche Einarbeitung in der BS- Architektur, ergänzt werden können. TAP- Controller, Befehlsregister, Decodierer, Bypass- Register, Runbist- Schaltung und der TDO- Ausgang sind die Komponenten die jede BS- Implementierung vorweisen muß. Aus diesem Grunde sind die erwähnten Einheiten bereits zu Modulen zusammengefaßt, und dem Benutzer stehen insgesamt 6 Module zur Verfügung.

Die Module unterscheiden sich in der Zahl der anschließbaren Testdaten- Register und Selbsttestschaltungen. Weiterhin unterscheiden sich die Module vom benutzten Befehlsregister.(IR_BY oder IR_ID)

Folgende Module sind vorhanden :

1) BS4BY --> *Bef. Bypass wird geladen, max. 4 Testdatenregister*
2) BS4ID --> *Bef. Idcode wird geladen, max. 4 Testdatenregister*
3) BS8BY --> *Bef. Bypass wird geladen, max. 8 Testdatenregister*
4) BS8ID --> *Bef. Idcode wird geladen, max. 8 Testdatenregister*
5) BS16BY --> *Bef. Bypass wird geladen, max. 10 Testdatenregister*
6) BS16ID --> *Bef. Idcode wird geladen, max. 10 Testdatenregister*

Die Module BS16BY und BS16ID könnten insgesamt 16 Testdatenregister verwalten, aber die Befehlscodierung erlaubt maximal 10 Register. (7 anwendungsspezifische Register, das Bypass- Register,das ID- Register und das BS- Register)

Die Benennung der Module :

BS --> *Boundary- Scan*
4,8,16 --> *Multiplexergröße*
ID --> *die Befehlsregisterversion IR_ID ist implementiert*
BY --> *die Befehlsregisterversion IR_BY ist implementiert*

Signaleingänge :

1) TCK --> *Taktfrequenz für die Testlogik*
2) TMS --> *Steuereingang für den TAP- Controller*
3) TRST --> *Reset- Eingang*
4) TDI --> *ser. Dateneingang*
5) CLK --> *Systemtakt*
6) ID_IN --> *Anschluß für das Befehlsregister*
7) BS_IN --> *Anschluß für das BS- Register*
8) DR1..7_IN --> *Anschlüsse für die anw. Spezifische Register*

Signalausgänge :

1) TDO --> *ser. Datenausgang*
2) SHIFTDR --> *Schiebesignal für die Testdatenregister*
3) UPDATEDR --> *Übernahmesignal für die Testdatenregister*

4) SEL_CLK --> *Steuersignal für den Systemtakt*
5) SHIFT_TDR --> *Schiebesignal für die anw. -spezifischen Register*
6) EXTEST --> *Steuersignal für die BS- Ausgabezellen*
7) MODE_EX --> *Steuersignal für die BS- Zellen*
8) MODE_IN --> *Steuersignal für die BS- Zellen*
9) CLOCKBS --> *Taktsignal für das BS- Register*
10) CLOCKID --> *Taktsignal für das ID- Register*
11) CLOCKDR1..7 --> *Taktsignale für die anw. -spezifischen Register*

Bemerkung : Am TDO- Ausgang dürfen keine weitere Komponenten angeschlossen werden, da bereits eine Pad- Zelle integriert ist.

Neben den sechs Modulen stehen dem Benutzer 6 BS- Zellen und das ID- Register zur Verfügung. Abbildung 5-1 zeigt den Aufbau eines Moduls.

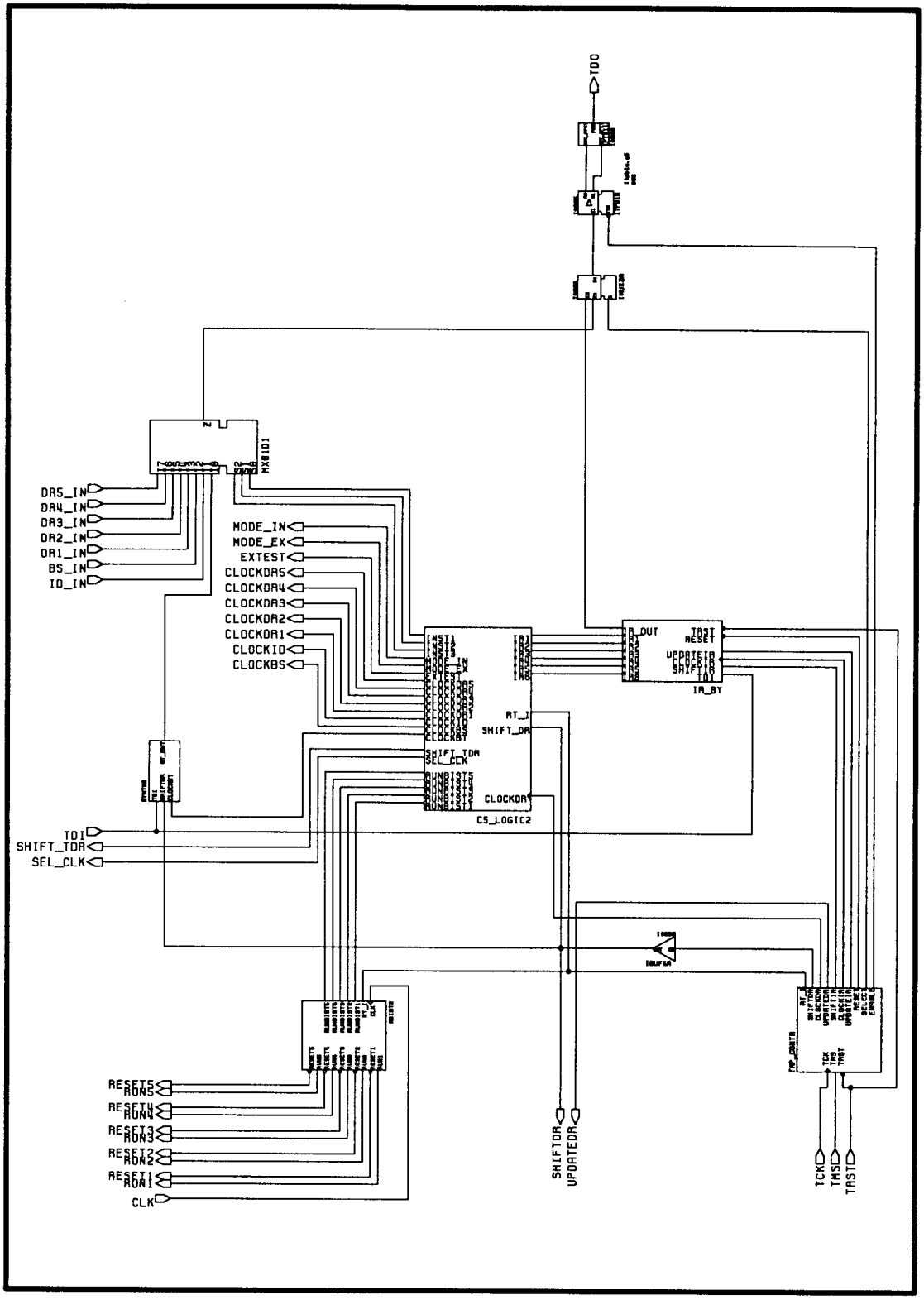


Figure 5-1 BS8BY

6. Das BS- Verfahren verglichen mit dem In- Circuit & Funktionstest

Im Gegensatz zum In- Circuit- Test benötigt das Boundary- Scan- Verfahren sehr wenige Anschlüsse. Damit ergibt sich die Frage : " Wie effektiv kann geprüft werden ?". Ein kritischer Faktor, des Boundary- Scan- Verfahrens, ist die Zeit, die benötigt wird, Testergebnisse zu lesen und neue Testdaten zu laden. Je mehr Bausteine sich auf einer Platine befinden, umso ungünstiger wird das Zeitverhalten in Bezug auf dynamische Schaltungen. Ein weiteres Maß für die Effektivität ist die Größe der benötigten Datenströme, eine Schaltung vollständig zu prüfen. Sehr komplexe Bausteine werden auch mit dem Boundary- Scan- Verfahren nie vollkommen testbar sein.

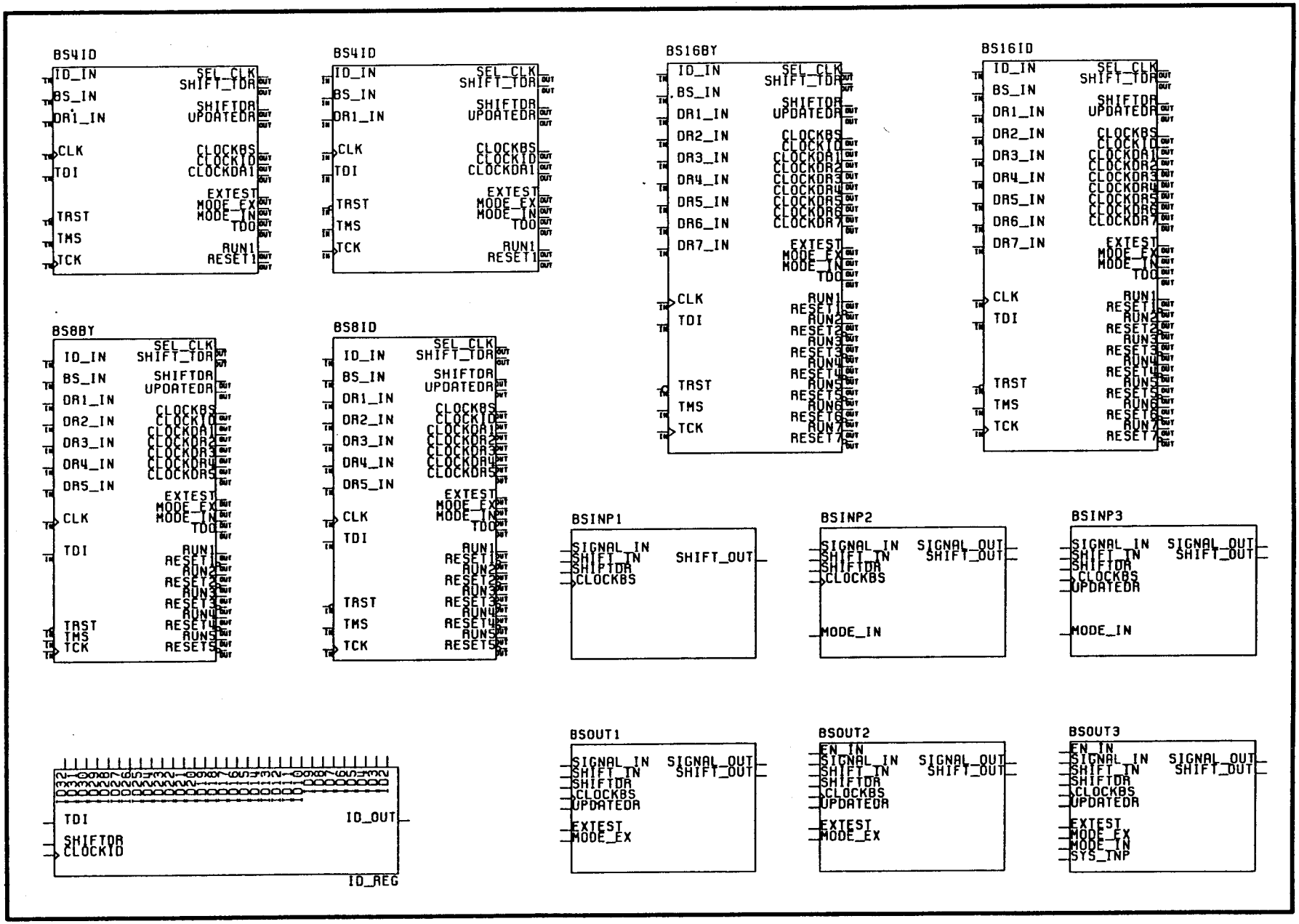
Ein großer Vorteil des Boundary- Scan- Verfahrens ist der sehr geringe Hardwareaufwand während der Testphase. Verglichen mit dem In- Circuit- Test wird an dieser Stelle ein Vielfaches der Zeit zurückgewonnen, die beim Entwurf der Bausteine länger investiert wurde. Betrachtet man die BS- Zellen als virtuelle Testnadeln, so stehen bei sehr großen Schaltungen häufig mehr Testpunkte zur Verfügung als ein Nagelbrett bieten kann. Jeder Bausteinanschluß der Schaltung ist ein Testpunkt und als Resultat können alle Bausteinverbindungen geprüft werden, egal welche Herstellungstechnologie verwendet wird. Der In- Circuit- Test kann diese Eigenschaft, wegen den technischen Grenzen, nicht immer bieten. Bausteine, die dynamische Schaltungen enthalten, können mit dem Boundary- Scan- Verfahren nur unzureichend geprüft werden. Ursache ist die serielle Abarbeitung der Testdaten, was sehr viel Zeit beansprucht. Abhilfe bringt ein integrierter Selbsttest, der die Startdaten von den BS-Zellen bekommt und das Endresultat ein vorgesehenes Anwendungsregister übergibt. Komplexe Bausteine, die über einen Selbsttest verfügen, beschleunigen den Testablauf, und die benötigten Datenströme werden kleiner.

Die Mächtigkeit des Boundary- Scan- Verfahrens läßt den In- Circuit Test überflüssig wirken, aber der Funktionstest muß weiterhin verwendet werden, um Fehler bei sehr hohen Betriebsgeschwindigkeiten zu lokalisieren.

7. Literaturverzeichnis

1) The Test Access Port and Boundary- Scan Architecture.
Colin M. Maunder and Rodham E. Tullos,
IEEE Computer Society Press Tutorial.

2) IEEE Standard Test Access Port and Boundary- Scan Architecture.
Institute of Electrical and Electronics Engineers.


BS4ID
ID_IN
BS_IN
DR1_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
BS4ID
ID_IN
BS_IN
DR1_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
BS16BY
ID_IN
BS_IN
DR1_IN
DR2_IN
DR3_IN
DR4_IN
DR5_IN
DR6_IN
DR7_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
CLOCKDR2
CLOCKDR3
CLOCKDR4
CLOCKDR5
CLOCKDR6
CLOCKDR7
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
RUN2
RESET2
RUN3
RESET3
RUN4
RESET4
RUN5
RESET5
RUN6
RESET6
RUN7
RESET7
BS16ID
ID_IN
BS_IN
DR1_IN
DR2_IN
DR3_IN
DR4_IN
DR5_IN
DR6_IN
DR7_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
CLOCKDR2
CLOCKDR3
CLOCKDR4
CLOCKDR5
CLOCKDR6
CLOCKDR7
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
RUN2
RESET2
RUN3
RESET3
RUN4
RESET4
RUN5
RESET5
RUN6
RESET6
RUN7
RESET7
BS8BY
ID_IN
BS_IN
DR1_IN
DR2_IN
DR3_IN
DR4_IN
DR5_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
CLOCKDR2
CLOCKDR3
CLOCKDR4
CLOCKDR5
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
RUN2
RESET2
RUN3
RESET3
RUN4
RESET4
RUN5
RESET5
BS8ID
ID_IN
BS_IN
DR1_IN
DR2_IN
DR3_IN
DR4_IN
DR5_IN
CLK
TDI
TRST
TMS
TCK
SEL_CLK
SHIFT_TDR
SHIFTDR
UPDATEDR
CLOCKBS
CLOCKID
CLOCKDR1
CLOCKDR2
CLOCKDR3
CLOCKDR4
CLOCKDR5
EXTEST
MODE_EX
MODE_IN
TDO
RUN1
RESET1
RUN2
RESET2
RUN3
RESET3
RUN4
RESET4
RUN5
RESET5
BSINP1
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
SHIFT_OUT
BSINP2
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
MODE_IN
SIGNAL_OUT
SHIFT_OUT
BSINP3
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
UPDATEDR
MODE_IN
SIGNAL_OUT
SHIFT_OUT
BSOUT1
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
UPDATEDR
EXTEST
MODE_EX
SIGNAL_OUT
SHIFT_OUT
BSOUT2
EN_IN
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
UPDATEDR
EXTEST
MODE_EX
SIGNAL_OUT
SHIFT_OUT
BSOUT3
EN_IN
SIGNAL_IN
SHIFT_IN
SHIFTDR
CLOCKBS
UPDATEDR
EXTEST
MODE_EX
MODE_IN
SYS_INP
SIGNAL_OUT
SHIFT_OUT
TDI
SHIFTDR
CLOCKID
ID_OUT
ID_REG