# PLD-Einsteiger-Set Teil 2

***Mit der PLD-Entwicklungssoftware ist eine übersichtliche Eingabe und Darstellung der Logik-Funktionen möglich, zur Realisierung von Logik-Bausteinen nach eigenen Vorgaben.***

## PLD-Entwicklungssoftware

Für den Anwender von programmierbarer Logik ist eine effiziente und leicht zu handhabende Software von großer Bedeutung. Am Markt gibt es heute eine Vielzahl von Softwarelösungen zur Entwicklung von programmierbaren Logikbausteinen. Das Spektrum reicht dabei vom einfachen PLD-Compiler bis hinauf zu komplexen Softwaresystemen mit Preisen von über 100.000,- DM.

Die Anbieter dieser Software lassen sich in 2 Gruppen einteilen: Zum einen treten die Halbleiter-Hersteller mit speziell angepaßten Entwicklungs-Systemen auf, und zum anderen bieten Softwarehäuser mit herstellerunabhängigen Lösungen ihre Produkte an.

Die vom Hersteller angebotene Software hat meist den Vorteil, daß sie optimal auf die jeweilige Logikarchitektur abgestimmt ist, dafür aber oftmals nur die Bausteine eines Herstellers unterstützt. Herstellerunabhängige Tools erlauben Designs für eine Vielzahl von Logikfamilien, sind aber meistens teurer als die vom Logikhersteller gelieferten Systeme. Die Entscheidung für ein bestimmtes System hängt letztendlich von den individuellen Gegebenheiten ab, wobei wesentliche Faktoren sicherlich die Leistungsfähigkeit/Komplexität und der Preis sind.

## Entwicklungsablauf

Der Entwicklungsablauf beim Entwurf komplexer Logik verläuft in den nachfolgend aufgeführten Schritten:

1. Logikeingabe
2. Verifikation und Minimierung
3. Place and Route
4. Simulation (optional)
5. Programmierung

## Logikeingabe

Der erste Schritt eines Logikdesigns ist die Logikeingabe. Es gibt verschiedene Verfahren wie z.B.:

- Boolesche Gleichungen
- Wahrheitstabellen
- Beschreibung als sog. State-Machine
- Schematische Schaltplaneingabe
- Beschreibung in Hochsprachen wie z. B. VHDL
- Beschreibung mit Makros

## Verifikation und Minimierung

Verifikation bedeutet Überprüfung, daher wird in dieser Stufe das Design nach logischen Kriterien überprüft, wie z.B. nach Syntaxfehlern, nicht angeschlossenen oder mehrfach definierten Signalen etc.

Ist das Design formal fehlerfrei, beginnt die Minimierung, d.h. die logischen Gleichungen werden mit unterschiedlichen Verfahren verkleinert, um z.B. die Anzahl der Produktterme möglichst gering zu halten. Produktterme sind in den Logikbausteinen nur in begrenzter Zahl vorhanden, deshalb ist es wichtig, an dieser Stelle eine dem Bausteintyp angepaßte Minimierung durchzuführen, um einen optimalen Ausnutzungsgrad des Bauelementes zu erreichen.

## Place and Route

Unter Place and Route wird eine Anpassung des Logikentwurfs an die physikalische Struktur des Bauelementes verstanden. Die Logik wird in die vorhandenen Blöcke des Logikbausteins plaziert (Place), um diese anschließend untereinander zu verbinden (Routing).

## Simulation

Die Simulation ermöglicht eine Überprüfung des Designs. Es werden 2 Arten der Simulationen unterschieden:

- Logische Simulation
- Timing-Simulation

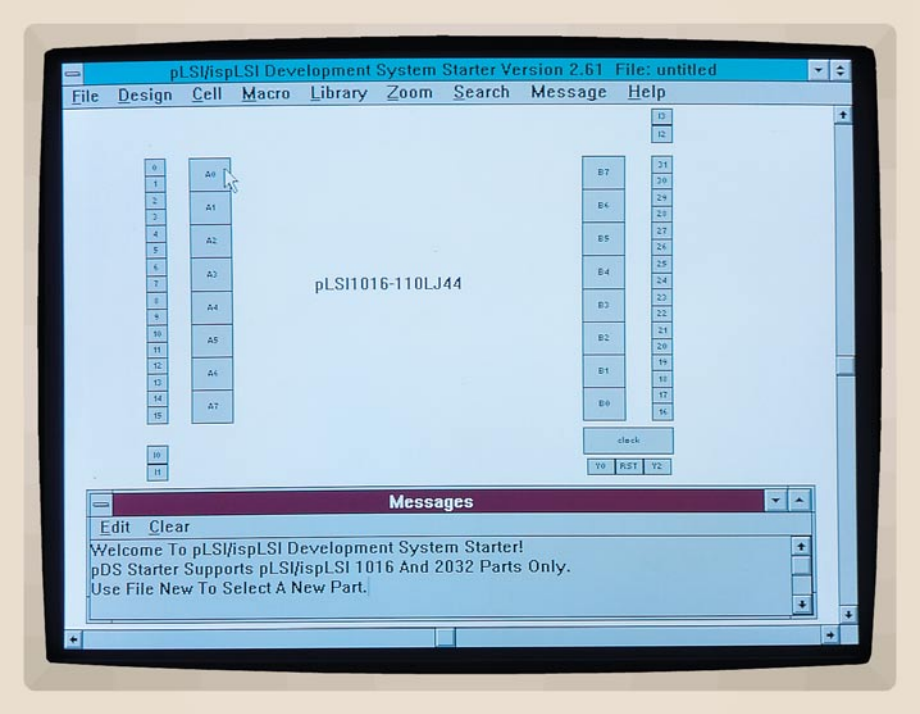
Bei der logischen Simulation wird die Laufzeit des Logikbausteins nicht berücksichtigt, sie überprüft daher nur den logischen Entwurf.

Die Timing-Simulation benutzt zusätzlich die physikalischen Daten des Logikbausteins (Gatterlaufzeiten etc.), und sie entspricht daher eher der Realität.

Die Realisierung einer Simulation ist sehr zeitaufwendig, denn für einen vollständigen Test sind eine Vielzahl von sogenannten Testpatterns erforderlich, die vom Anwender zu erstellen sind. Unabdingbar ist die Simulation beim Entwurf von Gate-Arrays und nicht wiederprogrammierbaren FPGAs.

Dagegen sind bei in-system-programmierbaren-PLDs, wie z.B. der ispLSI-Serie von Lattice, zum einen die Übersetzungszeiten der Software sehr kurz, und zum anderen wird das Bauelement von der Software in wenigen Sekunden direkt im System programmiert. Da die ispLSIs mindestens 1000mal programmierbar sind, können der Test und die Inbetriebnahme interaktiv, d.h. Schritt für Schritt vorgenommen werden.

**Bild 6: ISP-Sk2-Startmeldung**



## Programmierung

Bisher mußte ein Programmiergerät eingesetzt werden, um die Logikgleichungen in den Logik-Baustein zu programmieren. Wie bereits erwähnt, ist die ispLSI-Serie im System programmierbar. Ein Programmiergerät kann entfallen, da die Software das Bauelement direkt im Zielsystem programmieren kann.

Um einen möglichst übersichtlichen und praxisnahen Einblick in die Thematik der PLDs zu geben, wollen wir uns nachfolgend mit der anwenderfreundlichen Entwicklungssoftware der Firma Lattice näher befassen, die speziell für die ispLSI-Serie ausgelegt ist.

## Entwicklungssoftware für die ispLSI-Serie

Die ispLSI-Serie von Lattice wird von einer großen Zahl unterschiedlicher Softwarelösungen unterstützt. Neben der eigenständigen Software „PDS" bietet Lattice eine Reihe von sog. Fittern („Fitten" bedeutet passend machen) für Fremdsoftware an. Die Fitter übernehmen dann über Normschnittstellen den minimierten Logikentwurf und setzen diesen dann in die ispLSI-Struktur um (Place and Route).

Für folgende Softwarelösungen werden von Lattice passende Fitter angeboten: ABEL, Synario, LOG/IC, Viewlogic, CUPL, ORCAD, Mentor, Synopsis, Cadence und weitere.

Als eigenständiges Programm wird von Lattice die Software pDS1101 angeboten, die speziell für die ispLSI-Serie entwickelt wurde. Diese Software ist auch als Starter-Version mit der Bezeichnung isp-SK2 erhältlich, die jedoch nur 2 Typen, und zwar ispLSI1016 und ispLSI2032, unterstützt.

Nachfolgend wollen wir eine Übersicht über die Lattice-Software geben, die eine einfache und übersichtliche Bedienung erlaubt. Eine ausführliche Beschreibung kann dem dem Starter-Kit beiliegenden Handbuch entnommen werden.

## Systemvoraussetzungen

Die pDS1101 / isp-Sk2 benötigt folgende Rechnerausstattung:

- IBM-kompatibler PC mit mind. 80368-CPU
- MS-Windows 3.1 oder höher
- Min. 4 MB RAM, besser 8 MB (pDS1101)
- Min. 5 MB freie Festplattenkapazität
- Parallelport (LPT) zur Programmierung

## Benutzeroberfläche

Da die Software eine Standard-Windows-Anwendung ist, findet sich jeder erfahrene PC-Benutzer schnell zurecht. Nach dem Start der Software erscheint ein Bildaufbau wie in Abbildung 6 dargestellt.

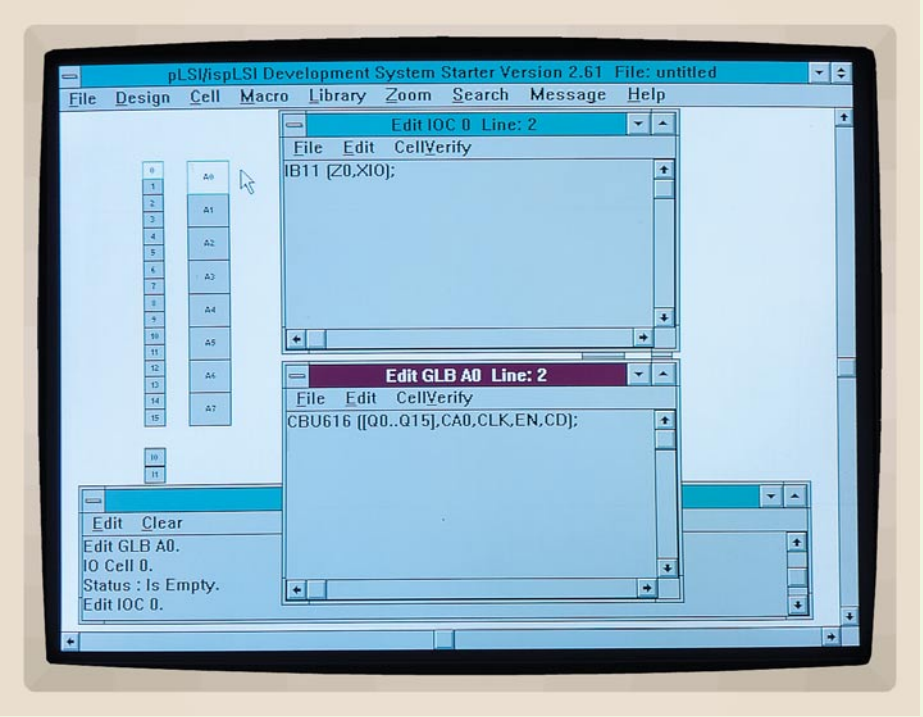



**Bild 7: Geöffnete I/O- und GLB-Zelle**

Im oberen Teil des Bildes werden einige Blöcke (Zellen) gezeigt, die der Struktur des Bauelementes entsprechen:

Die GLB-Zellen sind mit A 0 bis B 7 bezeichnet. Die IO-Zellen sind von 0 bis 31 numeriert, die dedizierten Input-Zellen haben die Bezeichnung I 0 bis I 4. 2 Takteingänge stehen unter der Bezeichnung Y 0 und Y 2 zur Verfügung. Ein weiterer Eingang hat eine Doppelfunktion: Als Reset trägt er die Bezeichnung RST, als weiterer Takteingang die Bezeichnung Y1, wobei die Zuordnung in der Software festgelegt wird.

## Eingabe eines Designs

Eine Designeingabe wird durch Einträge in die obigen Zellen realisiert, welche als Editierfenster auf dem Bildschirm geöffnet werden. Abbildung 7 zeigt eine geöffnete IO-Zelle (IOC 0) und eine geöffnete GLB-Zelle (GLB A0).

Die GLB-Zellen nehmen die Logikgleichungen auf. Hier können entweder Boolesche Gleichungen oder vordefinierte Makros eingetragen werden. In die IO-Zellen werden ausschließlich spezielle IO-Makros eingetragen.

Die Software kennt über 275 Makros, die in dem zum PLD-Einsteigerset gehörenden Handbuch beschrieben sind. Einige Beispiele: Zähler, Decoder, Multiplexer, Addierer als Funktionsblöcke für GLBs sowie IO-Makros zur Steuerung der IO-Zelle wie z.B. Eingang, Ausgang oder Tri-State-Funktion etc.

## Eingabe mit Booleschen Gleichungen

Um Boolesche Gleichungen zu beschreiben, werden von der Software u.a. folgende Zeichen als Logiksymbole benutzt:

& = logische UND-Verknüpfung
# = logische ODER-Verknüpfung
! = Invertierung
$ = logische XOR-Verknüpfung

Beispielsweise soll die Funktion

Y = (A & B) # (!C & D)

realisiert werden. Der Eintrag in der GLB-Zelle sieht dann wie folgt aus:

```
SIGTYPE Y OUT;
EQUATIONS
Y = (A & B) # (!C & D);
END;
```

Bedeutung der Sprachelemente:

| | |
|---|---|
| SIGTYPE = | Art des Ausgangs mit den Optionen: |
| OUT = | Kombinatorischer Ausgang |
| REG OUT = | Register-Ausgang |
| EQUATIONS = | Start der Logikbeschreibung |
| END = | Ende der Logikbeschreibung |

Abbildung 8 zeigt den Eintrag einer Booleschen Gleichung in die GLB-Zelle A 0.

## IO-Makros

Die IO-Zellen der ispLSI-Serie sind sehr vielseitig konfigurierbar. Die Funktion der Zellen wird durch Plazieren entsprechender IO-Makros festgelegt.

Folgende Makros definieren die Elementarfunktionen Input und Output.

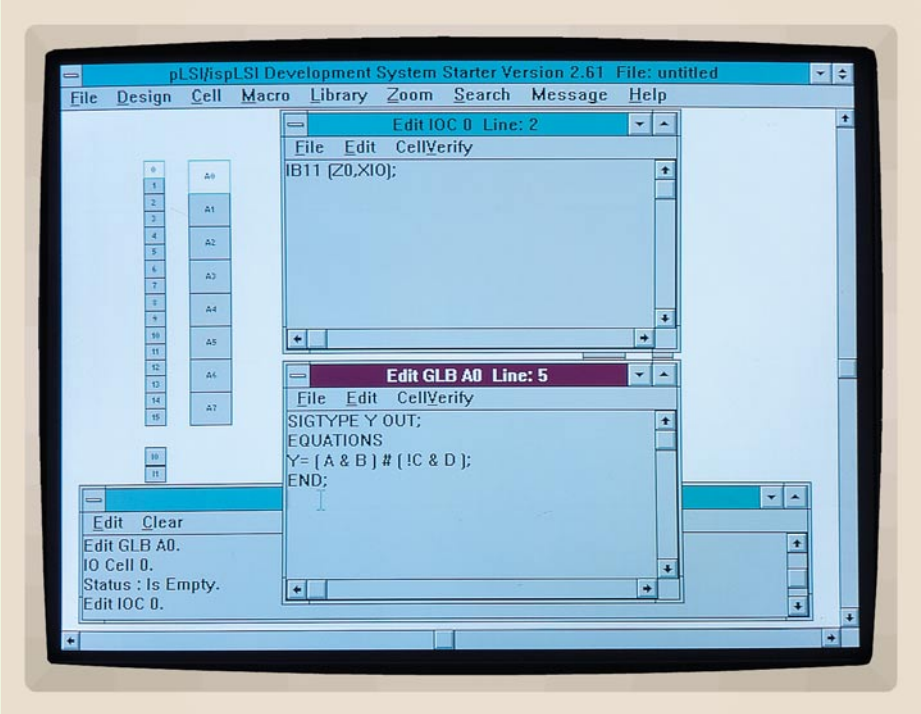



**Bild 8: Eingabe einer Booleschen Gleichung in die GLB-Zelle A0**

| Makroname: | Funktion: |
|---|---|
| IB11 (out,in) | Inputbuffer nicht invertiert |
| IB21 (out,in) | Inputbuffer invertiert |
| OB11 (out,in) | Outputbuffer nicht invertiert |
| OB21 (out,in) | Outputbuffer invertiert |

Vor der Klammer steht der Makroname wie z. B. IB11. Innerhalb der Klammern folgen zuerst die Signalnamen der Ausgänge und im Anschluß die Signalnamen der Eingänge. Bild 9 zeigt ein Inputbuffer

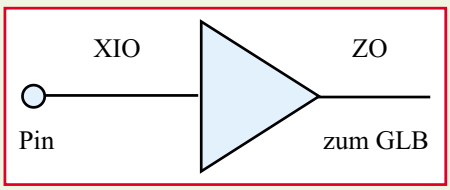



**Bild 9: Blockschaltbild eines Inputbuffer-Makros**

IB11 als Blockschaltbild, dessen Signalnamen folgende Bedeutungen haben:

ZO = Bezeichnung im Baustein (Ausgang Inputbuffer)
XIO = Bezeichnung am Pin des Bauelementes (Eingang Inputbuffer)

Der Ausgang einer IO-Zelle läßt sich wie folgt definieren:
OB11(Y_OUT,Y_IN);
Die Signalnamen sind beliebig wählbar z. B.:

Y_IN = Ausgang (Output vom GLB)
Y-OUT = Ausgang des Bausteins

Der Eintrag eines Makros in die IO-Zelle ist denkbar einfach: Mit einem Mausklick auf die gewünschte IO-Zelle erscheint ein Editierfenster auf dem Bildschirm, in welches dann die Makrobeschreibung eingetragen wird. (Ausführliche Makro-Beschreibungen finden sich im Handbuch). Bild 7 zeigt einen plazierten Inputbuffer IB11 in der IO-Zelle 0.

Eine weitere Möglichkeit zur Plazierung von Makros ist das Laden der Makrodefinition aus der Makrolibrary.

Nach der Aktivierung des Makros mit der oberen Auswahlleiste erscheint ein Makrofenster mit einer Makroübersicht (Abbildung 10).

Über Rollbalken läßt sich das gewünschte Makro auswählen. Danach stehen 2 Optionen zur Verfügung, um das Makro zu laden:

Funktion „Softplace" zur Plazierung von IO-Makros, kleinerer funktioneller Makros sowie von Makrosegmenten, wenn das Makro mehr als ein GLB belegt.

Funktion „Hardplace" zur Plazierung von hardwareoptimierten Makros wie z.B. Zähler. Die Plazierung auf einzelne GLBs erfolgt dann automatisch. Wurde eine der beiden Funktionen gewählt, fordert die Software die Auswahl einer entsprechenden Zelle an, in die dann per Mausklick das Makro geladen wird.

Der Anwender muß jetzt nur noch die Default-Signalnamen der Makros durch eigene Namen ersetzen.

## Eingabe mit funktionellen Makros

Mit der vorgestellten Methode soll jetzt ein komplexes Makro (16-Bit-Zähler) geladen werden. Ein 16-Bit-Zähler ist z. B. unter der Bezeichnung CBD616 in der Makrolibrary enthalten und wird über „Hardplace" plaziert. Abbildung 7 zeigt das geöffnete GLB A0 mit dem Makro CBD616, das wie folgt beschrieben ist:

CBD616 (Q0..Q15, CAO, CLK, EN, CD)
CBD = Counter Binary Down

Default-Signalnamen:

Q0..Q15 = Zählerausgänge 0 bis 15
CAO = Carry Out

**Bild 10: Übersicht über die von der Software bereitgestellten Makros**

CLK = Clock
EN = Enable
CD = Clear Direct

Ist das Makro plaziert, werden die Makrosignalnamen entsprechend dem Design geändert. Makroeingabe und Boolesche Eingabe lassen sich auch gemischt verwenden, wobei die IO-Zellen wie im vorherigen Beispiel konfiguriert werden.

Die Taktleitung ist für die Software ein normaler Input, welcher allerdings nur in Y-Zellen (Y 0-Y 3) plazierbar ist. Als Beispiel für Y 0 (Standard-Takteingang) IB11 (CLK, CLK_IN):

CLK_IN = Takteingang am Y 0-Pin
CLK = Signalname im Design

## Verify-Funktion

Ist die Designeingabe abgeschlossen, folgt die Verifikation. Die Software überprüft an dieser Stelle die eingegebenen Gleichungen, minimiert das Design, stellt fest, ob Signale nicht benutzt wurden und splittet die mit „Hardplace“ geladenen Makros in einzelne GLBs auf (Beispiel: CBD616 in 5 einzelne GLBs).

## Routing

In der Funktion Routing werden die internen Verbindungen festgelegt. Die Software kann die Pinzuordnung des Bausteins automatisch festlegen. Der Anwender kann aber auch die Pins ganz oder teilweise festlegen. Die Festlegung der einzelnen Pins erfolgt durch Mausklick auf einen der Signalnamen im Routing-Fenster. Abbildung 11 zeigt das geöffnete Routing-Fenster mit einer manuell gewählten Pinzuordnung.

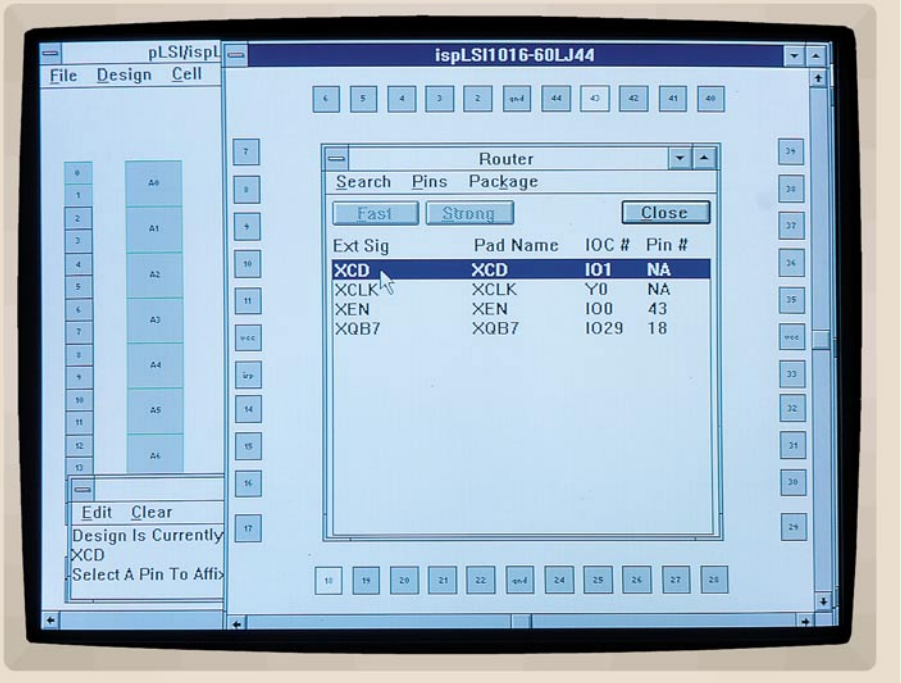



**Bild 11: Geöffnetes Routing-Fenster mit der Pinzuordnung**

## Fusemap

Die Fusemap enthält die Programmierinformationen für den Baustein. Hier wird aus dem Ergebnis des Logikentwurfs die JEDEC-Datei erstellt, welche dann per Download in den Baustein geladen wird.

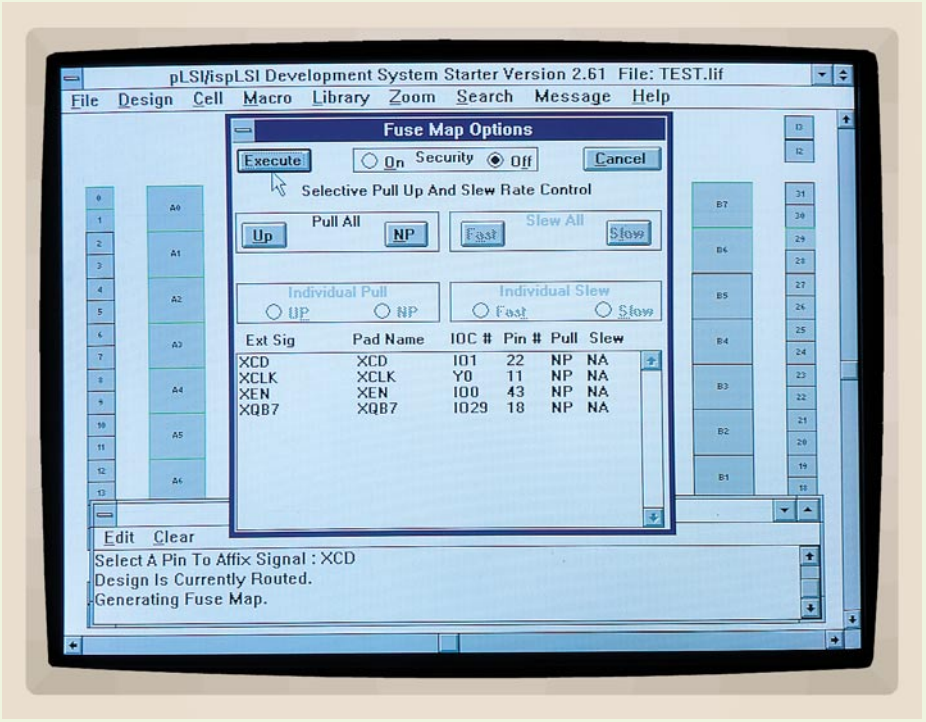



**Bild 12: Die Fusemap-Funktion der isp-SK2-Software**

Der Anwender hat in dieser Softwarefunktion, wie aus Abbildung 12 ersichtlich, zusätzlich folgende Wahlmöglichkeiten:

- Festlegung der Pullups pro IO-Pin
- Setzen des Security-Bits, um das Auslesen des Bauelements zu verhindern.

## Download

Mit einem angeschlossenen Down-Load-Kabel kann mit dem Download in das Zielsystem begonnen werden. Für die Produktion steht auch eine einfach zu bedienende Down-load-Software unter DOS zur Verfügung.

## Zusätzliche Tools der isp-SK2

Für ein komfortables Arbeiten mit der isp-SK2-Software sind darin besonders nützliche Tools enthalten:

- Download-Software als C-Quellcode für eigene Adaption (z. B. Mikrocontroller).
- Download-Programm unter MS-DOS für die Produktion und den Service.
- Daisy-Chain-Software zur Programmierung von mehr als einem Baustein auf einer Platine.
- Software zur Unterstützung von weiteren **i**n-**s**ystem-**p**rogrammierbaren Bauelementen von Lattice (ispGAL22V10, ispGDS).

Im dritten Teil dieses PLD-Artikels beschreiben wir den Aufbau und die Funktionsweise des von ELV entwickelten PLD-Entwicklungs-Kits, mit dessen Hilfe sich bereits eindrucksvolle Beispiele programmieren lassen.