5 Programmierbare Logikelemente (PLD)

Die Digitaltechnik wird in einem Maße durch die Fortschritte der Halbleitertechnik, speziell der Mikroelektronik geprägt und beeinflußt, wie es selten in anderen Bereichen der Fall war. Dabei handelt es sich keinesfalls "nur" um den Mikroprozessor, sondern um eine Vielzahl moderner Halbleiterbauelemente, die das "klassische" Angebot an Standardschaltungen der Hersteller ergänzen und erweitern. Begriffe wie 'ASIC', 'Semi-custom-IC', 'full-custom- IC' und andere werden in der Fachpresse genüßlich ausgebreitet, in der stillen Hoffnung, ein jeder werde sich schon im 'silicon valley' schlau machen. Da es sich bei der Umsetzung dieser Begriffe weniger um die Digitaltechnik als vielmehr um strategische und konzeptionelle Fragen der Elektronikentwicklung handelt, sollen an dieser Stelle die Begriffe nur erläutert werden. Ein Teilbereich dieser neuen Konzepte wird mit dem Kürzel PLD (**P**rogrammable **L**ogic **D**evice) umschrieben. Wegen ihrer breiten Vorzüge konnten sich diese "ASICs des kleinen Mannes" in der Praxis durchsetzen. Ihre Grundlagen werden in den folgenden Abschnitten behandelt.

5.1 Einführung

Mit fortschreitender Entwicklung der Halbleitertechnik wurde es möglich, immer komplexere Schaltungen wie z.B. Rechenschaltungen (ALUs), Multiplexer, sowie weitere Funktions- und Speicherschaltungen zu verwirklichen. Ein Nachteil dieses Trends besteht darin, daß mit wachsender Komplexität der Schaltungen deren Anwendungsbreite immer geringer wurde. Bild 5.1 läßt diese Zusammenhänge erkennen.

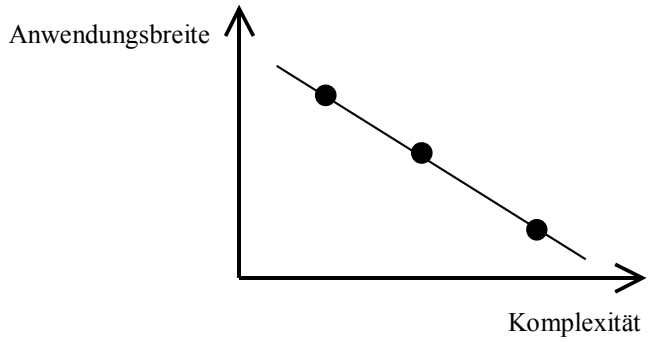


Bild 5.1: Zusammenhang zwischen Komplexität und Anwendungsbreite von integrierten Halbleiterschaltungen

Für den Entwickler von Digitalsystemen bedeutet dies, daß er neben den klassischen, heute schon als "diskret" bezeichneten ICs der Integrationsstufe SSI eine Vielzahl weiterer, höherer integrierter Bausteine der Stufen MSI und LSI verwenden mußte.

Um nun zu besonders flexiblen und gleichzeitig wirtschaftlichen Lösungen zu gelangen, schuf die Halbleiterindustrie -beginnend Mitte der 70-er Jahre- die programmierbaren Logikelemente PLD (**P**rogammable **L**ogik **D**evice). Die Eingruppierung dieser neuen Bauelemente in die große Familie integrierter Halbleiterschaltungen zeigt Bild 5.2.

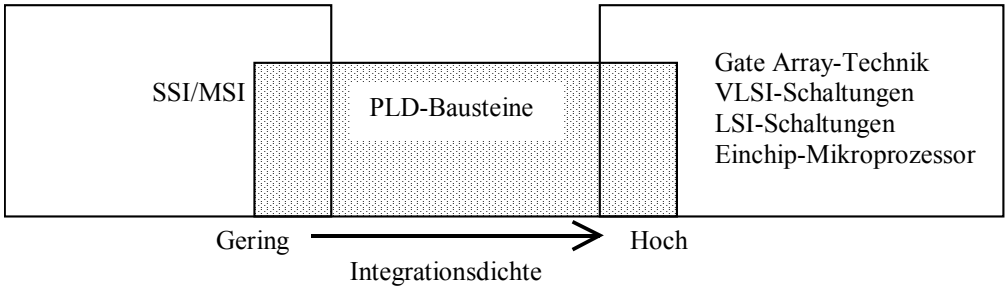


Bild 5.2: Eingruppierung und Anwendungsbreite von PLDs

Die bekanntesten programmierbaren Logikelemente sind die PALs (**P**rogrammable **A**rray **L**ogik). Im folgenden wollen wir Struktur, Aufbau und Programmierung dieser modernen Halbleiterchips untersuchen. Da die gesamte Literatur in diesem Bereich englischsprachig ist, haben sich hier auch noch nicht die Schaltungssysmbole nach DIN bzw. IEC durchsetzen können. Statt dessen sind immer noch die amerikanischen Symbole gebräuchlich.

5.2 Grundstruktur von PALs

In Bild 5.3 ist ein einfaches UND-Gatter mit den beiden Eingängen A bzw. B dargestellt. Die logische Funktionsgleichung ist:

$P = A \cdot B$ in CUPL: P = A & B

Bild 5.3: UND-Gatter mit zwei Eingängen

Das Symbol '&' wird bei der Aufstellung von Funktionsgleichungen in PAL-Technik bzw. bei der PAL-Programmierung Hilfe des speziellen Programms CUPL benutzt. Der noch weit verbreitete PAL-Assembler PALASM sieht hierfür das Symbol '*' vor. Logisches ODER entspricht dem bereits bekannten '+'. CUPL erwartet in diesem Fall das Zeichen '#' zur Darstellung einer Disjunktion. Im nachfolgenden Bild 5.4 ist die UND-Verknüpfung einmal in der typischen PAL-Darstellungsweise skizziert.

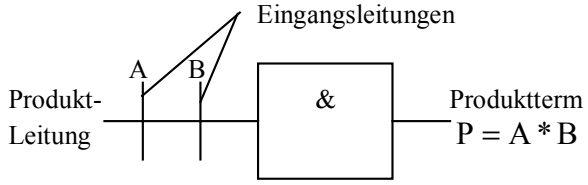


Bild 5.4: UND-Gatter in PALASM-Darstellung

Die Bezeichnungsweisen "Produktterm" bzw. "Produktleitung" rühren von der algebraischen Gleichung

PALASM: $P = A * B$, CUPL: P = A & B; (logisches Produkt)

her. Das Arbeitsprinzip der PALs ist nun recht einfach. Eingangsleitungen und Produktleitungen bilden "logische" Kreuzungspunkte, die entweder verbunden oder nicht verbunden sind. Damit

das Ganze auch physikalisch funktioniert, ist es nur erforderlich, die Eingangsleitungen mit Hilfe von Pufferverstärkern (Buffer) von den Produktleitungen zu entkoppeln und zusätzlich die Eingangssignale jeweils zu negieren. Bild 4.22 läßt dieses Arbeitsprinzip erkennen. Im Unterschied zu Bild 4.21 sind jetzt zusätzlich die negierten Eingangssignale $\overline{A}$ bzw. $\overline{B}$ vorhanden, die in der PALASM- bzw. CUPL-Systematik folgendermaßen gekennzeichnet werden:

PALASM: $/A \mathrel{\hat{=}} \overline{A}$ $/B \mathrel{\hat{=}} \overline{B}$ CUPL: $!A \mathrel{\hat{=}} \overline{A}$ $!B \mathrel{\hat{=}} \overline{B}$

Diese Darstellungsweise folgt aus der Notwendigkeit zu einzeiliger Eingabe bei EDV-Geräten (Überstreichungszeichen standardmäßig nicht vorgesehen).

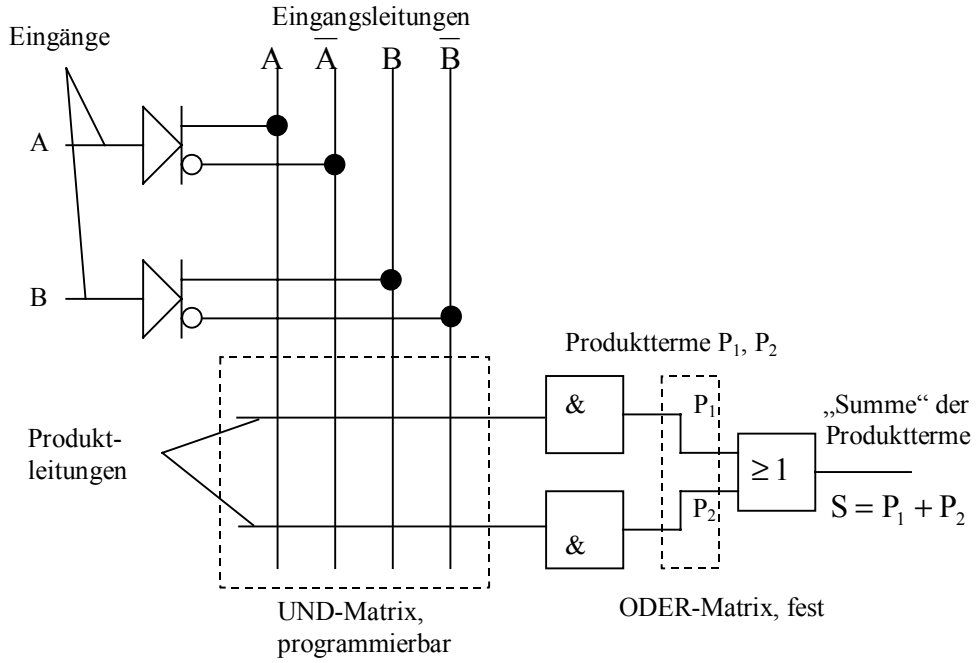


Bild 5.5: Grundstruktur eines PAL

In Bild 5.5 sind die beiden Produktterme P_1 bzw. P_2 vorhanden, die über ein ODER-Gatter verknüpft sind, an dessen Ausgängen die "Produktsumme" $S = P_1 + P_2$ gebildet wird. Damit ist auch schon die Grundstruktur aller PAL-Bausteine vorgegeben. Die Ausgänge aller UND-Gatter sind mit den Eingängen der ODER-Gatter fest verbunden. Dieser Teil der mikroelektronischen Schaltung kann also nicht verändert werden. Die Programmierbarkeit dieser Bauelemente liegt vielmehr in der Verknüpfung der Eingangsleitungen mit den Produktleitungen. Diese sind über schmale Metallstege oder "Sicherungen" miteinander verbunden. Bild 5.6 zeigt den Aufbau.

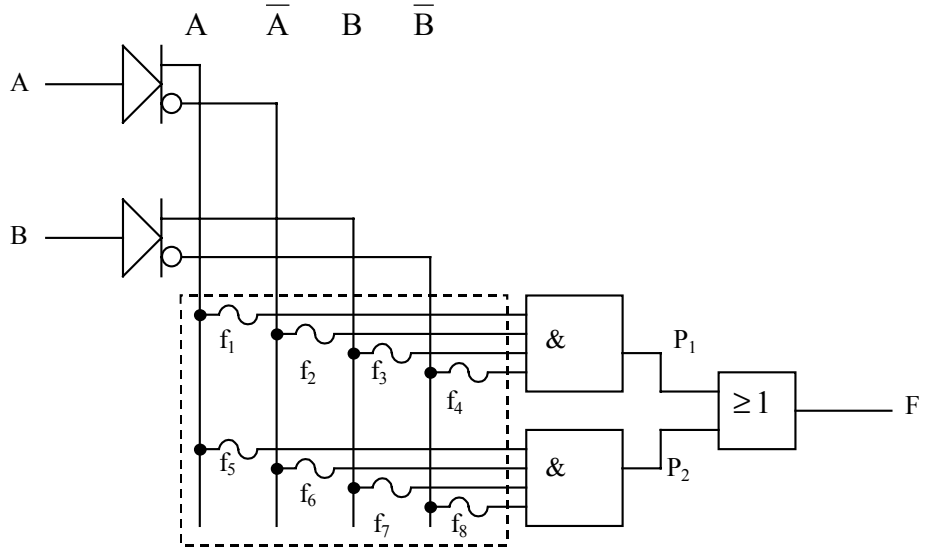


Bild 5.6: Prinzip eines PAL mit programmierbaren Sicherungen f_1 bs f_8

Es sind die Mikro-Sicherungen f_1 bis f_8 zu erkennen, die über elektronische Impulse aufgetrennt werden können. Das Programmieren eines PAL bedeutet also :

- bewußtes Unterbrechen von mikroskopisch feinen Sicherungen nach einem vorgebenen Bildungsgesetz.

Wie aber ist der Zusammenhang zwischen gewünschter Schaltung und dem "Fuse Pattern", d. h. dem Muster von intakten sowie durchgetrennten Sicherungen herzustellen? Betrachten wir ein einfaches Beispiel, die Antivalenz S (Exklusiv-ODER-Verknüpfung) der beiden Variablen A sowie B.

PALASM: S = A * /B + /A * B CUPL: S = A & !B # !A & B:

Damit nun die gewünschte Schaltung verwirklicht werden kann, müssen die Sicherungen f2, f3, f5, f8 durchtrennt, hingegen die übrigen Sicherungen f_1, f_4, f_6, f_7 intakt bleiben. Das Ergebnis dieses Programmiervorganges ist in Bild 5.7 zu erkennen.

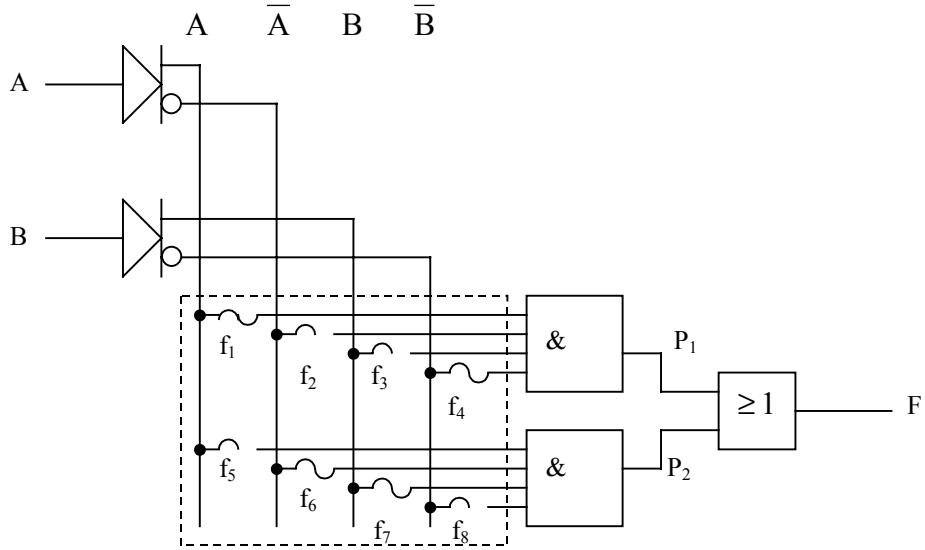


Bild 5.7: Antivalenz als Beispiel für PAL-Programmierung

Aus Gründen übersichtlicher Darstellung wird allgemein ein gleichwertiges System entsprechend Bild 5.8 benutzt. Alle physisch in der Schaltung vorhandenen Produktleitungen eines UND- Gatters, die Eingangsleitungen also, sind durch ein einziges Leitungssymbol dargestellt.

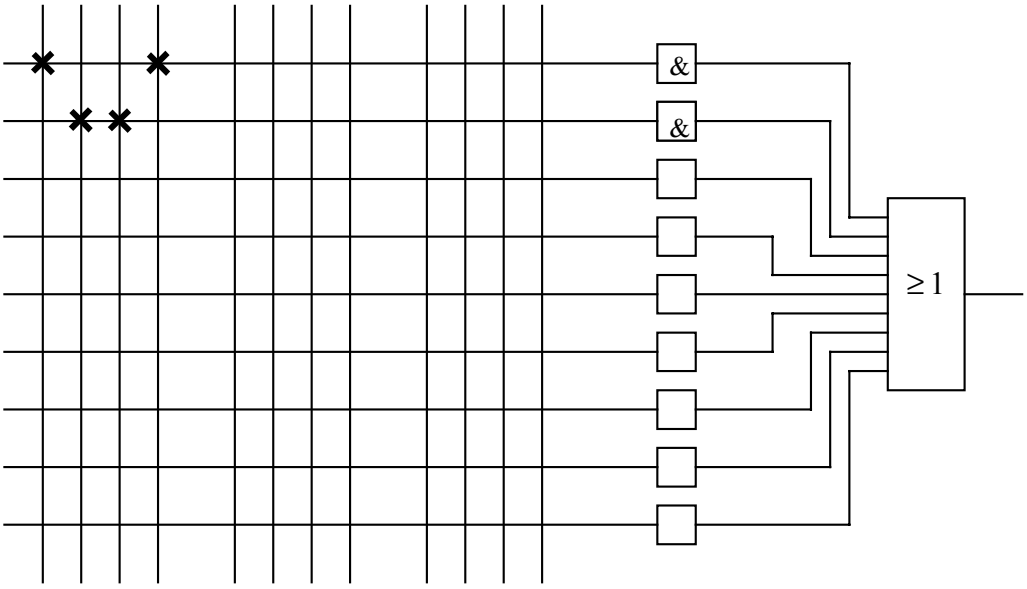


Bild 5.8: Beispiel für PAL-Programmierung (Antivalenz), spezielle PAL-Darstellungsweise

Eine intakte Sicherung wird daher zunächst mit einem 'X', eine durchtrennte mit einem Symbol für Leitungskreuzung '+' versehen. Bei der Programmierung technischer PAL-Bausteine wird für die Notation wegen der vorhandenen Ausgangsinverter eine erweiterte Vereinbarung getroffen. Im Folgenden sollen noch die beiden Spezialfälle betrachtet werden:

(a) keine Sicherung durchtrennt, d.h. alle noch intakt. Damit ergibt sich für S:

S = A*/A + B*/B = 0 (PALASM-Syntax) S= A&!A # B&!B = 0 (CUPL-Syntax)

(b) alle Sicherungen durchtrennt. In diesem Fall ergibt sich für das UND-Gatter ein offener Eingang, der sich schaltungstechnisch als logische 1 auswirkt.

Diese beiden Spezialfälle sind in Bild 5.9 dargestellt.

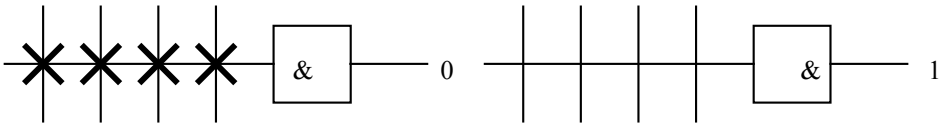


Bild 5.9: Zwei Spezialfälle der PAL-Programmierung

Aus diesen beiden Grenzfällen folgt, daß noch unbenutzte Summen- bzw. einzelne Produktterme eines PAL nachprogrammiert werden können. Fehlerhaft programmierte Terme (Fall (b)) hingegen können nicht "blockiert" werden, der PAL-Baustein ist zu verwerfen. Bei Versuchschaltungen wird in einem derartigen Fall häufig der Ausgang eines noch freien Terms verwendet.

5.3 Die Systematik der programmierbaren Logikbausteine

Die folgenden drei Grundtypen von Logikbausteinen bieten für jeden Anwendungsbereich eine optimale Lösung, wobei teilweise von den Vereinbarungen der Fa. MMI ausgegangen wurde.

PLE	**P**rogrammable **L**ogic **E**lement	Feste UND-Matrix programmierbare ODER-Matrix
PLA	**P**rogrammable **L**ogic **A**rray	Beide Matrixfelder programmierbar
PAL	**P**rogrammable **A**rray **L**ogic	Programmierbare UND-Matrix feste ODER-Matrix

Die nachfolgende Tabelle 5.1 enthält die wichtigen Eigenschaften der verschiedenen Bausteinarten.

Tabelle 5.1 Eigenschaften verschiedener PLD

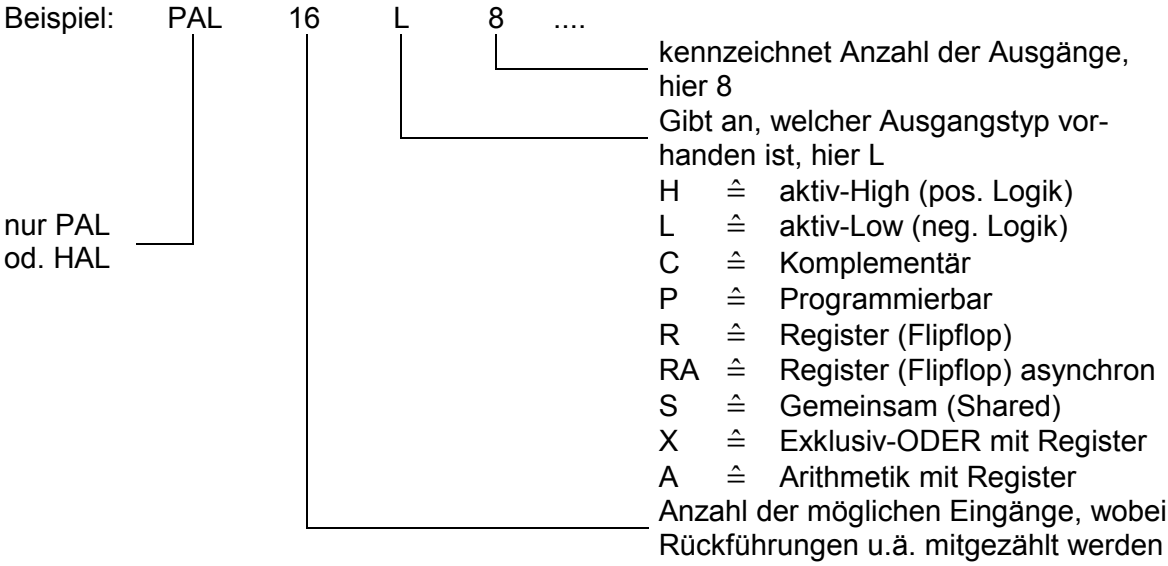
Typ	UND-Matrix	ODER-Matrix	Programmierung	Ausstattungsart
EPROM	fest	progr.	Ladungsträger	TS
PROM	fest	progr.	Hardwaresich.	TS
PLE	fest	progr.	"	TS, Reg, Pol
PAL	progr.	fest	"	TS, Reg, Pol, Fb.
GAL	progr.	fest	Ladungsträger	alle PAL-Strukt.
HAL	progr.	fest	Maske	"
PLA	progr.	progr.	Hardwaresich.	TS, OC, POL.
FPGA	progr.	nicht vorh.	"	"
FPLS	progr.	progr.	"	TS, Reg., Fb.

TS = Tri-State
OC = Open Collector
Reg = Register
Pol = programmierbare Polarität
Fb = Rückführungs-Ausgang (Feedback I/O)

Übersicht: PAL-Strukturen

Wie bereits in Abschnitt 5.2 dargestellt, besitzen die PALs von ihrem grundsätzlichen Aufbau her ein programmierbares UND- Matrixfeld, wobei jeweils die Ausgänge auf ein festverdrahtetes ODER-Feld geführt sind. Den ODER-Termen sind je nach Aufbau des PALs unterschiedlich viele Produktterme (UND-Gatter) zugeordnet. Die Zahl der Produktterme pro Ausgang liegt je nach PAL-Typ bei 2-16. Die Anzahl der Eingänge variiert zwischen 10 und 64.

Für die praktische Anwendung steht eine großes Spektrum verschiedener PAL-Typen zu Verfügung, die alle die o.a. gemeinsame Grundstruktur besitzen. Starke Unterschiede sind allerdings bei der jeweiligen Ausgangsschaltung zu bemerken. Hierdurch ist es möglich, für die unterschiedlichsten Aufgabenstellungen stets den zweckmäßigsten PAL-Typ auszuwählen. Für die verschiedenen PAL- Schaltungsstrukturen wurde von MMI ein sehr praktisches System zur Kennzeichnung vorgegeben, welches auch bei den übrigen Herstellern Eingang gefunden hat. Am Beispiel einer Typnummer sollen hier nur die wichtigsten Merkmale behandelt werden



Nachfolgend eine kurze Betrachtung der wichtigsten zu Verfügung stehenden PAL-Ausgangsbeschaltungen.

L PALs mit dieser Ausgangsstruktur besitzen einen sehr einfachen Aufbau ohne Speicherelemente. Daher sind nur Schaltnetze geringer Komplexität (z.B. Adreßdecoder u.ä) realisierbar.

P Mit dieser E-/A-Struktur lassen sich wesentlich komplexere Schaltungen vewirklichen. Ausgang A kann durch die Rückführleitung in das UND-Matrixfeld einbezogen werden. Die Bedingung für die Tri-State Funktion ist programmierbar. Im Tri- State Zustand befindliche Ausgänge, die eine Rückführleitung besitzen, können als zusätzliche Eingänge benutzt werden (bidirektionaler Datenverkehr).

R Zähler, Schieberegister und ähnliches lassen sich mit den **R-** PAL-Bausteinen verwirklichen, bei denen die Ausgänge über Register zurückgeführt sind (Registered Feedback). Zwischen das ODER-Matrixfeld und die Ausgangsinverter sind bei den R- Typen D-Flipflops geschaltet, deren Ausgänge in die Matrix zurückgeführt sind. Für alle Flipflops steht eine gemeinsame Taktleitung CLK zur Verfügung. Die Ausgangsinverter können über einen gemeinsamen Enable-Eingang hochohmig geschaltet werden. Die PALs vom Typ RA (**R**egister **A**synchron) ermöglichen den Aufbau asynchroner Schaltungen. Bei den PAL-Typen der Serie RA, S sind nämlich sowohl die Taktleitungen als auch die Tri-State Funktionen der Ausgangsinverter für jeden Ausgang programmierbar.

Es stehen weiterhin PALs zur Verfügung, bei denen 2 ODER-Terme auf ein Exclusiv-ODER Gatter geführt werden.

PALs der P-Serie (z.B. 16P8,16RP6) ermöglichen es, die Polarität des Ausgangssingnals zu programmieren (Low-Aktiv, High-Aktiv). Arithmetische Funktionen (addieren, subtrahieren, vergleichen) lassen sich mit den Pal-Typen 16A4, 16X4 realisieren. Die Ausgänge sind mit den Eingängen EXOR verknüpft auf die Matrix zurückgeführt.

PALs der S-Serie (z.B. 20RS4, 20S4) bieten die Möglichkeit, die Anzahl der Produktterme je Ausgang zu verändern. Darüberhinaus liefern manche Halbleiterhersteller PALs mit eingangsseitigen Registern (Latches), die weitere spezielle Anwendungen erlauben.

GAL-Bausteine

Eine Weiterentwicklung der PALs stellen die GAL-Bausteine dar. Sie wurden von der amerikanischen Halbleiterfirma Lattice Semiconductors erfunden und besitzen bezüglich der Sicherungsmatrix (Fuse-Map) den gleichen logischen Aufbau wie die PALs. Im Gegensatz zu diesen haben sie jedoch keine fest verdrahteten, sondern frei programmierbare Ausgangszellen. Durch diesen flexiblen Aufbau ist möglich, mit nur zwei verschiedenen GAL-Typen fast alle Standard-PALs zu ersetzen.

Die Struktur der Ausgangszelle, auch "**O**utput **L**ogic **Mac**rocell" (OLMC) genannt,ist in Bild 5.10 dargestellt.

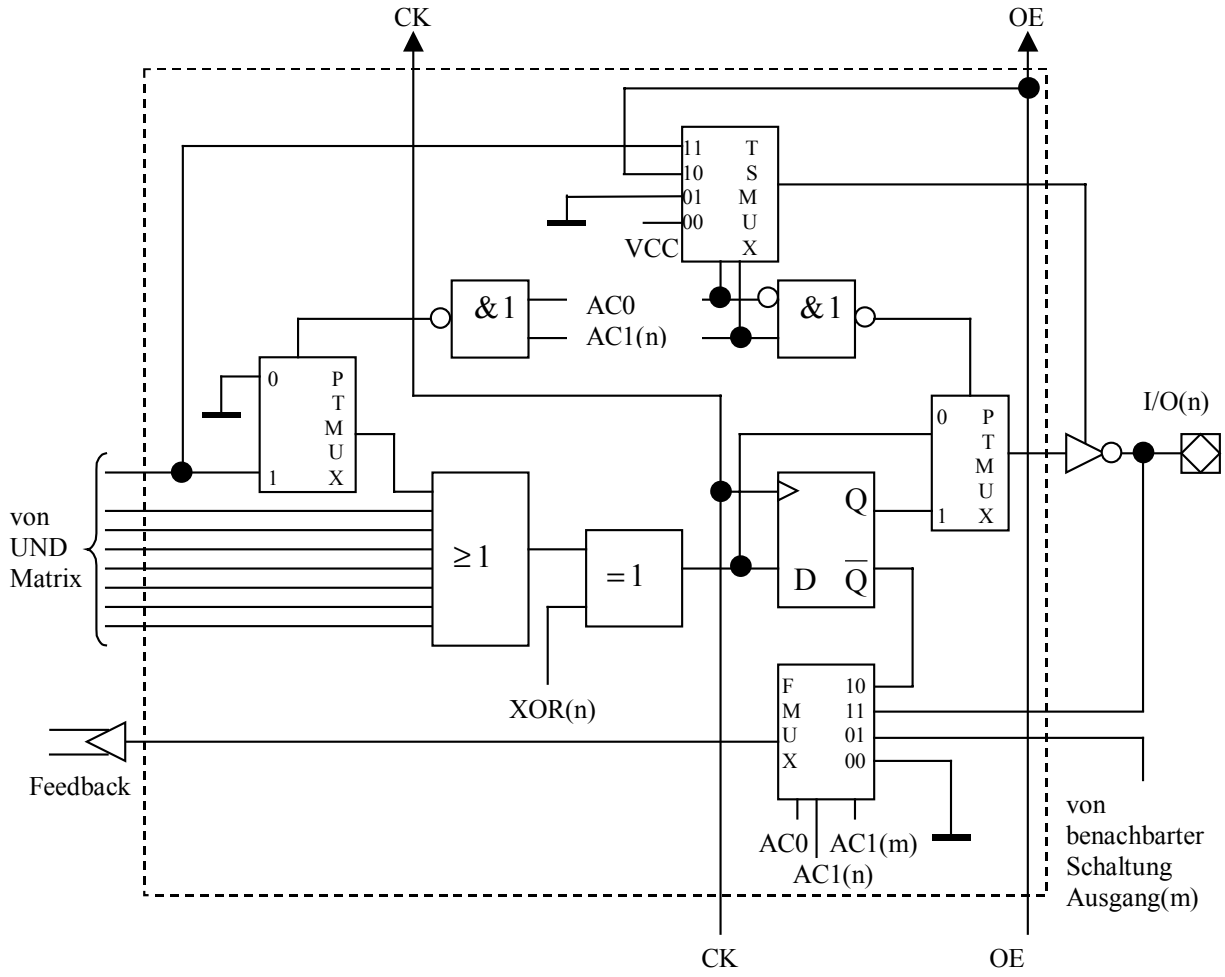


Bild 5.10: Struktur der Ausgangszelle eines GALs

Diese "Macrocell" kann durch gezielte Programmierung bestimmter zugehöriger Sicherungen eingestellt werden. Folgende Ausgangsstrukturen können programmiert werden:

1. Normaler Eingang
2. Kombinatorischer Ausgang
3. Kombinatorischer Ausgang mit Tri-State-Möglichkeit
4. Register-Ausgang
5. Register-Ausgang mit Rückführung und Eingangsmöglichkeit

Die Programmierung wird auf verschiedene Steuersignale zurückgeführt, die im einzelnen erläutert werden sollen:

AC0	AC1 (n)	
0	0	Anschluß ist Ausgang
0	1	Anschluß ist Eingang
1	0	Tri-State Freigabe durch Pin
1	1	Tri-State Freigabe durch Produktterme

Das SYN -Bit legt fest, ob der Baustein synchrone Ausgänge (Ausg. mit Flipflops) besitzen soll. Das EXOR -Bit jedes Ausgangs wird zur Programmierung der Polarität des Ausganges benutzt.

Diese hier genannten Steuersignale werden in der Beschreibung der GAL-Bausteine durch ein "Architecture Control Word" dargestellt. Es enthält aber auch die Informationen über die Freigabe der Produktterme.

Da die GAL-Bausteine in der energiesparenden CMOS-Technik (Kapitel 6) gefertigt werden, ist der notwendige Leistungsbedarf von Digitalschaltungen, die mit GAL anstelle von PAL bestückt sind, deutlich niedriger. Weil auch keine "Schmelzsicherungen" wie bei PAL irreversibel durchtrennt werden, können diese Bausteine wiederholt neu programmiert werden.

Nachfolgend die wichtigsten Vorteile von GALs gegenüber PALs:

- GALs sind mehrfach programmierbar
- 2 GAL-Typen (16V8 und 20V8) ersetzen 42 Standard PALs
- wesentlich geringerer Stromverbrauch
- höhere Schaltgeschwindigkeit

Weit verbreitet sind höher integrierte GAL-Bausteine wie z.B, 22V10 oder 26V12. Die praktischen Vorzüge dieser programmierbaren Bauelemente sind so groß, daß praktisch alle PAL-Hersteller inzwischen auch GAL-Bausteine vertreiben. Durch die Konzentration auf PAL und GAL haben die weiteren Typen von PLDs stark an Bedeutung verloren. Hinzu kommt das Interesse der Halbleiterindustrie, bei komplexen Digitalschaltungen in mittleren bis großen Stückzahlen ASICs (**A**pplication **s**pecific **I**ntegrated **C**ircuit ≙ anwenderspezifische Schaltungen) zu entwickeln. Naturgemäß stehen diese nur dem Auftraggeber zu Verfügung.

HAL-Bausteine

Wenn der Anwender sich für die Verwendung von HAL-Bausteinen (**H**ard-**A**rray-**L**ogic) entscheidet, so erhält er, ähnlich wie beim Gate-Array, fertig programmierte Bausteine mit seiner Schaltung vom Hardwarehersteller. Beim HAL wird die Programmierung schon beim Herstellungsprozeß über die Metallisierungsmaske durchgeführt. Der Vorteil hiebei ist, daß bei großen Stückzahlen niedrige Kosten erreicht werden und der Anwender selbst nichts programmieren muß, da er fertige Bausteine bezieht; damit werden auch keine Schaltungen durch Programmierfehler unbrauchbar. Die Entwicklungszeit und -kosten sind geringer als beim Gate-Array, die Komplexität ist die gleiche wie beim PAL.

PLE-Bausteine

PLEs (**P**rogr. **L**ogic-**E**lement) sind bezüglich der Struktur des UND- /ODER-Feldes identisch mit der eines PROM-Bausteins. Alle möglichen Kombinationen der Eingänge sind fest im UND-Matrifeld enthalten. Jeder Ausgang ist über das UND-Feld mit 2^n Produkttermen verbunden (n = Anzahl der Eingänge). Solche Bausteine sind dann vorteilhaft, wenn große Zahlen von Eingangskombiationen oder eine große Anzahl von Produkttermen pro Ausgang benötigt werden.

Als Beispiel sei PLE 12P8 genannt, dieses PLE besitzt 4096 Produktterme (ODER) und 8 Ausgänge.

PLEs mit ausgangsseitigen Registern ermöglichen die Verwirklichung komplexer sequenzieller Logikschaltkreise, bei denen eine große Anzahl von Eingangsvariablen in der Zustandsgleichung enthalten sind.

Wie schon bei den PROM-Bausteinen erläutert, lassen sich nur Gleichungen realisieren, bei denen die einzelnen Produktterme alle Eingangsvariablen enthalten. Da die Gleichungen für die logischen Funktionen in der Regel diese Voraussetzung nicht erfüllen, müssen die fehlenden Variablen ergänzt werden. "Von Hand" wäre dies eine mühselige Angelegenheit. Leider ist die Umsetzung der Gleichungen mit üblichen PAL-Assemblern nicht möglich. Hierfür gibt es spezielle PLE-Assembler (PLEASM), die die Gleichungen in eine für PLEs geeignete Form bringen, und die Fuse-Matrix generieren.

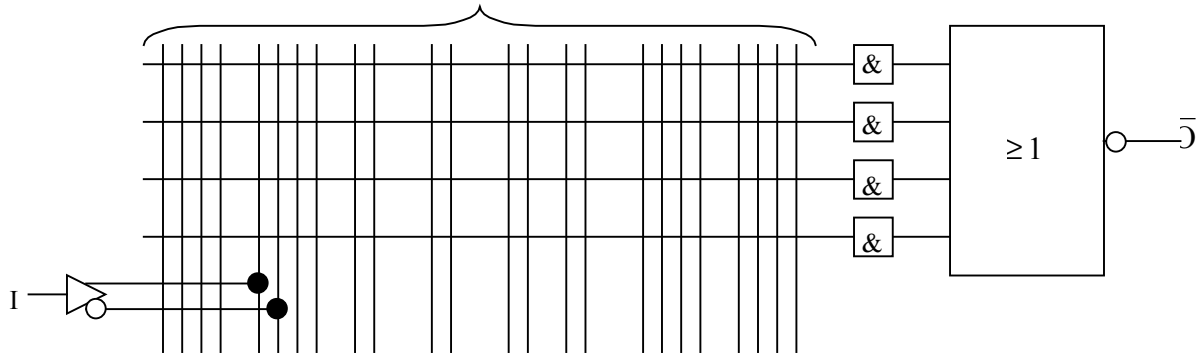
FPLA (PLA)-Bausteine

Diese Bausteine vereinen die Vorzüge der PALs sowie der PLEs (PROMs), da bei ihnen beide Matrixfelder programmierbar sind. Komplexe FPLA-Typen besitzen eine zusätzliche Matrix für die Programmierung der Kontrollterme, wobei auch der Flipflop-Typ des Ausgangs programmierbar ist. Sie erfordern jedoch vom Anwender ein hohes Maß an Kenntnissen, um die Bausteine voll ausnutzen zu können.

5.4 Arbeiten mit PAL- und GAL-Bausteinen

Um sinnvoll mit PALs arbeiten zu können, ist die Kenntnis der PAL-Grundstrukturen und der gebräuchlichen Schaltsymbole erforderlich. Diese sind in den Bildern 5.11a und 5.12 zusammengefaßt.

(a) PAL mit Ein-/Ausgangsstruktur



(b) PAL I/O Feedback und progr. Tri-State

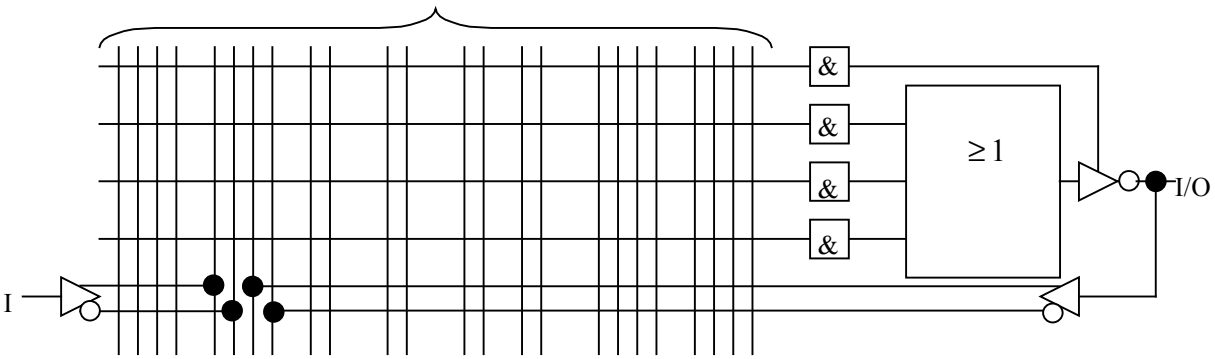


Bild 5.11a/b: PAL-Symbole/Strukturen, kombinatorische Logik

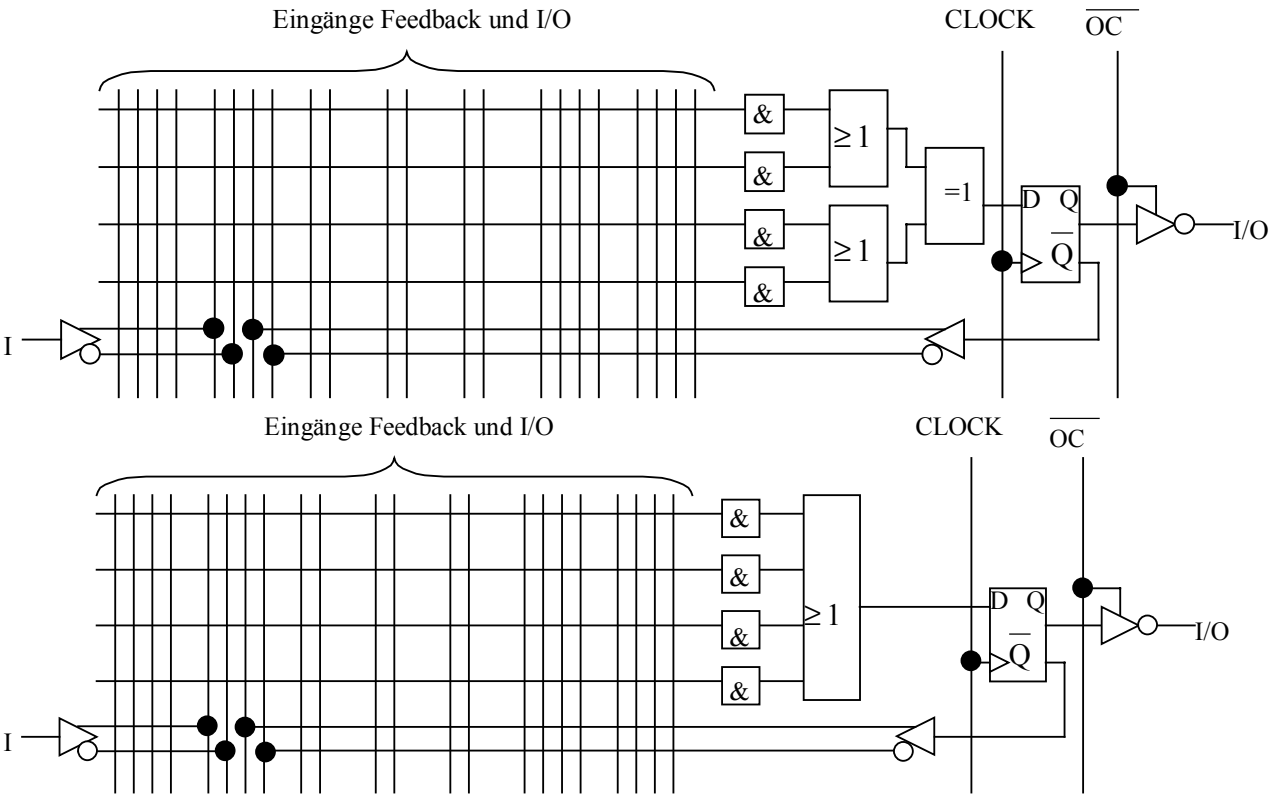
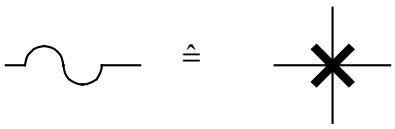
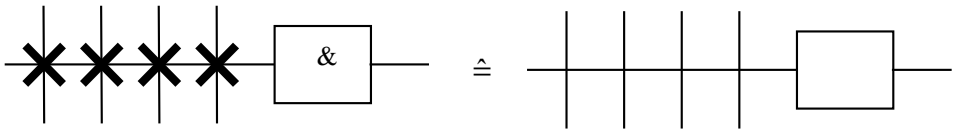


Bild 5.12: PAL-Symbole/Strukturen, Registerausgang mit Rückführung (Feedback) und invertierendem Tri-State-Treiber

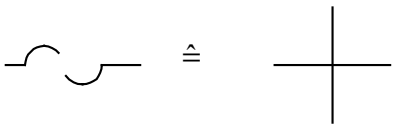
Eine intakte Sicherung wird wie folgt dargestellt:



Sind alle Sicherungen einer Produktleitung intakt, ergibt sich folgende vereinfachte Darstellung:



Eine durchgetrennte Sicherung wird wie folgt dargestellt:



5.5 PLD-Programmierung mit CUPL

Zur Beschreibung einer Digitalschaltung als PLD-Entwurf wären prinzipiell auch die bekannten Hardware-Beschreibungssprachen VHDL bzw. Verilog geeignet. Zwecks Programmierung müßte ein entsprechendes Entwurfsprogramm vom „Silicon Compiler“ übersetzt und die notwendigen Programmierinformationen zur PLD-Programmierung mit Hilfe eines entsprechenden Programmiergeräts in einer Datei gespeichert werden. Da die bisherigen PLD-Architekturen einen verhältnismäßig einfachen Aufbau besitzen, ist es nicht erforderlich, die Vielfalt und

Komplexität einer allgemeinen Beschreibungssprache anzuwenden. Es gibt daher in diesem Bereich spezielle, recht einfache PLD-Beschreibungssprachen, die häufig als lizenzfreie Programme im PC-Bereich verfügbar sind oder aber mit den entsprechenden Programmiergeräten ausgeliefert werden. Bekannte Programme sind :

- PALASM (Der „PAL-Assembler“, entwickelt bzw. gepflegt von den Halbleiterfirmen MMI und AMD); Der begrenzte Sprachumfang vom PALASM wurde für die klassischen PAL-/GAL-Typen entwickelt, es gibt daher nur wenig Erweiterungsmöglichkeiten im Hinblick auf leistungsfähigere PLDs.
- ABEL
- CUPL

Das Programm CUPL ist im PC-Bereich schon seit längerer Zeit verfügbar; zunächst als reine DOS-Version und später unter dem Namen WinCUPL auch unter den aktuellen Windows Versionen 9x sowie /NT, 2000 und XP. Es ist sehr flexibel und erlaubt im Gegensatz zum bekannten PAL-Assembler PALASM die Programmierung unterschiedlicher PLD-Strukturen. Mit diesem „Compiler für Programmierbare Logik“ ist es möglich, die nachfolgenden PLD-Typen zu programmieren:

- PAL (Programmable Array Logic)
- GAL (Generic Array Logic)
- FPGA (Field Programmable Array Logic)
- PROM (Programmable Read Only Memory)

Zu dem eigentlichen PLD-Compiler CUPL gehört auch ein Simulationsprogramm CSIM, mit dessen Hilfe des logische Verhalten des PLD-Entwurfs durch Simulation überprüft werden kann. Hinzu kommt ein weiteres kleines Programm PTOC zur Umsetzung vorhandener PALASM-Entwurfsdateiein in das logische und technische CUPL-Format.

Die Eigenschaften der betreffenden zu programmierenden Bauelemente sind in einer Bibliothek gespeichert (Datei CUPL.DL), welche jeweils von den Bauelementeherstellern aktualisiert wird. Die Bibliothek beschreibt die physischen Eigenschaften jedes Bauelements, d.h. seinen inneren Aufbau, die Anzahl der Anschlüsse und gültige Ein- bzw. Ausgänge. Wie auch bei jeder anderen Programmiersprache ist es erforderlich, eine Quellendatei zu erstellen, welche die logische Beschreibung des Entwurfs enthält. Diese Datei trägt die Bezeichnung <dateiname>.PLD. Es handelt sich hierbei um eine reine ASCII-Datei, welche mit jedem entsprechenden Texteditor erstellt werden kann.

Bei fehlerfreiem Übersetzungslauf einer .PLD-Datei erstellt der Compiler optional mehrere Ausgabedateien zur Steuerung von PLD-Programmiergeräten. Die wichtigste ist die JEDEC-Datei. Diese besitzt einen nach JEDEC genormten Dateiaufbau (JEDEC-File), der von allen bekannten Herstellern universeller Programmiergeräte unterstützt wird. Es handelt sich bei den JEDEC-Dateien ebenfalls um normale ASCII-Dateien, die auch mit jedem Texteditor erstellt bzw. nachbearbeitet werden könnten. Bei der Realisierung von PLD-Entwürfen in PROM-Technik werden häufig auch Dateien im sog. Hex-Format verwendet (.HEX-Datei).

Datenfluss in CUPL

Der logische Zusammenhang zwischen dem Programm CUPL, dem zum Programmpaket zugehörigen Simulationsprogramm CSIM und den Ein- sowie Ausgabedateien ist in Bild 5.13 dargestellt.

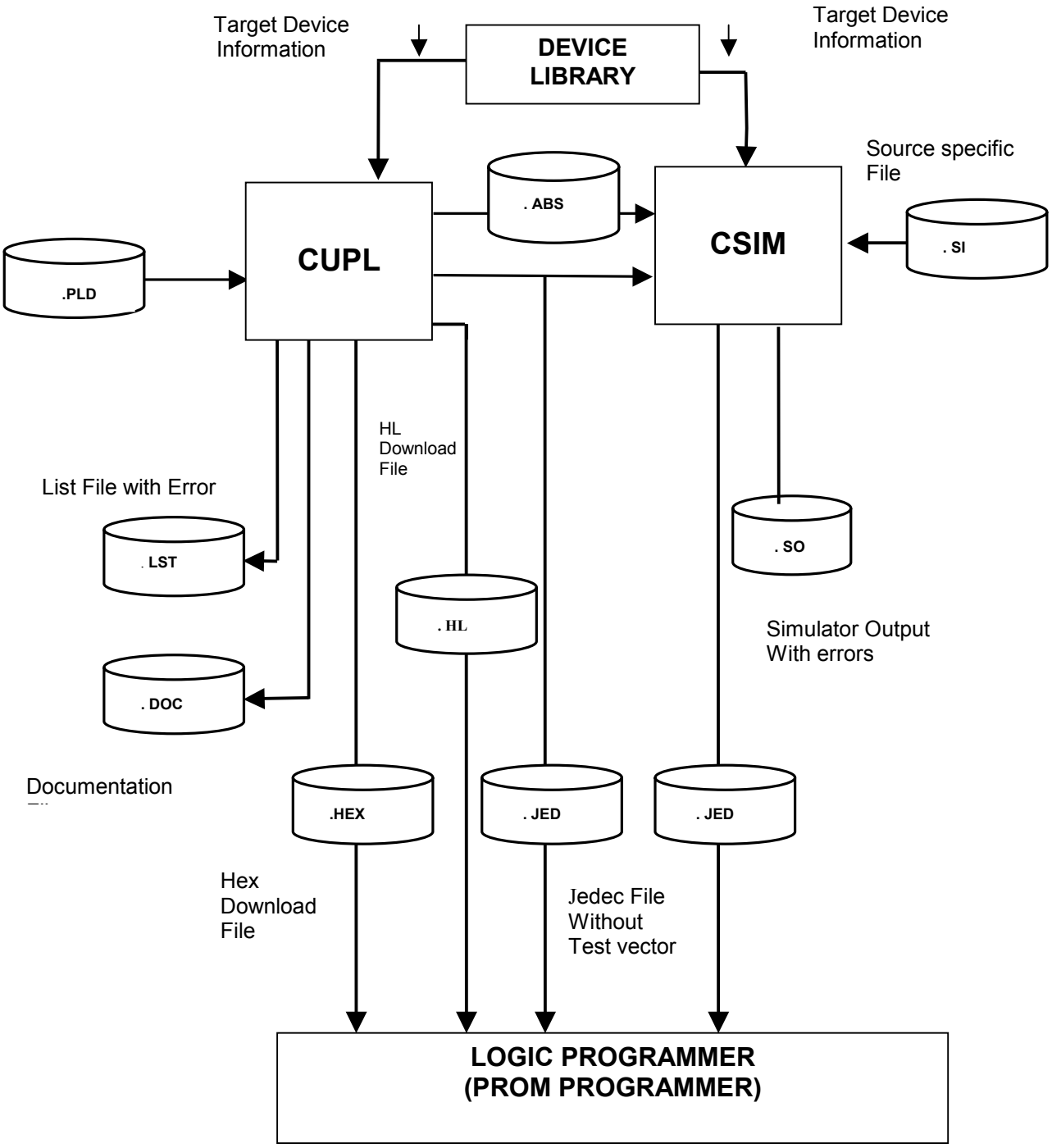


Bild 5.13: Datenfluß bei CUPL und CSIM

Syntax der CUPL-Anweisungen

Bei dem Progrmm CUPL kann die Beschreibung eines Schaltungsentwurfs als Kombination von drei Verfahren erfolgen:

- State Machines (Formalisiertes Zustandsdiagramm)
- Tabellen (Funktionstabellen)
- Gleichungen (formalisierte Boole'sche Gleichungen)

In den meisten Fällen wird man die Beschreibung mit Hilfe von Gleichungen bevorzugen, wobei deren Aufbau, sowie Syntax und Semantik zu beachten sind. Eine Gleichung besteht aus Variablen und Operatoren, wobei optionale Kommentare zur Beschreibung recht freizügig benutzt werden können.

Verwendete Operatoren in hierarchischer Reihenfolge

;	- Ende einer Zeile
!	- Negierung
&	- AND
#	- OR
$	- XOR
=	- Zuweisung (taktunabhängig)

Kommentare werden wie in C üblich, dargestellt:

```
/*  Dies ist ein Kommentar  */
```

Die folgenden Transferfunktionen zur Zuweisung von Eingängen zu Ausgängen können verschiedene Formen haben:

```
ident                = Ausdruck (normale Gatter)
IF (Produkt) ident     Ausdruck (Tri-State Bedingung)
APPEND ident.typ  =  Ausdruck (Flipflops)
```

Beispiele:

```
OUT             = INA # INB   ;     /* einfache ODER Funktion   */
IF INA & INB    OUTX          ;     /* IF_Abfrage               */
APPEND Y.d   = !Q1 & Q0 & !INA ;    /* D_type Flip-Flop         */
```

Für die Erweiterung gemäß 'ident.typ' gibt es je nach PLD-Architektur und -Typ verschiedene Erweiterungstypen 'typ':

D	D-Eingang eines D-Flipflops
L	D-Eingang eines transparenten Latches
J	J-Eingang eines JK-Flipflops
K	K-Eingang eines JK-Flipflops
S	S-Eingang eines RS-Flipflops
R	R-Eingang eines RS-Flipflops
T	T-Eingang eines T-Flipflops (Toggle-FF)
DQ	D-Ausgang eines D-Flipflops
LQ	Q-Ausgang eines transparenten Latches
AP	Asynchroner Setzeingang eines Flipflops (Asynchronous Preset)
AR	Asynchroner Rücksetzeingang eines Flipflops (Asynchronous Reset)
SP	Synchroner Setzeingang eines Flipflops (Synchronous Preset)
SR	Synchroner Rücksetzeingang eines Flipflops (Synchronous Reset)
CK	Programmierbarer Takt eines Flipflops (Clock)
OE	Programmierbare Freigabe eines Ausgangs (Output Enable)
CA	Complement Array (Bei PLDs vom Typ IFL)

NODE-Deklaration
Mit dem Schlüsselwort NODE lassen sich für bestimmte PLD-Architekturen "Knotenwerte" deklarieren:

```
NODE variable_name            /* Einfacher Knoten */
NODE [variable_liste]         /* Vielfache Knoten */
```

Distributives Gesetz

CUPL unterstützt die Umformung Boole'scher Ausdrücke gemäß dem Distributiven Gesetz:

```
A & (B # C) = A&B # A&C
```

Der vom Programm gebildete Ausdruck ist eindeutig, da der UND-Operators '&' gegenüber ODER '#' eine höhere Priorität aufweist.

DE MORGAN-Theorem

Auch das häufig benutzte De Morgan Theorem der Boole'schen Algebra wird von CUPL beherrscht:

```
! (A # B) = !A & !B
! (A & B) = !A # !B
```

Makro Ersetzung

CUPL erlaubt die Festlegung symbolischer Namen, die nicht dem jeweiligen Ein- oder Ausgangsanschluß eines PLD zugeordnet sein müssen. Dieser Name kann dann in anderen Ausdrücken benutzt werden, wobei CUPL jeweils die ursprüngliche Festlegung als Makro ersetzt.

Beispiel:

```
MEMMREQ = MEMW # MEMR; /* Der Ausdruck MEMW # MEMR wird immer  */
                       /* dann eingesetzt, wenn MEMREQ in   */
                       /* einem Ausdruck verwendet wird     */
```

Die Listen-Kurzform (List Notation)

CUPL erlaubt es, eine Gruppe von Variablen in einer Kurzform anzugeben:

[A,B,C,D]	wie z.B.	[MEMR, MEMW, IOR, IOW] oder
[X,N..Z]	wie z.B.	[ADR10..3] was folgenden Ausdruck ersetzen würde
		[ADR10, ADR9, ADR8, ADR7, ADR6, ADR5, ADR4, ADR3]

Bitfelder (Bit fields)

Ähnlich wie in der Programmiersprache C vorgesehen, bietet CUPL die Möglichkeit, eine Gruppe zusammengehöriger Bits unter einem symbolischen Namen als als Bitfeld zu verwalten. Dies geht beispielsweise folgendermaßen:

FIELD EAADR = [A10..3]

Hierbei kann EAADR in Ausdrücken anstelle von [A10..3] verwendet werden.

Der Vergleichsoperator ':'

Der Operator ':' erlaubt bei CUPL den Vergleich eines Bitfeldes mit einer hexadezimalen Konstanten oder aber einer Liste konstanter Werte.

Beispiele: EAADR : B5

EAADR: [10..3F] /* Wahr für den Adreßbereich zwischen 0x10 bis 0x3F */

Der Vergleichsoperator kann auch verwendet werden, um ein Bitfeld als Ausdruck verwenden zu können.

Beispiele:

EAADR: & /* Ersetzt A10 & A9 & A8 & A7 & A6 & A5 & A4 & A3 */
EAADR: # /* Ersetzt A10 # A9 # A8 # A7 # A6 # A5 # A4 # A3 */

Programmierung von GAL mit CUPL

Im Gegensatz zum PAL-Assembler lassen sich mit CUPL auch GAL und andere einfache PLD-Strukturen problemlos programmieren. CUPL erzeugt im Falle einer GAL-Programmierung das benötigte und für GAL charakteristische Architekturwort, welches in der Ausgabedatei (JEDEC-Datei) an das PLA (identisch zu PALs) angehängt wird. Dieses Architekturwort befindet sich in der JEDEC-Datei beim GAL 16V8 an den Adressen 2048- 2055 und 2120-2193, beim GAL 20V8 an den Adressen 2560-2567 und 2632-2705.

Adreßzuordnung für GALs in der JEDEC-Datei:

16V8:

Bit-Adresse	Funktion
0000-2047	Produktterme
2048-2055	Bit EXOR der Ausgangszellen
2056-2119	Freie Bits für eigene Anwendungen
2120-2127	Bit AC1 der Ausgangszellen
2128-2191	Freigabe der Ausgangszellen
2192	Bit SYN der Ausgangszellen
2193	Bit AC0 der Ausgangszellen

20V8:

Bit-Adresse	Funktion
0000-2559	Produktterme
2560-2567	Bit XOR der Ausgangszeile
2568-2631	Freie Bits für eigene Anwendungen
2632-2639	Bit AC1 der Ausgangszeilen
2640-2703	Freigabe der 64 Produktterme
2704	Bit SYN der Ausgangszeilen
2705	Bit AC0 der Ausgangszeilen

JEDEC-Format

Als Schnittstelle zum Programmiergerät dient die JEDEC-Datei (**J**oint **E**lectronic **D**evice **E**ngineering **C**ouncil, dies ist ein am. Gremium zur Schaffung von Industrienormen bei elektronischen Bauelementen). Diese wird mit Hilfe des Compilers erzeugt, kann jedoch auch durch selbständige Entwicklungssoftware erstellt werden. Die JEDEC-Datei ist aus ASCII-Zeichen aufgebaut und muß folgenden Vereinbarungen entsprechen.

Kurzübersicht der Syntax, die verarbeitet wird:

(n/a)	Design Spezifikation
N	Kommentar
QF	Anzahl der Fuses in den Zuweisungen
QP	Anzahl der Pins in den Testvektoren
QV	Maximale Anzahl der Testvektoren (kleiner 150)
F	Standard "Fuse-Zustand"
L	Adresse, Digit.Zeichen "Fuse Liste"
C	Hexsumme "Fuse Prüfsumme"
X	Testvektoren (siehe Abschnitt Testvektoren)
P	Pinsequenz
G	"Security Fuse"
R,S,T	Signatureanalyse

Im allgemeinen beginnt jede Zeile mit einer Identifikation, gefolgt von einem "Informationsfeld" und wird mit einem "*" abgeschlossen. Jeder "Fuse" des Bauelementes ist eine dezimale Nummer zugewiesen, mit den beiden möglichen Zuständen:

- 0 (X): logische Verbindung zwischen zwei Punkten, entspricht vorhandener Sicherung bzw. markiertem Kreuzungspunkt im Array;
- 1: keine logische Verbindung, Sicherung durchtrennt.

Die "Fuse"-Werte beginnen bei Null und sind fortlaufend bis zur maximalen "Fuse"-Nummer durchnummeriert. Ein Baustein mit 2048 "Fuses" besitzt "Fuse"-Nummern von 0 bis 2047.

Das JEDEC-Format überträgt nur alle Zeilen, in denen mindestens eine Sicherung zu durchtrennen ist. Dabei steht für jede zu durchtrennende Sicherung eine '1' in ASCII-Darstellung (=31h), sonst '0' (=30h). Die jeweilige Produktzeile wird dadurch adressiert, indem die Verbindungen in der Matrix von Null an durchnummeriert werden und diese Anzahl an den Beginn einer Zeile geschrieben werden und mit L markiert werden. Bei einem 20-poligen PAL hätte die erste Zeile als Kennung demnach *L0* die zweite *L32* und so weiter.

Am Beginn des JEDEC-Formats steht ein "STX" als Startkennung, gefolgt von *F0*, bzw. *F1*. Den Abschluß bildet auch hier wieder eine Prüfsumme. Man kann mit einem Texteditor auch einzelne Sicherungen nachbrennen, wobei jeder Wechsel von 1 nach 0 die Prüfsumme um eins vermindert und umgekehrt natürlich erhöht.

Testvektoren

Testinformationen werden durch Testvektoren angegeben, indem sie Testinformationen für den ausgewählten Baustein beinhalten. Im folgenden ist eine kurze Übersicht über die möglichen Testbedingungen angegeben:

0	- Eingang Low
2-9	- Eingang auf Super Voltage 2-9, oder VCC
C	- Eingang "Low","High","Low"
F	- float Eingang oder Output
H	- Ausgangstest auf "High"
K	- Eingang "High","Low","High"
L	- Ausgangstest auf "Low"
N	- Power pins und Ausgänge werden nicht getestet
	- Vorbesetzen des Registers

X - Ausgang nicht getestet
Z - Teste Eingang oder Ausgang auf Hochohmigkeit
B - buried register preload

Syntaxerklärung

Das X-Feld definiert den nicht explizit definierten logischen Eingangspegel für Testvektoren und "don't care"-Testbedingungen. Das X-Feld setzt Testvektoren von 1 bis zum Maximum (gesetzt bei QV) auf die Standard-Eingang-Testbedingungen. Wird das X-Feld benutzt, so muß es nach dem QF- und QP-Feld und vor dem ersten Testvektor spezifiziert sein.

Jeder Testvektor enthält m Testbedingungen, wobei m die Zahl der Anschlußkontakte des Bausteins ist. In der Tabelle für die Testbedingungen sind die verschiedenen Spezifikationen aufgeführt.

Das V-Feld beginnt mit einer dezimalen Vektornummer, gefolgt von einem Leerzeichen. Dem Leerzeichen folgt eine Serie von Testbedingungen. Abgeschlossen wird der Vektor durch ein "*". Die Vektorzahl kann vorangestellte Nullen besitzen.

Beispiel:
```
V0001 000000XXXNXXXHHHLXXN*
V0002 010000XXXNXXXHHHLXXN*
V0003 100000XXXNXXXHHHLXXN*
```

Pinsequenz

Die Bedingungen, die in den Testvektoren enthalten sind, werden beim Baustein in numerischer Reihenfolge von links nach rechts verwendet. Die am weitesten links stehende Bedingung wird für Pin 1 und die am weitesten rechts stehende für Pin 20 bei einem 20- poligen Baustein verwendet. Die zeitliche Sequenz ist festgelegt, d.h. daß Pin 1 vor Pin 2 usw. gesetzt wird.

Ausgänge (Outputs) werden generell erst nach allen Eingangs- (Input-) Setzvorgängen getestet.

Beispiel:

Pin	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
V0001	1	1	1	0	0	0	H	L	H	N	N	N	N	N	N	N	N	N	N	N*
V0002	1	0	0	0	0	0	H	H	H	N	N	N	N	N	N	N	N	N	N	N*

Vektor 1 verwendet 111000 von Pin 1 bis Pin 6 und HLH von Pin 7 bis Pin 9. Pin 10 bis 20 werden nicht getestet.

Testbedingungen

Der logische Testpegel ist durch die Bausteintechnologie festgelegt (z.B. TTL-Pegel).

1 legt den entsprechenden Eingangsanschluß auf H-Pegel
0 legt den entspr. Eingangsanschluß auf L-Pegel
X oder "don't care" Testbedingung läßt den Pegel unverändert, bzw. der Ausgang wird nicht getestet.
F Testbedingung verwendet "High Impedance" beim Bausteinpin.
2..9 Die Testbedingungen von 2 bis 9 verwenden eine "non standard" oder "super voltage" beim Baustein. Dies wird bei speziellen Testformen verwendet.
C gibt einen positiven Clock-Puls mit 0-1-0 Folge aus.

- **K** wechselt von 1-0-1 in gleicher Weise.
- **N** Anschluß wird nicht getestet.
- **L** testet einen logischen 0-Pegel.
- **H** testet einen logischen 1-Pegel.
- **Z** testet, ob der Ausgang hochohmig ist.
- **P** Vorbesetzen von Ausgangsregistern.
- **B** Vorbesetzen von "versteckten" Registern.

Security-Fuse

Entspricht einem Leseschutz; Wenn ein PAL bzw. GAL mit Security-Fuse ON (=G1) programmiert wird, läßt es sich nicht mehr auslesen. Auch ein Verifiziervorgang ist dann nicht mehr möglich.

Berücksichtigung von negierten Ausgängen (aktiv-L Ausgänge)

Ein Problem bei der praktischen Realisierung von Schaltungsentwürfen mit PAL/GAL ist die richtige Behandlung der ausgangsseitigen Polarität. Solange nur einfache H-PALs (z.B. 10H8) mit H- aktiven Ausgängen eingesetzt bzw. durch GAL nachgebildet werden, ist der Sachverhalt noch sehr einfach. Die Schaltgleichung wird in gewohnter Weise mit Hilfe der Boole'schen Algebra aufgestellt und vereinfacht. Anschließend erfolgt die syntaktisch richtige "Übersetzung" in die vom jeweiligen Programm verlangte Form der logischen Beschreibung. Leistungsfähigere Programme wie CUPL beherrschen einige Gesetzmäßigkeiten wie Distributivgesetz und De-Morgan-Theorem. Das Programm formt daher die Schaltgleichung entsprechend um und bildet die Terme ab auf die jeweilige PLD-Struktur. Dies ist besonders zu beachten, wenn die vorherrschenden L-aktiven PAL-Typen nachgebildet werden. Hierbei ist zu berücksichtigen, daß die Schaltungsstruktur aufgrund der ausgangsseitigen Negation jetzt einer konjunktiven Normalform KNF entspricht, die physische PAL-/GAL- Struktur indes einer DNF entspricht. Eine bloße Umformung nach De Morgan würde mehr Terme erzeugen, als im Bauelement physisch vorhanden sind. Das Ausgangssignal ist daher in derartigen Fällen in negierter Form zu verwenden. Dieses Prinzip wird auch beim Entwurf von Zählern in PAL-/GAL-Technik benutzt, wobei man als „goldene Regel“ die 0-Felder (Maxterme) im KV-Diagramm zusammenfasst.

Die Programmierung eines PLD-Entwurfs wird durch Bereitstellung einer leeren Musterdatei (Template file) TMPL.PLD wesentlich erleichtert. Diese Datei braucht nur kopiert, umbenannt und mit den Daten des eigenen Projektes ausgefüllt zu werden. Nachfolgend ein Beispiel für den erfolgreichen PLD-Entwurf einer einfachen Schaltung.

Programmliste einer CUPL-Quellendatei für GAL

```
/*                Kopf fuer Dokumentationszwecke                          */
Name              V9a    ;      /*              Dateiname                  */
Partno            IC1    ;      /*      z.B. IC-Bezeichnung im Schaltbild  */
Date          20.02.01   ;
Rev                  1   ;      /*      Entwicklungsstand des Projektes    */
Designer   Gruppe X     ;
Company FH Friedberg, Digitallabor ;
Assembly  Versuch 12    ;
Location                ;      /* Bei grossen Projekten : z.B. IC-Nummer  */

/*****************************************************************************/
/*                                                                           */
/*                        Beschreibung des Projektes                         */
/*                                                                           */
/*****************************************************************************/
/*                                                                           */
/*****************************************************************************/
/*                Allowable Target Device Taypes : PAL16R8,GAL16v8           */
/*****************************************************************************/

/**                             Inputs                                      **/
/*      Hier steht die deklaration fuer alle verwendeten Eingangspins        */

pin1 = takt     ;      /*      Auf Gross-/Kleinschreibung achten   !!!      */
pin2 = A        ;      /*      Eingang ist aktiv Low, entspricht            */
pin3 = !B       ;      /*      einem Inverter am Eingang                    */


pin1 = DEF      ;
pin5 = H1       ;

/**                             Ouputs                                      **/
/*      Hier steht die deklaration fuer alle verwendeten Ausgangspins        */

pin18 = X       ;
pin17 = !Y;     /* Ausgang ist aktiv low, entspricht                        */
                /* einem Inverter am Ausgang                                */
pin16 = Q1      ;
pin15 = Z1      ;
pin14 = Z2      ;

/*      Deklaration and Intermediate Variable Definitions                    */

E = !A # !B     ;      /* Zwischenvariable belegt keinen Ausgang            */

/**                             Logic equations                             **/
Q1.D    =    A & B  ;          /*      Flipflop-Ausgang                     */
X       =    DEF & H1 & !A  ;  /*      normaler Ausgang UND                 */
Y       =    A # B # !H1    ;  /*      ODER                                 */
Z1      =    A # !(B & H1)  ;  /*      beliebige Klammer                    */
Z2      =    !E & Z1        ;  /*Zwischenvariable und interne Rueckfuehrung */
```

Beispiel für JEDEC-Datei: Decodierer Dual in 3-Exzess-Code und Dual in Aiken-Code

In diesem Beispiel soll die JEDEC-Datei für zwei in einem GAL programmierte 4 Bit-Decodierer gezeigt werden:

Die durch CUPL erzeugte JEDEC-Datei besitzt den nachfolgenden Aufbau:

```
*F0*
L      0   11111111111111111111111111111111*
L     32   11111011011111111111111111111111*
L     64   10111011111111111111111111111111*
L     96   01110111101111111111111111111111*
L    128   01110111111110111111111111111111*
L    160   10111111101110111111111111111111*
L    256   11111111111111111111111111111111*
L    288   10110111011111111111111111111111*
L    320   11110111011101111111111111111111*
L    352   10110111111101111111111111111111*
L    384   01111011111110111111111111111111*
L    416   11111011101110111111111111111111*
L    512   11111111111111111111111111111111*
L    544   10111111101101111111111111111111*
L    576   10111111011101111111111111111111*
L    608   01111111011101111111111111111111*
L    640   01111011111101111111111111111111*
L    672   01110111101110111111111111111111*
L    768   11111111111111111111111111111111*
L    800   11111111111011111111111111111111*
L   1024   11111111111111111111111111111111*
L   1056   11111111111111111111011011111111*
L   1088   11111111111111111011101111111111*
L   1120   11111111111111110111011110111111*
L   1152   11111111111111110111011111111011*
L   1184   11111111111111110111111110111011*
L   1280   11111111111111111111111111111111*
L   1312   11111111111111110111011111111111*
L   1344   11111111111111111111011011111111*
L   1376   11111111111111110111011111110111*
L   1408   11111111111111111111011110110111*
L   1440   11111111111111110111011110111111*
L   1536   11111111111111111111111111111111*
L   1568   11111111111111110111011110111111*
L   1600   11111111111111110111011011111111*
L   1632   11111111111111110110111011111111*
L   1664   11111111111111110110111111111011*
L   1696   11111111111111111111011101111011*
L   1792   11111111111111111111111111111111*
L   1824   11111111111111111111111111111011*

33ED*     (Prüfsumme!)
```

Die programmierte Sicherungsmatrix sieht beispielsweise folgendermaßen aus:

```
00  ----  ----  ----  ----  ----  ----  ----  ----
01  ----  -X--  X---  ----  ----  ----  ----  ----
02  -X--  -X--  ----  ----  ----  ----  ----  ----
03  X---  X---  -X--  ----  ----  ----  ----  ----
04  X---  X---  ----  -X--  ----  ----  ----  ----
05  -X--  ----  -X--  -X--  ----  ----  ----  ----

08  ----  ----  ----  ----  ----  ----  ----  ----
09  -X--  X---  X---  ----  ----  ----  ----  ----
10  ----  X---  X---  ----  ----  ----  ----  ----
11  -X--  X---  ----  X---  ----  ----  ----  ----
12  X---  -X--  ----  -X--  ----  ----  ----  ----
13  ----  -X--  -X--  -X--  ----  ----  ----  ----

16  ----  ----  ----  ----  ----  ----  ----  ----
17  -X--  ----  -X--  X---  ----  ----  ----  ----
18  -X--  ----  X---  -X--  ----  ----  ----  ----
19  X---  ----  X---  X---  ----  ----  ----  ----
20  X---  -X--  ----  X---  ----  ----  ----  ----
21  X---  X---  -X--  -X--  ----  ----  ----  ----

24  ----  ----  ----  ----  ----  ----  ----  ----
25  ----  ----  ----  X---  ----  ----  ----  ----

32  ----  ----  ----  ----  ----  ----  ----  ----
33  ----  ----  ----  ----  ----  -X--  X---  ----
34  ----  ----  ----  ----  -X--  -X--  ----  ----
35  ----  ----  ----  ----  X---  X---  -X--  ----
36  ----  ----  ----  ----  X---  X---  ----  -X--
37  ----  ----  ----  ----  -X--  ----  -X--  -X--
40  ----  ----  ----  ----  ----  ----  ----  ----
41  ----  ----  ----  ----  -X--  -X--  ----  ----
42  ----  ----  ----  ----  ----  -X--  X---  ----
43  ----  ----  ----  ----  X---  X---  ----  X---
44  ----  ----  ----  ----  ----  X---  -X--  X---
45  ----  ----  ----  ----  X---  X---  -X--  ----

48  ----  ----  ----  ----  ----  ----  ----  ----
49  ----  ----  ----  ----  -X--  -X--  -X--  ----
50  ----  ----  ----  ----  X---  -X--  X---  ----
51  ----  ----  ----  ----  -X--  X---  X---  ----
52  ----  ----  ----  ----  -X--  X---  ----  -X--
53  ----  ----  ----  ----  ----  X---  X---  -X--

56  ----  ----  ----  ----  ----  ----  ----  ----
57  ----  ----  ----  ----  ----  ----  ----  -X—
```

5.6 Lerntest/Wissensfragen

1. Wie lassen sich Digitalschaltungen unterscheiden ?
2. Was versteht man unter PLD ?
3. Welche Unterschiede gibt es zwischen PAL und GAL ?

4. Welche Aufgabe besitzt die "security fuse" bei PAL bzw. GAL ?
5. Was ist eine JEDEC-Datei und welche Informationen sind in dieser enthalten ?
6. Was versteht man unter einem „Silicon Compiler“ ?
7. Welche Hard- und Softwaretools werden zum Entwurf und zur Realisierung von Schaltungen in PLD-Technik benötigt ?
8. Was sind Testvektoren und wo werden diese eingesetzt ?
9. Was ist ein ASIC ? Welche Geräte und Programme sind zum Entwurf von ASICs notwendig?