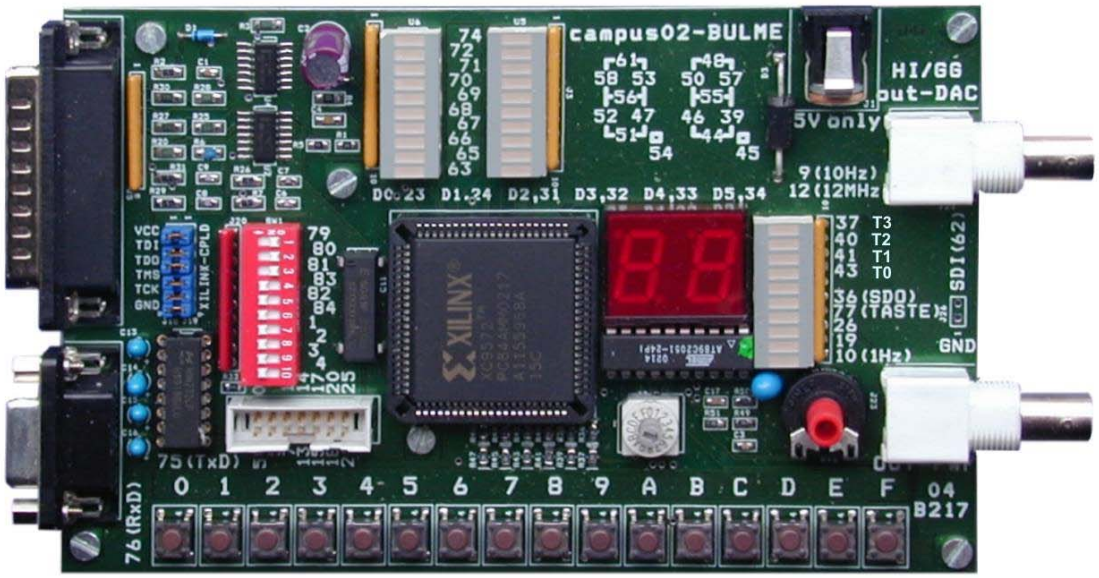


CPLD - Übungsboard für XC9572/108

Entwurf mit ABEL / Webpack 6.x, Dokumentation V1.0

B.Geiger, 13.02.2005

INHALT

Korrekturen, Fragen und Anregungen bitte an D.I. Bertram Geiger mailto:gg@bulme.at

1 Webpack 6.3

Hinweis:

Einige Bildschirmaufnahmen stammen noch von der Version 6.1, sollten sich aber nur unwesentlich von der Version 6.3 unterscheiden

1.1 Software - Installation

Derzeit (12 / 2004) aktuelle Version: Webpack 6.3i
Die Entwicklungssoftware erhalten Sie entweder mit dem Übungsboard oder kostenlos in der jeweils aktuellen Fassung von der Xilinx Homepage www.xilinx.com unter:
Products and Services / ISE Design Tools / ISE Webpack ...

ISE *WebPACK* Download Module *(service pack required - see information below)*	Download Size	PLD Design Environment	CPLD Device Support	FPGA Device Support
Complete ISE WebPACK Software - includes programming tools	207 Mb*	✔	✔	✔
Complete CPLD Tool Set - includes programming tools	130 Mb*	✔	✔	
Complete Programming Tools	45 Mb*		✔	✔

* Customers installing the above *WebPACK* modules also need to download and install the latest Service Pack. The latest service pack can be found here.

ModelSim XE Download Module	Download Size	MXE Simulator	CPLD VHDL Library	CPLD Verilog Library	FPGA VHDL Library	FPGA Verilog Library
Complete MXE Simulator	97 Mb	✔	✔	✔	✔	✔

Bild 1: Webpack Module

Für den Zugang zum Downloadbereich ist eine kostenlose Registrierung erforderlich.
Zum Arbeiten mit dem CPLD Board werden folgende Softwaremodule benötigt:

- „Complete CPLD Tool Set“ (130 MB) ... unbedingt erforderlich, keine zusätzliche Lizensierung
- „Complete MXE Simulator“ (97 MB) ... nur für Simulation benötigt. Diese Software erfordert eine zusätzliche (kostenlose) Lizensierung (*node locked*).

1.2 Neues Projekt anlegen

Hinweis: Bei `File Name` und `Module Name` DOS Konventionen einhalten !

Nach dem Start File / New Project aufrufen und Name sowie Speicherort angeben

weiter> New Source

Property Name	Value
Device Family	XC9500 CPLDs
Device	xc9572
Package	pc84
Speed Grade	-15
Top-Level Module Type	HDL
Synthesis Tool	XST (ABEL)
Simulator	Other
Generated Simulation Language	VHDL

Bild 2: Property einstellen bei neuem Projekt

weiter>

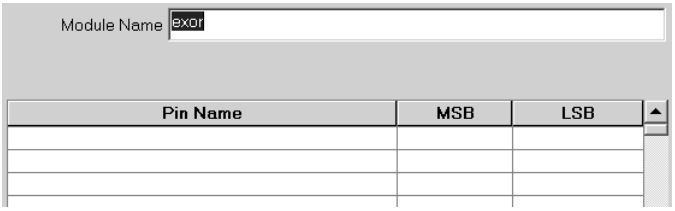


Bild 3: Hier könnte man die Pinnummern vergeben

weiter> Fertig stellen> weiter> weiter>

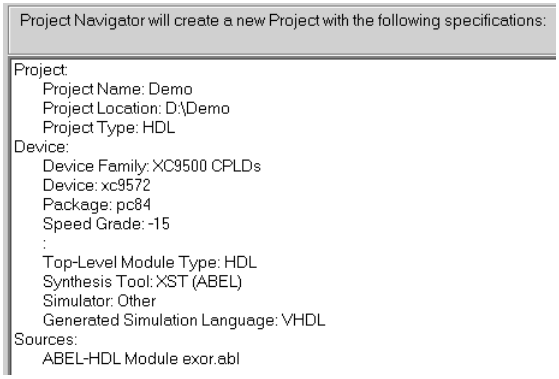


Bild 4: Zusammenfassung der Projektkonfiguration

Fertig stellen>

Damit öffnet sich wieder der Projekt Navigator, in dem alle weiteren Arbeiten stattfinden

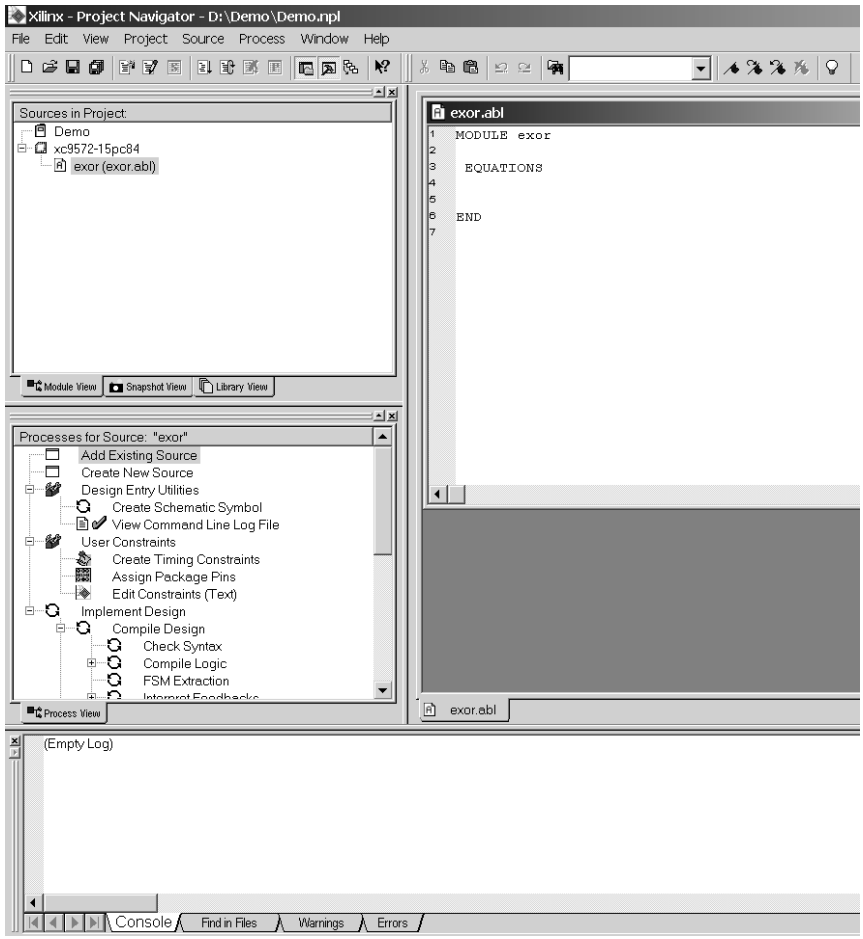


Bild 5: Die Standard - Arbeitsoberfläche: Projektnavigator

1.3 Source erstellen

Im Editorfeld wird nun der ABEL Sourcefile erstellt und gespeichert.
Als Beispielprogramm dient hier ein einfaches EXOR.
Die beiden Eingänge kommen vom Tastaturdecoder, der Ausgangszustand wird am obersten LED (L1) der LED Reihe angezeigt:

Taste	x0	x1	y
0	0	0	0
1	1	0	1
2	0	1	1
3	1	1	0

```
1  MODULE exor
2
3  DECLARATIONS
4
5  x0 PIN 43;
6  x1 PIN 41;
7  y  PIN 74 ISTYPE'com';
8
9  EQUATIONS
10
11 y = x0 $ x1;
12
13 END
14
```

Bild 6: Sourcecode für das Demoprogramm (EXOR)

1.4 Kompilieren

Als nächstes wird der Sourcecode übersetzt. Bei syntaktisch fehlerfreiem Code schaut das Kompiler Listing gleich aus wie der Sourcecode. Ansonsten werden hier allfällige Fehler (mit oft begrenzter Aussagekraft) angezeigt. Es ist empfehlenswert, nur die ersten ein bis zwei Fehler zu beachten sowie auszubessern und dann neu zu kompilieren, da die weiteren Fehlermeldungen meist unbrauchbar sind.

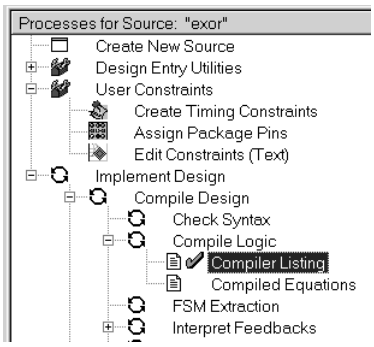


Bild 7: Kompiler starten

```
1  0002 |
2  0003 |DECLARATIONS
3  0004 |
4  0005 |x0 PIN 43;
5  0006 |x1 PIN 41;
6  0007 |y  PIN 74 ISTYPE'com';
7  0008 |
8  0009 |EQUATIONS
9  0010 |
10 0011 |y = x0 $ x1;
11 0012 |
12 0013 |END
13
```

Bild 8: Fehlerfreies Ergebnis

1.5 Fitter

Der Fitter realisiert die zuvor übersetzte Logik in der konkret ausgewählten Hardware. Der Fitter Report beschreibt das Ergebnis, er kann als das Datenblatt des neu konfigurierten Bausteins angesehen werden. Der Fitter meldet Fehler wie: zu wenig Platz oder Wahl falscher Anschlusspins

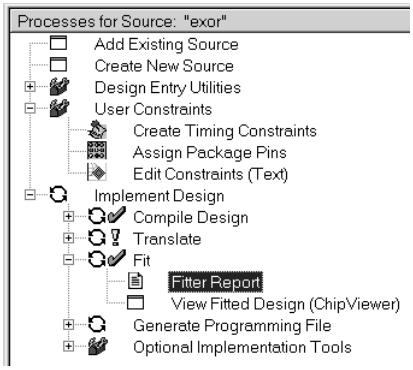


Bild 9: Fitter Report anfordern

1.6 Download

Die Schritte „Kompilieren“ und „Fitter“ müssen nicht separat aufgerufen werden, das diente hier nur der Anschaulichkeit. Man kann nach der Erstellung / Änderung des Sourcecode direkt die Konfiguration starten. dabei werden automatisch alle nötigen Zwischenschritte nachgeholt.

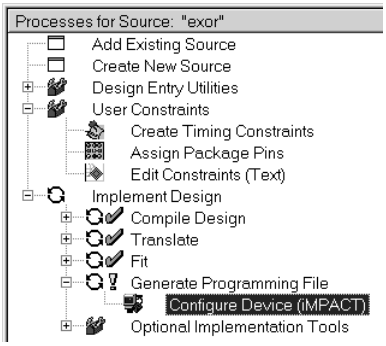
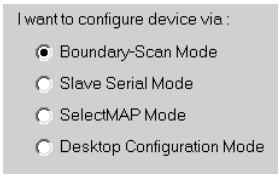


Bild 10: Konfiguration starten



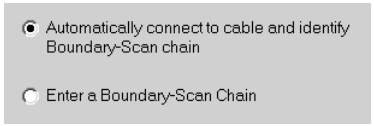


Bild 11: Betriebsartenabfrage (Default Einstellung)

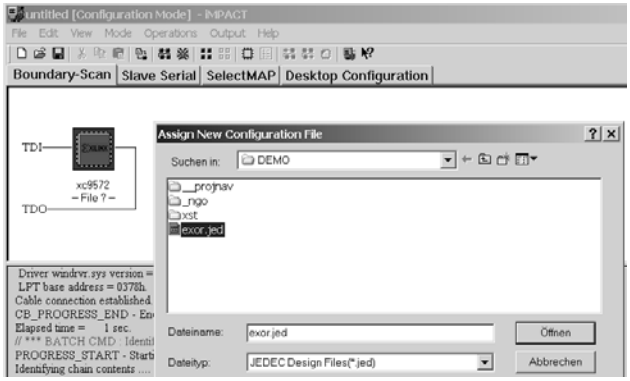
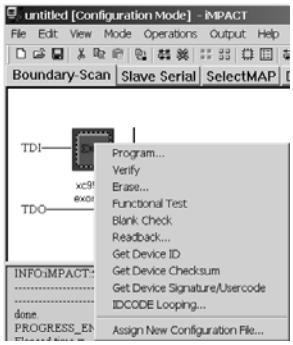
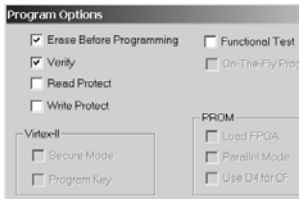
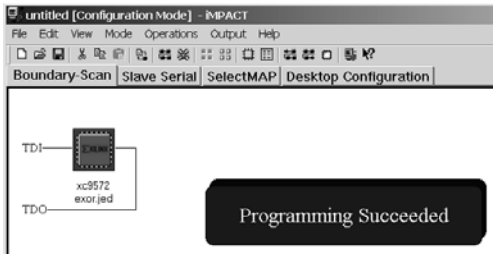


Bild 12: JEDEC Datei auswählen







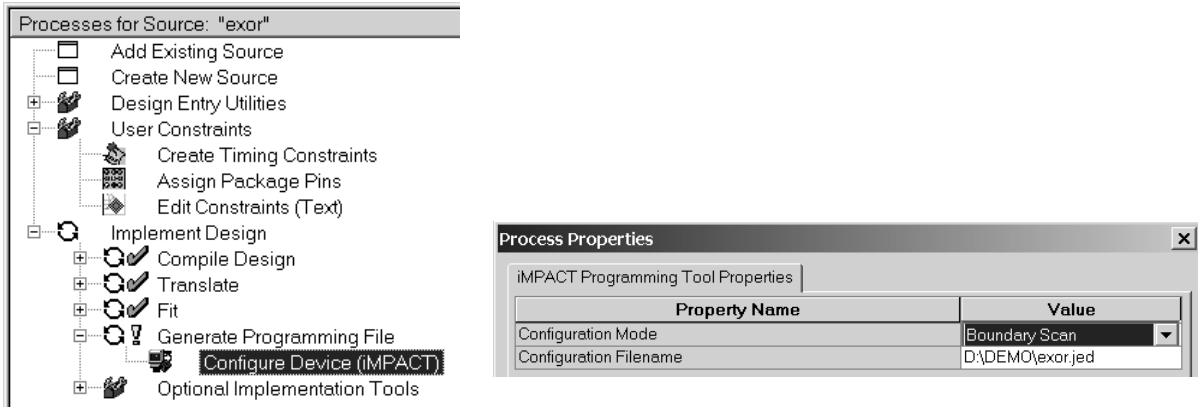
Programmierstart Letzte Optionen (Default) Fertig !

Bild 13: „Brennen“

Und jetzt sollte das EXOR funktionieren !

```
Device #1 selected
// *** BATCH CMD : Program -p 1 -e -v
PROGRESS_START - Starting Operation.
Validating chain...
Boundary-scan chain validated successfully.
'1': Putting device in ISP mode...
done
'1': Erasing device...
done.
'1': Erasure completed successfully.
'1': Putting device in ISP mode...
done
'1': Programming device...
done.
'1': Putting device in ISP mode...
done
'1': Verifying device...
done.
'1': Verification completed successfully.
'1': Programming completed successfully.
PROGRESS_END - End Operation.
Elapsed time =     14 sec.
```

Bild 14: Hier kann man den Programmiervorgang mitverfolgen (ev. Fehlermeldungen)



auf *Configure* zeigt „*Properties*“ *Properties* einstellen erspart die Startabfragen

Bild 15: Property für die Konfiguration vorgeben

2 ABEL -Sprachbeschreibung

Allgemeiner Aufbau des ABEL-HDL Sourcefiles (*filename.*ABL)

- Kopf
  ```
  MODULE modulname
  TITLE 'Ueberschrift'
  ```
  ... Optional
- Zuweisungen
  ```
  DECLARATIONS
  ....;
  ```
- Logische Beschreibungen
  ```
  EQUATIONS,  TRUTH_TABLE,  STATE_DIAGRAM
  ....;
  ```
- Endanweisung
  ```
  END
  ```

2.1 Syntax

2.1.1 Schreibweise und Kommentare

- Kommentare:

WICHTIG: **Umlaute und ß auch in Kommentaren bzw. TITLE vermeiden !!!**

Bei Nichtbeachtung "kryptische", d.h. nicht deutbare Fehlermeldungen (z.B. ... File corrupted)

`"Kommentar oder  //Kommentar` ... nur einmal pro Zeile
`"Kommentar"` ... kann beliebig eingefügt werden

- Symbolische Namen:

Das 1. Zeichen muss ein Buchstabe oder sein, restliche Zeichen auch Zahlen oder "_"
Es wird zwischen Groß- und Kleinschreibung unterschieden - daher am besten immer Kleinschreibung !
Modulname und Filename sollten nicht zusätzlich als symbolische Namen verwendet werden.
Die folgenden Bezeichnungen sind reserviert und dürfen nicht verwendet werden (führt zu nicht deutbaren Fehlermeldungen !):

async_reset	flag (obsolete)	property
attributes	fuses	state
case	goto	state_diagram
declarations	if	state_register
device	in (obsolete)	sync_reset
directive	istype	test_vectors
else	library	then
enable (obsolete)	macro	title
end	module	trace
endcase	node	truth_table
endwith	options	when
equations	pin	with

Tabelle 1: Verbotene Bezeichnungen für Operanden

2.1.2 I/O Zuweisungen, nähere Signaldefinitionen

Äußere Anschlüsse eines Bauteils oder eines Makros werden mit `PIN` bezeichnet, innere Punkte (Knoten, *buried logic*) mit `NODE`

Eingänge: *pinname* PIN;
pinname PIN *pinnummer*;

Ausgänge: *pinname* PIN ISTYPE '*type*';
pinname PIN *pinnummer* ISTYPE '*type*';

Knoten: *nodename* NODE ISTYPE '*type*';

Die Festlegung der PIN - Nummer kann:
- dem System überlassen werden
- in einem Constraint File festgelegt werden
- im Sourcecode festgelegt werden

Letztere Vorgangsweise ist im Übungsbetrieb aus Gründen der Übersichtlichkeit sehr empfehlenswert. Das Plazieren und Routen bzw. das Fitting wird bei den XC9500 Bausteinen dadurch nicht beeinträchtigt.

```
DECLARATIONS
in0 PIN 3;                          "Eingang 0
in1 PIN 4;                          "Eingang 1
in2 PIN;                            "Eingang 2
in2,in1,in2 PIN 3,4,5;              "Eingänge 0 bis 2
in0,in1,in2 PIN;                    "Eingänge 0 bis 2
in7..in0 PIN 1,2,3,4,5,6,7,8;       "8 Eingänge (in7 bis in0)
aus5 PIN 12 ISTYPE 'COM';           "Ausgang aus5 (kombinatorisch)
q3 PIN ISTYPE 'REG';                "Ausgang q3, Registerausgang
q15..q0 NODE ISTYPE 'REG';          "16 interne Flip Flop
```

Bei den Ausgängen kann man bei Registerausgängen zur Vermeidung von Zweideutigkeiten die Konfiguration der Ausgangsmakrozelle angeben (Attribute). Ohne nähere Angabe wählt das Optimierungsprogramm z.B. `ISTYPE 'BUFFER'` bzw. `'INVERT'` wie es für die Optimierung am günstigsten ist (De Morgan). Die eigentliche Logikfunktion bleibt dadurch zwar unbeeinflusst, beim Reset eines Registers liegt dann allerdings der entsprechende Ausgangspin nicht wie erwartet auf "0" !
Anders ausgedrückt: Bei `ISTYPE 'REG,BUFFER'` wird das Register immer direkt, d.h. ohne dazwischen geschalteten Inverter zum Pin geführt.

```
out4  PIN 9  ISTYPE  'REG';         "Registerausgang
out4  PIN 9  ISTYPE  'REG,BUFFER';  "Registerausgang, erzwingt 0 bei Reset
```

●	'BUFFER'	nicht invertierend
●	'INVERT'	invertierend
●	'COM'	kombinatorisch
●	'REG'	Register (D-FF)
	'RETAIN'	Signal darf nicht logisch vereinfacht werden >>ABEL6<<
	'NEG'	Unbestimmte Logik ist "1"
	'POS'	Unbestimmte Logik ist "0" ... default
	'DC'	Unbestimmte Logik ist ".X."

Tabelle 2: Attribute (● ... architekturunabhängig)

Symbolische Zuweisungen:

Zuweisungen können auch an Zwischengrößen erfolgen, um Ausdrücke zu vereinfachen. Dies dient nur der übersichtlicheren Darstellung, hat jedoch keinen Einfluss auf die Logik bzw. deren Realisierung.
Die nachstehenden Beispiele ergeben bei **gleicher logischer Funktion** jeweils eine unterschiedliche Realisierung, die sich auch im Ressourcenverbrauch und in der Laufzeit auswirkt:

```
DECLARATIONS
a,b,c PIN;
aus PIN ISTYPE 'COM';
zwi = a & b;

EQUATIONS
ausg = zwi & c;
```

```
DECLARATIONS
a,b,c PIN;
zwi PIN ISTYPE 'COM';
aus PIN ISTYPE 'COM';

EQUATIONS
zwi = a & b;
ausg = zwi & c;
```

```
DECLARATIONS
a,b,c PIN;
zwi NODE ISTYPE 'COM';
aus PIN ISTYPE 'COM';

EQUATIONS
zwi = a & b;
ausg = zwi & c;
```

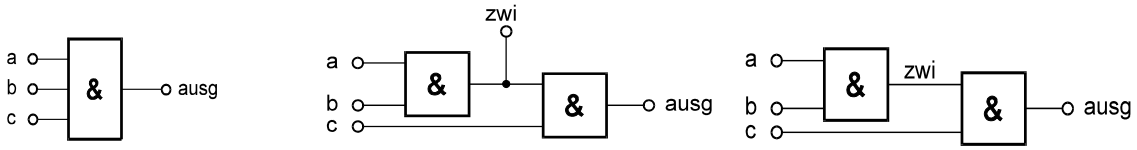


Bild 16: Beispiele für Zuweisungen

2.1.3 Operatoren

	Operator	Beispiel	Beschreibung	Priorität
Logische:				
	!		NOT (1-er Komplement)	1
	&		AND	2
	#		OR	3
	$		EXOR	3
	!$		EXNOR	3
Arithmetische:				
	+	a+b	Addition	3
	-	-a	Negation (2-er Komplement)	1
	-	a-b	Subtraktion (a+(-b))	3
Vergleichende:				
	==		gleich	4
	!=		ungleich (unsigned)	4
	<		kleiner "	4
	<=		kleiner gleich "	4
	>		größer "	4
	>=		größer gleich "	4
Zuweisungen:				
	=		Kombinatorisch	
	:=		Speichernd	

Tabelle 3: Operatoren

2.1.4 Operanden

Signale:
Einzelnes Signal mit dem logischen Pegel 0 / 1, Low / High, False / True

Datenformate:
`123` ... Dezimal 123 (Default)
`^h123` ... Hex 123
`^b101` ... Binär 101
'a' ... ASCII a (Hex 41)

Spezielle Konstanten:

`.X.` Don't care

Sets:
Signale können zu Sets (= Array oder Vektor) zusammengefasst werden

Interne Darstellung und Interpretation der verschiedenen Datentypen als Set mit 8 Bit Länge:

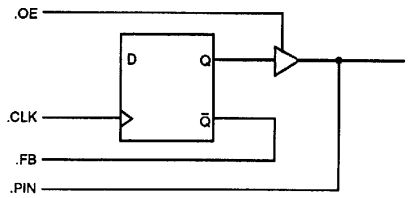
Signal 0	→	...0000	0	0	0	0	0	0	0	0	
Signal 1	→	...1111	1	1	1	1	1	1	1	1	!
Zahl 0	→	...0000	0	0	0	0	0	0	0	0	Auffüllen mit 0
Zahl 1	→	...0000	0	0	0	0	0	0	0	1	! Auffüllen mit 0
Zahl -1	→	...1111	1	1	1	1	1	1	1	1	2-er Komplement
Zahl 5	→	...0000	0	0	0	0	0	1	0	1	Auffüllen mit 0
Zahl -5	→	...1111	1	1	1	1	1	0	1	1	2-er Komplement
Zahl 255	→	...0000	1	1	1	1	1	1	1	1	
Zahl 256	→	...0001	0	0	0	0	0	0	0	0	Abschneiden
Zahl 259	→	...0001	0	0	0	0	0	0	1	1	Abschneiden

Tabelle 4: interne Darstellung von Sets

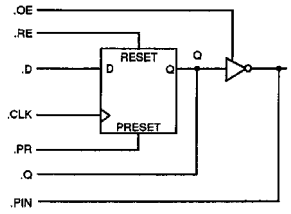
Hinweis:
Wird einem Signal eine Zahl zugeordnet, so wird nur das LSB ausgewertet und die höherwertigen Stellen abgeschnitten (grau hinterlegte Felder)

Punkterweiterungen (*Dot Extensions*):

Diese erlauben eine nähere Beschreibung von Signalen. Architekturunabhängige Beschreibungen (*pin to pin*) sind vorzuziehen, da sie leichter übertragbar sind. Zur Anpassung an spezielle Bausteinarchitekturen und zur Vermeidung von Mehrdeutigkeiten können jedoch detaillierte Beschreibungen erforderlich sein.



Architekturunabhängige Punkterweiterungen



Beispiel: Detaillierte Punkterweiterungen

Bild 17: Punkterweiterungen, D-FF

Kombinatorisch:				
	●	.OE	Output-enable	
	●	.PIN	Pin-Rückführung	
	●	.COM	Kombinatorische Rückführung	ABEL 6

(D)-Flipflop:				
	●	.CLK	Takt (flankengetriggert)	
	●	.FB	Register-Rückführung	
	●	.PIN	Pin-Rückführung	
	●	.OE	Output-enable	
	●	.CLR	Synchronous Clear	ABEL 6
	●	.ACLR	Asynchronous Clear	ABEL 6
	●	.SET	Synchronous Set	ABEL 6
	●	.ASET	Asynchronous Set	ABEL 6

		.D	D-Eingang des FF	

		.AR	Asynchroner Reset	
		.AP	Asynchroner Preset	
		.SR	Synchroner Reset	
		.SP	Synchroner Preset	
		.RE	Reset (Asynchron/Synchron)	
		.PR	Preset (Asynchron/Synchron)	

Tabelle 5: Punkterweiterungen (● ... architekturunabhängig)

2.1.5 Sets und Setoperationen

Mehrere Operanden können auch zu einem Set zusammengefasst werden, was die weitere Verarbeitung meist übersichtlicher gestaltet und bedeutend vereinfacht.
In der abgekürzten Darstellung müssen Signalnamen mit Buchstaben beginnen und für die Indizierung mit einer ein- oder zweistelligen Zahl enden.

```
adresse=[a7,a6,a5,a4,a3,a2,a1,a0];  "ausfuehrliche Darstellung
adresse=[a7..a0];                   "abgekuerzte Darstellung
ausgang=[q3,q2,z5,z6];
eingang=[0,0,e2,e1];                "Platzhalter zur Laengenanpassung
xadr=[ad7..ad3,.x.,.x.,.x.]         "teilweise abgekuerzte Darstellung und
                                    "Platzhalter zur Laengenanpassung
```

Mit Sets kann ähnlich wie mit normalen Operanden gearbeitet werden.
Dabei werden die einzelnen Operatoren immer wie folgt bewertet: [MSB, ... , LSB].

> Sets müssen immer gleiche Länge haben !

Um dies zu erreichen, müssen nicht signifikante Stellen entsprechend belegt werden .

Da grundsätzlich nur **Sets mit Sets** verarbeitet werden können, werden andere Datentypen zuvor intern in Sets umgewandelt.

> SETOPERATIONEN
>
> **verknüpfen** Signale der Sets Bitweise miteinander
> oder
> **vergleichen** die als vorzeichenlose Binärzahlen interpretierten Sets

Beispiele

```
DECLARATIONS
a2..a0 PIN ISTYPE 'COM';
b2..b0 PIN ISTYPE 'COM';
c4..c0 PIN ISTYPE 'COM';
y PIN ISTYPE 'COM';
A=[a2,a1,a0];
B=[b2,b1,b0];
C=[c4..c0];
```

Set	Signale	Set	Signale
A=3	A=[0,1,1]	!A	[!a2, !a1, !a0]
A=9	A=[0,0,1]	A.OE	[a2.OE, a1.OE, a0.OE]
B=6	B=[1,1,0]	A&y	[a2&y, a1&y, a0&y]
A=16	A=[0,0,0]	A&B	[a2&b2, a1&b1, a0&b0]

```
A=-1    [1,1,1]          A=7     [1,1,1]
B=3     [0,1,1]          B=15    [1,1,1]
A<B     ... Falsch !!!   A==B    ... Richtig !!!
```

```
C=A+B  ... Sets ungleich lang !
Abhilfe:
AA=[0,0,a2,a1,a0]
BB=[0,0,b2,b1,b0]
C=AA+BB
```

```
E=[1,0,1,1]
F=[1,0,0,1]
G=[1,0,.X.,1]
G==E   ... Richtig !
G==F   ... Richtig !
```

In Zuweisungen bei
`IF-THEN-ELSE, WHEN-THEN-ELSE, CASE-ENDCASE`
Set-Konstanten immer explizit angeben z.B. `[1,0,0,1]`
- nicht als Zahl z.B. dezimal `9`

Anstelle der Konstanten `0/1` können selbstverständlich auch symbolische Namen verwendet werden.

Beispiel:

```
DECLARATIONS
ausgang=[a2,a1,a0]
EQUATIONS
WHEN start THEN ausgang=1 ELSE ausgang=0;
```

Abel formt den Ausdruck wie folgt um:

```
ausgang = start &  1      #    !start & 0

         Signal & Zahl
            ↓      ↓
         Signal & Signal
         \_____  ______/
               \/
            Signal
```

→ Falsches Ergebnis: `ausgang=[start,start,start]`

```
DECLARATIONS
ausgang=[a2,a1,a0]
EQUATIONS
WHEN start THEN ausgang=[0,0,1] ELSE ausgang=0;
```

Abel formt den Ausdruck wie folgt um:

```
ausgang = start & [0,0,1]     #    !start & 0

         Signal & Set
            ↓     ↓
           Set & Set
           \___  ___/
               \/
             Set
```

→ Richtiges Ergebnis: `ausgang=[0,0,start]`

2.2 Logikentwurf

Die folgend beschriebenen Logikbeschreibungen können gemischt und auch mehrfach innerhalb eines Entwurfs angewandt werden.

2.2.1 Gleichungen

Überschrift: `EQUATIONS`

`EQUATIONS` bezieht sich immer auf Signale (`PIN` oder `NODE`), nicht auf Zwischengrößen (Symbolische Zuweisungen) - diese stehen unter `DECLARATIONS` !

- Aufstellen entsprechender Zuweisungen: `=` bei kombinatorischer Logik
  `:=` bei Registern

```
z.B.:  y = a&b;    "UND Verknüpfung
       q0 := !q0;  "T-Flipflop
```

- `WHEN` *`bedingung`* `THEN` *`zuweisung`* `ELSE` *`zuweisung`*

```
z.B.:  WHEN a==b THEN y=1 ELSE  y=0;            "1 Bit Komparator
       WHEN enable THEN q0:=!qo ELSE q0:=q0;    "T-Flipflop mit Enable
```

Verketten von `WHEN-THEN-ELSE:`

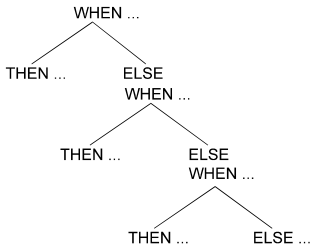


Bild 18: Verketten der `WHEN-THEN-ELSE` Anweisungen

Bei verketteten (*chained*) `WHEN-THEN-ELSE` Gleichungen ist von selbst sichergestellt, dass genau eine Bedingung erfüllt ist !

```
z.B.:  WHEN reset THEN q0:=0 ELSE
       WHEN enable THEN q0:=!qo ELSE q0:=q0;
```

Wird das `ELSE` weggelassen, so können keine, eine oder mehrere Bedingungen erfüllt sein:

```
WHEN bedingung THEN zuweisung;
WHEN bedingung THEN zuweisung;
...
WHEN bedingung THEN zuweisung;
```

- **Mehrfachzuweisungen** (*multiple assignments to the same identifier*):
Wird für ein Signal mehr als eine Zuweisung durchgeführt, so werden die einzelnen Zuweisungen zuerst getrennt berechnet und dann miteinander ODER-verknüpft !

- **Blöcke** (*equation blocks*): eine einzelne Zuweisung kann durch Klammerung durch weitere Gleichungen erweitert werden

zuweisung	{*zuweisung*;	{
	zuweisung;}	*zuweisung*;
		zuweisung;
		}

2.2.2 Wahrheitstabelle

Innerhalb eines Entwurfs können auch mehrere, voneinander unabhängige Wahrheitstabellen definiert werden, jeweils mit eigener Überschrift (TRUTH_TABLE)
Die Signale einer Wahrheitstabelle müssen naturgemäß immer Konstanten sein (`0, 1, .X.`), für die jedoch auch symbolische Namen stehen können !
Die Signale können als Einzelsignale oder als Sets dargestellt werden.

- Kombinatorisch:

```
TRUTH_TABLE
(eingangssignal -> ausgangssignal)
```

- Sequentiell:

```
TRUTH_TABLE
(eingangssignal :> ausgangssignal)
```

- Gemischt:

```
TRUTH_TABLE
(eingangssignal -> ausgangssignal :> ausgangssignal)
```

- Unvollständig bestimmte Wahrheitstabellen:

Eine Wahrheitstabelle muss nicht vollständig ausgeführt sein, d.h. mit weniger als 2^n Zeilen bei n Eingangssignalen:
Für die unbestimmten Zeilen wird ohne weitere Angabe für das Ausgangssignal der Wert 0 angenommen.
Der Ausgangszustand unbestimmter Zeilen kann auch individuell definiert werden:

```
ISTYPE 'pos,com'  ⇒ 1
ISTYPE 'neg,com'  ⇒ 0
ISTYPE 'dc,com'   ⇒ .x.  (Zustand wird der Optimierung überlassen)
```

Mit der Compileranweisung `@DCSET` wird für unbesetzte Terme die Ausgangsvariable generell entsprechend der Optimierung 0 oder 1 gewählt. Mit `@ONSET` kann dieser Modus wieder ausgeschaltet werden.
Für kombinatorische Entwürfe ist `@DCSET` möglicherweise empfehlenswert - bei sequentieller Logik ist es besser, nach undefinierten Übergängen in einen definierten (*default*, *reset*, 0) Status zu kommen.

- Beispiele:

Kombinatorisch

```
TRUTH_TABLE
([e0,e1,e2] -> [a0,a1])
[0,0,0] -> [0,1];
[1,0,1] -> [0,0];
.....
```

Kombinatorisch, mit SET's

```
DECLARATIONS
e_set=[e0,e1,e2];
a_set=[a0,a1];
TRUTH_TABLE
(e_set -> a_set])
0 -> 1;
5 -> 0;
.....
```

Sequentiell

```
TRUTH_TABLE
([e0,e1,e2] :> [a0,a1])
[0,0,0] :> [0,1];
[1,0,1] :> [0,0];
.....
```

Kombinatorisch und Sequentiell

```
TRUTH_TABLE
([e0,e1,e2] :> [a0,a1] -> [a2,a3,a4])
[0,0,0] :> [0,1] -> [0,0,1,1];
[1,0,1] :> [0,0] -> [1,1,0,1];
.....
```

	@DCSET	@DCSET	@DCSET	@DCSET
TRUTH_TABLE	TRUTH_TABLE	TRUTH_TABLE	TRUTH_TABLE	TRUTH_TABLE
([a,b]->c)	([a,b]->c)	([a,b]->c)	([a,b]->c)	([a,b]->c)
[0,0]->0;	[0,0]->0;	[0,1]->0;	[0,0]->0;	[1,1]->1;
[0,1]->0;	[0,1]->0;	[1,0]->0;	[1,0]->0;	
[1,0]->0;	[1,0]->0;	[1,1]->1;	[1,1]->1;	
[1,1]->1;	[1,1]->1;			
ergibt:				
c=a&b	c=a&b	c=a&b	c=b	c=1

Tabelle 6: unvollständige Wahrheitstabellen definieren

2.2.3 Zustandsdiagramm

Der jeweilige Status (*state*) kann direkt als Set oder Zahl angegeben werden.
Empfehlenswerter ist die Vergabe symbolischer Namen:
Dies erleichtert die Lesbarkeit und erlaubt eine flexiblere Zuordnung der Statuskodierung
z.B.: binär, binär-einschrittig oder one hot

```
STATE_DIAGRAM statusregister

STATE state: zuweisung;
             ....
             zuweisung;
      übergangsanweisung;

STATE state: zuweisung;
             ...
             zuweisung;
      übergangsanweisung;
```

Formulierung der Übergangsanweisungen (*transition statement*):

- Unbedingter Übergang:

```
GOTO state;
```

- Bedingter Übergang mit: `IF-THEN-ELSE`

```
IF bedingung THEN state ELSE   state;
```

Wird das `ELSE` weggelassen, so entspricht das in der Wirkungsweise der `CASE` Bedingung, wobei die selben Regeln wie bei `CASE` zu beachten sind !

```
IF bedingung THEN state;
IF bedingung THEN state;
...
IF bedingung THEN state;
```

Bei vollständigen verketteten (*chained*) `IF-THEN-ELSE` Gleichungen ist von selbst sichergestellt, daß genau eine Bedingung erfüllt ist:

```
IF bedingung THEN state ELSE
IF bedingung THEN state ELSE
...
IF bedingung THEN state ELSE   state;
```

Dies gilt auch für die ebenfalls erlaubten verschachtelten (*nested*) `IF-THEN-ELSE` Gleichungen:

```
IF bedingung THEN
      IF bedingung THEN state ELSE   state
ELSE   state
```

Verkettet und verschachtelt:

```
IF bedingung THEN state ELSE
      IF bedingung THEN
            IF bedingung THEN state ELSE   state
      ELSE   state
IF bedingung THEN state ELSE   state;
```

- Bedingter Übergang mit: `CASE-ENDCASE`

```
CASE
   bedingung: state;
   bedingung: state;
   ....
   bedingung: state;
ENDCASE
```

Die CASE Bedingungen müssen eindeutig sein, es muss genau eine Bedingung wahr sein !
Für die Einhaltung dieser Regel muss selbst gesorgt werden, bei Nichteinhaltung ergeben sich unvorhersehbare Ergebnisse.

`IF-THEN-ELSE` und `CASE` Bedingungen können gemischt und verschachtelt (*nested*) werden.

- Zuweisungen mit: `WITH-ENDWITH`

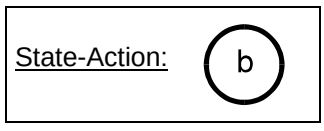
Bei den Übergangsanweisungen `IF-THEN-ELSE` und `CASE` können jeweils auch noch Zuweisungen für den Folgestatus gemacht werden. Bei nur einer Zuweisung kann das `ENDWITH` auch entfallen:

```
      IF bedingung THEN state WITH zuweisung;        "ohne ENDWITH
                  ELSE state WITH zuweisung;         "mehrere Zuweisungen -
                              ....;                  "mit ENDWITH
                              zuweisung;
                        ENDWITH
```

>> ABEL6 <<: Bei Zusammenfassung von Zuweisungen zu einem Block kann `ENDWITH` ebenfalls weggelassen werden.

```
      IF bedingung THEN state WITH {
                                   zuweisung;
                                   ....;
                                   zuweisung;
                                   }
                  ELSE state WITH   zuweisung;
```

- Zeitverhalten von Zuweisungen (Statusaktionen):




```
      ...
STATUS b:
       port=x;
       IF stop THEN b ELSE c;
STATUS c:
       ...
```

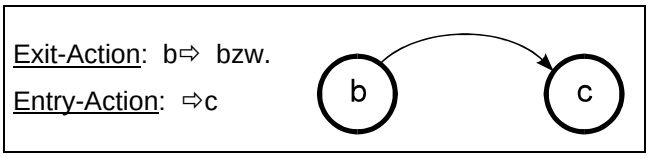
`port=x` gilt exakt solange wie der aktuelle Status (b) aktiv ist

```
      ...
STATUS b:
       port:=x;
       IF stop THEN b ELSE c;
STATUS c:
       ...
```

`port=x` beginnt im aktuellen Status (b) um einen Takt verzögert und wirkt im Folgestatus (c) einen Taktzyklus nach

```
       ...
STATUS b:
       IF stop THEN b WITH port:=x; ELSE c;
STATUS c:
       ...
```

`port=x` beginnt zum aktuellen Status (b) um einen Takt verzögert und endet zugleich mit dem aktuellen Status




```
       ...
STATUS b:
       IF stop THEN b
       ELSE c WITH port:=x;
STATUS c:
       ...
```

`port=x` gilt für den ersten Taktzyklus des Folgestatatus (c)

2.3 Compiler Anweisungen

Compiler Anweisungen werden mit einem vorgestellten @ gekennzeichnet

- **@DCSET, @ONSET**

Schaltet den "dont care mode" bei Wahrheitstabellen oder Zustandsdiagrammen mit `@DCSET` ein und mit `@ONSET` aus

- **@CARRY**

Begrenzt "Bit-Breite" (*look ahead*, *carry chain*) bei Addierern, Komparatoren und Zählern

```
DECLARATIONS
a7..a0 NODE;
aa=[a7..a0];
@CARRY 4;   "bewirkt 2 in Kette geschaltete 4-Bit Zaehler anstelle eines
                "einzelnen 8-Bit Zaehlers
EQUATIONS
aa:=aa+1
```

Anmerkung: Die Auswirkung dieser Einstellung kann durch die anschließend im Fitter durchgeführte Optimierung wieder aufgehoben werden, wenn dort nicht entsprechende Einstellungen vorgenommen werden. Teilweise ergibt sich hier ein nicht immer gänzlich durchschaubares Verhalten.

3 Übungsbeispiele

Die nachfolgenden Beispiele stellen eine Auswahl dar, um die Möglichkeiten der Sprache ABEL und des vorliegenden Übungsboards zu demonstrieren. Sie stellen keinen Anspruch auf optimales Design (Bausteinausnutzung, Geschwindigkeit) sondern sollen eine Einstiegshilfe sein und zu weiteren eigenen Versuchen anregen. Alle Beispiele wurden mit Webpack 6.x auf dem Übungsboard getestet.

3.1 DIL Schalter direkt an LED Reihe durchschalten

Aufgabe:
Einführungsbeispiel und Test der DIL Schalter:
Die Stellung der 10 DIL Schalter ist direkt auf der LED Reihe anzuzeigen.
Wenn sich ein DIL Schalter in der linken Stellung befindet, soll die entsprechende linke LED leuchten

```
Module dil_led

DECLARATIONS
dil9..dil0 PIN 79,80,81,83,82,84,1,2,3,4;                  // Dilschalter
dil=[dil9..dil0];
led9..led0 PIN 74,72,71,70,69,68,67,66,65,63 ISTYPE ‘com’; // Ledreihe
led=[led9..led0];

EQUATIONS
led = dil;

END
```

3.2 Tastatur auf Siebensegment Anzeige ausgeben

Aufgabe:
Ein Tastendruck auf die 16-stellige Tastatur soll auf der Siebensegmentanzeige in hexadezimaler Form angezeigt werden. Das Strobe Signal des Tastaturdecoders soll über den Dezimalpunkt signalisiert werden.

```
MODULE hex7seg

DECLARATIONS
t_d,t_c,t_b,t_a PIN 37,40,41,43;                           // 4 Bit von Tastaturenkoder
taste PIN 77;                                      // Strobe Signal von Tastaturenkoder
dezimal PIN 45 ISTYPE 'COM';                               // anzeigen an DP
a,b,c,d,e,f,g PIN 48,57,39,44,46,50,55 ISTYPE 'COM';       // 7-Segment Anzeige


//            a
//         --------
//        I        I
//    f   I    g   I  b
//         --------
//        I        I
//    e   I        I  c
//         --------
//            d

on,of = 0,1;                                               // 0 = on fuer gemeinsame Anode

TRUTH_TABLE
([t_d,t_c,t_b,t_a] -> [a,b,c,d,e,f,g])
[0,0,0,0] -> [on,on,on,on,on,on,of];  "0
[0,0,0,1] -> [of,on,on,of,of,of,of];  "1
[0,0,1,0] -> [on,on,of,on,on,of,on];  "2
[0,0,1,1] -> [on,on,on,on,of,of,on];  "3
[0,1,0,0] -> [of,on,on,of,of,on,on];  "4
[0,1,0,1] -> [on,of,on,on,of,on,on];  "5
[0,1,1,0] -> [on,of,on,on,on,on,on];  "6
[0,1,1,1] -> [on,on,on,of,of,of,of];  "7
[1,0,0,0] -> [on,on,on,on,on,on,on];  "8
[1,0,0,1] -> [on,on,on,on,of,on,on];  "9
[1,0,1,0] -> [on,on,on,of,on,on,on];  "A
[1,0,1,1] -> [of,of,on,on,on,on,on];  "b
[1,1,0,0] -> [on,of,of,on,on,on,of];  "C
[1,1,0,1] -> [of,on,on,on,on,of,on];  "d
```

```
[1,1,1,0] -> [on,of,of,on,on,on,on];  "E
[1,1,1,1] -> [on,of,of,of,on,on,on];  "F

EQUATIONS
dezimal = !taste;

END
```

3.3 Addierer

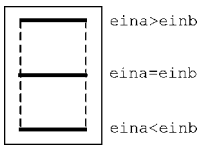
Aufgabe:
Es ist ein 3 Bit Addierer zu entwerfen:
Der erste Summand `eina` liegt am DIL Schalter, der zweite Summand `einb` liegt an der Tastatur.
Das Ergebnis `sum` wird an der LED Reihe vierstellig angezeigt

```
MODULE add3
TITLE '3-Bit Addierer ohne Carry-Eingang'

DECLARATIONS
a2,a1,a0 PIN 81,80,79;
eina = [0,a2..a0];
b2,b1,b0 PIN 40,41,43;
einb = [0,b2..b0];
cy,s2,s1,s0 PIN 70,71,72,74 ISTYPE 'COM';
sum = [cy,s2..s0];

EQUATIONS
sum = eina + einb;

END
```

3.4 Komparator

Aufgabe:

Es sind zwei 4 Bit Zahlenwerte zu vergleichen.:
`eina` ... DIL Schalter
`einb` ... Tastatur



Das Ergebnis ist wie nebenstehend gezeigt auf der Siebensegmentanzeige zu signalisieren

```
MODULE comp4

DECLARATIONS
a3..a0 PIN 83,81,80,79;
eina = [a3..a0];
b3..b0 PIN 37,40,41,43;
einb = [b3..b0];
groesser,gleich,kleiner PIN 61,56,51 ISTYPE 'COM';

EQUATIONS
!groesser=(eina>einb);
!gleich=(eina==einb);
!kleiner=(eina<einb);

END
```

3.5 Parity Generator

Aufgabe:
Eine über die Tastatur eingegebene 4 Bit Zahl ist um ein Parity Bit (gerade Parität) zu ergänzen und das Ergebnis an der LED Reihe anzuzeigen.

```
MODULE parity4

DECLARATIONS
d,c,b,a PIN 37,40,41,43;
led4..led0 PIN 69,70,71,72,74 ISTYPE 'COM';

EQUATIONS
led4=(d+c+b+a);
led3=d;
led2=c;
led1=b;
led0=a;

END
```

3.6 Dualzähler

Aufgabe:
Es ist ein 4 Bit Dualzähler zu entwerfen. Als Takt wird der interne 1 Hz Taktgeber verwendet, die Anzeige erfolgt auf der LED Reihe

```
MODULE dual4

DECLARATIONS
q3..q0 PIN 70,71,72,74 ISTYPE 'REG';        // 4 D-FF
zaehler=[q3..q0];
takt PIN 10; "1 Hz Takt

EQUATIONS
zaehler:=zaehler+1;
zaehler.clk=takt;

END
```

3.7 Dualer Auf / Ab Zähler mit Siebensegment Anzeige

Aufgabe:

Es ist ein in der Richtung umschaltbarer 4 Bit Dualzähler aufzubauen. Als Takt wird der interne 1 Hz Taktgeber verwendet, der Zählerstand soll auf einer Siebensegment Anzeige dargestellt werden. Die Richtungsumschaltung erfolgt durch die Tastatur (LSB):
„0“ ... abwärts
„1“ ... aufwärts

```
MODULE aufab7seg

DECLARATIONS
q3..q0 NODE ISTYPE 'REG';                   // 4 D-FF
zaehler=[q3..q0];
takt PIN 10;                                // 1 Hz Takt
auf PIN 43;                                 // Auf/Ab Umschaltung -> Tastatur: 0=Ab, 1=Auf
a,b,c,d,e,f,g PIN 48,57,39,44,46,50,55 ISTYPE 'COM'; // 7-Segment Anzeige

EQUATIONS
zaehler.clk=takt;
WHEN auf THEN zaehler:=zaehler+1 ELSE zaehler:=zaehler-1;

TRUTH_TABLE
(zaehler -> [a,b,c,d,e,f,g])
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9
[1,0,1,0] -> [0,0,0,1,0,0,0];  "A
[1,0,1,1] -> [1,1,0,0,0,0,0];  "b
[1,1,0,0] -> [0,1,1,0,0,0,1];  "C
```

```
[1,1,0,1] -> [1,0,0,0,0,1,0];  "d
[1,1,1,0] -> [0,1,1,0,0,0,0];  "E
[1,1,1,1] -> [0,1,1,1,0,0,0];  "F

END
```

3.8 BCD Zähler mit Siebensegment Anzeige

Aufgabe:
Es ist ein einstelliger BCD Zähler zu entwerfen. Als Takt wird der interne 1 Hz Taktgeber verwendet, der Zählerstand soll auf einer Siebensegment Anzeige dargestellt werden.

```
MODULE dez4

DECLARATIONS
q3..q0 NODE ISTYPE 'REG';                                    // 4 D-FF
zaehler=[q3..q0];
takt PIN 10; "1 Hz Takt
a,b,c,d,e,f,g PIN 48,57,39,44,46,50,55 ISTYPE 'COM';       // 7-Segment Anzeige

EQUATIONS
zaehler.clk=takt;
WHEN zaehler>=9 THEN zaehler:=0 ELSE zaehler:=zaehler+1;

TRUTH_TABLE
(zaehler -> [a,b,c,d,e,f,g])
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9

END
```

3.9 Stoppuhr

Aufgabe:
Es ist eine Stoppuhr mit zweistelliger Anzeige zu entwerfen (0 - 59 s). Als Takt wird der interne 1 Hz Taktgeber verwendet, der Zählerstand soll auf zwei Siebensegment Anzeigen dargestellt werden.
Die Ablaufsteuerung erfolgt über die Tastatur:

Taste 0: Reset
Taste 1: Start
Taste 2: Stop

```
MODULE stoppuhr

DECLARATIONS

takt PIN 10; "interner 1Hz takt
t1,t0 PIN 41,43; "Tastatur
tastatur=[t1,t0];
reset=(tastatur==0);                 // Steuersignale definieren
start=(tastatur==1);
stop=(tastatur==2);
e3..e0 NODE ISTYPE 'REG';            // Zaehler fuer Einerstelle
einer=[e3..e0];
z3..z0 NODE ISTYPE 'REG';             // Zaehler fuer Zehnerstelle
zehner=[z3..z0];
ae,be,ce,de,ee,fe,ge PIN 48,57,39,44,46,50,55 ISTYPE 'COM';  // Anzeige Einerstelle
az,bz,cz,dz,ez,fz,gz PIN 61,53,47,51,52,58,56 ISTYPE 'COM';  //Anzeige Zehnerstelle

EQUATIONS

einer.clk=takt;
zehner.clk=takt;
```

```
WHEN reset THEN einer:=0 ELSE
WHEN stop THEN einer:=einer ELSE
WHEN einer==9 THEN einer:=0 ELSE einer:=einer+1;

WHEN reset THEN zehner:=0 ELSE
WHEN !(start&(einer==9)) THEN zehner:=zehner ELSE
WHEN zehner==5 THEN zehner:=0 ELSE zehner:=zehner+1;

TRUTH_TABLE
(einer -> [ae,be,ce,de,ee,fe,ge])
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9

END
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9

END
```

3.10 Sägezahngenerator

Aufgabe:
Es ist ein Sägezahngenerator für ca. 1,5 kHz zu entwerfen.

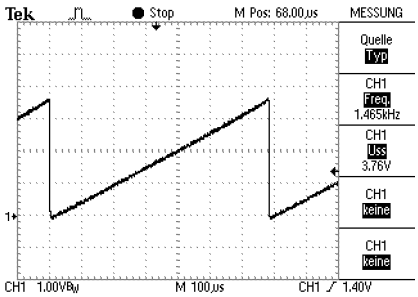


Bild 19: Gemessene Sägezahnspannung

```
MODULE saegez

// 6 Bit Zaehler mit Ausgabe an R2R
// 7 Bit Vorteiler von 12MHZ auf ca. 1465 Hz Saegezahn
// 12.000.000 MHz / 2E7 / 2E6 =      1.465 Hz

DECLARATIONS

da5..da0 PIN   34,33,32,31,24,23 ISTYPE 'REG';      // Zaehler und DA Anschluss
da=[da5..da0];
takt PIN 12;                                        // 12 MHz Takt
vt6..vt0 NODE ISTYPE'REG';                          // 12.000.000 / 2E7 = 94 kHz Takt fuer Zaehler
vorteiler=[vt6..vt0];


EQUATIONS

vorteiler.clk=takt;
vorteiler:=vorteiler+1;
da.clk=vt6;
da:=da+1;
END
```

3.11 Dreieckspannungsgenerator

Aufgabe:
Es ist ein Dreieckgenerator für ca. 1,5 kHz zu entwerfen.

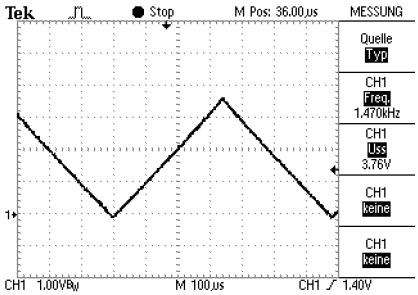


Bild 20: Gemessene Dreieckspannung

```
MODULE dreieck

// 6 Bit Zaehler mit Ausgabe an R2R
// 6 Bit Vorteiler von 12MHZ auf ca. 1465 Hz Dreieck
// 12.000.000 MHz / 2E6 / 2E6 / 2 =  1.465 Hz

DECLARATIONS

q6..q0 NODE ISTYPE 'REG';           // Zaehler (6 Bit fuer DA + 1 Bit fuer Dreieck)
zaehler=[q6..q0];
da5,da4,da3,da2,da1,da0 PIN 34,33,32,31,24,23 ISTYPE'COM';  // DA Anschluss
da=[.x.,da5..da0];
takt PIN 12;   "12 MHzTakt
vt5..vt0 NODE ISTYPE 'REG';         // 12.000.000 / 2E6 = 188 kHz Takt fuer Zaehler
vorteiler=[vt5..vt0];

EQUATIONS

zaehler.clk=vt5;
zaehler:=zaehler+1;
vorteiler.clk=takt;
vorteiler:=vorteiler+1;
WHEN q6 THEN da=!zaehler ELSE da=zaehler;

END
```

3.12 Sinusgenerator

Aufgabe:
Es ist ein Sinusgenerator für ca. 1,5 kHz zu entwerfen.

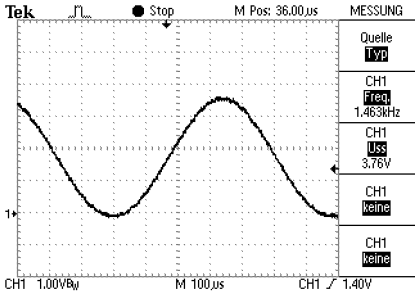


Bild 21: Gemessene Sinusschwingung

```
MODULE sinusgen

// 6 Bit Zaehler
// Dreieckgenerator mit nachgeschalteter Sinustabelle
// Signalausgabe an R2R
// Vorteiler von 12MHZ auf ca. 1465 Hz Dreieck
// 12.000.000 MHz / 2E6 / 2E6 / 2 =  1.465 Hz
```

```
DECLARATIONS

q6..q0 NODE ISTYPE 'REG';     // Saegezahn
zaehler=[q6..q0];
dr5..dr0 NODE ISTYPE 'COM';   // Dreieck
dreieck=[.x.,dr5..dr0];
da5,da4,da3,da2,da1,da0 PIN 34,33,32,31,24,23 ISTYPE'COM';  //DA Anschluss (Sinus)
da=[.x.,da5..da0];
takt PIN 12;                  //12 MHz Takt
vt5..vt0 NODE ISTYPE 'REG';   // 12.000.000 / 2E6 = 188 kHz Takt fuer Zaehler
vorteiler=[vt5..vt0];

EQUATIONS

zaehler.clk=vt5;      // Saegezahn
zaehler:=zaehler+1;
vorteiler.clk=takt;
vorteiler:=vorteiler+1;
WHEN q6 THEN dreieck=!zaehler ELSE dreieck=zaehler;  //Dreieck


TRUTH_TABLE
([dr5..dr0] -> [da5..da0])
0       ->     63;
1       ->     63;
2       ->     63;
3       ->     63;
4       ->     62;
5       ->     62;
6       ->     62;
7       ->     61;
8       ->     61;
9       ->     60;
10      ->     59;
11      ->     58;
12      ->     58;
13      ->     57;
14      ->     56;
15      ->     55;
16      ->     53;
17      ->     52;
18      ->     51;
19      ->     50;
20      ->     49;
21      ->     47;
22      ->     46;
23      ->     44;
24      ->     43;
25      ->     42;
26      ->     40;
27      ->     39;
28      ->     37;
29      ->     35;
30      ->     34;
31      ->     32;
32      ->     31;
33      ->     29;
34      ->     28;
35      ->     26;
36      ->     24;
37      ->     23;
38      ->     21;
39      ->     20;
40      ->     19;
41      ->     17;
42      ->     16;
43      ->     14;
44      ->     13;
45      ->     12;
46      ->     11;
47      ->     10;
48      ->     8;
49      ->     7;
50      ->     6;
51      ->     5;
52      ->     5;
53      ->     4;
54      ->     3;
55      ->     2;
56      ->     2;
57      ->     1;
58      ->     1;
59      ->     1;
60      ->     0;
```

```
61      ->      0;
62      ->      0;
63      ->      0;

END
```

3.13 Pulsweitenmodulation

Aufgabe:
Es ist eine Schaltung zu entwerfen, die je nach Tastendruck eine Gleichspannung in 15 Schritten zwischen 0 V und etwa 3,5 V (~ maximale Ausgangsspannung des CPLD) ausgibt.
Anzeige mit Oszilloskop oder Multimeter.

```
MODULE pwm

DECLARATIONS

takt PIN 12;
q15..q0 NODE ISTYPE'REG';
teiler=[q15..q0];
d,c,b,a PIN 37,40,41,43;
tastatur=[d,c,b,a];
pwm PIN 35 ISTYPE 'com';      // PWM Ausgang

EQUATIONS

teiler.clk=takt;
teiler:=teiler+1;
pwm=([q15..q12]<tastatur);     // PWM Bildung

END
```

3.14 Einfacher Sigma / Delta Wandler

Aufgabe:
Es ist ein einfaches Demonstrationsmodell eines AD Wandlers nach dem Sigma/Delta Prinzip aufzubauen. An Stelle des analogen Integrators wird ein RC Glied verwendet. Als Komparator dient ein normaler Logikeingang des CPLD und der 1 Bit D/A wird durch einen normalen Digitalausgang des CPLD angenähert. Als digitales Filter kommt ein Mittelwertbildner (Integrator) zur Anwendung
Die zu messende analoge Eingangsspannung wird durch ein auf dem Board befindliches Trimmpotentiometer erzeugt. Das Messergebnis soll mit einer Auflösung von 4 Bit auf einer Siebensegment Anzeige in hexadezimaler Form angezeigt werden.

```
MODULE sigmad

DECLARATIONS

aus PIN 36 ISTYPE'REG';
t6..t0 NODE ISTYPE'REG';
teiler=[t6..t0];
takt PIN 12;                                        // 12 MHz Takt
ein PIN 62;                                         // Summenpunkt, Komparatoreingang
ein_s NODE ISTYPE'REG';
i7..i0 NODE ISTYPE'reg';                            // Integrator (Digitalfilter)
integrator=[i7..i0];
tz7..tz0 NODE ISTYPE'reg';                          // Integrationsdauer
torzeit=[tz7..tz0];
d3..d0 PIN 67,66,65,63 ISTYPE'reg';                 // Ergebnis (dual)
daten=[d3..d0];
a,b,c,d,e,f,g PIN 61,53,47,51,52,58,56 ISTYPE'COM';  // 7-Seg.Display

EQUATIONS

teiler.clk=takt;                                    // Vorteiler
teiler:=teiler+1;
ein_s:=ein;                                         // Komparatoreingang  abtasten
ein_s.clk=takt;
aus:=!ein;                                          // 1-Bit DA
aus.CLK=t6;

torzeit.clk=t6;                                     // Integrationsdauer
torzeit:=torzeit+1;
```

```
integrator.clk=t6;                                  // Integrator
integrator.clr=(torzeit==0);
WHEN aus THEN integrator:=integrator+1 ELSE integrator:=integrator;
daten.clk=(torzeit==255);           // Die obersten 4 Bit des Integrators werden
daten:=[i7..i4];                                    // als Ergebnis in HEX Form angezeigt

TRUTH_TABLE
(daten -> [a,b,c,d,e,f,g])                          // HEX Anzeige
 0 -> [0,0,0,0,0,0,1]; "0
 1 -> [1,0,0,1,1,1,1]; "1
 2 -> [0,0,1,0,0,1,0]; "2
 3 -> [0,0,0,0,1,1,0]; "3
 4 -> [1,0,0,1,1,0,0]; "4
 5 -> [0,1,0,0,1,0,0]; "5
 6 -> [0,1,0,0,0,0,0]; "6
 7 -> [0,0,0,1,1,1,1]; "7
 8 -> [0,0,0,0,0,0,0]; "8
 9 -> [0,0,0,0,1,0,0]; "9
10 -> [0,0,0,1,0,0,0]; "A
11 -> [1,1,0,0,0,0,0]; "b
12 -> [0,1,1,0,0,0,1]; "C
13 -> [1,0,0,0,0,1,0]; "d
14 -> [0,1,1,0,0,0,0]; "E
15 -> [0,1,1,1,0,0,0]; "F

END
```

3.15 UART - Sender

Aufgabe:

Es ist der Sendeteil einer seriellen Schnittstelle (UART) mit folgenden Eigenschaften zu entwerfen:
Übertragungsrate: 9600 Baud
Datenformat: 1 Startbit, 7 Datenbit, kein Parity, 1 Stoppbit
Die zu sendenden Daten werden an der Tastatur des Übungsboard eingegeben und sollen als hexadezimaler Wert (0 - F) mittels einer Terminalemulation (z.B. Hyperterm) am PC angezeigt werden.

```
MODULE ser_tx

// Baudrate = 9600 Baud
// Format: 1 Startbit, 7 Datenbit, 1 Stopbit (7N1)
// Druck auf Tastatur sendet 0 - F als ASCII Zeichen

DECLARATIONS

takt PIN 12;                                // 12 MHz Takt
q10..q0 NODE ISTYPE 'REG';                  // Vorteiler für 9.200 Baud
baudgen=[q10..q0];
baud=q10;                                   // Baudrate
!start PIN 77;  // Startimpuls
startv PIN 19 ISTYPE 'REG,BUFFER';
trigger=start&!startv;                      // Flankentriggerung aus Tastensignal
ausg PIN 75 ISTYPE 'COM';                   // TX an Max232
s3,s2,s1,s0 NODE ISTYPE 'REG,BUFFER';       // Statusregister
status=[s3..s0];
d6,d5,d4,d3,d2,d1,d0 NODE ISTYPE 'COM';     // zu sendendes Zeichen
ascii=[d6..d0];
d,c,b,a PIN 37,40,41,43;                    // Tataturdaten (0 - F)
tastatur=[0,0,0,d,c,b,a];

EQUATIONS

startv:=start;                              // Start
startv.clk=baud;
status.clk=baud;                            // Register fuer Statemachine
baudgen.clk=takt;                           // Baudrategenerator: 12.000.000 / 9600 = 1250
WHEN baudgen==1250 THEN baudgen:=0 ELSE baudgen:=baudgen+1;
// Dual von Tastatur -> HEX -> ASCII:
WHEN tastatur<10 THEN ascii=tastatur + ^H30 ELSE ascii=tastatur + ^H37;

                                            // Sendeteil
STATE_DIAGRAM status

STATE 0:
        ausg=1;
        IF trigger THEN 1 ELSE 0;
```

```
STATE 1:                                    // Startbit
        ausg=0;
        GOTO 2;
STATE 2:
        ausg=d0;                            // LSB
        GOTO 3;
STATE 3:
        ausg=d1;
        GOTO 4;
STATE 4:
        ausg=d2;
        GOTO 5;
STATE 5:
        ausg=d3;
        GOTO 6;
STATE 6:
        ausg=d4;
        GOTO 7;
STATE 7:
        ausg=d5;
        GOTO 8;
STATE 8:
        ausg=d6;                            // MSB
        GOTO 9;
STATE 9:                                    // Stopbit
        ausg=1;
        GOTO 0;
END
```

3.16 UART - Empfänger

Aufgabe:
Es ist der Empfangsteil einer seriellen Schnittstelle (UART) mit folgenden Eigenschaften zu entwerfen:
Übertragungsrate: 9600 Baud
Datenformat: 1 Startbit, 7 Datenbit, kein Parity, 1 Stoppbit
Die zu empfangenden ASCII Daten werden mittels einer Terminalemulation (z.B. Hyperterm) vom PC an das Übungsboard gesendet und sollen wie folgt angezeigt werden:

- Als 7 Bit Wert an einer LED Reihe
- Für die Werte 0 - 9 und a - f an der Siebensegment Anzeige.

```
MODULE ser_rx

// Das serielle Eingangssignal wird mit der  8-fachen Baudrate abgetastet.
// Nach Erkennen der fallenden Startflanke wird das Startbit in der Mitte
// abgetastet -> Fehlermeldung und Ruecksetzen wenn <> 0
// Anschliessend werden 7 Datenbits jeweils in der Mitte des Bits abgetastet
// und in das Empfangsschieberegister geschoben.
// Das Stopbit wird ebenfalls abgetastet -> Fehlermeldung wenn <> 1
// Die Anzeige erfolgt zweifach:
// 7 Bit an der LED Reihe für sämtliche ASCII Codes
// Für die Werte 0 - 9 sowie a - f an der Siebensegmentanzeige
// Andere werte werden an der Siebensegment Anzeige als "8" angezeigt

DECLARATIONS

takt PIN 12;                                // 12 MHz Takt
q7..q0 NODE ISTYPE 'REG';                   // Baudratengenerator
baud=[q7..q0];
baud8=q7;                                   // Abtastung mit 8-facher Baudrate
din PIN 76;                                 // Serieller Eingang,
!err PIN 54 ISTYPE 'REG';                   // Framing Error signalisieren
d6..d0 PIN 67,68,69,70,71,72,74 ISTYPE 'REG'; "Empfangsdatenregister
rxdat=[d6..d0];

s3..s0 NODE ISTYPE 'REG,BUFFER';            // Statusregister fuer Statemachine
status=[s3..s0];
              // Statuskodierung:
wst,pst,ls0,ls1,ls2,ls3,ls4,ls5,ls6,stp,pstp=0,1,2,3,4,5,6,7,8,9,10;
t2..t0 NODE ISTYPE 'REG';                   // Timer zur Abtastung der Bit-Mitte
timer=[t2..t0];
dins,dind NODE ISTYPE 'REG';                // Synchronisierte / verzoegerte Eing.Daten
startfl NODE ISTYPE 'COM';                  // Startflanke (fallend)
bit=[1,1,1];                                // 1 Bit = 1+7 Takte
hbit=[0,1,1];                               // 1 halbes Bit = 1+3 Takte
sample NODE ISTYPE 'REG';                   // Abtastzeitpunkt
a,b,c,d,e,f,g PIN 61,53,47,51,52,58,56 ISTYPE'COM';  // 7-Seg.Display
```

```
EQUATIONS

baud.CLK=takt;                              // Baudratengenerator: 12.000.000 / (9600*8) ~ 156
WHEN baud==156 THEN baud:=0 ELSE baud:=baud+1;
rxdat.CLK=baud8;
status.CLK=baud8;
timer.CLK=baud8;
[dins,dind,sample,err].CLK=baud8;
dins:=din;                                  // Eingangsdaten synchronisieren
dind:=dins;                                 //   -   verzoegern
startfl=!dins&dind;                         // Startflanke bilden
WHEN sample THEN rxdat:=[dind,d6..d1] ELSE rxdat:=rxdat;
WHEN status==pst THEN err:=0 ELSE err:=err; // Framingerror

STATE_DIAGRAM status

STATE wst:                                  // Warten auf Startbit
      IF startfl THEN pst WITH timer:=hbit ELSE wst WITH timer:=0;

STATE pst:                                  // Startbit pruefen
      IF timer!=0 THEN pst WITH timer:=timer-1 ELSE
      IF dind==0 THEN ls0 WITH timer:=bit ELSE wst WITH err:=1;

STATE ls0:                                  // Bit 0
      IF timer!=0 THEN ls0 WITH timer:=timer-1 ELSE ls1  WITH
      {timer:=bit; sample:=1;}

STATE ls1:                                  // Bit 1
      IF timer!=0 THEN ls1 WITH timer:=timer-1 ELSE ls2  WITH
      {timer:=bit; sample:=1;}

STATE ls2:                                  // Bit 2
      IF timer!=0 THEN ls2 WITH timer:=timer-1 ELSE ls3  WITH
      {timer:=bit; sample:=1;}

STATE ls3:                                  // Bit 3
      IF timer!=0 THEN ls3 WITH timer:=timer-1 ELSE ls4  WITH
      {timer:=bit; sample:=1;}

STATE ls4:                                  // Bit 4
      IF timer!=0 THEN ls4 WITH timer:=timer-1 ELSE ls5  WITH
      {timer:=bit; sample:=1;}

STATE ls5:                                  // Bit 5
      IF timer!=0 THEN ls5 WITH timer:=timer-1 ELSE ls6  WITH
      {timer:=bit; sample:=1;}

STATE ls6:                                  // Bit 6
      IF timer!=0 THEN ls6 WITH timer:=timer-1 ELSE stp  WITH
      {timer:=bit; sample:=1;}

STATE stp:                                  // Stopbit pruefen
      IF timer!=0 THEN stp WITH timer:=timer-1 ELSE
           IF dind==1 THEN 0 ELSE 0 WITH err:=1;

TRUTH_TABLE
(rxdat -> [a,b,c,d,e,f,g])                  // 7-Segm. Anzeige
'0' -> [0,0,0,0,0,0,1];       "0
'1' -> [1,0,0,1,1,1,1];       "1
'2' -> [0,0,1,0,0,1,0];       "2
'3' -> [0,0,0,0,1,1,0];       "3
'4' -> [1,0,0,1,1,0,0];       "4
'5' -> [0,1,0,0,1,0,0];       "5
'6' -> [0,1,0,0,0,0,0];       "6
'7' -> [0,0,0,1,1,1,1];       "7
'8' -> [0,0,0,0,0,0,0];       "8
'9' -> [0,0,0,0,1,0,0];       "9
'a' -> [0,0,0,1,0,0,0];       "A
'b' -> [1,1,0,0,0,0,0];       "b
'c' -> [0,1,1,0,0,0,1];       "C
'd' -> [1,0,0,0,0,1,0];       "d
'e' -> [0,1,1,0,0,0,0];       "E
'f' -> [0,1,1,1,0,0,0];       "F

END
```

3.17 Systemtest

Mit dem nachfolgenden Programm kann das Übungsboard auf seine Funktionen getestet werden:

- Die Stellung der DIL Schalter wird auf den beiden LED Reihen angezeigt
- Die Tastaturdaten werden parallel auf den beiden Siebensegment Anzeigen ausgegeben (Dezimalpunkt = „Taste gedrückt“)
- ... DA + RS232 sind noch in Ausarbeitung

```
MODULE systest

DECLARATIONS
"DIL -> LED
dil9..dil0 PIN 79,80,81,83,82,84,1,2,3,4; "Dilschalter
dil=[dil9..dil0];
led9..led0 PIN 74,72,71,70,69,68,67,66,65,63 ISTYPE 'com'; "Ledreihe
led=[led9..led0];
"Tastatur -> Display
ae,be,ce,de,ee,fe,ge,dpe PIN 48,57,39,44,46,50,55,45 ISTYPE 'COM';  "Anzeige Einerstelle
az,bz,cz,dz,ez,fz,gz,dpz PIN 61,53,47,51,52,58,56,54 ISTYPE 'COM';  "Anzeige Zehnerstelle
d,c,b,a PIN 37,40,41,43; "4 Bit von Tastaturenkoder
tastatur=[d,c,b,a];
taste PIN 77; "Strobe Signal von Tastaturenkoder

EQUATIONS

"DIL -> LED
led = dil;
"Tatatur -> Display
dpe=taste;
dpz=taste;


TRUTH_TABLE
"tastatur -> Display: Einer
(tastatur -> [ae,be,ce,de,ee,fe,ge])
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9
[1,0,1,0] -> [0,0,0,1,0,0,0];  "A
[1,0,1,1] -> [1,1,0,0,0,0,0];  "b
[1,1,0,0] -> [0,1,1,0,0,0,1];  "C
[1,1,0,1] -> [1,0,0,0,0,1,0];  "d
[1,1,1,0] -> [0,1,1,0,0,0,0];  "E
[1,1,1,1] -> [0,1,1,1,0,0,0];  "F

TRUTH_TABLE
"tastatur -> Display: Zehner
(tastatur -> [az,bz,cz,dz,ez,fz,gz])
[0,0,0,0] -> [0,0,0,0,0,0,1];  "0
[0,0,0,1] -> [1,0,0,1,1,1,1];  "1
[0,0,1,0] -> [0,0,1,0,0,1,0];  "2
[0,0,1,1] -> [0,0,0,0,1,1,0];  "3
[0,1,0,0] -> [1,0,0,1,1,0,0];  "4
[0,1,0,1] -> [0,1,0,0,1,0,0];  "5
[0,1,1,0] -> [0,1,0,0,0,0,0];  "6
[0,1,1,1] -> [0,0,0,1,1,1,1];  "7
[1,0,0,0] -> [0,0,0,0,0,0,0];  "8
[1,0,0,1] -> [0,0,0,0,1,0,0];  "9
[1,0,1,0] -> [0,0,0,1,0,0,0];  "A
[1,0,1,1] -> [1,1,0,0,0,0,0];  "b
[1,1,0,0] -> [0,1,1,0,0,0,1];  "C
[1,1,0,1] -> [1,0,0,0,0,1,0];  "d
[1,1,1,0] -> [0,1,1,0,0,0,0];  "E
[1,1,1,1] -> [0,1,1,1,0,0,0];  "F

EN
```

3.18 Beispiele in Ausarbeitung

Modulation: ASK, PSK, FSK
Pseudo Random Noise Generator
Frequenzzähler mit Ziffern und Balkenanzeige
Inkrementalgeber (mit Hardwareerweiterung)
Scrambler (zwei Boards gekoppelt)

4 Literaturhinweise

Data IO: ABEL HDL Reference
Data IO

Xilinx: Xilinx ABEL User Guide
Xilinx

Lattice: ABEL-HDL Reference Manual
Lattice

Datenblätter und Applikationsschriften:

http://www.xilinx.com