Versuch 3: Programmierbare Logikbausteine

Einführung in die Logiksynthese und Simulation

1 Einleitung

Ziel des Labors ist die Vertiefung der Lehrinhalte der Vorlesung Digitaltechnik. In diesem Versuch wird der Ablauf der Logiksynthese und Simulation anhand rechnergestützter Werkzeuge (easyABEL, EZ-ABEL 4.3) gezeigt. Nach der Einführung in das Thema Logiksynthese und Simulation wird das Werkzeug vorgestellt, mit dem die Abbildung der Entwurfsdaten auf eine bestimmte Bausteinarchitektur durchgeführt wird. Die Eingabe der Entwurfsbeschreibung erfolgt hier mit der weit verbreiteten Hardware-Beschreibungssprache ABEL-HDL (Advanced Boolean Expression Language - Hardware Description Language).

Um bestimmte Sachverhalte verständlich darzustellen, wird abschließend anhand mehrerer Beispiele die Vorgehensweise bei der Logiksynthese, der Simulation und der Umgang mit den Entwicklungswerkzeugen demonstriert. Der Umfang der dabei verwendeten Komponenten der Entwicklungswerkzeuge ist ausreichend für die Bearbeitung der gestellten Aufgaben. Für weiterreichende Informationen stehen im Labor die Handbücher zur Verfügung.

2. Logiksynthese

Unter Logiksynthese ist die Umsetzung eines gegebenen Problems in eine Hardwarestruktur zu verstehen. Zu diesem Zweck muß zunächst entschieden werden, ob ein Schaltnetz oder ein Schaltwerk zu realisieren ist.

2.1 Schaltnetze

Ein Schaltnetz verfügt über rein kombinatorische Logik. Änderungen an den Eingängen haben unmittelbar Änderungen an den Ausgängen zur Folge. Das Schaltnetz kennt im Gegensatz zum Schaltwerk keine Register und somit auch keine Zustände. Das Verhalten eines Schaltnetzes wird später noch am Beispiel einer einfachen Rohrpoststeuerung veranschaulicht.

2.2 Schaltwerke

Ein Schaltwerk setzt sich aus einem Schaltnetz und einem Speicher zusammen. Durch die Zwischenspeicherung der Schaltzustände und durch die Rückkopplung auf ein Schaltnetz entsteht dieses Schaltwerk, das sich allgemein als „Endlicher Automat“ (auf Neudeutsch auch FSM Finite State Machine) beschreiben läßt. Im Gegensatz zum statischen (kombinatorischen) Schaltnetz, dessen Ausgaben allein durch die Eingaben bestimmt sind, verfügt ein Schaltwerk über ein Gedächtnis (Speicher, beliebige Flipflops), in dem vorangegangene Schritte gespeichert werden. Außerdem handelt es sich beim Schaltwerk um ein sequentielles (dynamisches) System, da ja eine Folge von Zuständen durchlaufen wird.

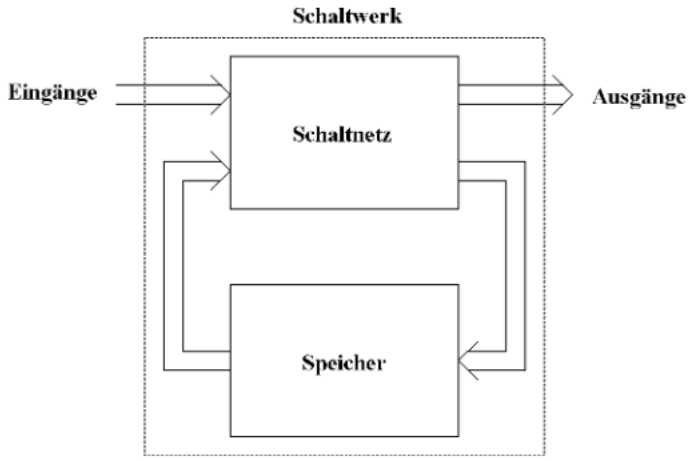


Bild 2.1: Struktur eines Schaltwerks

2.2.1 Endliche Automaten als theoretische Grundlage der Schaltwerke

Ein endlicher Automat ist eine sequentielle Maschine. Der endliche Automat kann nur eine endliche Menge Z an Zuständen annehmen. Zu einem bestimmten Zeitpunkt t_n wird in den Automaten ein Eingabeelement e_i aus der endlichen Eingabemenge E eingegeben. Mit dem Zustand z_i gibt der Automat das Ausgabeelement a_i der Ausgabemenge A aus. Dabei geht der Automat vom Zustand z_i in den Zustand z_j über.

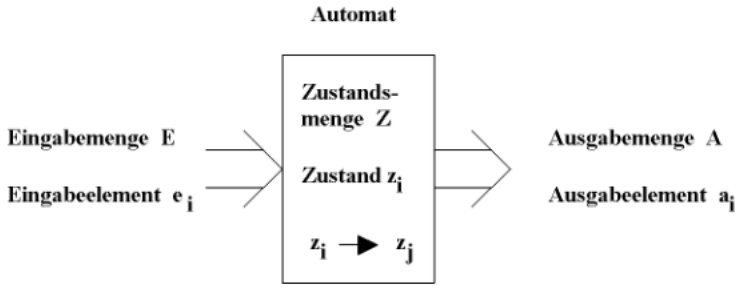


Bild 2.2: Grundkonzept des allgemeine endlichen Automaten

Die entscheidenden Begriffe zur Beschreibung des endlichen Automaten sind folgendermaßen definiert:

Eingabe:

Bei der Eingabe liest der Automat das Eingabezeichen e_i. Das Element e_i ist eines aus endlich vielen Zeichen der Eingabemenge E. Es gilt $e_i \in E$.

Zustand:

Der Automat kann zu einem Zeitpunkt t_n einen Zustand z_i aus endlich vielen Zuständen der Zustandsmenge Z annehmen. Der Automat geht vom Zustand z_i über in den Folgezustand z_j. Es gilt $z_i \in Z$, $z_j \in Z$.

Ausgabe:

Der Automat gibt im Zustand z_i bei einem Eingabeelement e_i das Ausgabeelement a_i aus. Das Ausgabeelement a_i ist ein Zeichen der Ausgabemenge A, die aus endlich vielen Zeichen besteht. Es gilt $a_i \in A$.

Für die Darstellung des Verhaltens eines Automaten gibt es zwei gebräuchliche Darstellungsformen:

- Zustandstabelle oder Automatentabelle (-tafel)
- Zustandsgraph (-diagramm)oder Netz

Automatentabelle:

Zustand zur Zeit t_n	Folgezustand zur Zeit t_{n+1}	Eingabeelement	Ausgabeelement
z_i	z_j	e_i	a_i
..	..	..	..

Durch diese Automatentabelle ist das Verhalten des Automaten zu jeder Zeit eindeutig beschrieben.

Eine andere Form der Beschreibung ist die durch einen Zustandsgraphen. Jeder Zustand wird durch einen Kreis (man spricht von "Knoten") dargestellt. Von jedem Kreis gehen so viele Linien mit Pfeil aus, bzw. enden dort (man spricht von "gerichteten Kanten"), wie es Eingabeelemente gibt. Eine Kante endet in dem Zustand, in den der Automat nach Lesen eines Zeichens übergeht. Im folgenden Bild ist der Übergang vom Zustand z_i in den Zustand z_j dargestellt. Das Eingabeelement e_i erzeugt den Zustandswechsel. Im Zustand z_j wird dann das

Ausgabeelement a_i ausgegeben. Damit die Darstellung übersichtlich bleibt werden oft nur die Eingaben dargestellt, die auch zu einem Zustandswechsel führen bzw. eine Ausgabe erzeugen.

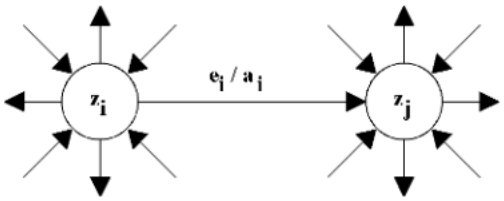


Bild 2.3 Übergang vom Zustand z_i in den Folgezustand z_j im Zustandsgraphen

Formal läßt sich der Automat beschreiben durch

die Eingabemenge $E = \{ e_1, e_2, ... \}$,

die Zustandsmenge $Z = \{ z_1, z_2, ... \}$,

die Ausgabemenge $A = \{ a_1, a_2, ... \}$

und die Funktionen, die zu Zustandsübergängen bzw. zu Ausgaben führen

die Übergangsfunktion $z(t_{n+1}) = f_ü \{ z(t_n), e(t_n) \}$ und

die Ausgabefunktion $a(t_n) = f_a \{ z(t_n), e(t_n) \}$

Diese Form beschreibt den nach dem amerikanischen Mathematiker G.H. Mealy benannten "Mealy-Automaten" (1955 "Bell Systems Technical Journal"). Einen bezüglich der Ausgabe etwas anderen Automaten nennt man "Moore-Automaten" nach E.F. Moore (1955 Annals of Mathematical Studies). Beim Moore-Automaten wird das Ausgabeelement a zur Zeit t_n nur durch den Zustand des Automaten angegeben.

=> Die Ausgabefunktion $a(t_n) = f_a \{ z(t_n) \}$

In der Digitaltechnik entsprechen den Elementen e und a Binärwerte. Der Übergang vom Zustand $z(t_n)$ in den Zustand $z(t_{n+1})$ erfolgt zu diskreten Zeitpunkten, die durch ein Taktsignal bestimmt sind. Der Zustand des Automaten ist für die Dauer des Taktsignals gespeichert und kann sich nur beim nächsten Taktsignal ändern. Damit kann man die Blockschaltbilder und Zustandsdiagramme für diese unterschiedlichen Automaten angeben.

2.2.2 Moore-Schaltwerk

Beim Moore-Schaltwerk werden die Zustandsvariablen, d.h. die Register ohne weitere logische Verknüpfung mit den Eingangssignalen direkt oder über ein Ausgangsschaltnetz auf die Ausgänge geführt (Bild 2.4).

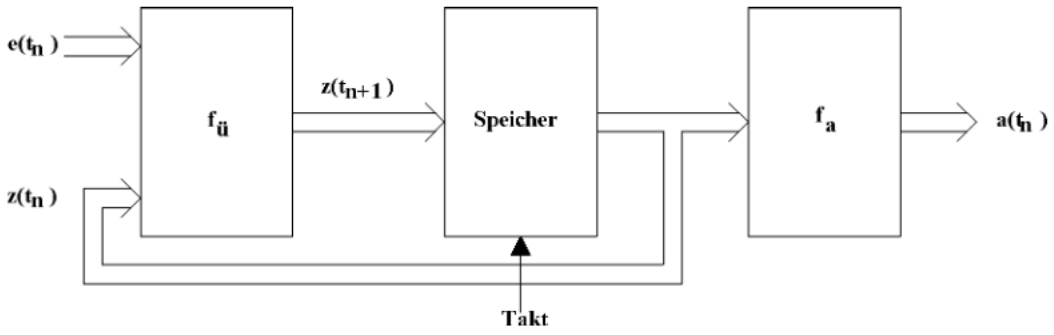


Bild 2.4: Struktur des Moore-Schaltwerks

Dadurch kann eine Signaländerung an den Ausgängen nur durch eine Zustandsänderung, also frühestens mit dem nächsten Takt auftreten. Man sagt, das Moore-Schaltwerk ist zustandsorientiert. Im Zustandsdiagramm des Moore-Schaltwerks (Bild 2.5) wird daher das Ausgangssignal mit in den Kreis für den Zustand aufgenommen und durch einen Strich getrennt unterhalb der Zustandsvariablen angegeben.

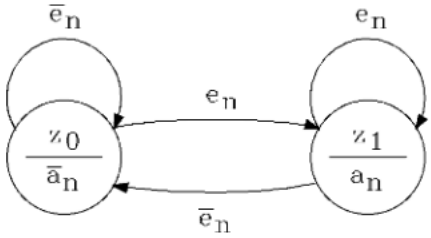


Bild 2.5: Zustandsdiagramm für ein Moore-Schaltwerk mit zwei Zuständen

2.2.3 Mealy-Schaltwerke

Reine Mealy-Schaltwerke unterscheiden sich von Moore-Schaltwerken dadurch, daß die Zustandsvariablen, d.h. die Ausgänge der Speicherelemente, noch kombinatorisch mit Eingangssignalen verknüpft sind, bevor sie auf die primären Ausgänge der Schaltung geführt werden (Bild 2.6).

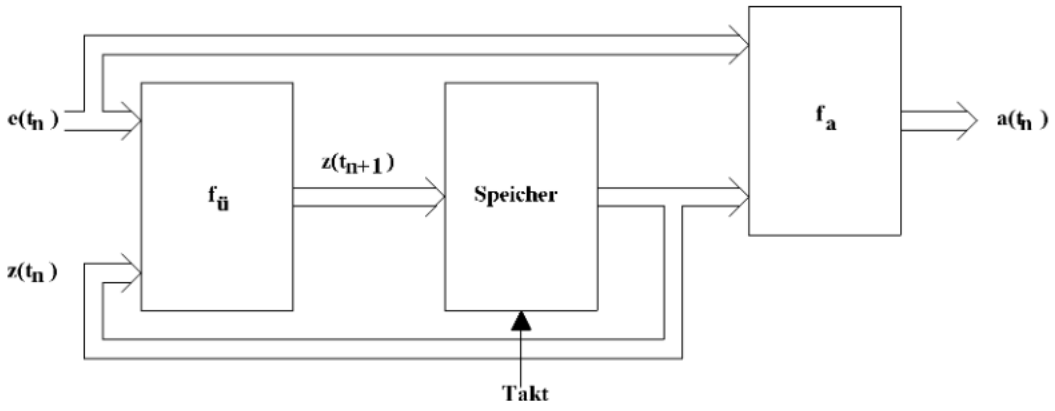


Bild 2.6 Struktur des Mealy-Schaltwerks

Dadurch kann die Änderung eines Eingangssignals sofort eine Änderung der Ausgangssignale bewirken. Man sagt, das Mealy-Schaltwerk ist übergangsorientiert. Mealy-Schaltwerke reagieren schneller auf Änderungen der Eingangssignale, jedoch können bei dieser Art der Realisierung auch dynamische Hazards auftreten. Da das Ausgangssignal nicht ausschließlich vom Zustand abhängt, muß beim Zustandsdiagramm des Mealy-Schaltwerks der Ausgangswert in Abhängigkeit von den Eingangsvariablen dargestellt werden (Bild 2.7).

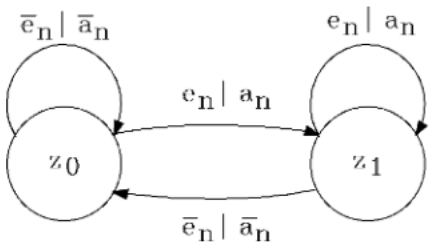


Bild 2.7: Zustandsdiagramm für ein Mealy-Schaltwerk mit zwei Zuständen

Aus dem Zustandsdiagramm läßt sich sowohl für Moore- als auch für Mealy-Schaltwerke direkt die Übergangstabelle herleiten, die, analog zur Funktionstabelle der Schaltnetze, für die Minimierung der Schaltung benötigt wird. Bei größeren Schaltungen ist es sehr zeitaufwendig, die Minimierung selbst (per Hand) durchzuführen. Diese (auch sehr fehlerträchtige) Arbeit übernehmen spezielle Software-Tools, die gewöhnlich in Softwareprodukten zur Logiksynthese integriert sind.

3 Synthesewerkzeuge

Synthesewerkzeuge verlangen eine strukturierte und einheitliche Schaltungsbeschreibung. Dabei besteht generell die Möglichkeit der sprachorientierten oder der graphischen (schematischen) Eingabebeschreibung.

3.1 Sprachorientierte Entwurfswerkzeuge (ABEL, easyABEL)

Die Eingabe der Schaltungsbeschreibung erfolgt mit einer Hardware Description Language (HDL) namens ABEL-HDL. Die Syntax ähnelt streckenweise vertrauten Komponenten gängiger Programmiersprachen. Die mit ABEL erzeugten HDL-Dateien erhalten immer die Extension „.abl“. Die Dateien müssen wie folgt strukturiert sein:

- Kopfteil mit Namen des zu entwerfenden Schaltungsmoduls
- Deklarationsteil für Zielarchitektur, Signaldefinitionen, Signal-Pin-Zuweisungen, Variablendefinitionen, ..
- Logikbeschreibung in Form von Booleschen Gleichungen

  und / oder
- Logikbeschreibung in Form von Funktionstabellen oder Zustandsübergangskonstrukten

„easyABEL“ ist ein eigenständiger PC-Abkömmling des Softwarepaketes ABEL der Firma „DATA I/O“, was speziell auf die Programmierung von PLD´s optimiert ist.

Nach der vollständigen Eingabe der Schaltungsbeschreibung mit einem Texteditor bereitet ein Sprachprozessor (Compiler) nach einem Syntaxcheck das AHDL-File für die anschließende Logikminimierung (Bündelminimierer ESPRESSO) auf. Nach der Minimierung erfolgt die Ausrichtung der Netzliste auf die geplante Zielarchitektur (ein bausteinunabhängiger Entwurf ist auch möglich). „easyABEL“ erzeugt ein File mit der Extension „.jed“ zur Programmierung der Bausteine. Zusätzlich können noch verschiedenste Dateien, z.B. zur Simulation, zur Dokumentation usw. erzeugt werden.

3.2 Graphische Entwurfswerkzeuge

Graphisch orientierte Synthesewerkzeuge erlauben die schematische Eingabe des Entwurfs in Form von Schaltbildern auf Registertransfer- oder Gatterebene. Das bedeutet, daß in einer Bibliothek die gängigsten Gattergrundtypen, Zählerschaltungen, Register, Flipflops usw. zur Verfügung gestellt werden, die aufgerufen und auf graphischem Weg verdrahtet werden. Das Problem bei der graphischen Eingabe liegt in der Tatsache, daß man, im Vergleich zur sprachorientierten Eingabe, bereits zu Beginn der Synthese eine exakte Vorstellung der Schaltungshardware haben muß. Das zeitintensive Minimieren und die Ermittlung der benötigten Produktterme muß zuvor „per Hand“ erfolgen. Andererseits hat man beim graphischen Entwurf immer die direkte Sicht auf die realisierte Hardwarestruktur. Die ist bei der Sprache AHDL nicht immer der Fall.

Im Labor wird nur die sprachorientierte Eingabe verwendet.

4 Simulation

Beim Entwurf einer digitalen Schaltung werden zwei Schritte ausgeführt. In der Logiksynthese wird eine Schaltung entworfen, die die in der Entwurfsaufgabe beschriebene Funktion erfüllt. Hierbei gibt es mehrere Ansätze, z.B. die Synthese eines Automaten, der Entwurf auf Gatterebene, der Entwurf mit Bausteinen oder eine Mikroprogrammsteuerung. Der zweite Schritt ist die Abbildung des logischen Entwurfs auf eine bestimmte Zieltechnologie, z.B. GAL, FPGA, Gate Array, Full Custom usw. Nach jedem dieser Entwurfsschritte erfolgt eine Überprüfung, ob die synthetisierte Schaltung noch der Aufgabenstellung entspricht (Validierung). Wenn ja, kann zum nächsten Entwurfsschritt übergegangen werden, andernfalls muß mindestens der letzte Schritt noch einmal durchlaufen werden. Die Validierung ist also notwendig, um sicherzustellen, daß ein Entwurf die Aufgabenstellung erfüllt. Das wichtigste Werkzeug hierzu ist der Simulator. Je nachdem, welche Entwurfsphase überprüft wird, unterscheidet man zwischen der Logiksimulation oder funktionalen Simulation, der Timingsimulation und der Fehlersimulation. Im Labor werden wir nur die Logiksimulation näher betrachten, die von easyABEL auch unterstützt wird.

4.1 Funktionale Simulation (Logiksimulation)

Der Einsatz moderner Entwurfswerkzeuge wie easyABEL befreit den Entwickler von so fehlerträchtigen Aufgaben, wie der Logikminimierung. Es treten aber immer wieder Entwurfsfehler durch fehlerhafte Eingaben oder falsch verstandene Aufgabenstellungen auf. Die funktionale Simulation (Logiksimulation) dient dazu, solche (statischen) Fehler während des Entwurfs möglichst früh zu erkennen.

4.2 Timingsimulation

Es kann immer wieder vorkommen, daß in einer korrekt entworfenen Schaltung dynamische Fehler auftreten, die anhand der logischen Gleichungen des Entwurfs nicht erkannt werden können. Dynamische Fehler treten durch Signallaufzeiten innerhalb der Schaltung auf.

Probleme mit dem Timing treten bei Schaltungen z.B. dann auf, wenn die Signallaufzeiten bzw. Gatterverzögerungszeiten (propagation delays) innerhalb der Schaltung variieren oder wenn ein Pfad zu einem Ausgang länger oder kürzer als die anderen Pfade ist. Es existieren viele Methoden um diese Hazards zu eliminieren, die einfachste Methode ist Redundanz in die Schaltung einzubringen. Durch die Optimierung der Schaltung (bei easyABEL durch PLAOpt) die durch die Entwurfswerkzeuge durchgeführt wird und dazu dient, Redundanz in der Schaltung zu eliminieren, wird auch diese gewollte Redundanz eliminiert. Daher dürfen in diesen Fällen keine Optimierungen durchgeführt werden (bei easyABEL z.B. kein PLAOpt durchführen bzw. -reduce none verwenden). In der folgenden Abbildung ist eine Schaltung gezeigt, bei der Probleme mit dem Timing auftreten, da sowohl das nicht invertierte Signal C als auch das invertierte Signal $\overline{C}$ verwendet wird. Beachten Sie, daß in dem Teil der Schaltung in dem $\overline{C}$ verwendet wird eine zusätzliche Laufzeit auftritt.

Die Funktionsgleichung für F lautet:

$$F = B \cdot \overline{C} \vee \overline{A} \cdot C$$

Im Zeitdiagramm *) sieht man, daß beim Signal F, daß eigentlich konstant 1 sein müßte, ein Übergang von 1 nach 0 und wieder nach 1 auftritt (-> glitch). Durch Einfügen des redundanten Terms $\overline{A} \cdot B$, der die benachbarten Produktterme der ursprünglichen Funktion überdeckt (verbindet) kann dieser Hazard beseitigt werden. Die resultierende Funktionsgleichung lautet:

$$F = B \cdot \overline{C} \vee \overline{A} \cdot C \vee \overline{A} \cdot B$$

Durch diese Funktionsgleichung wäre also das Timingproblem beseitigt. Aber wenn jetzt eine Optimierungsroutine durchgeführt wird, würde der redundante Term eliminiert werden und es ergebe sich wieder die ursprüngliche Funktion mit dem Hazard.

*) Es wurde nur die Laufzeit 1 x t_p des Inverters berücksichtigt, ergänzen Sie das Zeitdiagramm unter der Voraussetzung gleicher Laufzeiten 1 x t_p für alle Gatter!

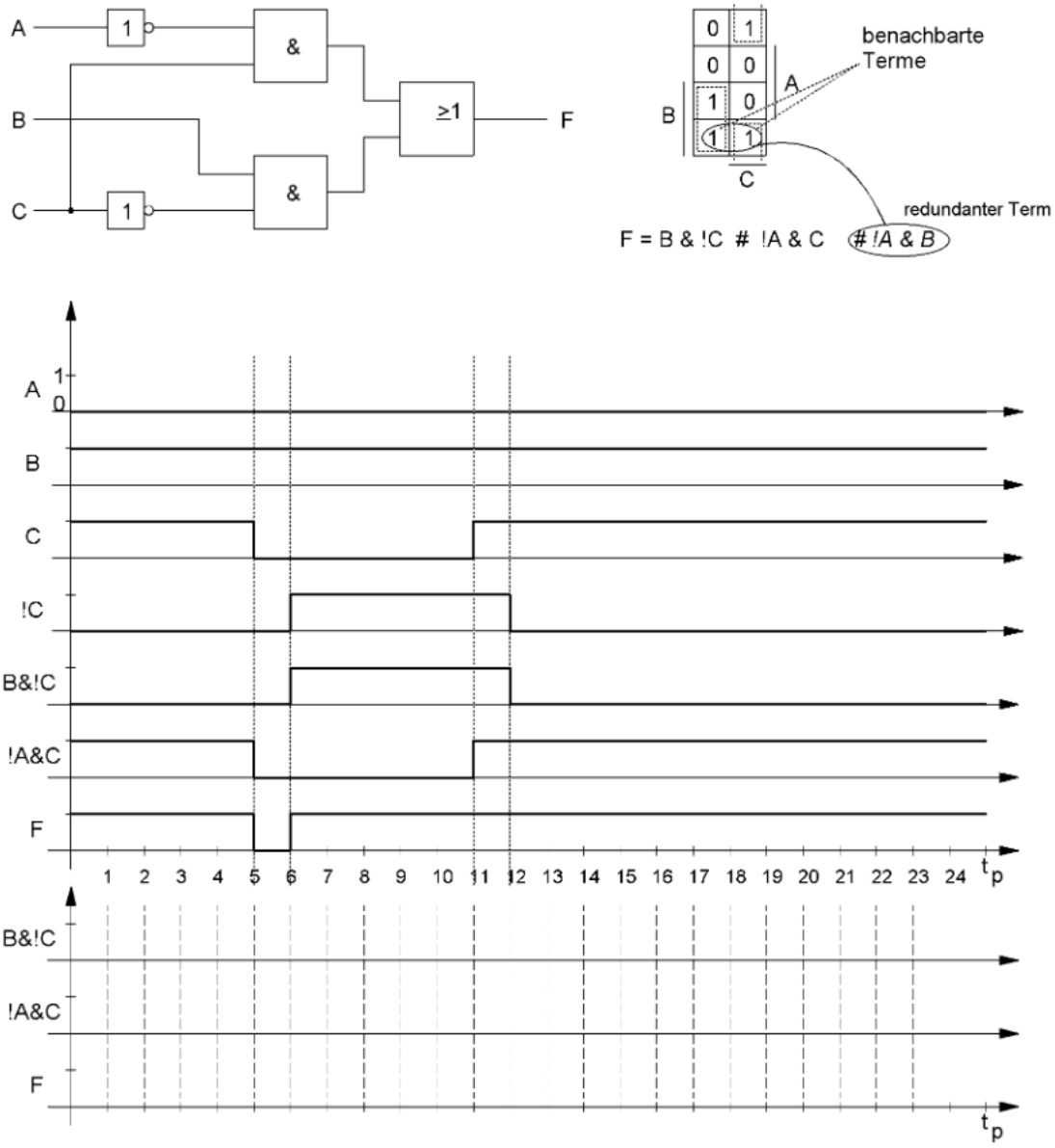


Bild 4.1 Beispielschaltung und Zeitdiagramm mit nicht invertierter und invertierter Variablen C

4.2.1 Timingsimulation vor dem Plazieren und Routen

Hier werden die resultierenden Funktionsgleichungen für die Simulation herangezogen und es existieren Angaben über die zu erwartenden Gatterlaufzeiten der verwendeten Gatter. Dadurch lassen sich bereits die meisten dynamischen Fehler lokalisieren.

4.2.2 Timingsimulation nach dem Plazieren und Routen

Jetzt ist der Entwurf auf einen konkreten Baustein abgebildet worden. Die Bausteinhersteller liefern für ihre unterschiedlichen Bausteine sehr genaue Simulationsmodelle bezüglich Laufzeiten bei Verknüpfungen, bei Flipflops, aber auch bezüglich der Auswirkung der Länge von Signalpfaden oder der effektiven Impedanz der programmierten Verbindungen (Widerstand im Ein-Zustand bzw. Kapazität im Aus-Zustand), die eine viel exaktere Timingsimulation ermöglichen. Das Ergebnis der Simulation sind sog. Timing-Diagramme (traces), in denen Ausgangssignalwerte in Abhängigkeit vom Simulationszeitraum aufgetragen sind. Die vom Simulator gelieferten Diagramme werden mit den erwarteten Ausgangssignalen verglichen. Stimmen die Ergebnisse nicht überein, so kann einer der folgenden Fehler vorliegen:

- Entwurfsfehler

- Schaltungsbeschreibung fehlerhaft (falsche Netzliste)
- Zeitbedingungen falsch gewählt
- Fehler im Simulator oder den Synthesewerkzeugen (sehr unwahrscheinlich)

Sind die erwarteten und simulierten Ausgangssignale identisch, so ist die Schaltung für die verwendeten Modelle und Testmuster korrekt. Es erfolgt immer nur eine partielle Verifikation. Eine erschöpfende Verifikation, gerade bei umfangreichen Schaltungen, ist praktisch nicht möglich.

4.3 Fehlersimulation

Meist ist es nicht ausreichend, das korrekte Design einer Schaltung durch die oben beschriebenen Logik- bzw. Timingsimulation zu validieren. Nach der Herstellung (sprich hier Programmierung) muß jede Schaltung auch noch auf ihre korrekte elektrische Funktion überprüft werden. Dazu werden Testmuster benötigt, deren ausschließlicher Zweck das Erkennen elektrischer Fehler ist. Dementsprechend muß die Testmustermenge möglichst viele aller möglichen elektrischen Fehler aufdecken. Die Fehlersimulation ermöglicht dabei eine Aussage darüber zu machen, welche Fehler man mit einer bestimmten Testmustermenge erkennen kann und welche nicht.

Fehlermodelle

Die Fehlersimulation basiert auf einem einfachen Modell möglicher Schaltungsfehler. Folgende Annahmen liegen diesem Modell zugrunde:

1. Eine Schaltung enthält nur statische Fehler (dynamische Fehler, z.B. Beeinflussung der Schaltung durch Betriebsparameter wie Temperatur- oder Betriebsspannungsschwankungen, Wackelkontakte oder ähnliches werden nicht berücksichtigt).

2. Es existieren nur bestimmte Fehlerarten, z.B.:
   - Ein Schaltungsknoten hält ständig den Signalwert 0 (stuck-at-zero Fehler oder s-0)
   - Ein Schaltungsknoten hält ständig den Signalwert 1 (stuck-at-one Fehler oder s-1)
   - Es existiert ein Kurzschluß zwischen zwei Schaltungsknoten
   - Gatterein- oder Gatterausgänge sind nicht beschaltet

3. Eine Schaltung enthält maximal einen Fehler (Einzelfehlerannahme)

Bei der Fehlersimulation wird nun ein Fehler (nach obigem Fehlermodell) in die Schaltung eingebaut und eine Logiksimulation durchgeführt. Weicht das erhaltene Ergebnis vom zuvor ermittelten Ausgangsmuster im fehlerfreien Fall ab, so kann der injizierte Fehler mit Hilfe der verwendeten Testmuster erkannt werden. Stimmen die Ergebnisse überein, so reicht die verwendete Testmenge zur Feststellung des Fehlers nicht aus und es muß gezielt ein Testmuster berechnet werden und der Testmenge angefügt werden, um diesen Fehler zu erkennen.

4.4 Testmustergenerierung

Sowohl für die Logik- als auch für die Fehlersimulation wird eine Menge von Testmustern benötigt. Während die Ermittlung dieser Testmuster für den Funktionstest (Logiksimulation) zumindest für Schaltungen von geringer Komplexität noch recht einfach ist, gestaltet sich die Berechnung von Stimuli für die Fehlersimulation u.U. sehr schwierig. Bei der Fehlersimulation müssen Eingangsmuster gefunden werden, die injizierte Fehler an den Ausgängen sichtbar werden lassen. Diese Aufgabe wird besonders für Schaltwerke in der Regel so komplex, daß sie manuell kaum bzw. gar nicht lösbar ist. In der industriellen Praxis werden daher Algorithmen eingesetzt, die eine automatische Bestimmung von Testmustern für logische Schaltungen ermöglichen. Die in diesem Versuch verwendeten Beispielschaltungen sind von geringer Komplexität und es wird hier nur eine reine funktionale Simulation (Logiksimulation) durchgeführt. Die zur Durchführung dieser Simulation erforderlichen Testmuster können noch „von Hand“ ermittelt werden. Bei den Beispielen werden jeweils einfache Methoden zur Gewinnung der Testmuster für Schaltnetze und -werke vorgestellt.

5. Beispielaufgaben

Damit die Vorbereitung der im Labor zu lösenden Aufgaben leichter fällt, werden in diesem Kapitel die Lösung zweier Beispielaufgaben erklärt. Zunächst wird ein kombinatorisches Schaltnetz entworfen und in ABEL beschrieben. Die zweite Beispielaufgabe beschreibt den Entwurf eines Schaltwerkes mit ABEL, sowohl als Mealy- als auch als Moore-Schaltwerk. Es werden jeweils die Methoden zur Gewinnung von Testmustern für die funktionale Simulation vorgestellt.

5.1 Schaltnetz: Rohrpost

Aufgabenstellung:

Die Steuerung eines kleinen Rohrpostsystems ist zu entwerfen. Das System besteht aus vier Stationen (i = 1 .. 4), von denen jede sowohl Sender als auch Empfänger einer Rohrpostnachricht sein kann (Bild 5.1).

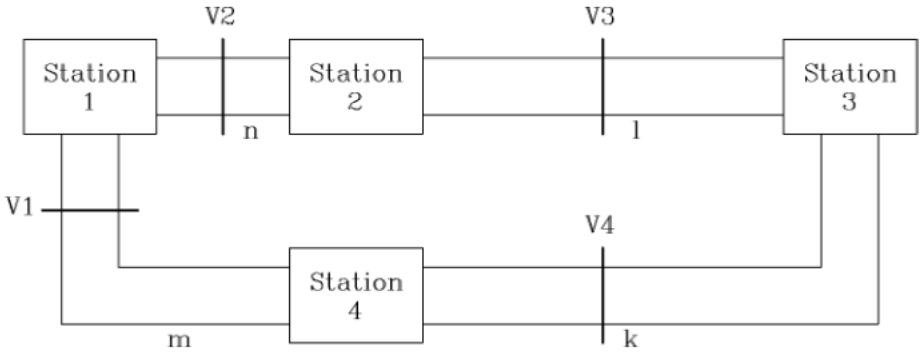


Bild 5.1: Blockschaltbild der Rohrpostanlage

Um eine Nachricht von einer Station zu einer anderen zu schicken, meldet die sendende Station der Steuerung sowohl ihre eigene Nummer, als auch die Nummer der Zielstation. Die Steuerung soll die vier in den Rohren zwischen den Stationen angebrachten Schließvorrichtungen V_j (j = 1 .. 4) so ansteuern, daß die Nachricht von der Sendestation zur Empfangsstation gelangen kann. Ein Signal $V_j = 0$ bedeutet dabei eine geöffnete Schließvorrichtung (Nachricht kann passieren), $V_j = 1$ schließt die Vorrichtung, so daß die Nachricht eine Station nicht mehr passieren kann. Der Eingang der Sendestation kann unberücksichtigt bleiben. Zusätzlich soll ein Signal R generiert werden, das die Bewegungsrichtung der Nachricht festlegt (R = 1: Bewegungsrichtung im Uhrzeigersinn), so daß eine möglichst kurze Strecke zwischen Sender und Empfänger zurückgelegt wird. Die Länge der Strecken zwischen den einzelnen Stationen ist durch die Variablen k, l, m und n gegeben. Für die Strecken soll gelten: $n < m < l < k$ und $k < (l + m + n)$. Sind Sende- und Empfangsstation identisch, soll ein Fehlersignal F erzeugt werden.

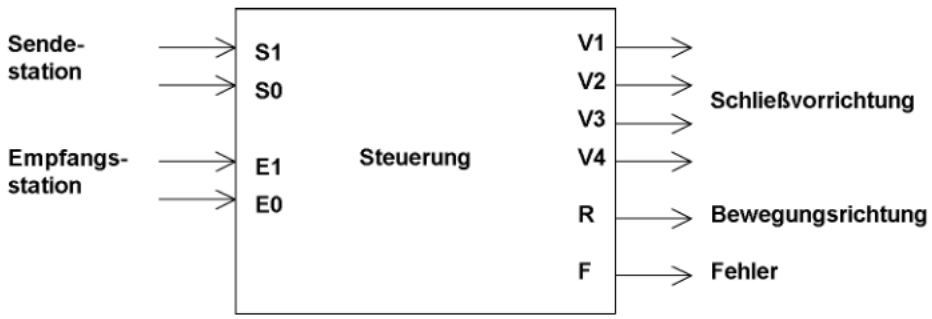


Bild 5.2: Blockschaltbild des Steuermoduls der Rohrpost

Die Kodierung der Stationsnummern soll dabei wie folgt vorgenommen werden:

Tabelle 5.1: Codierung der Stationsnummern

Station	Sendestation		Empfangsstation	
I	S1	S0	E1	E0
1	0	0	0	0
2	0	1	0	1
3	1	0	1	0
4	1	1	1	1

Aus bisher dargestellten Randbedingungen ergibt sich die Funktionstabelle 5.2 für die Eingangssignale S1, S0, E1 und E0.

Tabelle 5.2: Funktionstabelle der Rohrpoststeuerung

Nr.	S1	S0	E1	E0	V1	V2	V3	V4	R	F
0	0	0	0	0	X	X	X	X	X	1
1	0	0	0	1	X	0	1	X	1	0
2	0	0	1	0	X	0	0	1	1	0
3	0	0	1	1	0	X	X	1	0	0
4	0	1	0	0	1	0	X	X	0	0
5	0	1	0	1	X	X	X	X	X	1
6	0	1	1	0	X	X	0	1	1	0
7	0	1	1	1	0	0	X	1	0	0
8	1	0	0	0	1	0	0	X	0	0
9	1	0	0	1	X	1	0	X	0	0
10	1	0	1	0	X	X	X	X	X	1
11	1	0	1	1	1	X	X	0	1	0
12	1	1	0	0	0	1	X	X	1	0
13	1	1	0	1	0	0	1	X	1	0
14	1	1	1	0	X	X	1	0	0	0
15	1	1	1	1	X	X	X	X	X	1

X = don´t care

Die anschießende Minimierung mittels KV-Diagramm liefert die folgenden Produkttermvarianten für die Ausgänge:

$$R = \overline{S1} \cdot \overline{S0} \cdot \overline{E1} \vee S1 \cdot S0 \cdot \overline{E1} \vee S1 \cdot \overline{S0} \cdot E1 \vee \overline{S1} \cdot E1 \cdot \overline{E0}$$

$$F = \overline{S1} \cdot \overline{S0} \cdot \overline{E1} \cdot \overline{E0} \vee \overline{S1} \cdot S0 \cdot \overline{E1} \cdot E0 \vee S1 \cdot S0 \cdot E1 \cdot E0 \vee S1 \cdot \overline{S0} \cdot E1 \cdot \overline{E0}$$

$$V1 = \overline{S1} \cdot \overline{E1} \vee S1 \cdot \overline{S0}$$

$$V2 = S1 \cdot S0 \cdot \overline{E0} \vee S1 \cdot \overline{S0} \cdot E0$$

$$V3 = \overline{S1} \cdot E0 \vee S1 \cdot S0$$

$$V4 = \overline{S1}$$

Erzeugung der Testmuster für ein Schaltnetz
Bei einem Schaltnetz reicht es aus, auf die Wahrheitstabelle, die zur Erstellung der Schaltung benutzt wurde, zurückzugreifen und alle Kombinationen aus Eingangs- und Ausgangssignalen zu simulieren. Wenn allerdings die Wahrheitstabelle schon fehlerbehaftet ist, so wird sich der Fehler in die Simulation fortpflanzen und unentdeckt bleiben. Um dies zu vermeiden, sollte man die Testmustertabelle aus der Aufgabenstellung neu entwic??keln (Im Anhang B befindet sich das AHDL-File für die Rohrpoststeuerung.).

5.2 Beispielschaltwerk: Treppenhauslicht

Gegeben ist das Blockschaltbild einer Beleuchtungssteuerung (Bild 5.3).

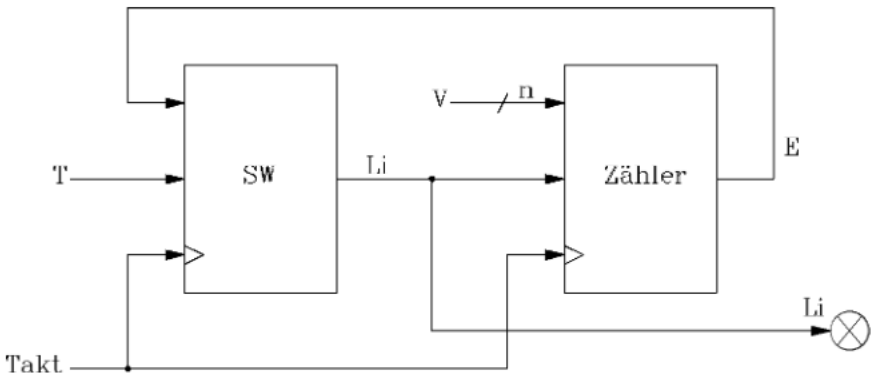


Bild 5.3: Blockschaltbild der Treppenhausbeleuchtung

Es ist nur der Block SW zu entwerfen. Der Zähler ist vorgegeben und arbeitet nach folgendem Prinzip: Bei gelöschtem Licht (Li = 0) ist der Zähler dauerhaft rückgesetzt. Es wird E = 0 ausgegeben. Ist die Beleuchtung eingeschaltet (Li = 1), so zählt der Zähler aufwärts, bis entweder der Vergleichswert V erreicht wird oder der Zähler durch Li = 0 wieder zurückgesetzt wird. Wird der Vergleichswert V erreicht, bleibt der Zähler auf diesem Stand stehen und erzeugt das Endesignal E = 1. Das synchrone Schaltwerk SW soll so entworfen werden, daß die Beleuchtung per Tastendruck (Taste gedrückt T = 1, Taste nicht gedrückt T = 0) abwechselnd eingeschaltet (Li = 1) und ausgeschaltet (Li = 0) werden kann. Diese manuelle Schaltfunktion soll unabhängig von der Dauer der Tastenbetätigung sein. Zusätzlich soll die Beleuchtung nach der vorgegebenen Zeit (wenn der Zähler das Signal E = 1 liefert) automatisch verlöschen.
Die Aufgabe ist als Mealy- und als Moore-Schaltwerk zu realisieren.

5.2.1 Realisierung als Moore-Schaltwerk

Aus der Aufgabenstellung resultiert das folgende Zustandsdiagramm für das Moore-Schaltwerk:

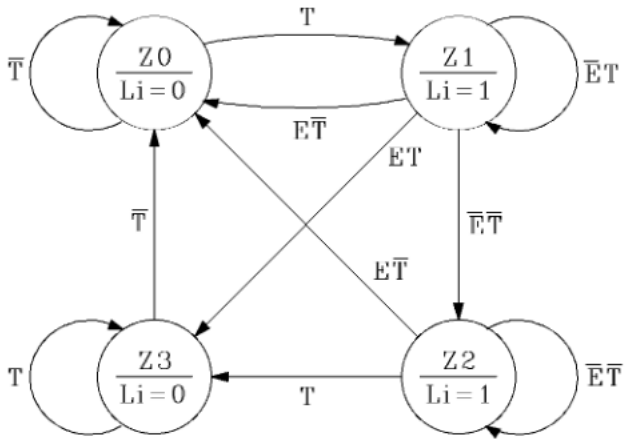


Bild 5.4: Zustandsdiagramm des Moore-Schaltwerks

Im Anhang D befindet sich das AHDL-File der Moore-Realisierung. ABEL erlaubt neben der Eingabe in Form einer Übergangstabelle (Tabelle 5.3), auch die Beschreibung der Zustandswechsel mit den Sprachelementen „case“, „if then else“ und „goto“.

5.2.2 Realisierung als Mealy-Schaltwerk

Für die Realisierung als Mealy-Schaltwerk ergibt sich folgendes Zustandsdiagramm:

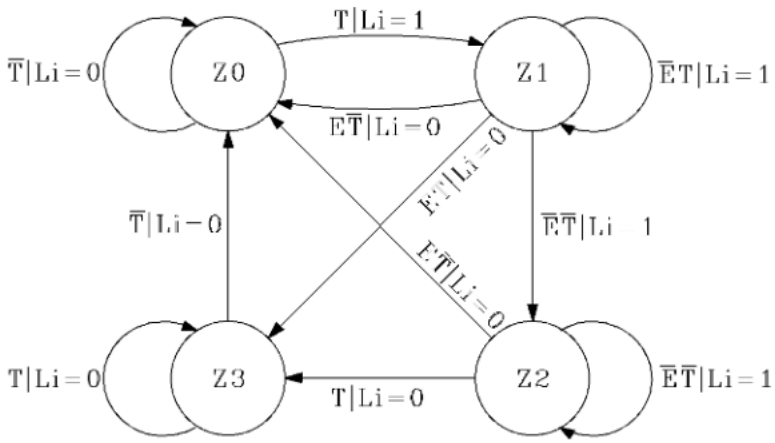


Bild 5.5: Zustandsdiagramm des Mealy-Schaltwerks

Der generelle Unterschied zum Moore-Schaltwerk besteht darin, daß aufgrund der nicht gepufferten Ausgänge die Ausgangswerte direkt von den Eingängen abhängen können. Das bedeutet, einem bestimmten Zustand kann kein fester Ausgangswert zugeordnet werden, es sei denn, es besteht keine Korrelation zwischen Eingang und Ausgang wie im Zustand Z3.

Die Umsetzung der Mealy-Bedingungen in AHDL ist sehr anschaulich (Anhang E). Die Umsetzung des Zustandsdiagramms wird mit der zusätzlichen Sprachkomponenten „with“ erreicht, welche eine Ausgangsänderung **vor** dem Wechsel in einen Folgezustand bewirkt.

Die Zustandsübergangstabelle der Beleuchtungssteuerung ist in der folgenden Tabelle dargestellt:

Tabelle 5.3: Zustandsübergangstabelle der Beleuchtungssteuerung

E	T	Q_1^t	Q_0^t	Z^t	Q_1^{t+1}	Q_0^{t+1}	Z^{t+1}	Ausgang Li	
								Mealy	Moore
X	0	0	0	Z0	0	0	Z0	0	0
X	1	0	0	Z0	0	1	Z1	1	0
0	1	0	1	Z1	0	1	Z1	1	1
1	0	0	1	Z1	0	0	Z0	0	1
1	1	0	1	Z1	1	1	Z3	0	1
0	0	0	1	Z1	1	0	Z2	1	1
0	0	1	0	Z2	1	0	Z2	1	1
1	0	1	0	Z2	0	0	Z0	0	1
X	1	1	0	Z2	1	1	Z3	0	1
X	1	1	1	Z3	1	1	Z3	0	0
X	0	1	1	Z3	0	0	Z0	0	0

Erzeugung von Testmustern für ein Schaltwerk

Bei einem Schaltwerk ist, gegenüber einem Schaltnetz, zusätzlich noch der Zustand zu berücksichtigen, in dem sich das Schaltwerk zu einem bestimmten Zeitpunkt befindet. Die Eingangsmuster müssen so gewählt werden, daß jeder Übergang von einem Zustand zum nächsten mindestens einmal durchlaufen wird. Dabei steht T für gedrückten Taster und E für das Endesignal. Wenn man die Zustände mit A, B, C und D kodiert, ergibt sich folgendes Zustandsdiagramm für die Übergänge:

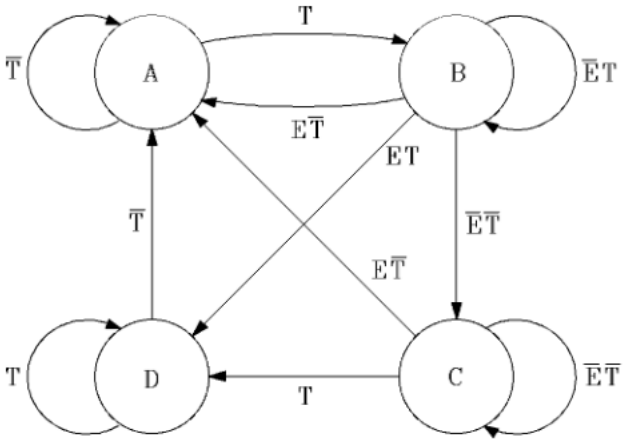


Bild 5.6: Zustandsdiagramm für das Schaltwerk

Die Tabelle 5.4 enthält die ausgewählten Testmuster für das Beispielschaltwerk. Um alle von einem Zustand (z.B. B) ausgehenden Zustandsübergänge durchlaufen zu können, ist es im Beispiel notwendig, manche Zustandsübergänge (z.B. von A nach B) mehrfach durchzuführen. In der Tabelle steht X für don´t care, d.h. der Pegel des entsprechenden Signals hat keinen Einfluß auf das Übergangsverhalten und kann beliebig gewählt werden. Falls für den Zustandsübergang mehrere Möglichkeiten existieren, d.h. mehrere Übergangsbedingungen sind ODER verknüpft, so sind alle Bedingungen getrennt zu testen. Dieser Zustandsübergang muß also mehrfach durchlaufen werden.

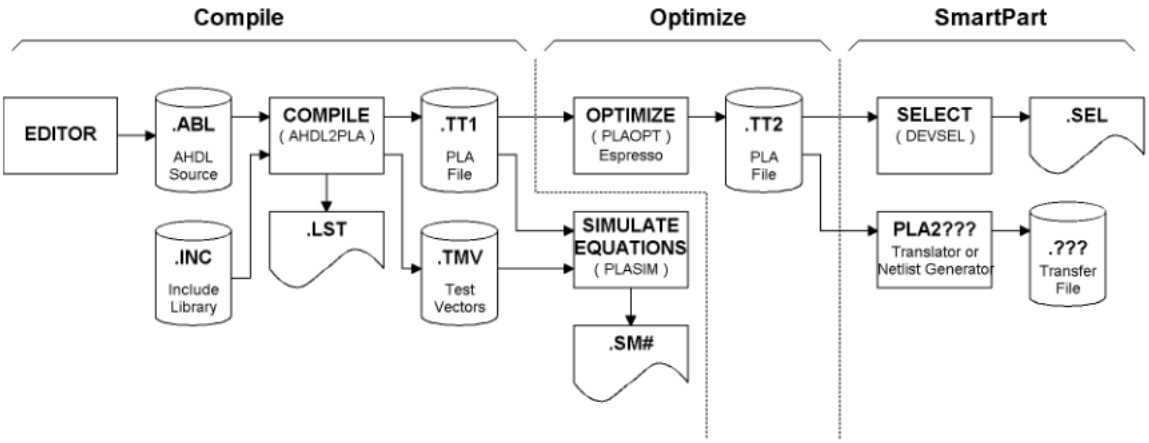
Tabelle 5.4: Testmuster für das Beispielschaltwerk

Zustand	T	E	Folgezustand
A	0	X	A
A	1	X	B
B	1	0	B
B	0	1	A
A	1	X	B
B	1	1	D
D	1	X	D
D	0	X	A
A	1	X	B
B	0	0	C
C	0	0	C
C	0	1	A
A	1	X	B
B	0	0	C
C	1	X	D

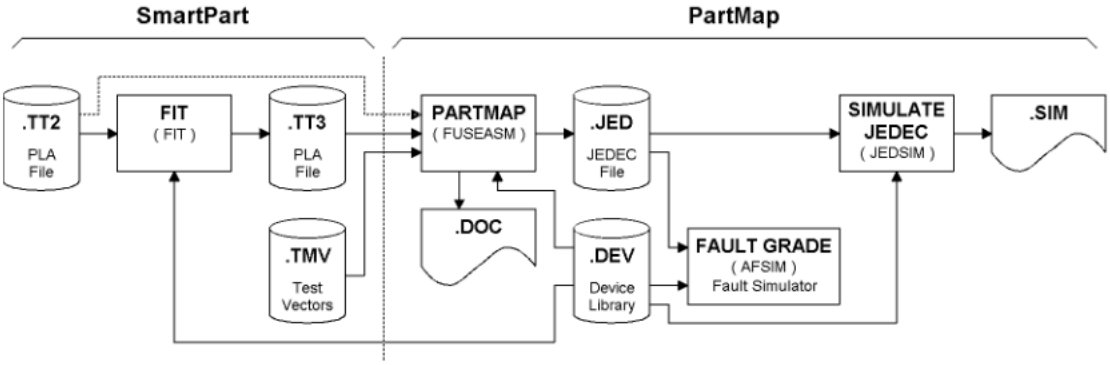
5.3 Schaltungsentwicklung mit easyABEL

Erstellen sie ein Unterverzeichnis in dem die jeweilige Schaltungsbeschreibung abgespeichert wird. Die Eingabe in ABEL kann auf zwei unterschiedliche Weisen erfolgen. Man kann die Funktionstabelle direkt eingeben oder man gibt die (vereinfachten) Booleschen Gleichungen für die Ausgangsvariablen vor. Sowohl die Tabelle, als auch die Gleichungen reichen in Verbindung mit der Definition der Ein- und Ausgänge aus, das Schaltnetz vollständig zu beschreiben. Im Anhang sind die AHDL-Source-File der Beispielschaltungen abgedruckt.
Für die Eingabe des AHDL-Files wird entweder der integrierte Editor oder ein beliebiger anderer Texteditor (z.B. den DOS-Editor) verwendet.
Hier wird beschrieben, wie die verschiedenen easyABEL Verarbeitungsmodule (processing modules) zusammenwirken und wie sie einzusetzen sind. Dabei kann nicht auf alle Möglichkeiten eingegangen werden. In der folgenden Abbildung ist der Programmablauf dargestellt.

Architecture-Independent



Device-Specific



Ausgabedateien (output files):

source_file.abl	AHDL-Datei (source file) des Designs
source_file.tmv	Datei mit Testvektoren
module_name.tmv	Datei mit Testvektoren die in der AHDL-Datei vorgegeben sind
source_file.lst	Datei des Listings nach dem Compilieren
source_file.dmc	Design manager control Datei, die den Zusammenhang zwischen allen Dateien beschreibt
source_file.aop	Alle gewählten Optionen sind hier gespeichert
module_name.tt*n*	PLA Dateien, die von easyABEL verwendet werden
module_name.sel	Liste der möglichen Bausteine (-> SmartPart)
module_name.fit	Zuordnung der Pins durch den Fitter
module_name.eq*n*	Gleichungen die von PAL2EQN erzeugt wurden
module_name.sm*n*	Simulationsergebnisse von PLASim (Funktionale Simulation)
device_id.sim	Simulationsergebnisse von JEDSim (Simulation nach dem „Mappen“)
device_id.jed	Programmierdatei im JEDEC-Format
module_name.doc	Dokumentation des Designs nach Fuseasm

Bild 5.7 Programmablauf des architekturunhabhängigen Teils und des bausteinspezifischen Teils Übersicht Ausgabedateien

Die verschiedenen Programmodule, die in der easyABEL-Entwurfsumgebung zusammengefaßt sind, erfüllen folgende Aufgaben:

- Syntaxüberprüfung
- Synthese des Designs
- Simulation der Gleichungen
- Logikminimierung
- Erzeugung einer Datei zur Programmierung des Bausteins
- Simulation des Designs für den programmierten Baustein
- Dokumentation des Designs

Die Oberfläche der easyABEL-Entwurfsumgebung hat folgendes Aussehen:

Es folgt eine Auflistung der in der easyABEL-Entwurfsumgebung (Design Environment) zusammengefaßten Programmteile. Zuerst ist der entsprechende Menübefehl und danach das entsprechende Programm aufgeführt:

1. **Compile** (AHDL2PLA) - Compiliert die AHDL-Datei, führt Syntaxüberprüfung durch, expandiert Macros, reagiert auf Directiven und synthetisiert das Design. Es erfolgt eine Konvertierung in das interne PLA Format.

2. **Simulate Equations** (PLASIM) - Funktionale Simulation der Gleichungen unter Verwendung der PLA-Datei und der Datei mit Testvektoren.

3. **Optimize** (PLAOPT) - Logikminimierung unter Verwendung der PLA-Datei.

4. **PartMap** (FUSEASM) - Erzeugt die JEDEC-Datei zur Programmierung und die Dokumentationsdatei aus der PLA-Datei.

5. **Simulate JEDEC** (JEDSIM) - Funktionale Simulation des Designs unter Verwendung der aktuellen JEDEC-Datei zur Programmierung und der Datei mit den Testvektoren.

6. **SmartPart** (DEVSEL und FIT) - Optionaler Programmteil zur Auswahl möglicher Bausteine und um das Design auf den Baustein abzubilden. Es erfolgt auch eine Zuordnung der Pins.

5.3.1 Starten

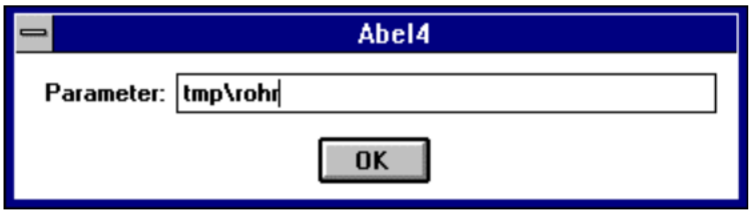
Das easyABEL Programm AHDL2PLA führt eine Syntaxüberprüfung durch und konvertiert die AHDL-Datei in ein PLA-Format, welches von den nachfolgenden Programmteilen benötigt wird.

Um das aktuelle Design zu bearbeiten, kann man entweder das Programm mit der AHDL-Datei aufrufen oder in der Entwurfsumgebung über den Menüpunkt **File** mit dem Untermenüpunkt **Open** eine Datei öffnen.

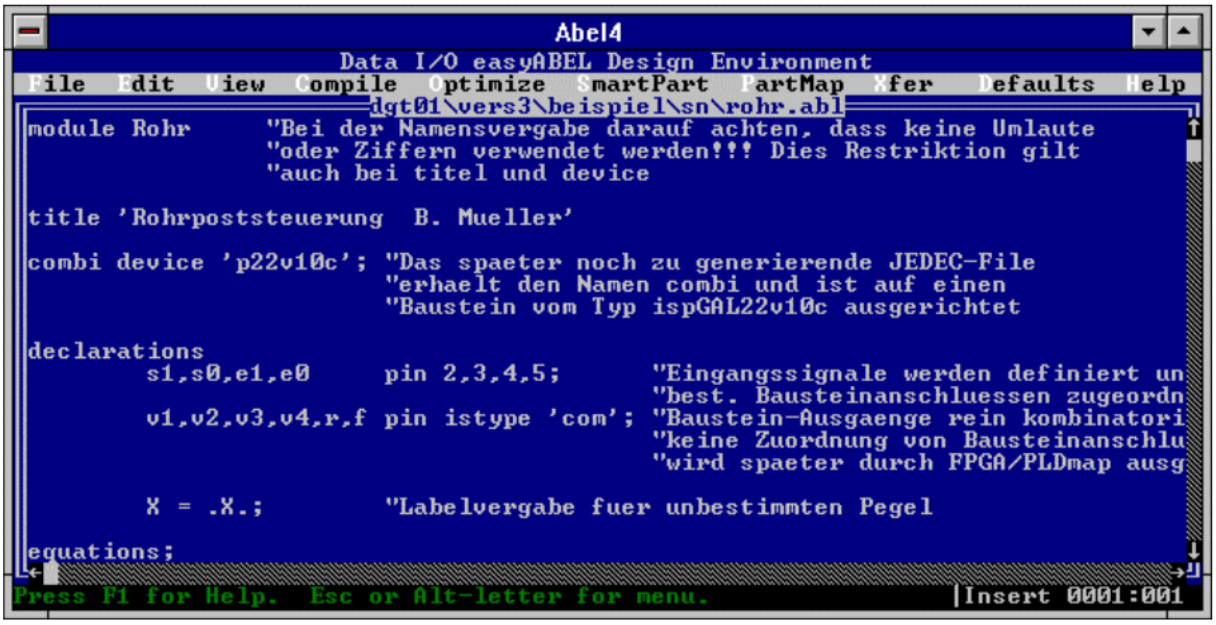
a) Aufruf mit aktueller Datei:

Unter Windows Ordner mit **easyAbel** wählen (auf Ordner **easyAbel** doppelklicken) und auf Icon **Abel4** doppelklicken.

Als Parameter wird der Name des zu bearbeitenden Designs (**rohr**.abl) eingegeben (einschließlich des entsprechenden Unterverzeichnisses „tmp\“) und dann ← gedrückt.



Daraufhin wird die easyABEL-Entwurfsumgebung mit der angegebenen Datei aufgestartet:



b) Aufruf ohne Datei:

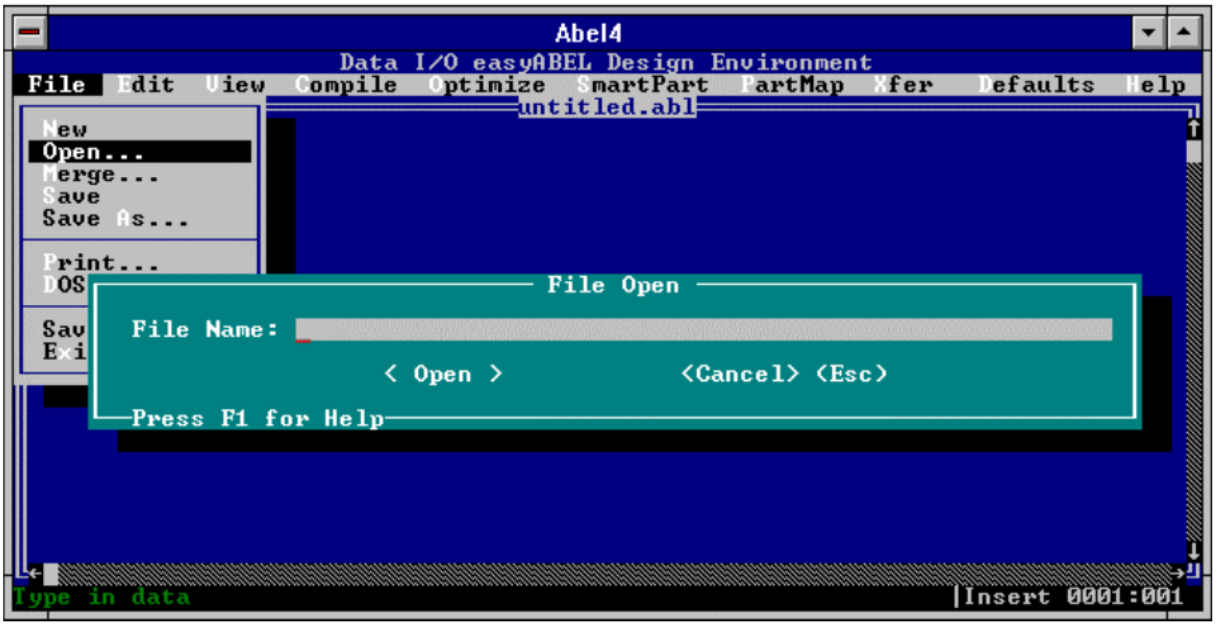
Vorgehensweise wie unter a). Es wird jedoch kein Parameter angegeben. Dadurch wird nur die easyABEL-Entwurfsumgebung aufgestartet:

Jetzt muß über den Menüpunkt **File** und Untermenüpunkt **Open** die aktuelle Datei geöffnet werden.

Eingabe: *f* F

Eingabe: **O** oder über Cursor-Steuertasten (′, ≤) Untermenüpunkt „**Open**“ auswählen und ←.

Es erfolgt die Aufforderung einen Dateinamen einzugeben:



Nach Eingabe des gültigen Dateinamens einschließlich des entsprechenden Unterverzeichnisses < **Open** > wählen. Die Datei wird geladen.

Die Erläuterung für alle Menü- und Untermenüpunkte findet man im Kapitel 7 des easyABEL-Manuals. In den folgenden Ausführungen wird nur noch auf die hier relevanten Punkte in stark gekürzter Form eingegangen, da die Eingabe der Anweisungen und die resultierenden Anzeigen vergleichbar sind.

5.3.2 Compilieren

Das easyABEL Programm AHDL2PLA führt eine Syntaxüberprüfung durch und konvertiert die AHDL-Datei in ein PLA-Format, welches von den nachfolgenden Programmteilen benötigt wird.

1. Menüpunkt Compile: *f*X.

2. Untermenüpunkt Options...: O

3. Mit Tab bzw. den vier Cursortasten können die Optionen (hier z.B. „**Standard Listing**“) angewählt werden. Durch Druck auf die Leertaste (**space**) erfolgt die Auswahl. Die Auswahl muß mit **< OK >** abgeschlossen werden.

4. Mit dem Untermenüpunkt **Compile** wird das Übersetzen („Compilieren“) gestartet. Es erscheint ein Fenster, in dem Meldungen zum aktuellen Stand des Übersetzens angezeigt werden.

5. Wurden Syntaxfehler während des Übersetzens festgestellt, kann man sich mit dem Untermenüpunkt **Compiler Listing** im Menü **View** die Listing-Datei (myfile.lst) anschauen, in der diese Fehler markiert sind (größere Listings evtl. ausdrucken). Die Fehler müssen in der **Ausgangsdatei** behoben werden, bevor fortgefahren werden kann. ***Häufig wurde nur ein Semikolon an den entsprechenden Stellen vergessen.***

5.3.3 Funktionale Simulation

Mit easyABEL hat man zwei Möglichkeiten ein Design zu simulieren. Mit **Simulate Equations** wird eine funktionale Simulation (Logiksimualtion) der Logikgleichungen unter Verwendung der PLA-Datei durchgeführt. **Simulate Equations** ist architekturunabhängig, d.h. es werden keine bausteinspezifischen Informationen verwendet, auch nicht wenn bereits ein bestimmter Baustein ausgewählt wurde. Dies gilt auch für **Simulate Optimized** bzw. **Simulate Fitted Design**. Mit **Trace Options** können jeweils Optionen für die Simualtion bzw. Darstellung der Simulationsergebnisse vorgegeben werden. Nachdem die JEDEC-Datei erzeugt wurde kann aber auch unter Verwendung dieser Datei mit **Simulate JEDEC** das Design im aktuellen Baustein simuliert werden. Im **View** Menü kann man sich mit dem Untermenüpunkt **Simulation Results** die Datei mit den aktuellen Simulationsergebnissen (z.B. myfile.sm1) anschauen.

(Achtung! Wurde zwischenzeitig eine andere AHDL-Datei geladen, werden bei Anwahl von **View** **Simulation Results** ***alle Schritte einschließlich der Simulation für dieses neue Design automatisch ausgeführt.)***

5.3.4 Optimierung

Im Menü **Optimize** kann eine Minimierung des Designs nach entsprechenden Vorgaben durchgeführt werden.

5.3.5 JEDEC-Datei

Wenn ein architekturunabhängiges Design bearbeitet wurde muß zuerst mit **SmartPart** ein geeigneter Baustein zur Realisierung ausgewählt werden. Mit dem Untermenüpunkt **Fit** wird der Entwurf in einen auszuwählenden oder vorgegebenen Baustein eingepaßt (“Fitting“). Die Zuordnung von Anschlüssen erfolgt automatisch oder kann vorgegeben werden. Wurde bereits in der AHDL-Datei ein Baustein (**Device**) ausgewählt und eine Zuordnung der Signale zu Anschlüssen (**Pins**) vorgenommen wird dies hier berücksichtigt. Ist die gewählte Zuordnung nicht realisierbar wird wenn möglich eine geänderte realisierbare Zuordnung vorgeschlagen.

Im Menü **PartMap** wird mit **FPGA/PLDmap** der Entwurf auf den vorgegebenen Baustein abgebildet (“Mapping“), d.h. es wird die Programmierung festgelegt. Die Untermenüpunkte **Fit** und **FPGA/PLDmap** sind die Einzigen, in denen bausteinspezifische Vorgaben verarbeitet werden. Es wird die JEDEC-Datei (myjedecfile.jed) zur Programmierung des Bausteins erzeugt. Mit dem Untermenüpunkt **Simulate JEDEC** kann eine bausteinspezifische Simulation (jedoch nur funktionale Simulation, keine Timing Simulation) durchgeführt werden.

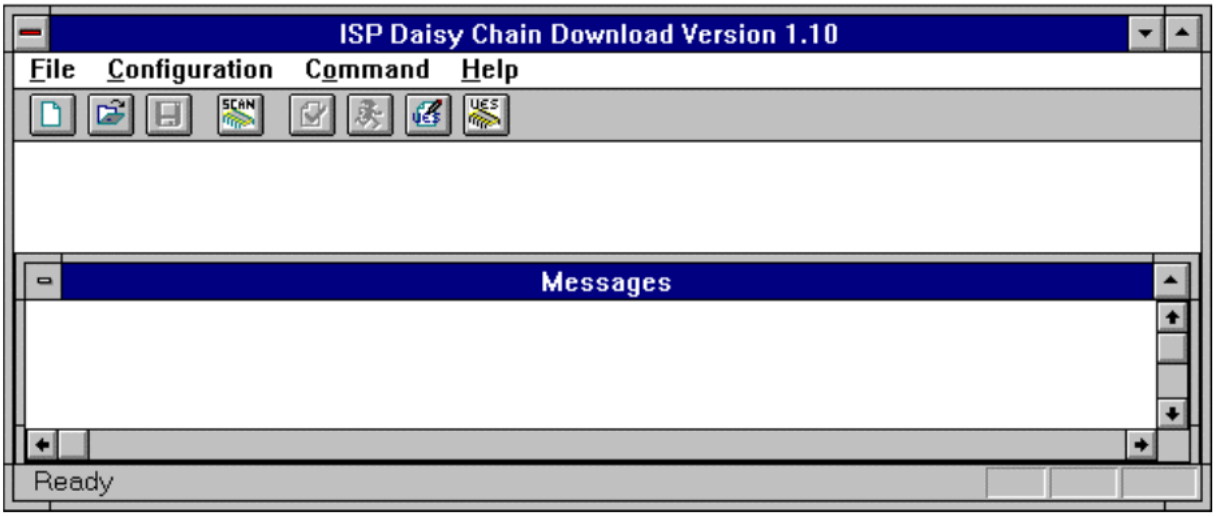
Die für das Entwurfsbeispiel der Rohrpost erzeugten Ausgabedateien sind im Anhang C als Beispiel zusammengestellt.

5.3.6 Programmierung des ispGAL22V10

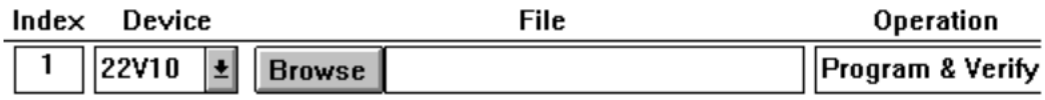
Wenn das Design erfolgreich simuliert wurde, wird die JEDEC-Datei (myjedecfile.jed) mit dem Programm **Download** von Lattice über die parallele Schnittstelle in den EPLD-Baustein geladen (der Baustein wird dadurch programmiert). Die Hardware kann dann auf dem Experimentierbrett ausgetestet werden.

Unter der Programmgruppe Lattice befindet sich das Programm ***ISP***IDCD zum Download der erzeugten JEDEC-Datei in das ispGAL22V10, das bereits in einem Baustein des Laborsystems steckt. Der Laborbaustein muß mit einem 25 poligen Kabel mit der parallelen Schnittstelle LPT2 des Arbeitsplatzrechners verbunden sein und die Betriebsspannung muß eingeschaltet sein.

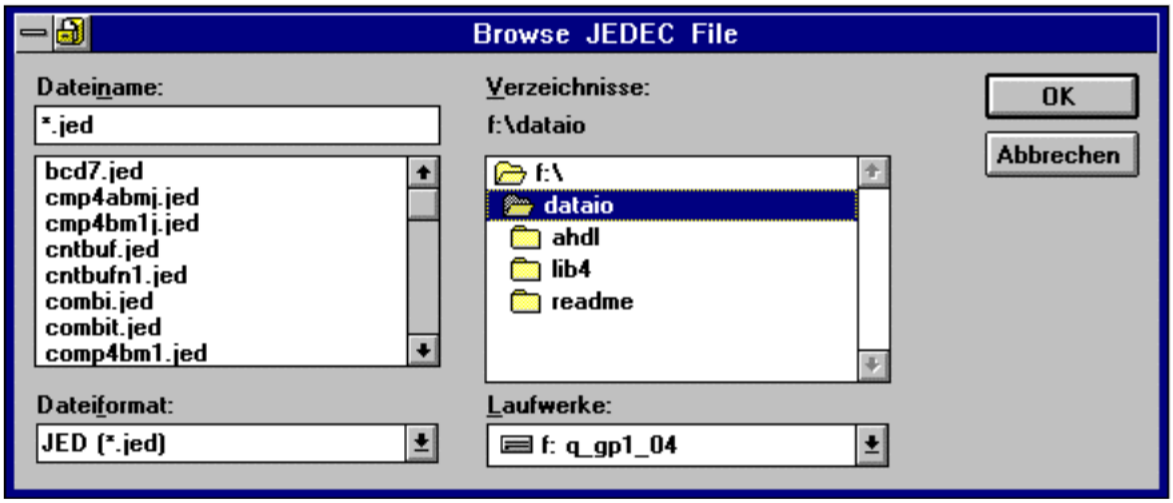
Wenn das Programm gestartet wurde erscheint folgende Oberfläche:



Um die aktuelle Konfiguration vorzugeben, klicken Sie auf das Icon „**SCAN**“ in der oberen Funktionsleiste. Dadurch wird der angeschlossene Baustein automatisch erkannt und die entsprechenden Einstellungen zur Programmierung vorgenommen. Es erscheint ein zusätzliches Fenster mit der Maske zur Bearbeitung des Bausteins.



Im Feld **File** wird der Name der zu programmierenden JEDEC-Datei eingetragen. Mit **Browse** können die Verzeichnisse auch nach JEDEC-Dateien durchsucht werden.

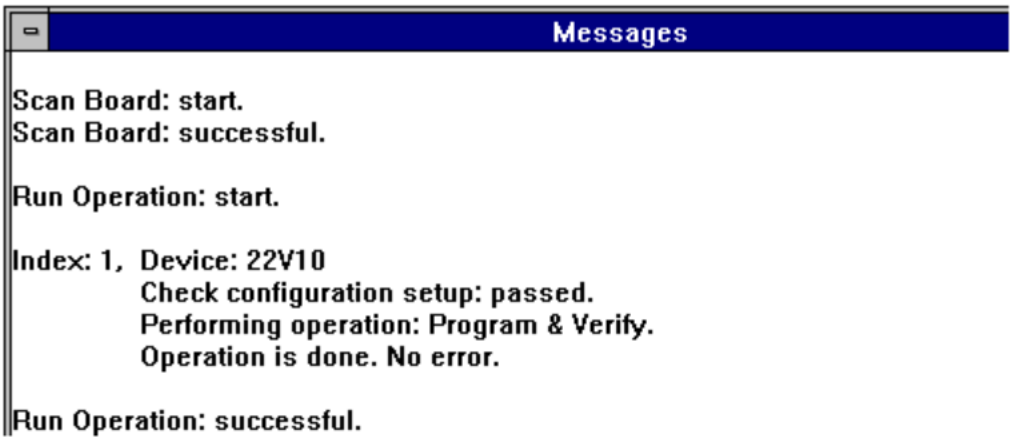


Wenn eine JEDEC-Datei ausgewählt wurde kann sie mit „OK“ übernommen werden.

Im Feld **Operation** ist als Operation **Program & Verify** vorzugeben. Klicken Sie jetzt auf das Icon „**RUN**“

zum Starten der Programmierung.

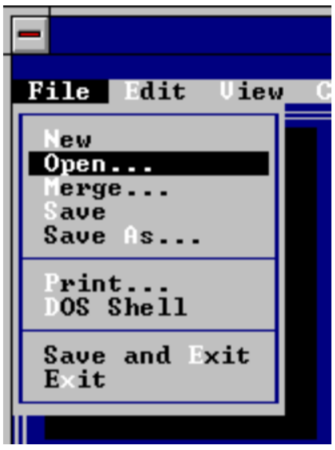
Im Fenster **Messages** wird der Fortgang der Programmierung angezeigt. Während der Programmierung leuchtet die rote Leuchtdiode des Laborbausteins. Die grüne Leuchtdiode dient zur Anzeige der Betriebsbereitschaft.



Wenn die Programmierung abgeschlossen ist erlischt die rote Leuchtdiode. Der Baustein ist jetzt mit der aktuellen Datei programmmiert und kann ausgetestet werden. Die Programmierung bleibt auch nach Ausschalten der Stromversorgung erhalten. Im Anhang H finden Sie eine Kurzanleitung des Entwicklungsablaufs.

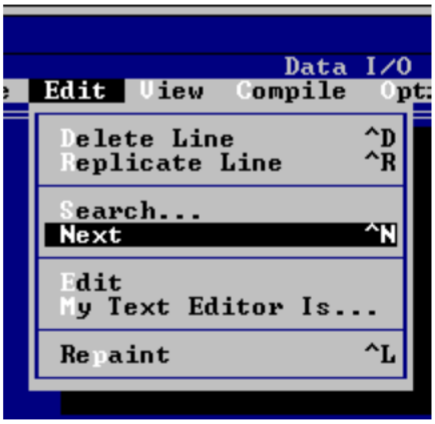
5.3.7 Die Menüs der easyABEL-Entwurfsumgebung

a) File



New:	Öffnen einer neuen ABEL-Datei.
Open:	Öffnen einer bereits existierende Datei.
Merge:	Zusammenfügen von Dateien.
Save:	Abspeichern einer ABEL-Datei.
Save As:	Abspeichern einer ABEL-Datei unter neuem Namen.
Print:	Drucken einer Datei.
DOS Shell:	Aufruf der DOS Shell (kann mit „exit“ wieder verlassen werden).
Save and Exit:	Verlassen der Entwurfsumgebung mit Speicherung der aktuellen Änderungen in der Datei.
Exit:	Verlassen der Entwurfsumgebung ohne Speicherung.

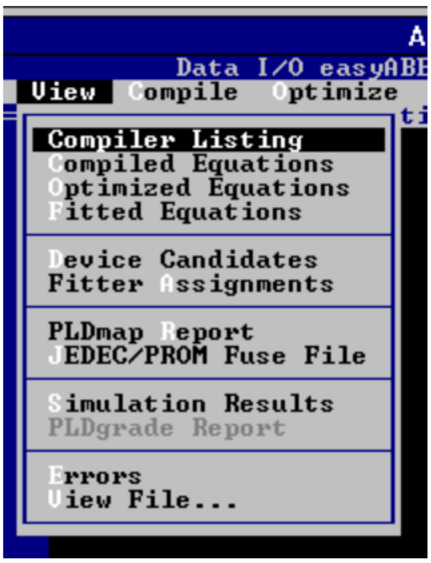
b) Edit



Befehle des integrierten Editors.

Mit „My Text Editor Is“ kann ein alternativer Editor (allerdings nur ein unter MS-DOS lauffähiger) eingesetzt werden.

c) View



Es können die während der verschiedenen Entwurfsschritte erzeugten Dateien zur Anzeige ausgewählt werden.

d) Compile

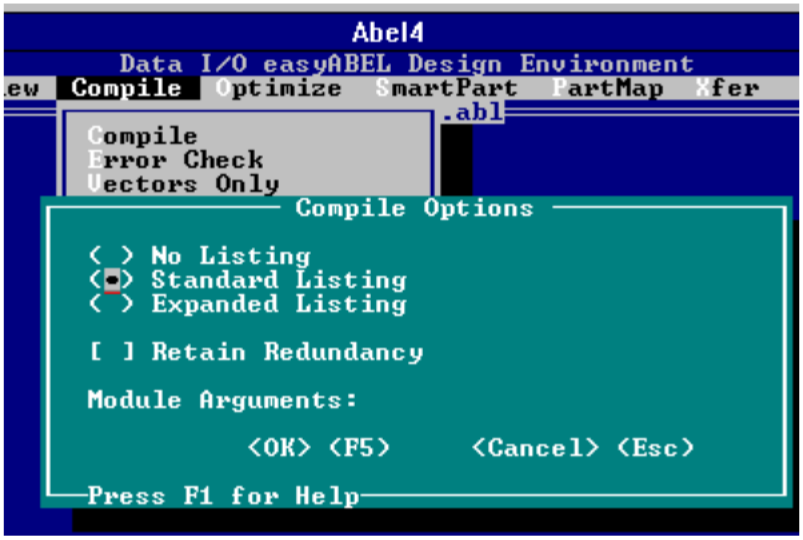
Aufruf des PLD-Compilers.

Compile:	Die ABEL-Datei wird übersetzt (compiliert).
Error Check:	Es erfolgt nur eine Syntaxüberprüfung.
Vectors Only:	Nur die Testvektoren für die Simulation werden übersetzt.
Options:	Es können Optionen ausgewählt werden (s.u.).

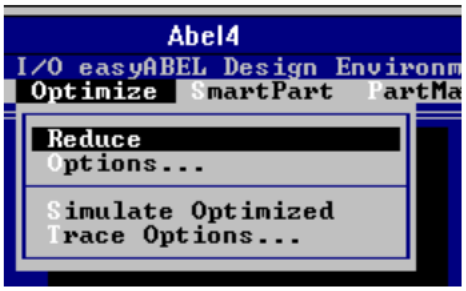
Simulate Equations: Funktionale Simulation nach dem Übersetzen.

Re-Simulate:	Erneute Simualtion
Trace Options:	Vorgaben für die Simulation.

d1) Compile Options



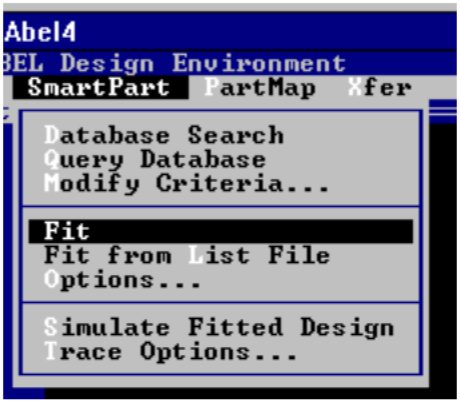
e) **O**ptimize



Optimierung des Entwurfs.

Einzelheiten siehe easyABEL-Handbuch.

f) SmartPart



Programmteil zur Auswahl eines, für den aktuellen Entwurf, geeigneten Bausteins.

Abbilden der aktuellen Entwurfsdaten auf den entsprechenden Baustein und Festlegung Der Anschlüsse für die Ein- und Ausgänge.

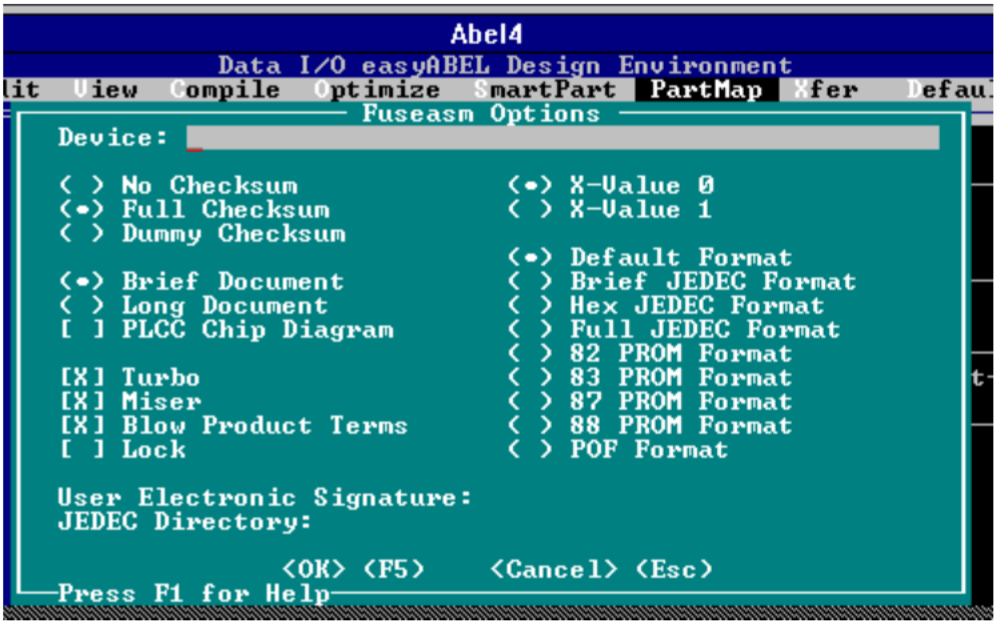
Simulation des „gefitteten“ Entwurfs.

g) PartMap

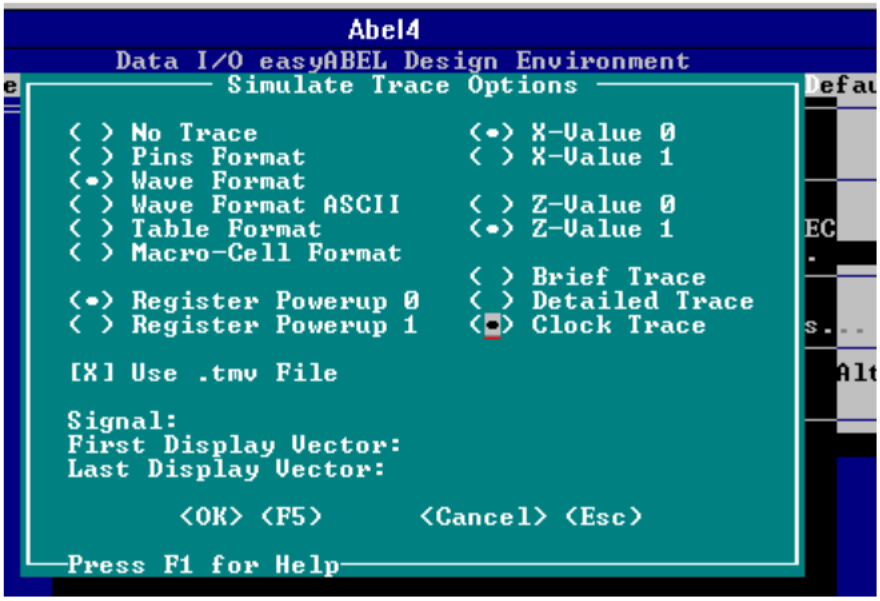
Erzeugung der zur Programmierung benötigten JEDEC-Datei. Festlegung der Programmierung („Mapping“).

Funktionale Simualtion des Entwurfs nach der Festlegung der Programmierung.

g1) Fuseasm Options

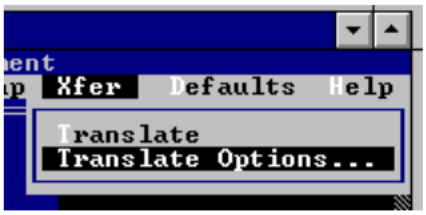


g2) Simulate Trace Options



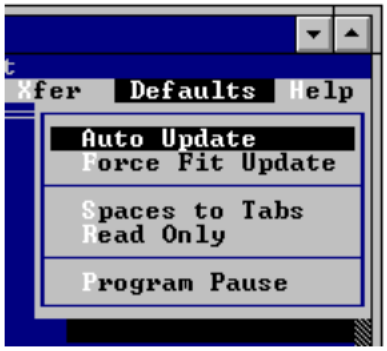
h) **X**fer

Übersetzung in anderes Netzlistenformat.

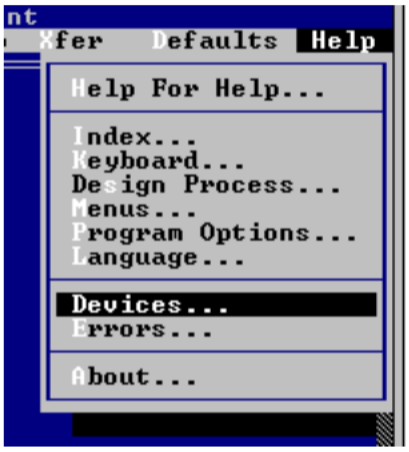


i) **D**efaults

Vorgaben für die Bearbeitung des Entwurfs.



j) **H**elp



Hilfe zum Ablauf des Entwurfs, den Menüs, der Sprache, den Programmoptionen, den verfügbaren Bausteinen und den Fehlermeldungen.

5.4 Einführung in ABEL

Hier soll an Beispielen gezeigt werden, wie einfach es ist, mit ABEL die verschiedensten digitalen Grundfunktionen, Schaltnetze und Schaltwerke zu beschreiben.

5.4.1 Beschreibung kombinatorischer Logik (Schaltnetze)

a) Logische Operatoren - Gatter

Die logische Grundoperatoren werden in ABEL einfach als !, # und &, für INVERTIERUNG, ODER-Verknüpfung und UND-Verknüpfung, dargestellt. Kombinatorische Logik wird dann durch ein einfaches Gleichheitszeichen (=) beschrieben. Links vom Gleichheitszeichen steht der Name der (Ausgangs-) Funktion und rechts die (Eingangs-) Variablen mit den Operatoren für die Verknüpfungen:

```
NOT_A = !A;                 INVERTER
A_OR_B= A # B;              ODER-Gatter
A_AND_B = A & B;            UND-Gatter
A_NOR_B = ! ( A # B );      NOR-Gatter
A_NAND_B = ! ( A & B );     NAND-Gatter
A_EXOR_B = A $ B;           EXOR-Gatter ( ANTIVALENZ )
A_EXNOR_B = A !$ B;         EXNOR-Gatter ( ÄQUIVALENZ )
```

b) Wahrheitstabellen

Die Beschreibung der gleichen Funktionen in Form einer Wahrheitstabelle (truth table) würde folgendermaßen aussehen:

```
truth_table ([ B, A ] -> [ NOT_A, A_OR_B, A_AND_B, A_NOR_B, A_NAND_B, A_EXOR_B, A_EXNOR_B ])
              [ 0, 0 ] -> [     1,      0,       0,       1,        1,        0,         1 ];
              [ 0, 1 ] -> [     0,      1,       0,       1,        0,        1,         0 ];
              [ 1, 0 ] -> [     1,      1,       0,       1,        0,        1,         0 ];
              [ 1, 1 ] -> [     0,      1,       1,       0,        0,        0,         1 ];
```

c) Relationen

Folgende Operatoren zur Darstellung von Relationen sind vorhanden:

```
==      gleich ( doppeltes Gleichheitszeichen )
!=      ungleich
<       kleiner als
<=      kleiner als oder gleich
>       größer als
>=      größer als oder gleich
```

Relationale Operatoren können zur Realisierung von kombinatorischer Logik aber auch bei Abfragen eingesetzt werden:

```
VECTOR_A = [A3, A2, A1, A0 ];
VECTOR_B = [B3, B2, B1, B0 ];

A_EQ_B = VECTOR_A == VECTOR_B;
A_NE_B = VECTOR_A != VECTOR_B;
A_NE_B = ! ( VECTOR_A == VECTOR_B );
..
oder

if ( VECTOR_A == VECTOR_B ) then ...
..
```

d) Arithmetische Operatoren

Arithmetische Operatoren sind ebenfalls vorhanden. Nähere Informationen dazu findet man im **easyABEL users manual** und in der **easyABEL Referenz**.

e) Multiplexer und Decoder

e1) Multiplexer

Ein Multiplexer für zwei 4 Bit breite Vektoren läßt sich z.B. folgendermaßen beschreiben:

```
DATA_0 = SEL & A0 # !SEL & B0;
DATA_1 = SEL & A1 # !SEL & B1;
DATA_2 = SEL & A2 # !SEL & B2;
DATA_3 = SEL & A3 # !SEL & B3;
```

e2) Decoder

Einen 1 aus 16 Decoder kann man wie folgt beschreiben:

```
DECODE_0 = !A3 & !A2 & !A1 & !A0;
DECODE_1 = !A3 & !A2 & !A1 & A0;

..
DECODE_15 = A3 & A2 & A1 & A0;
```

5.4.2 Beschreibung von getakteter Logik und Registern (Schaltwerke)

a) Register (Flipflops, FF)

Einfache Register (als Beispiel ein D-Flipflop) ergeben sich folgendermaßen:

```
FF_A.D = DATA_INPUT;        oder vereinfacht        FF_A := DATA_INPUT;
FF_A.CLK = CLOCK;           ( nur für D-FFs )       FF_A.CLK = CLOCK;
FF_A.RST = RESET;                                   FF_A.RST = RESET;
FF_A.Q ist der Ausgang des Flipflops.
```

Hier handelt es sich um ein einfaches D-Flipflop, dessen Vorbereitungseingang .D an DATA_INPUT, der Takteingang .CLK an das Signal CLOCK und der Rücksetzeingang .RST an das Signal RESET angeschlossen sind. Es werden D-, T-,RS- und JK-Flipflops unterstützt. Durch die Vektorschreibweise ergeben sich erheblich Vereinfachungen in der Beschreibung (siehe auch **easyABEL users manual**).

b) Schieberegister

Folgendermaßen lassen sich Register kaskadieren:

```
A := DATA_INPUT;
B := A;
C := B;
D := C;
A.CLK = CLOCK;
B.CLK = CLOCK;
C.CLK = CLOCK;
D.CLK = CLOCK;
A.RST = RESET;
B.RST = RESET;
C.RST = RESET;
```

D.RST = RESET;

Wenn die Signale A,B,C und D als Ausgänge (outputs) deklariert sind, werden sie mit jeweils mit einem Anschluß (pin) des Bausteins verbunden. Wenn sie als Knoten (nodes) deklariert sind, handelt es sich um interne Signale und sie werden durch die Software „vergraben“ (buried).

c) Zähler

Zähler lassen sich auf verschiedene Weisen beschreiben. Wenn sie als Automaten beschrieben werden sollen, wird durch das Schlüsselwort **goto** das einfache taktabhängige Fortschalten des Zählers beschrieben:

```
state S0:       goto S1;
state S1:       goto S2;

...

state S14:      goto S15;
state S15:      goto S0;
```

Wenn Macrozellen als T-Flipflops verwendet werden können lassen sich Zähler sehr elegant beschreiben:

```
module  Tcount

title '4 bit counter with load and clear'

D0 .. D3        pin;
Q3 .. Q0        pin istype 'reg_T';
CLK, I0, I1     pin;
Data    = [D3..D0];
Count   = [Q3..Q0];
Mode    = [I1,I0];
Clear   = [0,0];
Hold    = [0,1];
Load    = [1,0];
Inc     = [1,1];

equations

        Count.T =       ((Count.q+1     & (Mode==Inc)
                        # (Count.q      & (Mode==Hold)
                        # (Data)        & (Mode==Load)
                        # ( 0 )         & (Mode==Clear))
                        $ Count.q;
        Count.C = CLK;
end
```

d) Zustandsgesteuerte Flipflops (latches)

Ein transparentes zustandsgesteuertes D-Flipflop (D-latch) ergibt sich für:

Q = ENA&DATA # !ENA&Q # Q&DATA

Durch den Term Q&DATA werden Hazards unterdrückt.

Alternativ wird durch 'reg_G' ein zustandsgesteuertes (gated clock) Speicherelement beschrieben.

6. Aufgabenstellung

Die Arbeiten zu Versuch 3 lassen sich zum großen Teil zu Hause vorbereiten. Es wird dringenst empfohlen dies zu beherzigen, da der zum Labortermin zur Verfügung stehende Zeitraum begrenzt ist.

6.1 Versuchsvorbereitung

1. Erstellen Sie für die Ihnen zugeteilte Schaltnetzaufgabe die Funktionsgleichungen und für die Schaltwerksaufgabe das Zustandsdiagramm und die Zustandsübergangstabelle eines Moore-Schaltwerkes.

2. Setzen Sie die Funktionsgleichungen des Schaltnetzes und das Zustandsdiagramm (bzw. die Übergangstabelle) des Schaltwerkes nach Moore in AHDL-Files um.

Sollten während der Vorbereitung Verständnisprobleme auftreten, so bearbeiten Sie bitte die Aufgaben so weit wie möglich und besprechen, wenn möglich vor dem Labortermin, mit ihrem Betreuer die aufgetretenen Fragen.

6.2 Versuchsdurchführung

1. Rufen Sie easyABEL mit diesen Dateien auf und durchlaufen Sie alle Entwurfsschritte.

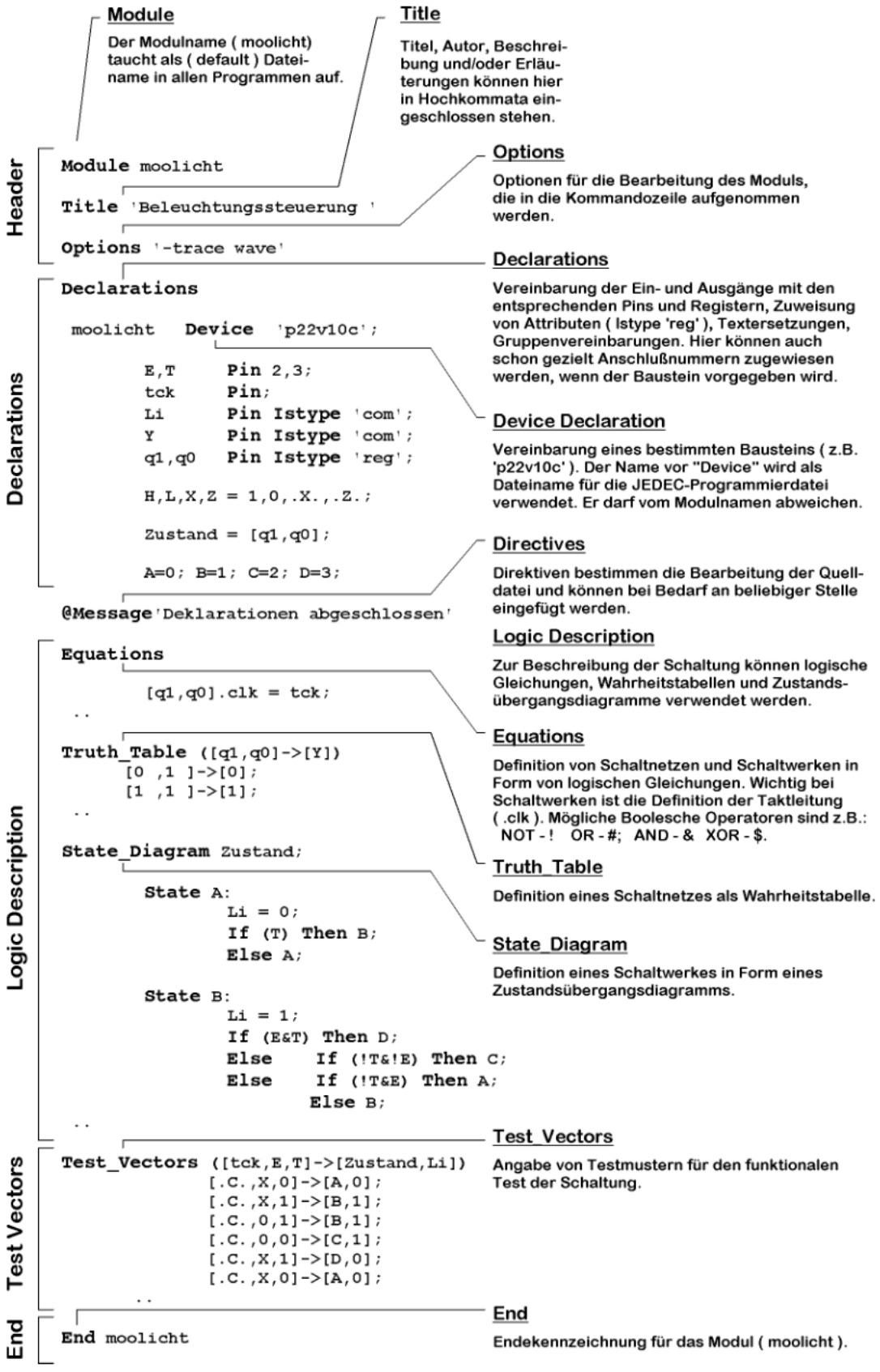
2. Bearbeiten Sie nun analog dazu die von Ihnen vorbereiteten Aufgaben.
   Tip: Sie können z.B. die existierenden AHDL-Dateien der Beispielaufgaben einfach abändern oder die Datei dummy.abl (siehe Anhang I) entsprechend ergänzen.

3. Laden Sie die JEDEC-Dateien mit dem Programm zum DOWNLOAD in den Baustein und überprüfen Sie die Funktion der Hardware.

Verzeichnisstruktur auf den Labor-PC´s:

C:\dataio\dgt0\vers3\beispiel\sn		
	\swmoore	Verzeichnisse der Beispiele
	\swmealy	
C:\dataio\tmp		Verzeichnis für Ihre Arbeit

Anhang A: Aufbau eines AHDL-Files



Anhang B: AHDL-File für die Rohrpoststeuerung

```
module Rohr     "Bei der Namensvergabe darauf achten, dass keine Umlaute
                "oder Ziffern verwendet werden!!! Dies Restriktion gilt
                "auch bei titel und device

title 'Rohrpoststeuerung  B. Mueller'

combi device 'p22v10c'; "Das spaeter noch zu generierende JEDEC-File
                        "erhaelt den Namen combi und ist auf einen
                        "Baustein vom Typ ispGAL22v10c ausgerichtet

declarations
        s1,s0,e1,e0     pin 2,3,4,5;      "Eingangssignale werden definiert und
                                          "best. Bausteinanschluessen zugeordnet
        v1,v2,v3,v4,r,f pin istype 'com'; "Baustein-Ausgaenge rein kombinatorisch
                                          "keine Zuordnung von Bausteinanschluessen
                                          "wird spaeter durch FPGA/PLDmap ausgefuehrt

        X = .X.;        "Labelvergabe fuer unbestimmten Pegel

equations;

truth_table     ([s1,s0,e1,e0]->[v1,v2,v3,v4,r,f])   "Funktionstabelle
                 [0 , 0, 0, 0]->[X ,X ,X ,X ,X,1];
                 [0 , 0, 0, 1]->[X ,0 ,1 ,X ,1,0];
                 [0 , 0, 1, 0]->[X ,0 ,0 ,1 ,1,0];
                 [0 , 0, 1, 1]->[0 ,X ,X ,1 ,0,0];
                 [0 , 1, 0, 0]->[1 ,0 ,X ,X ,0,0];
                 [0 , 1, 0, 1]->[X ,X ,X ,X ,X,1];
                 [0 , 1, 1, 0]->[X ,X ,0 ,1 ,1,0];
                 [0 , 1, 1, 1]->[0 ,0 ,X ,1 ,0,0];
                 [1 , 0, 0, 0]->[1 ,0 ,0 ,X ,0,0];
                 [1 , 0, 0, 1]->[X ,1 ,0 ,X ,0,0];
                 [1 , 0, 1, 0]->[X ,X ,X ,X ,X,1];
                 [1 , 0, 1, 1]->[1 ,X ,X ,0 ,1,0];
                 [1 , 1, 0, 0]->[0 ,1 ,X ,X ,1,0];
                 [1 , 1, 0, 1]->[0 ,0 ,1 ,X ,1,0];
                 [1 , 1, 1, 0]->[X ,X ,1 ,0 ,0,0];
                 [1 , 1, 1, 1]->[X ,X ,X ,X ,X,1];

test_vectors    ([s1,s0,e1,e0]->[v1,v2,v3,v4,r,f])   "Testvektoren
                 [0 , 0, 0, 0]->[X ,X ,X ,X ,X,1];   "Diese Testvektoren koennen nur vom
                 [0 , 0, 0, 1]->[X ,0 ,1 ,X ,1,0];   "in easyABEL eingebauten Logik-
                 [0 , 0, 1, 0]->[X ,0 ,0 ,1 ,1,0];   "simulator verwendet werden.
                 [0 , 0, 1, 1]->[0 ,X ,X ,1 ,0,0];
                 [0 , 1, 0, 0]->[1 ,0 ,X ,X ,0,0];
                 [0 , 1, 0, 1]->[X ,X ,X ,X ,X,1];
                 [0 , 1, 1, 0]->[X ,X ,0 ,1 ,1,0];
                 [0 , 1, 1, 1]->[0 ,0 ,X ,1 ,0,0];
                 [1 , 0, 0, 0]->[1 ,0 ,0 ,X ,0,0];
                 [1 , 0, 0, 1]->[X ,1 ,0 ,X ,0,0];
                 [1 , 0, 1, 0]->[X ,X ,X ,X ,X,1];
                 [1 , 0, 1, 1]->[1 ,X ,X ,0 ,1,0];
                 [1 , 1, 0, 0]->[0 ,1 ,X ,X ,1,0];
                 [1 , 1, 0, 1]->[0 ,0 ,1 ,X ,1,0];
                 [1 , 1, 1, 0]->[X ,X ,1 ,0 ,0,0];
                 [1 , 1, 1, 1]->[X ,X ,X ,X ,X,1];

end Rohr                                              "Modulende
```

Anhang C: Ausgabedateien des Beispiels der Rohrpoststeuerung

C-1 Compiler Listing-Datei ROHR.LST

```
0001 |module Rohr     "Bei der Namensvergabe darauf achten, dass keine Umlaute
0002 |                "oder Ziffern verwendet werden!!! Dies Restriktion gilt
0003 |                "auch bei titel und device
0004 |
0005 |title 'Rohrpoststeuerung  B. Mueller'
0006 |
0007 |combi device 'p22v10c'; "Das spaeter noch zu generierende JEDEC-File
0008 |                        "erhaelt den Namen combi und ist auf einen
0009 |                        "Baustein vom Typ ispGAL22v10c ausgerichtet
0010 |
0011 |declarations
0012 |        s1,s0,e1,e0     pin 2,3,4,5;      "Eingangssignale werden definiert und
0013 |                                          "best. Bausteinanschluessen zugeordnet
0014 |        v1,v2,v3,v4,r,f pin istype 'com'; "Baustein-Ausgaenge rein kombinatorisch
0015 |                                          "keine Zuordnung von Bausteinanschluessen
0016 |                                          "wird spaeter durch FPGA/PLDmap ausgefuehrt
0017 |
0018 |        X = .X.;        "Labelvergabe fuer unbestimmten Pegel
0019 |
0020 |equations;
0021 |
0022 |truth_table     ([s1,s0,e1,e0]->[v1,v2,v3,v4,r,f])   "Funktionstabelle
0023 |                 [0 , 0, 0, 0]->[X ,X ,X ,X ,X,1];
0024 |                 [0 , 0, 0, 1]->[X ,0 ,1 ,X ,1,0];
0025 |                 [0 , 0, 1, 0]->[X ,0 ,0 ,1 ,1,0];
0026 |                 [0 , 0, 1, 1]->[0 ,X ,X ,1 ,0,0];
0027 |                 [0 , 1, 0, 0]->[1 ,0 ,X ,X ,0,0];
0028 |                 [0 , 1, 0, 1]->[X ,X ,X ,X ,X,1];
0029 |                 [0 , 1, 1, 0]->[X ,X ,0 ,1 ,1,0];
0030 |                 [0 , 1, 1, 1]->[0 ,0 ,X ,1 ,0,0];
0031 |                 [1 , 0, 0, 0]->[1 ,0 ,0 ,X ,0,0];
0032 |                 [1 , 0, 0, 1]->[X ,1 ,0 ,X ,0,0];
0033 |                 [1 , 0, 1, 0]->[X ,X ,X ,X ,X,1];
0034 |                 [1 , 0, 1, 1]->[1 ,X ,X ,0 ,1,0];
0035 |                 [1 , 1, 0, 0]->[0 ,1 ,X ,X ,1,0];
0036 |                 [1 , 1, 0, 1]->[0 ,0 ,1 ,X ,1,0];
0037 |                 [1 , 1, 1, 0]->[X ,X ,1 ,0 ,0,0];
0038 |                 [1 , 1, 1, 1]->[X ,X ,X ,X ,X,1];
0039 |
0040 |test_vectors    ([s1,s0,e1,e0]->[v1,v2,v3,v4,r,f])   "Testvektoren
0041 |                 [0 , 0, 0, 0]->[X ,X ,X ,X ,X,1];   "Diese Testvektoren koennen nur vom
0042 |                 [0 , 0, 0, 1]->[X ,0 ,1 ,X ,1,0];   "in easyABEL eingebauten Logik-
0043 |                 [0 , 0, 1, 0]->[X ,0 ,0 ,1 ,1,0];   "simulator verwendet werden.
0044 |                 [0 , 0, 1, 1]->[0 ,X ,X ,1 ,0,0];
0045 |                 [0 , 1, 0, 0]->[1 ,0 ,X ,X ,0,0];
0046 |                 [0 , 1, 0, 1]->[X ,X ,X ,X ,X,1];
0047 |                 [0 , 1, 1, 0]->[X ,X ,0 ,1 ,1,0];
0048 |                 [0 , 1, 1, 1]->[0 ,0 ,X ,1 ,0,0];
0049 |                 [1 , 0, 0, 0]->[1 ,0 ,0 ,X ,0,0];
0050 |                 [1 , 0, 0, 1]->[X ,1 ,0 ,X ,0,0];
0051 |                 [1 , 0, 1, 0]->[X ,X ,X ,X ,X,1];
0052 |                 [1 , 0, 1, 1]->[1 ,X ,X ,0 ,1,0];
0053 |                 [1 , 1, 0, 0]->[0 ,1 ,X ,X ,1,0];
0054 |                 [1 , 1, 0, 1]->[0 ,0 ,1 ,X ,1,0];
0055 |                 [1 , 1, 1, 0]->[X ,X ,1 ,0 ,0,0];
0056 |                 [1 , 1, 1, 1]->[X ,X ,X ,X ,X,1];
0057 |
0058 |end Rohr                                          "Modulende
0059 |
```

C-2 Dateien der synthetisierten Gleichungen ROHR.EQ#

ROHR.EQ1

```
Product Term Usage:

            Terms    Signal
            -----    ------
              4      f
              8      r
              6      v4
              4      v3
              7      v2
              6      v1

Total:       35      (Total assumes no product term sharing)


Equations:

f = (s1 & s0 & e1 & e0
     # s1 & !s0 & e1 & !e0
     # !s1 & s0 & !e1 & e0
     # !s1 & !s0 & !e1 & !e0);

r = (s1 & s0 & e0
     # s1 & s0 & !e1
     # s1 & !s0 & e1
     # !s0 & e1 & !e0
     # !s1 & e1 & !e0
     # !s1 & !e1 & e0
     # !s1 & !s0 & !e0
     # !s1 & !s0 & !e1);

v4 = (s1 & s0 & !e1
     # !s0 & e1 & !e0
     # s0 & e0
     # !s1 & e1
     # !s1 & e0
     # !s1 & !s0);

v3 = (!s1 & !s0 & !e1
     # !s1 & e0
     # s1 & s0
     # s1 & e1 & !e0);

v2 = (!s1 & s0 & !e1 & e0
     # !s1 & !s0 & !e1 & !e0
     # s1 & e1 & !e0
     # s1 & s0 & e1
     # s1 & s0 & !e0
     # s1 & !s0 & !e1 & e0
     # !s1 & !s0 & e1 & e0);

v1 = (s1 & e1 & !e0
     # s1 & e1
     # s1 & !s0
     # !s0 & !e1 & !e0
     # !s1 & s0 & !e1
     # !s1 & !e1 & !e0);
```

ROHR.EQ2 und ROHR.EQ3 sind hier identisch

```
Product Term Usage:

          Default  Reverse   Signal
          Polarity Polarity  Signal
           ------   ------   ------
              4        4     f
              6        4     r
              5        4     v4
              4        4     v3
              7        7     v2
              4        3     v1

Total:       30       26

Best Total:     26     (Totals assume no product term sharing)


Equations:

f = (s1 & s0 & e1 & e0
     # !s1 & s0 & !e1 & e0
     # s1 & !s0 & e1 & !e0
     # !s1 & !s0 & !e1 & !e0);

r = (s1 & !s0 & e1
     # s1 & s0 & !e1
     # !s1 & !s0 & !e1
     # s1 & e1 & e0
     # !s1 & !e1 & e0
     # !s1 & e1 & !e0);

v4 = (!s1 & !s0
     # !s1 & e1
     # s1 & s0 & !e1
     # s0 & e0
     # !s0 & e1 & !e0);

v3 = (s1 & s0
     # !s1 & !s0 & !e1
     # !s1 & e0
     # s1 & e1 & !e0);

v2 = (s1 & s0 & e1
     # !s1 & !s0 & e1 & e0
     # !s1 & s0 & !e1 & e0
     # s1 & !s0 & !e1 & e0
     # s1 & s0 & !e0
     # s1 & e1 & !e0
     # !s1 & !s0 & !e1 & !e0);

v1 = (s1 & !s0
     # s1 & e1
     # !s1 & s0 & !e1
     # !s0 & !e1 & !e0);


Reverse-polarity Equations:

!f = (!s1 & e1
     # s1 & !e1
     # !s0 & e0
     # s0 & !e0);

!r = (s1 & !s0 & !e1
     # !s1 & e1 & e0
     # s1 & s0 & e1 & !e0
     # !s1 & s0 & !e1 & !e0);

!v4 = (s1 & !s0 & !e1
     # s1 & !s0 & e0
     # s1 & s0 & e1 & !e0
     # !s1 & s0 & !e1 & !e0);

!v3 = (!s1 & e1 & !e0
     # s1 & !s0 & !e1
     # s1 & !s0 & e0
     # !s1 & s0 & !e0);

!v2 = (!s1 & e1 & !e0
     # !s1 & s0 & e1
     # s1 & !s0 & e1 & e0
     # s1 & s0 & !e1 & e0
     # !s1 & !s0 & !e1 & e0
     # !s1 & s0 & !e0
     # s1 & !s0 & !e1 & !e0);

!v1 = (!s1 & e1
     # s1 & s0 & !e1
     # !s1 & !s0 & e0);
```

C-3 Simulationsdateien ROHR.SM#

ROHR.SM1 Ergebnisse in Tabellenform ROHR.SM2, ROHR.SM3 und COMBI.SIM sind identisch.

```
Simulate EZ-ABEL 4.30  Date: Sun Apr 14 20:42:53 1996
Fuse file: 'rohr.tt1'  Vector file: 'rohr.tmv'  Part: 'PLA'
Rohrpoststeuerung  B. Mueller


       s s e e   v v v v
       1 0 1 0   1 2 3 4 r f

V0001  0 0 0 0   H H H H H H
V0002  0 0 0 1   L L H H H L
V0003  0 0 1 0   L L L H H L
V0004  0 0 1 1   L H H H L L
V0005  0 1 0 0   H L L L L L
V0006  0 1 0 1   H H H H H H
V0007  0 1 1 0   L L L H H L
V0008  0 1 1 1   L L H H L L
V0009  1 0 0 0   H L L L L L
V0010  1 0 0 1   H H L L L L
V0011  1 0 1 0   H H H H H H
V0012  1 0 1 1   H L L L H L
V0013  1 1 0 0   L H H H H L
V0014  1 1 0 1   L L H H H L
V0015  1 1 1 0   H H H L L L
V0016  1 1 1 1   H H H H H H

16 out of 16 vectors passed.
```

ROHR.SM1 Ergebnisse als Zeitliniendiagramm ROHR.SM2, ROHR.SM3 und COMBI.SIM sind identisch.

```
Simulate EZ-ABEL 4.30  Date: Sun Apr 14 19:01:27 1996
Fuse file: 'rohr.tt1'  Vector file: 'rohr.tmv'  Part: 'PLA'
Rohrpoststeuerung  B. Mueller


        s    s    e    e         v    v    v    v
        1    0    1    0         1    2    3    4    r    f

V0001  |    |    |    |         ~|   ~|   ~|   ~|   ~|   ~|
V0002  |    |    |     ~|       |~   |~     |    |    |   |~
V0003  |    |     ~|  |~        |    |    |~     |    |   |
V0004  |    |      |   ~|       |    ~|   ~|     |   |~   |
V0005  |     ~|  |~   |~         ~|   |~   |~   |~   |    |
V0006  |      |  |    ~|          |   ~|   ~|   ~|   ~|   ~|
V0007  |      |   ~|  |~        |~   |~   |~    |    |   |~
V0008  |      |    |   ~|       |    |     ~|    |   |~   |
V0009   ~|  |~   |~   |~         ~|  |    |~   |~   |    |
V0010    |  |    |     ~|         |   ~|  |    |    |    |
V0011    |  |     ~|  |~          |    |   ~|   ~|   ~|   ~|
V0012    |  |      |   ~|         |   |~   |~   |~     |   |~
V0013    |   ~|  |~   |~        |~    ~|   ~|   ~|    |   |
V0014    |    |  |     ~|       |    |~     |    |    |   |
V0015    |    |   ~|  |~         ~|   ~|    |   |~   |~   |
V0016    |    |    |   ~|         |    |    |   ~|   ~|   ~|

16 out of 16 vectors passed.
```

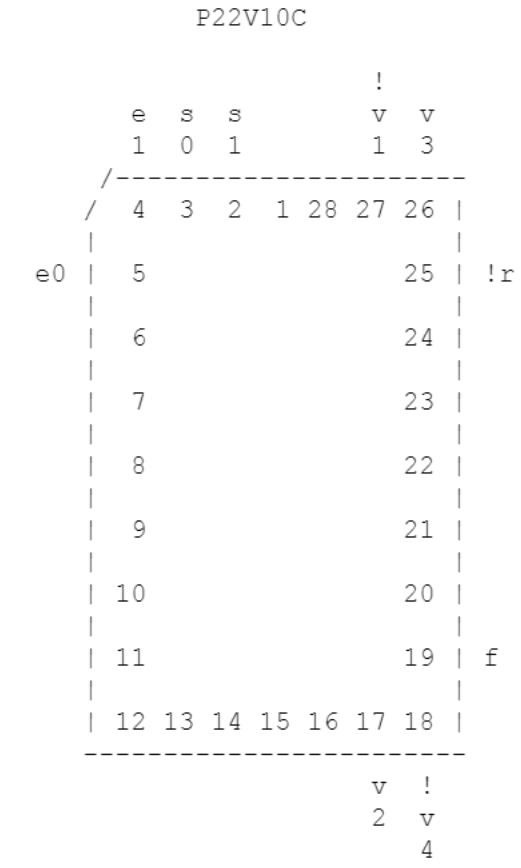
C-4 Dokumentationsdatei ROHR.DOC

```
                                                                    Page 1
EZ-ABEL 4.30  -  Device Utilization Chart         Sun Apr 14 20:44:10 1996

Rohrpoststeuerung  B. Mueller

    ==== P22V10C Programmed Logic ====


f      = (  s1 & s0 & e1 & e0
         #   !s1 & s0 & !e1 & e0
         #   s1 & !s0 & e1 & !e0
         #   !s1 & !s0 & !e1 & !e0 );

r      = !(  s1 & !s0 & !e1
         #   !s1 & e1 & e0
         #   s1 & s0 & e1 & !e0
         #   !s1 & s0 & !e1 & !e0 );

v4     = !(  s1 & !s0 & !e1
          #   s1 & !s0 & e0
          #   s1 & s0 & e1 & !e0
          #   !s1 & s0 & !e1 & !e0 );

v3     = (  s1 & s0
          #   !s1 & !s0 & !e1
          #   !s1 & e0
          #   s1 & e1 & !e0 );

v2     = (  s1 & s0 & e1
          #   !s1 & !s0 & e1 & e0
          #   !s1 & s0 & !e1 & e0
          #   s1 & !s0 & !e1 & e0
          #   s1 & s0 & !e0
          #   s1 & e1 & !e0
          #   !s1 & !s0 & !e1 & !e0 );

v1     = !(  !s1 & e1
          #   s1 & s0 & !e1
          #   !s1 & !s0 & e0 );
```

EZ-ABEL 4.30 - Device Utilization Chart Sun Apr 14 20:44:10 1996

Rohrpoststeuerung B. Mueller

==== P22V10C Chip Diagram ====



SIGNATURE: N/A

Page 3
EZ-ABEL 4.30 - Device Utilization Chart Sun Apr 14 20:44:10 1996

Rohrpoststeuerung B. Mueller

==== P22V10C Resource Allocations ====

Device Resources	Resource Available	Design Requirement	Part Utilization	Unused
Dedicated input pins	12	4	4	8 (66 %)
Combinatorial inputs	12	4	4	8 (66 %)
Registered inputs	-	0	-	-
Dedicated output pins	-	6	-	-
Bidirectional pins	10	0	6	4 (40 %)
Combinatorial outputs	-	6	-	-
Registered outputs	-	0	-	-
Reg/Com outputs	10	-	6	4 (40 %)
Two-input XOR	-	0	-	-
Buried nodes	-	0	-	-
Buried registers	-	0	-	-
Buried combinatorials	-	0	-	-

Page 4
EZ-ABEL 4.30 - Device Utilization Chart Sun Apr 14 20:44:10 1996

Rohrpoststeuerung B. Mueller

==== P22V10C Product Terms Distribution ====

Signal Name	Pin Assigned	Terms Used	Terms Max	Terms Unused
f	19	4	12	8
r	25	4	12	8
v4	18	4	10	6
v3	26	4	10	6
v2	17	7	8	1
v1	27	3	8	5

==== List of Inputs/Feedbacks ====

Signal Name	Pin	Pin Type
s1	2	CLK/IN
s0	3	INPUT
e1	4	INPUT
e0	5	INPUT

```
                                                                    Page 5
EZ-ABEL 4.30  -  Device Utilization Chart         Sun Apr 14 20:44:10 1996

Rohrpoststeuerung  B. Mueller

    ==== P22V10C Unused Resources ====
```

Pin Number	Pin Type	Product Terms	Flip-flop Type
6	INPUT	-	-
7	INPUT	-	-
9	INPUT	-	-
10	INPUT	-	-
11	INPUT	-	-
12	INPUT	-	-
13	INPUT	-	-
16	INPUT	-	-
20	BIDIR	NORMAL 14	D
21	BIDIR	NORMAL 16	D
23	BIDIR	NORMAL 16	D
24	BIDIR	NORMAL 14	D

```
                                                                    Page 6
EZ-ABEL 4.30  -  Device Utilization Chart         Sun Apr 14 20:44:10 1996

Rohrpoststeuerung  B. Mueller

    ==== I/O Files ====


Module: 'rohr'


Input files
===========
ABEL PLA file: rohr.tt3
Vector file: rohr.tmv
Device library: P22V10C.dev

Output files
============
Report file: rohr.doc
Programmer load file: combi.jed
```

C-5 JEDEC-Datei COMBI.JED

```
^EZ-ABEL 4.32 Data I/O Corp. JEDEC file for: P22V10C V9.0
Created on: Sun Apr 14 20:44:10 1996

Rohrpoststeuerung  B. Mueller
*
QP28* QF5828* QV16* F0*
 X0*
NOTE Table of pin names and numbers*
NOTE PINS s1:2 s0:3 e1:4 e0:5 v1:27 v2:17 v3:26 v4:18 r:25 f:19*
L0044 11111111111111111111111111111111111111111111*
L0088 10111111011111111111111111111111111111111111*
L0132 01110111101111111111111111111111111111111111*
L0176 10111011111101111111111111111111111111111111*
L0440 11111111111111111111111111111111111111111111*
L0484 01110111111111111111111111111111111111111111*
L0528 10111011101111111111111111111111111111111111*
L0572 10111111111101111111111111111111111111111111*
L0616 01111111011110111111111111111111111111111111*
L0924 11111111111111111111111111111111111111111111*
L0968 01111011101111111111111111111111111111111111*
L1012 10111111011101111111111111111111111111111111*
L1056 01110111011110111111111111111111111111111111*
L1100 10110111101110111111111111111111111111111111*
L4312 11111111111111111111111111111111111111111111*
L4356 01110111011101111111111111111111111111111111*
L4400 10110111101101111111111111111111111111111111*
L4444 01111011011110111111111111111111111111111111*
L4488 10111011101110111111111111111111111111111111*
L4884 11111111111111111111111111111111111111111111*
L4928 01111011101111111111111111111111111111111111*
L4972 01111011111101111111111111111111111111111111*
L5016 01110111011110111111111111111111111111111111*
L5060 10110111101110111111111111111111111111111111*
L5368 11111111111111111111111111111111111111111111*
L5412 01110111011111111111111111111111111111111111*
L5456 10111011011101111111111111111111111111111111*
L5500 10110111101101111111111111111111111111111111*
L5544 01111011101101111111111111111111111111111111*
L5588 01110111111101111111111111111111111111111111*
L5632 01111111011110111111111111111111111111111111*
L5676 10111011101110111111111111111111111111111111*
L5808 0111010000000010111 1*
V0001 N0000XXNXXXXXNNXNNHXXNXXNNNN*
V0002 N0001XXNXXXXXNNXLNLXXNXXHHNN*
V0003 N0010XXNXXXXXNNXLHLXXNXXHLNN*
V0004 N0011XXNXXXXXNNXNHLXXNXXLNLN*
V0005 N0100XXNXXXXXNNXLNLXXNXXLNHN*
V0006 N0101XXNXXXXXNNXNNHXXNXXNNNN*
V0007 N0110XXNXXXXXNNXNHLXXNXXHLNN*
V0008 N0111XXNXXXXXNNXLHLXXNXXLNLN*
V0009 N1000XXNXXXXXNNXLNLXXNXXLLHN*
V0010 N1001XXNXXXXXNNXHNLXXNXXLLNN*
V0011 N1010XXNXXXXXNNXNNHXXNXXNNNN*
V0012 N1011XXNXXXXXNNXNLLXXNXXHNHN*
V0013 N1100XXNXXXXXNNXHNLXXNXXHNLN*
V0014 N1101XXNXXXXXNNXLNLXXNXXHHLN*
V0015 N1110XXNXXXXXNNXNLLXXNXXLHNN*
V0016 N1111XXNXXXXXNNXNNHXXNXXNNNN*
CAC0A*
~2936
```

Anhang D: Moore-Realisierung der Beleuchtungssteuerung

```
Module moolicht

title 'Beleuchtungssteuerung b. mueller'

moolicht   device  'p22v10c';

declarations

        E,T     pin;                    "Baustein-Eingaenge
        tck     pin;                    "Kontroll-Eingang
        Li      pin istype 'com';       "Baustein-Ausgang

        q1,q0   pin istype 'reg';       "Variablen der Zustandskodierung

        H,L,X,Z = 1,0,.X.,.Z.;          "Labels fuer Signalpegel
                                        ".X. bedeutete don´t care
                                        "und .Z. steht für tri-state

        Zustand = [q1,q0];              "Zustandskodierungsvektor
                                        "alternativ zulaessig ist auch [q1,q0]

        A=0; B=1; C=2; D=3;             "Variablen der Zustandskodierung

equations

        [q1,q0].clk = tck;              "Def. der Taktabhaengigkeit

state_diagram Zustand;                  "Def. des endlichen Automaten

        state A:
                Li = 0;
                if (T) then B;
                else A;

        state B:
                Li = 1;
                if (E&T) then D;
                else    if (!T&!E) then C;
                else    if (!T&E) then A;
                        else B;

        state C:
                Li = 1;
                if (T) then D;
                else if (E&!T) then A;
                     else C;

        state D:
                Li = 0;
                if (!T) then A;
                else D;

test_vectors ([tck,E,T]->[Zustand,Li])  "Def. der Testvektoren fuer interne
               [.C.,X,0]->[A,0];        "Simulation
               [.C.,X,1]->[B,1];
               [.C.,0,1]->[B,1];
               [.C.,0,0]->[C,1];
               [.C.,0,0]->[C,1];
               [.C.,X,1]->[D,0];
               [.C.,X,1]->[D,0];
               [.C.,X,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,1,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,0,0]->[C,1];
               [.C.,1,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,1,1]->[D,0];
               [.C.,X,0]->[A,0];

end moolicht
```

Anhang E: Mealy-Realisierung der Beleuchtungssteuerung

```
module mealicht

title 'Beleuchtungssteuerung b. mueller'

mealicht   device  'p22v10c';

declarations

        E,T     pin;                    "Baustein-Eingaenge
        tck     pin;                    "Kontroll-Eingang
        Li      pin istype 'com';       "Baustein-Ausgang

        q1,q0   pin istype 'reg';       "Variablen der Zustandskodierung

        H,L,X,Z = 1,0,.X.,.Z.;          "Labels fuer Signalpegel
                                        ".X. bedeutete dont care
                                        "und .Z. steht für tri-state

        Zustand = [q1,q0];              "Zustandskodierungsvektor
                                        "alternativ zulässig ist auch [q1,q0]

        A=0; B=1; C=2; D=3;             "Variablen der Zustandskodierung

equations

        [q1,q0].clk = tck;              "Def. der Taktabhaengigkeit

state_diagram Zustand;                  "Def. des Endlichen Automaten

        state A:
                if (T) then B with Li=1;
                             endwith;
                else A;

        state B:
                if (E) then
                    if (T) then D;
                    else A;
                else
                    if(!T) then C with Li = 1;
                                  endwith;
                    else B with Li=1;
                           endwith;

        state C:
                if (T) then D;
                else
                    if (E) then A;
                    else C with Li=1;
                           endwith;

        state D:
                if (!T) then A;
                else D;

test_vectors ([tck,E,T]->[Zustand,Li])  "Def. der Testvektoren fuer interne
               [.C.,X,0]->[A,0];        "Simulation
               [.C.,X,1]->[B,1];
               [.C.,0,1]->[B,1];
               [.C.,0,0]->[C,1];
               [.C.,0,0]->[C,1];
               [.C.,X,1]->[D,0];
               [.C.,X,1]->[D,0];
               [.C.,X,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,1,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,0,0]->[C,1];
               [.C.,1,0]->[A,0];
               [.C.,X,1]->[B,1];
               [.C.,1,1]->[D,0];
               [.C.,X,0]->[A,0];

end mealicht
```

Anhang F: Programmierbare Logikbausteine am Beispiel des EPLD-Bausteins ispGAL22V10

In diesem Versuch wird ein ispGAL22V10 eingesetzt (in system programmable generic array logic) der über individuell konfigurierbare Makrozellen verfügt und im System, also ohne spezielles Programmiergerät in der Schaltung programmiert werden kann. Im Folgenden soll die besondere Bausteinarchitektur des ispGAL22V10 vorgestellt werden.

F-1 Logikdiagramm

Der GAL-Baustein 20V10 weist eine 44 x 120 - UND-Matrix auf, d.h. er verfügt über 120 UND-Gatter mit jeweils 44 programmierbaren Eingängen (siehe Bild F.2). Die 12 Eingangssignale (Pin 2 bis Pin 16) und die 10 Rückführungssignale aus den Ausgangs-Makrozellen OLMC (output logic macro cell) können jeweils nicht invertiert oder invertiert an ein beliebiges UND-Gatter geschaltet werden. Die 10 bidirektionalen Tristate-Ausgänge (Pin 17 bis Pin 27) können individuell als primäre Ausgänge oder als primäre Eingänge programmiert werden. Die Umschaltung erfolgt über ein Freigabesignal OE (output enable). Insgesamt geht die Anzahl der maximal verfügbaren Ein- und Ausgänge aus der Bausteinbezeichnung hervor: „22V10“ steht für 22 Eingänge und 10 Ausgänge, „V“ (variable) für konfigurierbare Ausgangszelle. Aus dem Logikdiagramm in Bild F.2 geht aber zugleich hervor, daß die Summe der Ein- und Ausgänge auf 22 begrenzt ist. Ferner wird der GAL-Baustein mit einem zentralen Taktsignal CLK betrieben, das synchron an allen Speicherelementen in den Makrozellen anliegt. Der Anschluß Pin 2 für das Taktsignal kann bei Bedarf auch als primäre Eingänge konfiguriert werden. Alle Flipflops können asynchron zurückgesetzt bzw. synchron gesetzt werden.

F-2 Ausgangs-Makrozelle OLMC

Die Entwurfsflexibilität eines GAL-Bausteins liegt in der leichten Konfigurierbarkeit seiner Ausgangszellen. Im Bild F.1 sind die OLMC und die programmierbaren Signalpfade dargestellt. Mit Hilfe von zwei programmierbaren Multiplexern können die Signalpfade für jede einzelne OLMC (über S_0 (n) und S_1 (n)) definiert werden. n bezeichnet dabei die Zuordnung der Ausgangszellen zu den Gehäusepins. Die Belegung der Steuersignale ist aber für den Anwender „transparent“, d.h. er braucht sich darüber während des GAL-Entwurfs keine Gedanken zu machen. Der PLD-Compiler stellt die Belegung entsprechend der gewünschten Schaltfunktion automatisch ein. Zum Verständnis der Konfigurationsmöglichkeiten ist es aber notwendig, näher auf die Multiplexerfunktionen einzugehen.

Jede der Makrozellen hat primär zwei verschiedene Betriebszustände (modes): Ausgang mit Register (registered)oder kombinatorischer Ein-/Ausgang (combinatorial I/O). Der Betriebszustand und die Ausgangspolarität wird durch die beiden Steuerbits S_0 und S_1 eingestellt.

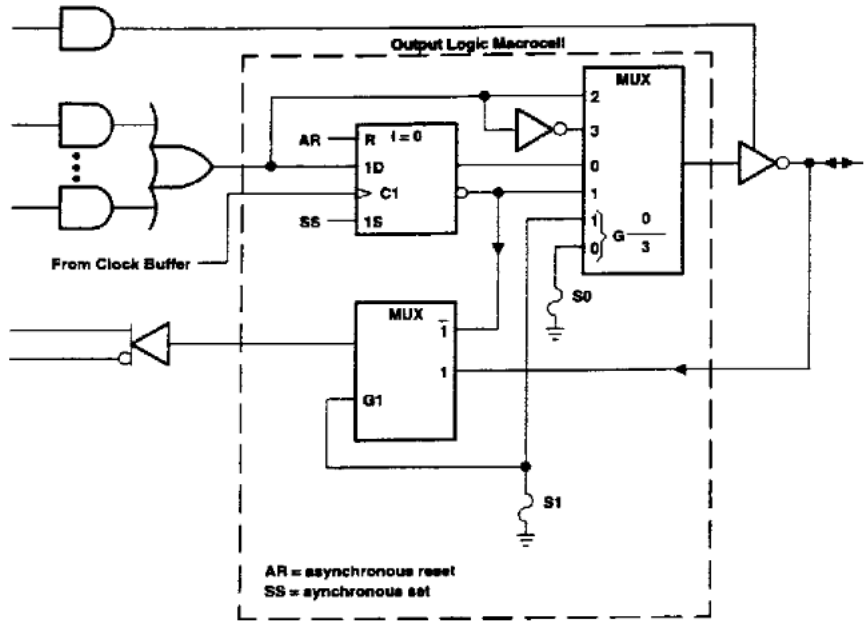
REGISTERED

Der Ausgangspin wird vom Q-Ausgang des D-Flipflops der Ausgangszelle angesteuert. Die Polarität des Ausgangs kann ebenfalls vorgegeben werden. Die Freigabe des Tristate-Puffers kann für jede Zelle über ein separates UND-Gatter erfolgen. Das Ausgangssignal des Flipflops wird in die UND-Matrix sowohl invertiert als auch nicht invertiert zurückgekoppelt. Diese Rückkopplung geschieht vom $\overline{Q}$ -Ausgang des Flipflops und nicht vom Ausgangspin, so daß hier der Pin tatsächlich nur ein Ausgang ist und nicht als dynamischer Ein-/Ausgang verwendet werden kann, wie dies bei beim kombinatorischen Betriebszustand möglich ist.

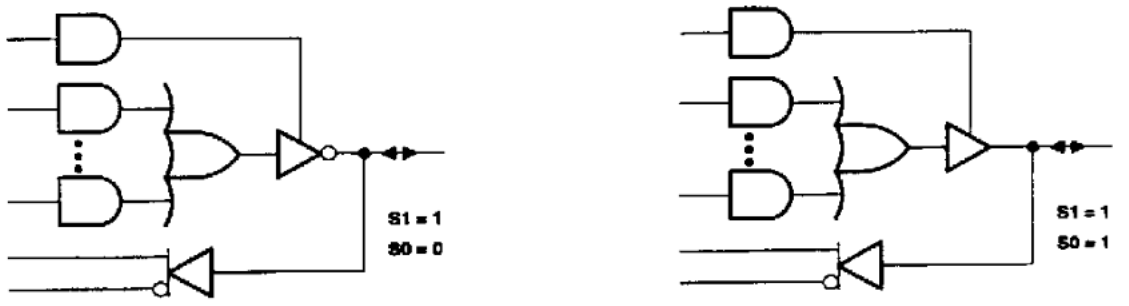
COMBINATORIAL I/O

Im rein kombinatorischen Betrieb wird der Pin durch das ODER-Gatter angesteuert. Auch hier kann die Polarität vorgegeben werden (active high oder active low). Die Kontrolle des Tristate-Puffers erfolgt auch hier über ein separates UND-Gatter. Der Pin kann als dedizierter Ausgang (dedicated output), dedizierter Eingang (dedicated input) oder, durch das Ausgangssignal des UND-Gatters bestimmt, als dynamisch Ein-/Ausgang (dynamic I/O) verwendet werden. Die Rückkopplung (feedback) in die UND-Matrix (AND array) erfolgt „pin-seitig“ vom Ausgang des Tristate-Puffers. Beide Polaritäten des Signals sind verfügbar.

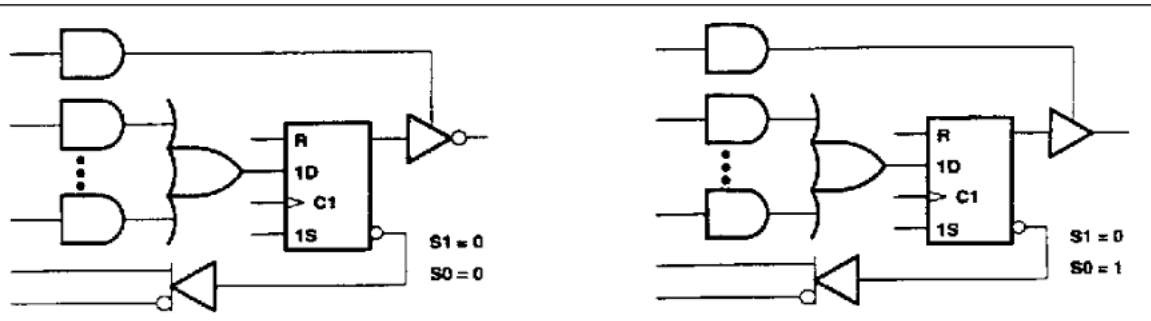
output logic macrocell diagram



ispGAL22V10 OUTPUT LOGIC MACROCELL (OLMC)



Schaltbilder der Ausgangs-Makrozelle für den Betriebszustand Combinatorial I/O



Schaltbilder der Ausgangs-Makrozelle für den Betriebszustand Registered Output

Bild F.1 Schaltbild der Ausgangszelle und die verschiedenen Betriebszustände des ispGAL22V10

ispGAL22V10 LOGIC DIAGRAM / JEDEC FUSE MAP

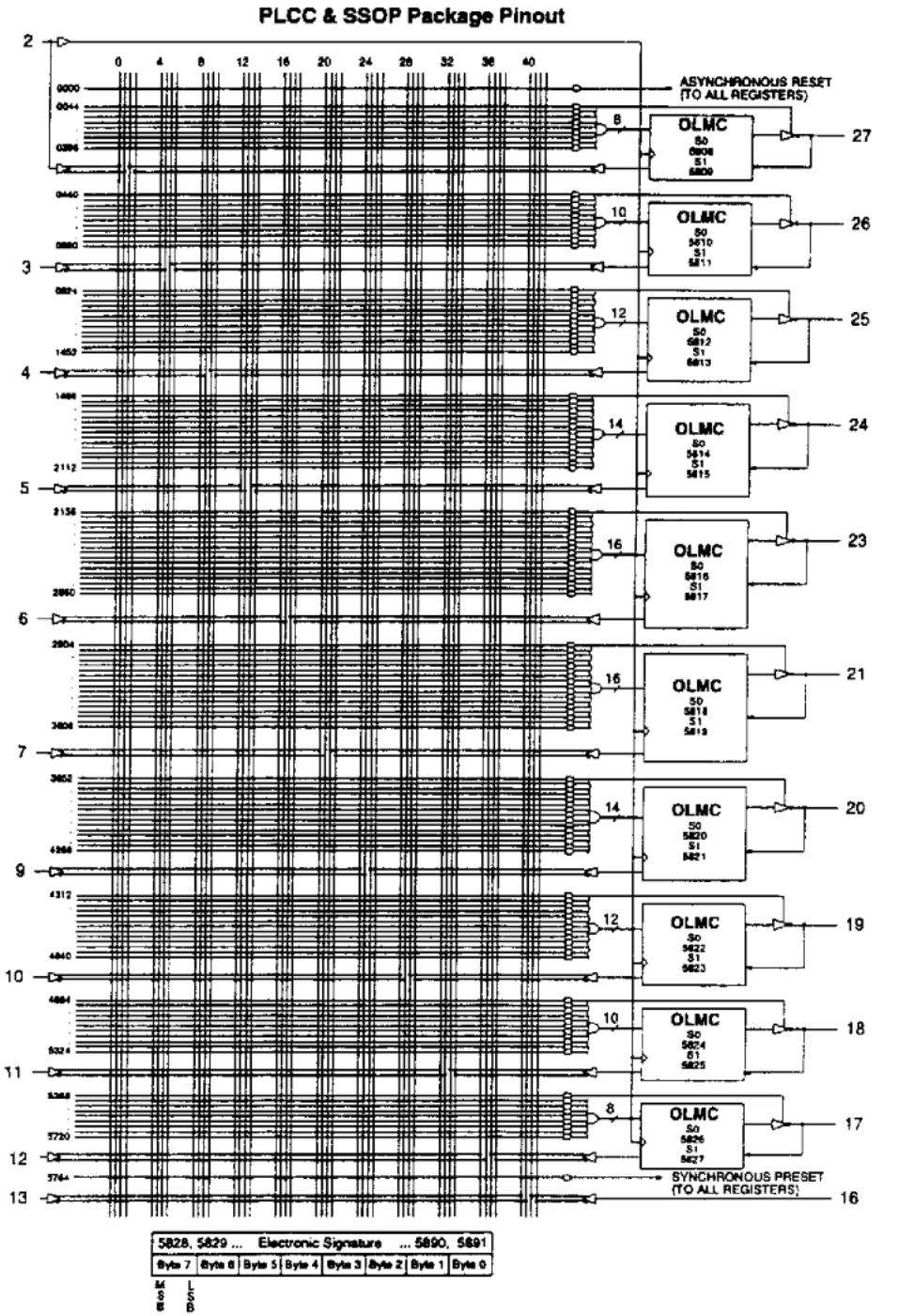


Bild F.2 Schaltbild des ispGAL22V10 und JEDEC fuse map

Anhang G: Literaturverzeichnis

(ohne Anspruch auf Vollständigkeit, wird laufend ergänzt)

[Ammon88] Ammon, Peter:
ASIC-Praxis: Grundlagen und Handhabung anwenderspezifischer Ics.
Franzis-Verlag, München, 1988

[Auer90] Auer, Adolf:
Programmierbare Logik-IC.
Hüthig Buch Verlag, Heidelberg, 1990

[Bitterle89] Bitterle, Dieter:
GALs: Programmierbare Logikbausteine in Theorie und Praxis.
Franzis-Verlag, München, 1991

[Eabl92] easyABEL Manual,
Data I/O Corporation, 10525 Willows Road N.E.
P.O. Box 97046, Redmond, Whashington 98073-9746 USA
Copyright 1992

[Elek91] Programmierbare Logi-ICs: Die Asics für den kreativen Entwickler.
Elektronik, Sonderausgabe Nr. 271, 1991

[LatticeD94] Lattice Data Book 1994
Lattice Semiconductor Corp.
5555 Northeast Moore Court
Hillsboro, Oregon 97124 USA

[LatticeH94] Lattice Handbook 1994
Lattice Semiconductor Corp.
5555 Northeast Moore Court
Hillsboro, Oregon 97124 USA

[Seifart88] Seifart, Manfred:
Digitale Schaltungen.
Hüthig Verlag, Heidelberg, 1988

[Weyer88] Weyerer, Manfred; Goldemund, Gerald:
Prüfbarkeit elektronischer Schaltungen:
Grundlagen, Verknüpfung und Anwendung von CAD und CAT.
Hanser-Verlag, München Wien, 1988

[Wojt88] Wojtkowiak, Hans:
Test und Testbarkeit digitaler Schaltungen.
Teubner-Verlag, Stuttgart, 1988

[Wund91] Wunderlich, Hans-Joachim:
Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test.
Springer-Verlag, Berlin Heidelberg, 1991

Anhang H: Kurzanleitung der Bearbeitungsschritte

- ABEL4 aufrufen → in Programmgruppe **easyABEL** auf „Icon“ **ABEL4** „doppelklicken“
  Dateinamen eingeben, um die Datei zu laden oder

- Entwurfsdatei laden →
  ƒ+F => **File**-Menü
  O => **Open**
  Dateinamen eingeben
  mit < Open > abschließen

- Übersetzen → **ƒ+C** => **Compile**-Menü
  (Compile) **C** => **Compile**
  (mit **O** Optionen bzw. mit **T** Trace Optionen vorgeben)

- Anzeige → **ƒ+V** => **View**-Menü
  C => **Compiler Listing** überprüfen
  (bzw. jeweils die im aktuellen Entwicklungsschritt erzeugten Dateien anzeigen lassen)

- Simulation → **S** => **Simulate** (des aktuellen Entwicklungsschritts)

- Optimieren → **ƒ+O** => **Optimize**-Menü
  R => **Reduce**

- Fitting → **ƒ+S** => **SmartPart**-Menü
  F => **Fit**

- Mapping → **ƒ+P** => **PartMap**-Menü
  (JEDEC-Datei erzeugen) **F** => **FPGA/PLDmap**

- Programmierung → in Programmgruppe **Lattice** auf „Icon“ **IDCD** „doppelklicken“
  „Icon“ **SCAN** anklicken zur automatischen Bausteinerkennung
  mit Browse die JEDEC-Datei im Verzeichnis C:\dataio auswählen
  Operation „Program & Verify“ auswählen
  „Icon“ **RUN** anklicken
  warten bis rote LED erloschen ist, d.h. Programmierung ist abgeschlossen

- Austesten der Hardware

Anhang I: Datei *dummy.abl*

```
module DUMMY                        "Name des Entwurfs vergl. Anhang A

title     'Informationen zu         "Informationen zum Entwurf
          Funktion
          Autor ..'



declarations                        "Hier stehen Ihre Deklarationen

    DUMMYJ device 'P22V10C';        "Der Baustein P22V10C wird verwendet
                                    "und der Name der JEDEC-Datei zum
                                    "Programmieren ist dummyj.jed

equations                           "Hier stehen wenn noetig logische
                                    "Gleichungen und die Taktabhaengigkeit


truth_table                         "Schaltnetz als Wahrheitstabelle


state_diagram                       "Schaltwerk als Zustandsuebergangs-
                                    "diagramm


test_vectors                        "Testmuster fuer die funktionale
                                    "Simulation

end DUMMY                           "Endekennzeichnung
```

DGT ABEL Referenzkarte

ABEL Referenz

General

Comments	*comment* or *comment < End-of-Line >
String	'string'
Blocks	{ block }
Identifiers	Letter or _ followed by letters, digits or _
Number notation	Binary - ^b## Octal - ^o## Hex - ^h## Decimal - [^d]##
Special Constants	.C. .D. .F. .K. .P. .sv2. .sv3. .sv4. .sv5. .sv6. .sv7. .sv8. .sv9. .U. .X. .Z.

Syntax of Elements

Module	MODULE module_name [(dummy_arg [,dummy_arg]...)]
Options	OPTIONS 'option [, option]...'[;]
Title	TITLE 'string'
Declarations	DECLARATIONS declarations ;
Device	device_id DEVICE real_device ;
Pin	[!]pin_id [, [!]pin_id] PIN [pin# [, pin#]] [ISTYPE 'attributes'] ;
Node	[!]node_id [, [!]node_id] NODE [node# [, node#]] [ISTYPE 'attributes'] ;
Constant	id [, id] ... = expr [, expr] ...
Macro	macro_id MACRO [(dummy_arg [dummy_arg]...)] {block} ;
Istype	signal [, signal] [PIN|NODE[##]] ISTYPE 'attr [, attr]' ;
Library	LIBRARY 'name'
Equations	EQUATIONS equations ;
When-Then-Else	WHEN condition THEN [!]element=expression ; [ELSE equation] ; or WHEN condition THEN equation ; [ELSE equation] ;
Truth Table	TRUTH_TABLE (inputs -> outputs)
State Diagramm	STATE_DIAGRAM state_reg [-> state_out] [STATE state_exp : [equation] [equation] : : trans_stmt ; ...]
IF-Then-Else	IF expression THEN state_exp [ELSE state_exp] ; or IF expression THEN state_expression ELSE IF expression THEN state expression ELSE state_expression;
Case	CASE expression : state_expr; [expression : state_exp;] [expression : state_exp;] ... ENDCASE ;
Goto	GOTO state_exp ;
With-Endwith	transition_stmt state_exp WITH equation [equation] ... ENDWITH ;
Fuses Section	FUSES fuse_number = fuse value; or fuse_number_set = fuse value ;
XOR Factors	XOR_FACTOR signal name=xor_factors ;
Test Vectors	TEST_VECTORS [note] (inputs -> outputs) [invalues -> outvalues ;] ...
Trace	TRACE (inputs -> outputs)
End	END [module_name]

Attributes

Attribute	Description
'buffer'	No inverter
'com'	Combinatorial
'inverter'	Inverter
'neg'	Complement signal
'pos'	Do not complement signal
'reg'	Clocked memory element
'reg_D'	Clocked memory element - D-type flip-flop
'reg_T'	Clocked memory element - T-type flip-flop
'reg_SR'	Clocked memory element - SR-type flip-flop
'reg_JK'	Clocked memory element - JK-type flip-flop
'reg_G'	Memory element - D-type flip-flop with gated clock
'xor'	XOR gate

Directives

@ALTERNATE	Alternate operators
@CONST id = expression ;	Constant definition
@DCSET	Don't care Set Adjustable
@EXIT	
@EXPR [block] expr;	Expression
@IF expr. block	Include block if true
@IFB (arg) block	Include block if blank
@IFDEF id block	Include block if defined
@IFIDEN (arg1,arg2) block	Include block if identical
@IFNB (arg) block	Include block if not blank
@IFNDEF id block	Include block if not defined
@IFNIDEN (arg1,arg2) block	Include block if not identical
@INCLUDE filespec	include file
@IRP dummy_arg (arg [, arg] ...) block	Indefinite repeat
@IRPC dummy_arg (arg) block	Indefinite repeat, character
@MESSAGE 'string'	Print message
@ONSET	Don't Care Set Fixed
@PAGE	Form feed in listing file
@RADIX expr ;	Change default base numb.
@REPEAT expr block	Repeat block *n* times
@STANDARD	Return to default operators

Dot Extensions

.AP	Asynchronous Preset
.AR	Asynchronous Reset
.CE	Clock-enable input to a gated-clock flip-flop
.CLK	Clock
.D	Data input to a D-type flip-flop
.FB	Registered feedback normalized to pin
.FC	Flip-flop mode control
.J	J input to a JK-type flip-flop
.K	K input to a JK-type flip-flop
.LD	Load input for registers
.LE	Active low latch enable
.LH	Active high latch enable
.OE	Output Enable
.PIN	Pin Feedback
.PR	Preset
.Q	Q feedback
.R	R input to a RS-type flip-flop
.RE	Reset
.S	S input to a RS-type flip-flop
.SP	Synchronous register preset
.SR	Synchronous register reset
.T	T input to a T-type flip-flop

Logical Operators

!	NOT : ones complement
&	AND
#	OR
$	XOR: exclusive OR
!$	XNOR: exclusive NOR

Operator Priorities

1	-	negate
1	!	NOT
2	&	AND
2	<<	shift left
2	>>	shift right
2	*	multiply
2	/	unsigned division
2	%	modulus
3	+	add
3	-	substract
3	#	OR
3	$	XOR
3	!$	XNOR
4	==	equal
4	!=	not equal
4	<	less than
4	<=	less than or equal
4	>	greater than
4	>=	greater than or equal

Relational Operators

==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

Arithmetic Operators

-A	twos complement
A-B	substraction
A+B	addition
A*B	multiplication
A/B	unsigned integer division
A%B	modulus: remainder from /
A<<B	shift A left by B bits
A>>B	shift A rigth by B bits

Assignment Operators

=	Combinatorial assignment
:=	Registered assignment (D-type flip-flop)

DGT ABEL Referenzkarte

Valid Set Operators

Operators	Expression	•	Result
!,?	![a, b, c]	•	[!a, !b, !c]
-(unary)	-[a, b, c]	•	![a, b, c] + 1
&, #, $, !$	[a, b, c] & [x, y, z]	•	[a & x, b & y, c & z]
all	[x, y, z] & a	•	[x, y, z] & [a, a, a]
.	[x, y, z].d	•	[x.d, y.d, z.d]
==	[a, b, c] == [x, y, z]	•	(a == x) & (b == y) & (c == z)
!=	[a, b, c] != [x, y, z]	•	(a != x) # (b != y) # (c != z)
+	[a3, a2, a1] + [b3, b2, b1]	•	[a3 $ b3 $ c2, a2 $ b2 $ c1, a1 $ b1 $ c0] where c2 = (a2 & b2) # (a2 &c1) # (b2 & c1) c1 = (a1 & b1) # (a1 &c0) # (b1 & c0) c0 = 0
-(binary)	[a, b, c] - [x, y, z]	•	[a, b, c] + (- [x, y, z])

<	[a3, a2, a1] + [b3, b2, b1]	•	c3
			where
			c3 = (!a3 & (b3 # c2) # a3 & b3 & c2) != 0
			c2 = (!a2 & (b2 # c1) # a2 & b2 & c1) != 0
			c3 = (!a1 & (b1 # c0) # a1 & b1 & c0) != 0
			c0 = 0

<=	[a, b, c] <= [x, y, z]	•	!([a, b, c] > [x, y, z])
>=	[a, b, c] >= [x, y, z]	•	!([a, b, c] < [x, y, z])
>	[a, b, c] > [x, y, z]	•	!([a, b, c] < [x, y, z])
all	6 & [a, b, c]	•	[1, 1, 0] & [a, b, c] ***expansion***
all	6 & [a, b]	•	[1, 0] & [a, b] ***truncation***
all	6 & [a, b, c,d]	•	[0, 1, 1, 0] & [a, b, c, d] ***padding***

Language Processor

ABEL Batch Processing

abel4bat *filename* [options]

Global Options

@ *response_filename*.rsp
-i *filename*
-o *filename*
-help
-usage
-silent
-menu
-errlog

AHDL2PLA

ahdl2pla *ahdlfile* [*options*]
-args *arguments*
-list [expand]
-ovector *filename*
-syntax
-vectors

PLASim

plasim *plafile* [*options*]
-ivectors *filename*
-break *first_vector* [*last_vector*]
-signal *signal_name pin# 's*
-trace [pins|waveltable|macro|none]
[brief|clock|detail]
-x (0|1|h|l)
-z (0|1|h|l)
-initial (0|1)

PLAOpt

plaopt *plafile* [*options*]
-reduce [bypin|group] [chooselfixed][dt] [exact]

Fuseasm

fuseasm *plafile* [*options*]
-ivectors *filename*
-idevice *device*
-ues *signature_word*
-document [brief|long] [plcc]
-checksum [none|full|dummy]
-format [default|82|83|87|88|hex|brief|full|pof]
-config (ptintact noturbo nomiser lock)
-x (0|1)

JEDSim

jedsim *jedecfile* [*options*]
-ivectors *filename*
-break *first_vector* [*last_vector*]
-signal *signal_name pin# 's*
-trace [pins|waveltable|macro|none]
[brief|clock|detail]
-x (0|1|h|l)
-z (0|1|h|l)
-initial (0|1)

Utilities

pla2eqn -i *infile* [-o *outfile*]
[-language abel|pds|lca|snap|none]
finddev4 *string*
cleanup4 [all]

Options

SmartPart - Devsel

devsel *plafile* [*options*] -manufacturer
mfr_name
- database *filename*
-device *device*
-sort *range*
-power *range*
-uf1 *range*
-uf2 *range*
-pins *range*
-price *range*
-speed *range*
-utilization *range*
-package [dip pdip cdip sdip lcc plcc clcc sogc
sogp]
-sps [com|mil|ind|std]
technology [cmos ttl ecl gaas]
-erasable [no|yes, uv ee]
-jamload [yes|no]
-log *filename*

range = exact ## | less ## | greater ## | range ##

SmartPart - Fit

fit *plafile* [options] fitter -i *filename*
-log *filename*
-device *device*
-idevice *filename*
-adhoc *filename*
-first
-supress
-preassign [keeplignore|try]

ABEL Design Enviroment

Starting

abel4 [*filename*] [-43|-50|bw]

Keyboard

Alt -letter	Pop up menu
F1	Help
F2	Pop up options
F4	Process through Simulate JEDEC
F5	Same as <OK>
F6	Clear entry field
F10	Same as <Cancel>
<space>	Toggle (*) or select [X]
Tab or ↵	Next selection
Shift Tab	Previous selection

Editor

Ctrl D	Delete current line
Ctrl R	Replicate current line
Ctrl N	Find next search string
Ctrl ←	Go to beginning of file
Ctrl →	Go to end of file