
Klaus Dembowski
Das Addison-Wesley Handbuch der Hardwareprogrammierung
Band1
Band 2
ADDISON-WESLEY
DVD
Dembowski
Band1
Das Addison-Wesley Handbuch der Hardwareprogrammierung
2305
Dembowski
Band 2
Das Addison-Wesley Handbuch der Hardwareprogrammierung
2305
Dembowski
DVD
Das Addison-Wesley Handbuch der Hardwareprogrammierung
Interaktive DVD
2305

Kapitel 3

Programmierbare Logikschaltungen

Die zunehmende Komplexität einer Logikschaltung erfordert eine Vielzahl an digitalen Bausteinen, die es als Standard-Chips mit unterschiedlichen Funktionen und für verschiedene Logikfamilien gibt, wie es im Kapitel 2 aufgezeigt wurde. Doch je mehr Bausteine verwendet werden, desto höher ist die Wahrscheinlichkeit für Fehlfunktionen, und auch der Leistungsverbrauch steigt an. In der Praxis kommt es außerdem immer wieder vor, dass die Funktionen eines Bausteins nicht ausgenutzt werden, weil beispielsweise nur ein oder zwei von vielleicht acht im Baustein vorhandenen Gattern oder Flip-Flops für die jeweilige Applikation notwendig sind.

Außerdem sind die einzelnen Logikbausteine elektrisch miteinander zu verbinden, was meist anhand von extra anzufertigenden Platinen erfolgt, die über entsprechende Leiterbahnen verfügen müssen. Anstatt für eine bestimmte Applikation nun eine hohe Anzahl von einzelnen Logikbausteinen zu verwenden, kann ein einziger ***programmierbarer Logikbaustein*** möglicherweise die gewünschte Funktion übernehmen. Notwendige Veränderungen oder Anpassungen an sich verändernde Randbedingungen lassen sich dann ohne eine Neukonstruktion der Hardware durchführen, denn es ist lediglich eine Anpassung der im Chip arbeitenden Software notwendig.

Im Jahre 1977, sechs Jahre nach der Vorstellung des ersten Mikroprozessors (Intel 4004), erschienen bereits die ersten kommerziellen programmierbaren Logikbausteine auf dem Markt. Sie wurden als ***Programmable Array Logic*** (PAL) bezeichnet und sind von der Firma Monolithic Memories entwickelt worden. Daraus haben sich im Laufe der Zeit immer komplexere Bausteine entwickelt, wobei die ***Field Programmable Arrays*** (FPGAs) heutzutage das obere Leistungsende dieser Bausteine repräsentieren.

Die ursprüngliche Idee bei der Verwendung von programmierbaren Logikbausteinen war, die notwendigen Logikgatter möglichst durch einen einzigen Baustein ersetzen zu können, wobei diese Lösung flexibler und auch kostengünstiger als der Aufbau einer Applikation mit diskreten Logikbausteinen sein sollte. Diese Tatsache kann in der Industrie bereits seit längerer Zeit als erfüllt angesehen werden. Darüber hinaus wurden mit programmierbaren Logikbausteinen aber auch neue Anwendungen möglich, die mit konventioneller Logik nicht zu bewältigen sind, wie beispielsweise »intelligente« Interfaces für PCI, SCSI oder auch den ***Universal Serial Bus*** (USB).

Es ist durchaus gängige Praxis, ein neues Interface, etwa für PCI-Express oder Hypertransport, zunächst mit einem FPGA zu realisieren, weil sich Änderungen in der Entwicklungsphase dann relativ schnell per Software durchführen lassen. Erst nach vollständiger Umsetzung der gewünschten Funktionalität fertigt der Hersteller den gewünschten Inter-

face-Chip, bei dem es dann für den Kunden keine Möglichkeit mehr gibt, an den internen Gegebenheiten etwas zu verändern. Dies ist auch nicht mehr notwendig, weil der Interface-Chip dann beispielsweise der PCI-Bus-Spezifikation Version 2.3 entspricht und dementsprechend zu verwenden ist. Wie der Chip-Hersteller intern die hierfür notwendigen logischen Verknüpfungen, Register und Speicherzellen realisiert hat, spielt für den Entwickler einer passenden Interface-Schaltung keine Rolle.

Der somit realisierte Interface-Chip kann auch als ASIC (Application Specific Integrated Circuit) angesehen werden. Der Bezeichnung nach ist es ein Applikations-spezifischer Baustein, der hier eben für PCI-Bus-Interfaces vorgehen ist, wobei er intern »fest verdrahtet« ist. Üblicherweise wird dieser Interface-Baustein dann jedoch nicht als ASIC, sondern als PCI-Bus-Dekoder oder ähnlich bezeichnet werden, also gemäß seinem Anwendungszweck.

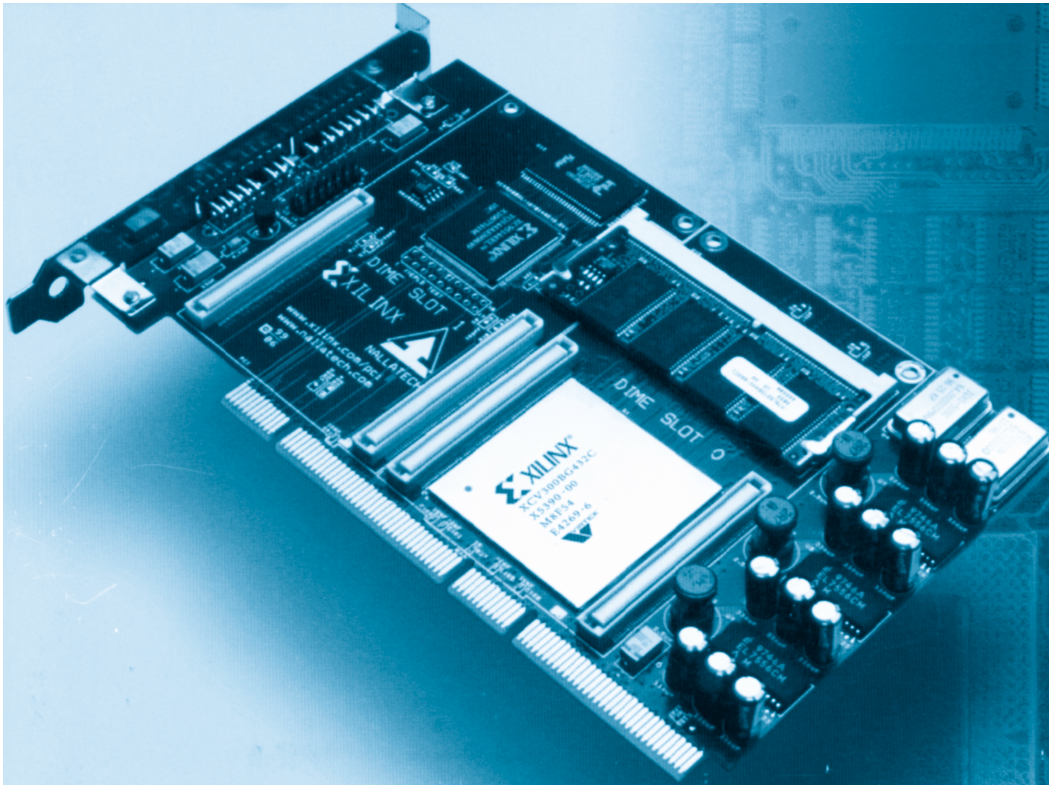

Abbildung 3.1: Eine 64-Bit-PCI-Karte mit einem FPGA aus der Virtex-Serie der Firma Xilinx als PCI-Bus-Interface

Jeder etwas speziellere Logikbaustein ist gewissermaßen aus einem ASIC oder auch FPGA entstanden, bevor er vielleicht zu einem Standardbaustein avanciert und dann in großen Stückzahlen preiswert zu fertigen ist. Die Entwicklung einer eigenen integrierten, anwendungsspezifischen Schaltung sollte dabei für den Entwicklungsingenieur möglichst so einfach sein wie der ihm vertraute Platinenentwurf. Deshalb sind für ASICs bestimmte Bibliotheken mit gewissen Standardfunktionen verfügbar, so dass diese Entwurfstechnik auch wieder die Selektierung bestimmter Gatter und anderer digitaler Elemente voraussetzt, wie es auf anderer Ebene mit den Standard-Logik-Chips der Fall ist. Die Platzierung und Verdrahtung der Elemente erfolgt dabei aber nicht auf einer Platine, sondern im Chip mithilfe spezieller CAE-Programme, meist unter Mitwirkung der IC-Entwickler beim Halbleiterhersteller.

Im Kapitel 3.3 wird noch etwas näher auf die *Field Programmable Gate Arrays* eingegangen. Die Bezeichnung »Field« soll dabei andeuten, dass sich diese Chips im Feld, also in üblicher Arbeitsumgebung eines Entwicklers, programmieren und/oder rekonfigurieren lassen, und keinerlei Halbleitertechnologie wie bei den ASICs für die Implementierung der jeweiligen Funktion notwendig ist.

3.1 Programmable Logic Devices

Die allgemein als *Programmable Logic Devices* (PLDs) bezeichneten Bauelemente gibt es in zahlreichen verschiedenen Ausführungen und mit unterschiedlicher Komplexität. Insbesondere viele ISA-Interfaces auf PC-Einsteckkarten werden mit Hilfe dieser programmierbaren Logikbausteine konstruiert, da diese Realisierungsmöglichkeit prinzipiell eine Reihe von Standard-TTL-Bausteinen ersetzen kann und sich bei größeren Kartenstückzahlen auch preiswerter produzieren lässt. Zu den einfacheren *Programmable Logic Devices* (PLDs) gehören im Wesentlichen:

- PROMs

Die PROMs (Programmable Read Only Memory) sind in erster Linie als Speicherelemente bekannt. Sie besitzen ein fest verdrahtetes AND-Array und ein programmierbares OR-Array. Entsprechendes gilt für die EPROMs (Erasable Programmable Read Only Memory), deren Inhalt durch UV-Licht löschbar ist.

- PLAs

Die PLA-Bausteine (Programmable Logic Based Sequencer) verfügen sowohl über ein programmierbares AND-Array als auch über ein programmierbares OR-Array. Aus Kostengründen ist für einfache logische Verknüpfungen In den meisten Fällen wird ein PAL einem PLA aus Kostengründen vorgezogen.

- PALs

Die PALs (Programmable Array Logic) haben ein programmierbares AND-Array und ein fest verdrahtetes OR-Array (ODER). Dadurch sind viele Eingänge durch ein UND verknüpfbar. Löschbare PALs werden auch als EPLD (Erasable) oder GAL bezeichnet, was vom jeweiligen Hersteller abhängig ist, denn die Bezeichnung PAL stammt vom Hersteller (Monolithic Memories) des ersten programmieren Logik-Bausteins.

Zur Programmierung dieser Logikbausteine werden eine Software und meist ein spezielles Programmiergerät benötigt. PLDs werden in TTL-, CMOS- und ECL-Technik hergestellt. Die CMOS-Versionen sind mit den TTL-Versionen zumeist kompatibel, haben jedoch einen geringeren Stromverbrauch. Einige der Bausteine besitzen interne Rückkopplungsleitungen, wodurch Ausgänge als Eingänge zurückgekoppelt werden können. Tri-State-Ausgänge können ebenfalls vorhanden sein, so dass die Ausgangsleitungen als Eingang, Ausgang und bidirektional verwendet werden können. Man unterscheidet vier verschiedene Funktionsgruppen.

- Einfache Logik für Gatterfunktionen (Combinatorial Devices).
- Logik mit Registern (Registered Devices).
- Sequenzer, D-,T-,J/K- oder RS-Flip-Flops (Sequencer Devices).
- Asynchrone Anwendungen, programmierbarer Takt, Leitungen zum Setzen und Rücksetzen der Flip-Flops (Asynchronous Devices).

Die Anwendungen für PLDs sind demnach sehr vielfältig. Von einer einfachen Dekodierung bis zu komplexen Interfaces und Controllern reichen die möglichen Einsatzgebiete. Im Folgenden wird auf die PALs näher Bezug genommen, da sie die häufigsten Vertreter in einfachen Interface-Schaltungen darstellen.

Exkurs

Auch für einfachere Applikationen, bei denen sich ein klassischer PLD-Baustein (PAL) als ausreichend darstellen würde, wird heutzutage üblicherweise auf einen Chip zurückgegriffen, der unter CPLD firmiert. Dieser Umstand ist darin begründet, dass sich diese Chips kostengünstiger als die älteren Typen herstellen lassen und auch kein spezielles Gerät für die Programmierung und für das Löschen benötigt wird.

Durch die Programmiersoftware wird eine JEDEC-Datei erzeugt, mit der der Baustein in einem Programmiergerät entsprechend zu programmieren ist. Die Verbindungsleitungen (Fuses) werden durch das Programmiergerät durchtrennt. Während der Programmierung kann außerdem eine *Security Fuse* durchtrennt werden, die ein Auslesen des Bausteininhalts verhindert. Elektrisch löschbare PALs, die bei einigen Herstellern auch als GALs (Generic Array Logic) bezeichnet werden, können mehrere Male programmiert und damit mehrfach verwendet werden. Wie erwähnt, können sich die Bezeichnungen von Hersteller zu Hersteller unterscheiden. Eine für die einfachen PALs gebräuchliche Kennzeichnung lautet wie folgt:

PAL 20 L 8 A L

- Power Level
- Geschwindigkeit
- Anzahl der Ausgänge oder Flip-Flops
- Architektur
- Anzahl der Eingänge

Power Level:

Keine Angabe: Full Power, 180-240 mA

A: Half Power, 90-105 mA

B: Quarter Power, 45-55 mA

Geschwindigkeit:

Keine Angabe: 35 ns

A: 25 ns

B: 15 ns

D: 10 ns

Architektur:

H: Aktiv HIGH Ausgang

L: Aktiv LOW Ausgang

P: Programmierbare Ausgangspolarität

C: Komplementär-Ausgang

XP: Exklusiv OR-Gatter, programmierbare Polarität

S: Produktterme können zwischen Ausgängen platziert werden

Als Beispiel für den internen Aufbau eines PALs ist in der Abbildung 3.2 der Typ PAL20L8 gezeigt. Es ist zu erkennen, dass der Baustein 20 Eingänge besitzt, lediglich 14 sind aber als reine Eingänge (Pin 1 bis 11, Pin 13, 14, 23) geschaltet, die anderen können sowohl als Ein- als auch als Ausgang verwendet werden. Die Pin-Angaben beziehen sich dabei auf einen Baustein im DIP-Gehäuse, die Werte in Klammern hingegen auf zwei verschiedene PLCC-Gehäuseformen, in denen der Baustein ebenfalls angeboten wird. Das AND-Array ist programmierbar, und die acht OR-Gatter sind fest verdrahtet. Jeder Ein- und Ausgang kann invertiert werden.

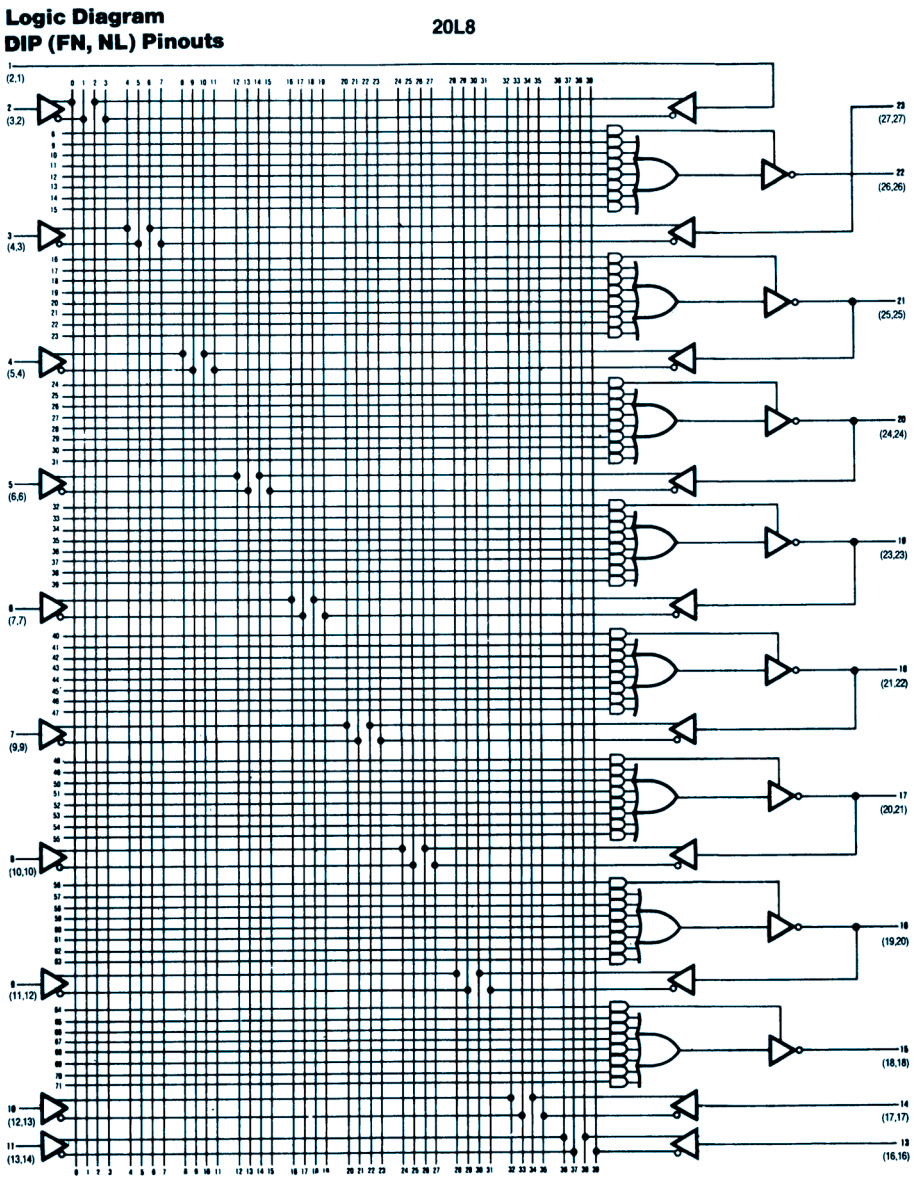

Abbildung 3.2: Der interne Aufbau eines PALs vom Typ 20L8

Die horizontalen Zeilen (8-71) werden als ***Produktterme*** bezeichnet. Jede AND-Verknüpfung kann maximal so viele Eingänge besitzen, wie Eingangsspalten (0–30) vorhanden sind. Diese Terme gelangen auf die OR-Gatter. Im unprogrammierten Zustand sind die Kreuzungspunkte der Zeilen und Spalten miteinander verbunden. Durch die Programmierung werden die Verbindungen (Fuses) aufgetrennt. Folgende Schritte sind dabei üblicherweise für die PLD-Programmierung notwendig:

1. **Create:** Erstellen der logischen Gleichungen mit einem Editor. Als Editor kann nahezu jeder gebräuchliche verwendet werden.
2. **Syntax Check:** Die Quelldatei wird auf ihre syntaktische Korrektheit hin überprüft.
3. **Expand:** Vom Programm wird die komplette Wertetafel für die Gleichungen aufgestellt.
4. **Minimize:** Anwendung der Schaltalgebra-Rechenregeln zur Minimierung der Ausgangsfunktion.
5. **Assemble:** Erstellen des JEDEC-Files.
6. **Simulate:** Testen, ob die Ausgangssignale zusammen mit den angelegten Eingangssignalen die gewünschte Funktion ergeben.
7. **Program:** Baustein mit der JEDEC-Datei programmieren.

Eingangs wurde erwähnt, dass PLDs bei PCs oftmals für Businterface-Schaltungen eingesetzt werden. Für die Dekodierung einer Adresse auf dem ISA-Bus (Kapitel 18) sind das Schreibsignal /IOW, das Lesesignal /IOR, das Signal AEN zur Unterscheidung zwischen Prozessor- und DMA-Zugriff sowie die Adressenleitungen A0-A9 zu dekodieren.

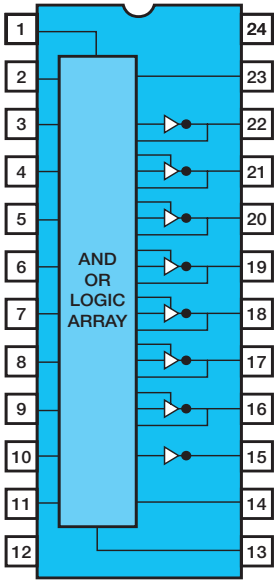

Abbildung 3.3: Die Anschlussbelegung des Bausteins PAL20L8

Als Ausgänge sollen acht ***aktiv Low*** Chip-Select-Signale (/CS) zur Verfügung stehen, an die sich dann Peripherie-Bausteine anschließen lassen. Das folgende Listing zeigt einen Ausschnitt aus der hierfür notwendigen Quelldatei im PALASM-Format.

```
TITLE        PCIO.PDS
CHIP  GATES  PAL20L8

;PIN  1   2   3   4   5   6   7    8    9    10   11   12
      A0  A1  A2  A3  A4  A5  A6   A7   A8   A9   AEN  GND

;PIN 13   14   15   16   17   18   19   20   21   22   23  24
     IOR  IOW  CS1  CS2  CS3  CS4  CS5  CS6  CS7  CS8  NC  VCC


EQUATIONS

/CS1= /IOR*/AEN*A9*A8*/A7*/A6*/A5              ;300H-31FH,READ
    + /IOW*/AEN*A9*A8*/A7*/A6*/A5              ;300H-31FH,WRITE

/CS2= /IOR*/AEN*A9*A8*/A7*/A6*/A5*/A4*/A3*/A2  ;300H-303H,READ
    + /IOW*/AEN*A9*A8*/A7*/A6*/A5*/A4*/A3*/A2  ;300H-303H,WRITE

.............

/CS8= /IOR*/AEN*A9*A8*/A7*/A6*/A5*/A4*A3       ;308H-30FH,READ
    + /IOW*/AEN*A9*A8*/A7*/A6*/A5*/A4*A3       ;308H-30FH,WRITE


SIMULATION

TRACE_ON  A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AEN IOR IOW CS1 CS2
          CS3 CS4 CS5 CS6 CS7 CS8

;SELECT CS1

SETF /IOR /AEN A9 A8 /A7 /A6 /A5
CHECK /CS1

SETF /IOW /AEN A9 A8 /A7 /A6 /A5
CHECK /CS1

;SELECT CS2

SETF /IOR /AEN A9 A8 /A7 /A6 /A5 /A4 /A3 /A2
CHECK /CS2

SETF /IOW /AEN A9 A8 /A7 /A6 /A5 /A4 /A3 /A2
CHECK /CS2

.............

;SELECT CS8

SETF /IOR /AEN A9 A8 /A7 /A6 /A5 /A4 A3
CHECK /CS8

SETF /IOW /AEN A9 A8 /A7 /A6 /A5 /A4 A3
CHECK /CS8

TRACE_OFF
```

Im *Title Block* wird mit *Chip Gates* der gewünschte PAL-Typ festgelegt. Der Assembler vergleicht diesen angegebenen Baustein später mit seiner internen Bibliothek. Wenn das PAL nicht vorhanden ist oder die Angaben für den Baustein nicht korrekt sind – Eingänge als Ausgänge deklariert worden sind o. Ä. –, werden entsprechende Fehlermeldungen ausgegeben. Kommentare werden mit einem »;« gekennzeichnet. Es ist sinnvoll, bei der Zuordnung der Pins zu den einzelnen Eingangs- und Ausgangsleitungen der besseren Übersicht wegen Kommentarzeilen voranzustellen. Nach der Eingabe von EQUATIONS werden die Gleichungen für die Dekodierung aufgestellt. Die einzelnen Eingangssignale werden zu einem Produktterm mit einem AND (*) verknüpft, diese Verbindungen werden später bei der Programmierung des Bausteins »gebrannt«.

Wenn in einen Adressbereich sowohl geschrieben als auch aus ihm gelesen werden soll, werden die einzelnen Adressen mit dem /IOW- oder mit dem /IOR-Signal verknüpft, denn beide Signale sind nie gleichzeitig aktiv. Das AEN-Signal wird dabei ebenfalls mit in die einzelnen AND-Verknüpfungen einbezogen. Nachdem alle Gleichungen eingegeben sind, kann optional eine Simulation durchgeführt werden.

Mit SIMULATION wird dies im Quellprogramm gekennzeichnet. Die Simulation ist zur Überprüfung der erzielten Ausgangsfunktion sehr nützlich. Eingeleitet wird sie mit TRACE_ON und der Angabe, welche Signale überprüft werden sollen. Mit SETF und den darauf folgenden Angaben werden die einzelnen Eingangszustände festgelegt, und mit CHECK wird dann der entsprechende Ausgang überprüft. Mit TRACE_OFF wird der Simulationsteil dann abgeschlossen, so dass das Quellprogramm daraufhin in den oben genannten Schritten weiterverarbeitet werden kann.

Nach dem kompletten, erfolgreichen PAL-Assembler-Durchlauf sind dann die folgenden Dateien vorhanden, und anhand des Fuseplot (XPLOT) können die durch das Programm erstellten Verbindungen noch manuell kontrolliert werden.

- PCIO.PDS: Die Quelldatei
- PCIO.XPT: Der Fuseplot, die Darstellung der Verbindungen
- PCIO.JED: Das JEDEC-File, die Programmierdatei
- PCIO.HST: Das Ergebnis der Simulation (History), evtl. mit Fehlermeldungen
- PCIO.TRF: Das Ergebnis der Simulation (Result)

Die ersten PLDs waren nur einmal programmierbar, wobei es aber auch Typen gibt, die elektrisch oder mit UV-Licht löschbar sind. Die programmierbaren Logikelemente sind bei den PLDs technologisch gesehen wie bei den bekannten Speicherbausteinen (ROM, EPROM, EEPROM) realisiert. Die hier behandelten PLDs werden auch als SPLDs, was für *Simple PLD* steht, bezeichnet. Sie verfügen über eine vergleichsweise geringe Logikdichte mit maximal 1000 Gatteräquivalenten, und alle zu verknüpfenden Logiksignale werden auf eine gemeinsame UND-Matrix geführt.

3.2 Complex PLDs

Die zuvor erläutertet PLDs werden nicht mehr weiterentwickelt. Stattdessen sind mittlerweile die CPLDs als Standard bei den PLDs anzusehen.

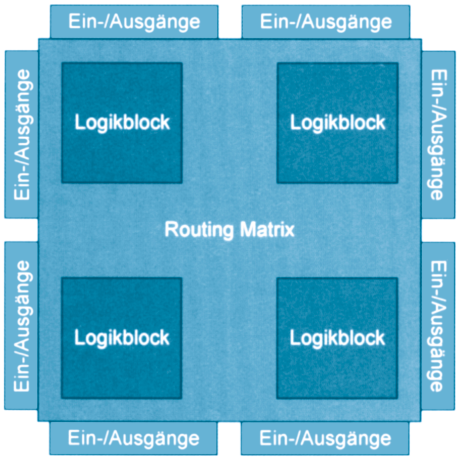

Abbildung 3.4: Ein CPLD verfügt über mehrere Logikblöcke, die über eine Routing Matrix miteinander verbunden werden können.

Diese ***Complex PLDs*** verfügen nicht nur wie die SPLDs über einen einzigen Logikblock, sondern über mehrere, wobei die einzelnen Blöcke in einer PAL-ähnlichen Struktur ausgelegt sind, so dass das grundlegende Prinzip das Gleiche ist wie bei den PALs. Die Blöcke werden hier als ***Function Blocks*** oder auch als LAB (Logic Array Block) bezeichnet und mithilfe einer Schaltmatrix (Routing Matrix) untereinander verbunden. Die LABs setzen sich dabei meistens aus den drei Komponenten UND-Matrix, Produktterm-Logik und Ausgangs-Makrozelle zusammen.

Bekannte CPLD-Familien sind ispMACH 4 der Firma Lattice, XC9000 der Firma Xilinx und MAX 7000 der Firma Altera, die bis zu 512 Makrozellen mit bis zu 20.000 Gatteräquivalenten bieten, wobei Taktfrequenzen von 100 bis 200 MHz hier als Standard gelten. Typische Gehäuseformen sind hierfür PLCC mit 44 Pins bis hin zu BGA-Gehäusen mit an die 400 Anschlüssen.

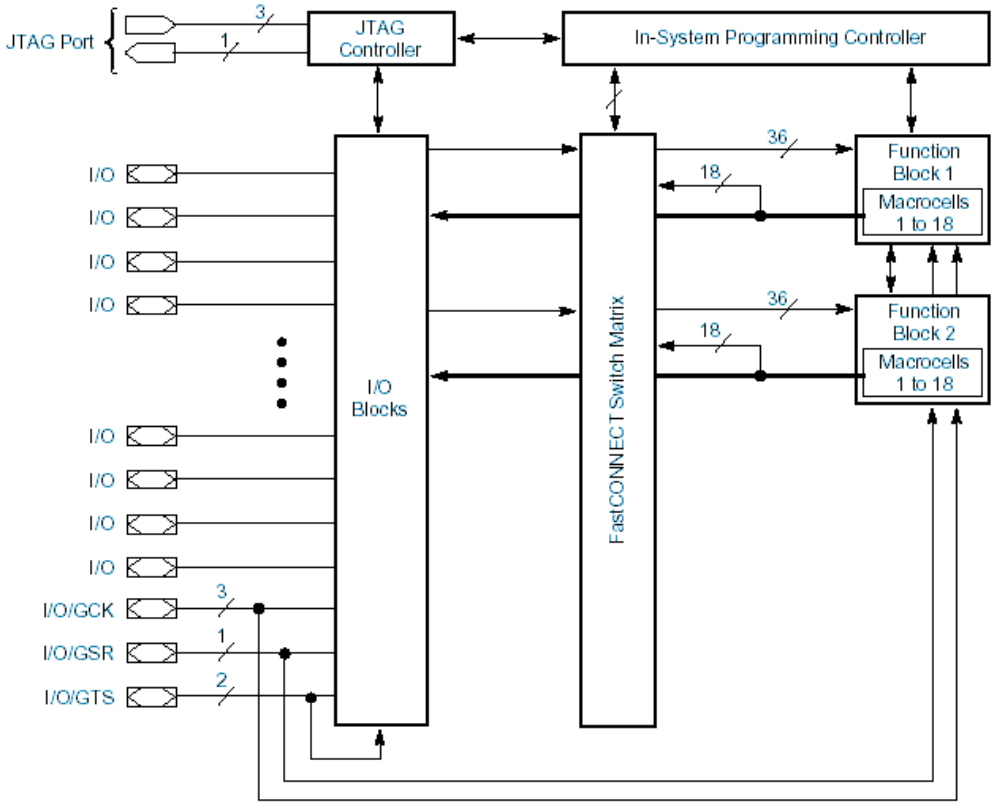

Abbildung 3.5: Der interne Aufbau eines CPLD

Üblicherweise sind die CPLDs in der Applikation programmierbar (ISP, In System Programmable), was bei den SPLDs meist nicht der Fall ist. Hierfür gibt es einfache Adapterlösungen (z.B. den ByteBlaster von Altera), die beispielsweise Signale vom Parallel-Port auf ein im Chip integriertes JTAG-Interface oder auch auf eine herstellerspezifische Schnittstelle umsetzen.

Die komplette Programmerstellung sowie die anschließende Programmierung erfolgt dabei mithilfe einer vom Hersteller des CPLD angebotenen Entwicklungsumgebung, die im Internet meist kostenlos zur Verfügung gestellt wird. Oftmals lässt sich mit der gleichen Entwicklungssoftware und dem Adapter auch eine FPGA-Applikation erstellen, wobei die Software dann zumindest für die aktuellsten und leistungsstärksten Typen nicht mehr kostenlos abgegeben wird.

3.3 Field Programmable Gate Arrays

In den achtziger Jahren wurden von der Firma Xilinx erstmalig die *Field Programmable Arrays* (FPGAs) vorgestellt, die mittlerweile die gleiche Leistungsfähigkeit wie »festverdrahtete« Logikchips erreichen. Aktuelle FPGAs, wie etwa die Virtex- oder Spartan-Chips der Firma Xilinx, verfügen über bis zu 1,6 Millionen Gatteräquivalente mit 150 Millionen Transistoren und mehr.

Exkurs

Unter einem »Gatteräquivalent« versteht man ein NAND-Gatter mit zwei Eingängen.

Die grundsätzlichen Unterschiede zwischen CPLD und FPGA bestehen im Aufbau der Logikblöcke und in der Architektur der Verbindungsmatrix. Während die Logikblöcke bei CPLDs aus einer PAL-ähnlichen UND/ODER-Struktur bestehen, sind FPGAs aus einfacheren, dafür aber mit sehr vielen Logikblöcken aufgebaut, die als *Configurable Logic Blocks* (CLB) bezeichnet werden. Statt über eine programmierbare Verbindungsmatrix verfügen FPGAs über einzelne programmierbare Verbindungselemente (Input Ouput Blocks), die direkt für die CLB-Verbindungen eingesetzt werden können, was eine fast beliebige Zusammenschaltung von zahlreichen CLBs ermöglicht. Dies hat zur Folge, dass das genaue zeitliche Verhalten einer Schaltung von der Aufteilung der gewünschten Logik in CLBs plus den notwendigen Verbindungen abhängig ist und nicht wie bei den CPLDs recht genau vorherbestimmt werden kann.

Die CLBs enthalten als grundlegende Elemente so genannte *Look Up Tables* (LUTs), die für kombinatorische Logik oder auch für Latches und Flip-Fops und somit auch als Speicherelemente und für arithmetrische Funktionen verwendet werden können. Je nach Hersteller ist der genaue FPGA-Aufbau und die dazugehörige Terminologie recht unterschiedlich, so dass auch die Unterschiede zwischen CPLDs und FPGAs nicht immer deutlich werden und gewissermaßen »verschwimmen«.

Die FLEX-Familien (6000, 8000, 10K) der Firma Altera verfügen beispielsweise über eine *Routing Matrix* wie die CPLDs; die Logikblöcke weisen jedoch eine LUT-Struktur auf, wie sie für FPGAs üblich ist. Altera nennt diese Chips EPLDs, was hier für *Embedded Programmable Logic Devices* steht.

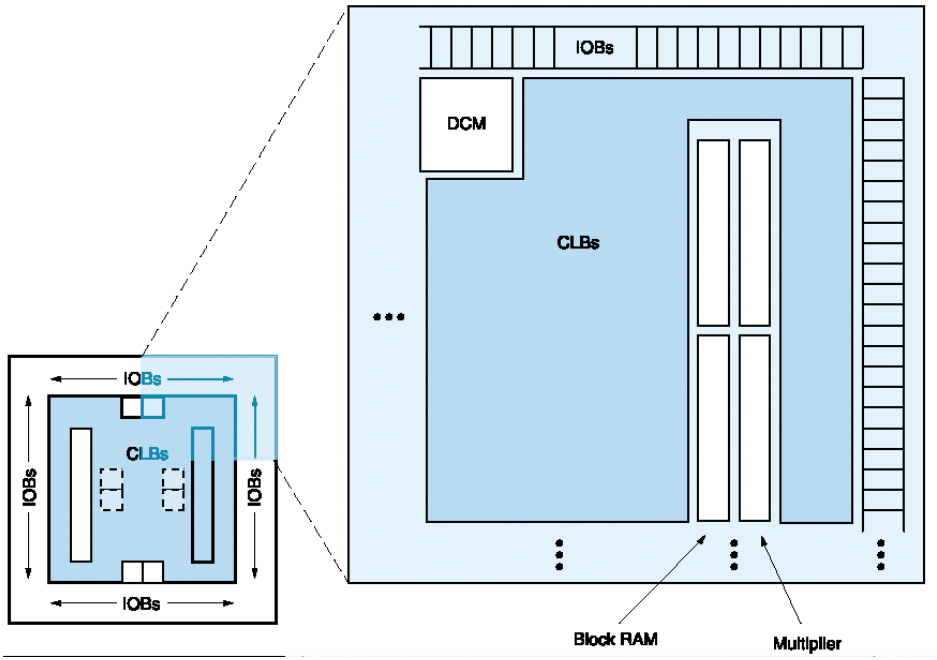

Abbildung 3.6: Aufbau eines FPGA

In der Abbildung 3.6 ist der Aufbau eines FPGA aus der Spartan 3E-Familie der Firma Xilinx gezeigt und in der folgenden Tabelle einige Daten zu den Vertretern aus dieser Familie. Jeder der *Configurable Logic Blocks* ist mit einer Reihe von I/O-Blocks umgeben, und je nach Typ gibt es eine unterschiedliche Anzahl von integriertem Block-RAM, Multiplizier-Blocks (Multiplier) für die schnelle Ausführung von Produktbildungen, mehrere *Digital Clock Manager* (DCM) für die Generierung unterschiedlicher Takte und bis zu über 300 Ein/Ausgabe-Anschlüsse (I/Os).

Typ	System-gatter	Gatter-äquivalent	CLBs	Block RAM	Multipliers	DCMs	I/Os
XC3S100E	100 k	2160	240	72 kBit	4	2	108
XC3S250E	250 k	5508	612	216 kBit	12	4	172
XC3S500E	500 k	10476	1164	360 kBit	20	4	232
XC3S1200E	1200 k	19512	2168	504 kBit	28	8	304
XC3S1600E	1600 k	33192	3688	648 kBit	36	8	376

Tabelle 3.1: Daten von FPGAs aus der Spartan 3E-Familie der Firma Xilinx

Als Speicherelemente kommen bei FPGAs vorwiegend SRAM-Zellen zum Einsatz, was bedeutet, dass die programmierte Logikfunktion nach der Abschaltung der Versorgungsspannung verschwunden ist. Die Logikfunktion wird deshalb bei jedem Hochfahren des Systems aus einem separaten Speicherbaustein in das FPGA geladen, was ca. 100–0 ms dauert. Die Firma Lattice hat im Jahre 2005 erstmalig FPGAs (Lattice eXpanded Programmability) vorgestellt, die über einen integrierten FLASH-Speicher verfügen, was die FPGA-Anwendung vereinfacht und zudem eine beschleunigte Verfügbarkeit (1 ms) zur Folge hat. Für CPLDs ist die Flash-Technologie bereits seit längerer Zeit im Einsatz.

Exkurs

Die Hersteller von programmierbaren Logikbausteinen stellen üblicherweise für die Software-Entwicklung kostenlose Programme (im Internet) zur Verfügung, die sowohl für CPLDs als auch für FPGAs geeignet sind.

Wie es bereits bei den CPLDs erwähnt wurde, gibt es auch für FPGAs entsprechende Entwicklungsumgebungen von den Chip-Herstellern. Für den CPLD-Entwurf genügen meist Beschreibungsmethoden auf einem niedrigen Abstraktionsniveau, beispielsweise grafisch, wie es mit der Erstellung von Schaltplänen (Schematic Design) vergleichbar ist, oder auch anhand der Boole´schen Algebra, wie es auch zuvor bei dem PAL-Beispiel gezeigt wurde.

3.3.1 Hardware Description Language und IP Cores

Mit steigender Komplexität der CPLDs und erst recht für FPGAs wird eine höhere Abstraktionsebene benötigt. Neben herstellerspezifischen Lösungen wird der Entwurf zumeist in einer *Hardware Description Language* (HDL) wie Verilog oder VHDL (Very High Level Description Language) durchgeführt, was fast alle Entwicklungsumgebungen beherrschen. Der Entwurf kann dabei zunächst unabhängig vom jeweiligen Chip erfolgen, und erst am Ende des Vorgangs wird ein passender Typ aus einer Bibliothek bestimmt, wobei jeder Chip-Hersteller aus seiner Entwicklungsumgebung heraus natürlich nur seine eigenen Bauelemente vorschlägt. Eine gewisse Standardisierung ergibt sich jedoch bei der Erstellung des Quellcodes durch die Verwendung der hierfür vorgesehenen IEEE-Bibliotheken (IEEE1076-1987/1999), wie es auch im folgenden Beispiel für einen Addierer im VHDL-Code angegeben ist. Es gibt hier zwei Eingangssignale (A, B) und das entsprechende Ausgangssignal (RESULT).

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity ADDER is
       generic (N: integer:=4);
       port (CLK, RESET: in std_logic;
      A, B: in std_logic_vector (N-1 downto 0);
```

```
        RESULT: out std_logic_vector (N-1 downto 0)
              ) ;
end ADDER
```

Nach der Erstellung des Quellcodes erfolgt wie bei einer üblichen Software-Programmierung die Übersetzung, was mit einer Simulation verbunden werden kann, so dass Fehler in einem möglichst frühen Stadium erkannt und daraufhin beseitigt werden können. Nach der Simulation erfolgt die Synthese, wo der Code als Verschaltung der Logikelemente (Netzliste) stattfindet, die bei der Implementierung dann in der jeweiligen FPGA-Hardware abgebildet werden. Als letzter Schritt erfolgt die Programmierung – meist in einem separaten Speicher-Chip – bzw. die Übertragung des Codes in den FPGA. Die Überprüfung der korrekten Funktion kann üblicherweise an mehreren Stellen während des Designs stattfinden, wobei Verification eine dieser Möglichkeiten darstellt.

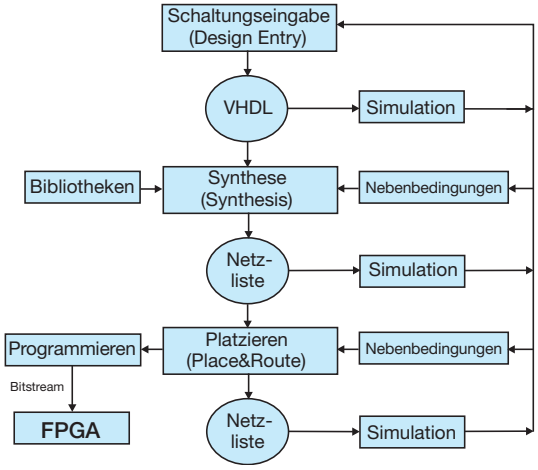

Abbildung 3.7: Die einzelnen Schritte für den VHDL-Entwurf

Mithilfe von CPLDs und FPGAs lassen sich eine Vielzahl an unterschiedlichen Anwendungen realisieren, wie etwa die Eingangs erwähnten Interface-Schaltungen für PCI oder den USB. Für unterschiedlichste Schnittstellen, Controller, CPUs und DSPs sind so genannte *IP Cores* verfügbar, die in Verilog oder VHDL geschriebene Designs darstellen und in unterschiedliche CPLDs und FGPAs geladen werden können. IP steht dabei für *Intellectual Property*, was soviel bedeutet wie »Eigentumsrechte an einer Idee«. Eine einfache (eigene) 8-Bit-CPU lässt sich beispielsweise mühelos in einem CPLD mit 32 Makrozellen wie einem Baustein aus der Lattice ispMACH 4-Serie unterbringen. Diese *IP Cores* oder auch *Soft Cores* werden teilweise direkt von den Herstellern der Logik-Chips angeboten, wie auch von darauf spezialisierten Firmen sowie auch von einzelnen Entwicklern, und das oftmals sogar kostenlos, wenn es sich bei den Cores bereits um ältere und etablierte Designs handelt.

Es macht aus Kostengründen aber sicher keinen Sinn, in ein FPGA für 20 € einen 8051-Microcontroller-Core zu laden, wenn dieser Prozessor bereits für 5 € im Elektronikhandel zu erwerben ist. Gleichwohl kann ein 8051-Core für ein CPLD hergenommen werden, um in diesem Prozessor dann zusätzliche Komponenten zu implementieren, wie etwa spezielle Adressdekoder, über die der handelsübliche 8051-Controller nicht verfügt. Die Verwendung bereits vorgefertigter Cores kann also auch aus Kosten/Nutzen-Sicht empfehlenswert sein, zumal es ohne diese vorgefertigten Cores für einen Entwickler kaum mehr möglich ist, 30.000 Gatteräquivalente und mehr überhaupt noch sinnvoll und optimal zu nutzen.