Entwurf digitaler Schaltungen (EDS)

Einführung

1. Wie erledigen wir das, was sich nicht mit Software erledigen läßt?
2. Wie sieht es im Innern der hochintegrierten Schaltkreise aus?
3. Wir stellen uns nicht so an und lassen uns etwas einfallen ...

Schwerpunkte:

- Wann brauchen wir Hardware?
- CPLDs und FPGAs als Grundlage moderner Schaltungslösungen
- Tricksen mit elementaren Digitalschaltungen
- Zeitbestimmende Hardware (Verzögerungsstufen, Takterzeugung usw.)
- Reguläre Steuerwerke
- Prinzipien der Mikroprogrammsteuerung
- Mikrocontroller als Hardware
- Leistungssteigernde Zusatzbeschaltung
- Digitale Elektronik – Impulse sind nicht einfach zerhackter Gleichstrom
- Entwerfen über Schaltplan

– Entwerfen heißt, eine bestimmte Aufgabe mit zuhandenen technischen Mitteln auf wirtschaftlich und anwendungspraktisch sinnvolle Weise zu lösen. –

1. Die Aufgabe:

- Realzeitraster: in welcher Zeit ist es zu tun?
- Funktionalität: was ist zu tun? (die auszuführenden Informationswandlungen).

2. Die Voraussetzungen und Nebenbedingungen:

- Zuhandenheit,
- Infrastruktur,
- Kosten,
- Zeitbedarf (Einordnung in eine Terminplanung, Time to Market),
- Akzeptanz.

Das Realzeitraster:

- in welcher Zeit ist dieAufgabe zu erledigen?
- einmalig oder zyklisch?
- Latenzzeit: von der Anforderung bis zum Beginn der Reaktion.

Das Ziel:
Durchführung der geforderten Informationswandlungen in der vorgegebenen Zeit t mit kostenoptimalen Mitteln (Kosten über alles = über den gesamten Lebenszyklus = Total Cost of Ownership).

Je geringer die Zeitvorgabe t und je größer die Kompliziertheit bzw. Komplexität, um so größer der Aufwand

Komplexität und Kompliziertheit sind keine Wechselworte:

Komplexität beschreibt Ressourcen-Anforderungen des Algorithmus in Abhängigkeit von der Problemgröße. Beschreibung hat die Form O (n) (Order of... = Größenordnung von...). *Komplex* = mehr als linear mit der Problemgröße wachsende Anforderungen (Verarbeitungsleistung, Speicherplatz, Gatter, Siliziumfläche usw.).

Kompliziertheit = Spitzfindigkeit, Verwickeltheit, Vielfalt der Funktionen. *Kompliziert* = verwickelt (sophisticated). Läßt sich näherungsweise durch Umfang der Problembeschreibung (= funktionelle Spezifikation) kennzeichnen.

Nutzung zuhandener Mittel und Entwicklungstechnologien:

- t groß: das kostengünstigste zuhandene Mittel aussuchen: reicht es aus?
- t extrem klein: ist die Aufgabe überhaupt realisierbar? - Es kommen nur schaltungstechnische Lösungen in Frage. Erfinderische Bemühungen gelegentlich notwendig.

Zeitraster	Beispiele	technische Mittel
10 ns	Befehlsausführung in Prozessoren, Speicheransteuerung, Videodarstellung	Hardware, Gatter-Ebene (auch: Transistor-Ebene)
100 ns	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung
1 µs	Gerätesteuerung	Hardware auf Gatter-Ebene, State Machines, Mikroprogrammierung, Maschinenprogrammierung (was sich mit 10...50 Maschinenbefehlen erledigen läßt)
1 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop)	Realzeit-Software (was sich mit wenigen tausend Maschinenbefehlen erledigen läßt)
10 ms	Gerätesteuerung, Regelungsaufgaben (Closed Loop), Bedienung/Anzeige	Realzeit-Software
100 ms	Regelungsaufgaben (Closed Loop), Bedienung/Anzeige, Prozeßsteuerung, Informationsbeschaffung (Retrival)	komplexe Realzeit-Software, Vernetzung
1 s	Prozeßsteuerung, Informationsbeschaffung (Retrival)	Realzeit- bzw. beliebige Software (je nach Hardware-Plattform)
kommt nicht drauf an	Fertigungssteuerung, allgemeine Verwaltung	beliebige Software (auch Windows etc.), Vernetzung (incl. Internet)

Tabelle 1 Typische Größenordnungen des Realzeitrasters

Zuhandenheit:

- Technologien,
- Beschreibungsmittel,
- Fachwissen (State of he Art), gedankliche Ansätze (Paradigmata), Problemverständnis,
- Verifizierungsmittel:
  - Testen,
  - Simulation,
  - Prüfen/Messen,
  - Fehlersuchen (Debugging).

Was man nicht prüfen kann (Funktionsnachweis), kann man eigentlich gar nicht realisieren.

Ebenen der Zuhandenheit:

- Basistechnologien (Si, GaAs usw.),
- Fertigungstechnologien (z. B. 0,35 µm CMOS),
- Bauelemente - fertige Schaltkreise, auch programmierbare:
  - Gatter-Ebene
  - Register-Transfer-Ebene
  - Funktionsblöcke (Building Blocks)
- Hardware-Plattformen,
- Entwicklungsumgebungen,
- Systemumgebungen.

Paradigmata:
Mit welchen geistigen und handwerklichen Voraussetzungen der Entwickler an die Aufgabe herangeht, wird oft nicht beachtet – es ist aber nicht selten von entscheidender Bedeutung. Uns geht es hier nicht um das subjektive Können, sondern um die Wahl der grundsätzlichen Lösungsansätze, d. h. sozusagen um die "Philosophie" der Problemlösung bzw. des Systementwurfs (Tabelle 2).

Hinweis:
Wichtig ist, jeweils einen *zweckmäßigen* Ansatz zu wählen – eine Binsenweisheit, die aber gelegentlich übersehen wird. Auch "Entwicklungsphilosophien" sind der Mode unterworfen (erkennbar anhand von Zeitschriften, Kongressen, Seminaren, Lehrplänen usw.). Davon nicht beeindrucken lassen! – Aber auch nicht allzu konservativ/beschränkt bleiben: ein professioneller Entwickler sollte in der Lage und willens sein, sich, wenn es sein muß, auch zügig in ungewohnte Verfahren und Prinzipien einzuarbeiten.

grundlegende Lösungsansätze und Beschreibungsmittel	technische Lösungsansätze	Programmier-"Philosophien"
• Boolesche Gleichungen, • Kontaktpläne (× herkömmliche Steuerungstechnik), • Schaltpläne, • Zustandsgraphen (State Machines, abstrakte Automaten), • Entscheidungstabellen, • Hardware-Beschreibungssprachen, • Petri-Netze, • analoge Signalverarbeitung (× Regelungstechnik, rückgekoppelte Systeme, Laplace-Transformation), • digitale Signalverarbeitung (× z-Transformation), • neuronale Netze, • Fuzzy-Logik	• kombinatorische Zuordnung, • Table Lookup, • State Machines (abstrakte Automaten), • Sequencer (einfache Folge-, Zeitplan- und Ablauf-steuerungen), • Mikrocontroller, • universelle Prozessoren, • spezielle Prozessoren, • Multiprozessorsysteme, • hochparallele Systeme (auch: zelluläre und systolische)	• herkömmliche Programmierung (Programmablauf, Flowchart- bzw. if-then-goto-Paradigma) • strukturierte Programmierung, • ereignisgesteuerte Programmabläufe, • Multitasking, • abstrakte Datentypen, • objektorientierte Programmierung, • relationale Ansätze (Datenbasis; Memory Based Reasoning), • regelbasierte Ansätze (Expertensysteme, Prolog usw.)

Tabelle 2 Lösungsansätze im Systementwurf (Auswahl)

Auf welcher Ebene der Zuhandenheit aufsetzen?
Das ist abhängig von der Realzeitraster-Vorgabe und somit ein fließendes Ziel in Abhängigkeit vom Stand der Technik:

- < 1 ns: Basis- (Halbleiter-) Technologie,
- wenige ns: Fertigungstechnologie (zuhandene Halbleiter-Technologien + Integrationsgrad + Kombinationen (Mixed Signal, Logik + DRAM usw.),
- < 1 µs: Bauelemenetechnologie (zuhandene Schaltkreise),
- vom ms-Bereich an: zuhandene Funktionseinheiten...fertige Systemumgebungen

Kostenoptimierung = nicht zuviel Overkill für die Problemlösung = gerade soviel, um Reserven für Änderungen zu haben (den gesamten Lebenszyklus bedenken!).

Software oder Hardware?
Im strengen Sinne gibt es die Alternative nicht, da jede Software eine Hardwareplattform braucht, um laufen zu können.

Beides zusammen = Kostenoptimum unter Erfüllung der vorgegebenen Realzeitanforderungen.

Stand der Technik:

- Hohe Anforderungen an die Funktionalität.
- Programmier-Paradigma hat sich soweit durchgesetzt, daß praktisch jedem Lösungsansatz für auch nur halbwegs komplexe funktionelle Anforderungen irgendeine Form der Programmierbarkeit zugrunde gelegt wird - kaum noch jemand baut hochkomplizierte Spezialschaltungen, die nicht irgendwie programmierbar bzw. durch Programmierung steuerbar (parametrisierbar) sind.
- Es geht also in der Praxis nicht um den extremen Gegensatz: fertige Systemumgebung vs. Einzweck-Spezialschaltung, sondern um kostenoptimierte Verbundlösungen.
    - Ausnutzung zuhandener Prozessoren - Auswahl, Gestaltung der Systemumgebung, Zusatzbeschaltung, Implementierung vorteilhafter Programmier-Paradigmata (Multitasking, Ereignissteuerung, State Machine usw.), Mehrprozessorkonfigurationen.
    - Gestaltung neuartiger Prozessoren.
    - programmierbare, funktionsvariable Hardware.

Das Entscheidungsproblem (Hardware vs. Software) stellt sich in der Praxis:

- zur Erfüllung extremer Relazeitanforderungen,
- zur Erfüllung gegebener Anforderungen auf kostengünstigere Weise (Funktionalität implementieren, die bisher im gegebenen Kostenrahmen nicht implementierbar war),
- zum “entwicklungsmäßig schneller sein” (Time to Market).

Weshalb ersetzen wir Hardware durch Software?

- der herkömmliche Trend seit den 60er Jahren,
- Rückgriff auf ein höheres Niveau der Zuhandenheit,
- Zuhandenheit hochleistungsfähiger Technologien (Taktfrequenzen),
- Verkürzung der Entwicklungszeit (“es ist nur zu programmieren” = Time to Market),
- Verringerung der Hardwarekosten (Controller usw.),
- Beherrschung von Kompliziertheit (durch Zerlegen in handliche Programmstücke),
- Ersparen der Hardware-Entwicklung (zuhandene funktionstüchtige Plattformen),
- kurze Änderungszyklen,
- flexible Entwicklungswerkzeuge,
- funktionelle Flexibilität allgemein (man darf programmieren - es läßt sich alles programmieren),
- Software kann nicht ausfallen (verschleißen) -- wohl aber Fehler enthalten (und auch veralten -- wenn es keine HardwarePlatform mehr gibt, auf der sie laufen könnte).

Weshalb ersetzen wir Software durch Hardware?

- ein (ab und zu wieder) neuerer Trend,
- ermöglicht durch hochintegrierte programmierbare Schaltkreise (FPGAs, CPLDs),
- mehr Leistung,
- Kontrolle über den gesamten Lebenszyklus (nicht mehr auf Zulieferungen, fremde Standards und fremde Schutzrechte angewiesen sein),
- Kostenoptimierung = Verbilligung der Hardware-Plattform, kein Software-Overhead (besserer Wirkungsgrad),
- Lösung der Aufgabe mit weniger Silizium, Strom, EMV (langsamst-mögliche Taktierung),
- besseres Realzeitverhalten,

- kürzere Latenzzeiten,
- echter Parallelismus (von vornherein = gemäß Struktur des Problems - der dem Problem überhaupt inhärente Parallelismus kann voll ausgenutzt werden),
- unmittelbare (evidente) Vergegenständlichung der Informationswandlungen (kein Zwang zum Programmieren (= unangemessen, unübersichtlich, "semantische Lücke" zwischen Problembeschreibung und Programmablauf, zuviel Overhead).

Realisierung von Digitalschaltungen

Standardisierte elementare Schaltfunktionen (Off-the-Shelf-Schaltkreise)
Schaltungen aus einfachen Elementen von Grund auf aufzubauen ist nach wie vor notwendig: zu Anpassungszwecken, für eher einfache Aufgaben (wofür ausgewachsene Computer zu teuer sind), für Einzelanfertigungen usw. Des weiteren finden wir solche Schaltungen nach wie vor auch in Funktionseinheiten und Systemen, die in großen Stückzahlen gefertigt werden. Entsprechende Schaltkreise können gleichsam sofort aus dem Regal (Off the Shelf) entnommen werden.

Elementare digitale Schaltkreise gibt es seit den 60er Jahren; es handelt sich also gleichsam um einen gesättigten Stand der Technik. Die Hersteller fertigen die Bauelemente, die nachgefragt werden. (Deshalb werden auch bestimmte "Uralt-Typen" noch in riesigen Stückzahlen hergestellt.)

Moderne Entwicklungen betreffen:

- verringerte Versorgungsspannungen (von 3,3 V an abwärts bis hin zu Werten unter 1 V),
- die Ausrichtung auf 2 Typensortimente: (1) Bustreiber und andere Interface-Koppelstufen, (2) Grundfunktionen zum Aufbauen von sog. Restlogik (Glue Logic),
- miniaturisierte Gehäuse (Anschlußabstände z. B. 1,25; 0,625; 0,5 und 0,4 mm),

Zum Integrationsgrad
Bei den hier in Rede stehenden Schaltkreisen unterscheidet man folgende Stufen:

- SSI (Small Scale Integration): ein Schaltkreis, der nur wenige Gatterfunktionen enthält (nach Texas Instruments: maximal 12 Gatterfunktionen). Die einzelnen Gatter oder Flipflops sind direkt an die Schaltkreisanschlüsse geführt (Zusammenschaltung zu komplizierteren Funktionen außerhalb des Schaltkreises).
- MSI (Medium Scale Integration): ein Schaltkreis, der eine mittlere Anzahl an Gatterfunktionen enthält (nach Texas Instruments: zwischen 13 und 99 Gatterfunktionen). Der Schaltkreis stellt typischerweise eine kompliziertere Funktion dar (Zähler, Decoder, Multiplexer usw.).
- LSI (Large Scale Integration): damit bezeichnen wir alle noch komplizierteren Schaltkreise (von 100 Gatterfunktionen an aufwärts).

Programmierbare Schaltkreise
Auch programmierbare Schaltkreise können typischerweise aus dem Regal genommen werden (Off the Shelf). Ein solcher Schaltkreis ist an sich universell nutzbar. Er enthält keine bestimmten Funktionen, sondern vielseitig nutzbare Elementarstrukturen (Zellen, schaltbare Verbindungen usw.). Erst durch Programmieren erhält man die jeweils gewünschten Funktionen. Die Schaltkreise unterscheiden sich u. a. hinsichtlich der Auslegung, Anzahl und Anordnung der programmierbaren Strukturen (Tabelle 3, Abb. 1 bis 4) und hinsichtlich des Programmierverfahrens. Hierbei kommt es vor allem darauf an, in welcher Umgebung und wie oft der Schaltkreis programmiert werden muß bzw. kann (Tabelle 4).

Bezeichnung	Erklärung
Programmable Logic Device (PLD)	der übliche Oberbegriff für programmierbare Logikschaltkreise
Programmable Logic Array (PLA)	kombinatorische UND-ODER-Strukturen; sowohl UND als auch ODER programmierbar. Ein PLA-Schaltkreis enthält eine bestimmte Anzahl von UND-Gattern mit programmierbaren Eingängen sowie ODER-Gatter, die den UND-Gattern nachgeschaltet werden können. Diese Verbindungen sind ebenfalls programmierbar. Veraltet
Programmable Array Logic (PAL)	kombinatorische UND-ODER-Strukturen; UND programmierbar, ODER fest verschaltet. Ein PAL-Schaltkreis enthält eine bestimmte Anzahl von UND-Gattern mit programmierbaren Eingängen. Die UND-Gatter sind fest mit ODER-Gattern verbunden. PAL-Schaltkreise werden zumeist mittels Durchschmelzverfahren (Fusible Link) programmiert
Generic Array Logic (GAL)	elektrisch programmier- und löschbare, mit universell nutzbaren Flipflops erweiterte PAL-Strukturen (Makrozellen)
EPLD	bezeichnet üblicherweise eine elektrisch (mehrmals) programmierbare und durch UV-Licht löschbare Schaltung
Complex Programmable Logic Device (CPLD)	üblicherweise faßt man unter diesem Begriff all das zusammen, was komplexer ist als eine einfache UND-ODER-Makrozellen-Struktur, aber nicht so komplex wie ein FPGA. (Solche Schaltkreise enthalten, , (1) komplexere Makrozellen (verglichen mit denen der GALs) und (2) Koppelnetzwerke zum Verbinden der Zellen untereinander.)*)
Field Programmable Gate Array (FPGA)	Sammelbegriff für Schaltkreise, die es ermöglichen, eine Vielzahl programmierbarer Zellen freizügig untereinander zu verbinden, so daß - wenigstens näherungsweise - ein ähnlicher Grad der Schaltungsintegration erreicht werden kann wie bei Nutzung kundenspezifischer Gate-Array-Schaltkreise*)

*): ein CPLD hat vergleichweise wenige komplexe, ein FPGA vergleichsweise viele einfache Zellen

Tabelle 3 Programmierbare Schaltkreise

Programmier-verfahren	Ausführung	Änderbarkeit	Bemerkungen
Maskenprogram-mierung	beim Halbleiterher-steller	im einzelnen Schaltkreis nicht mehr änderbar	nur bei extremen Stückzahlen von praktischer Bedeutung
Durchschmelzprin-zip (Fuse)	beim Anwender*)	im einzelnen Schaltkreis praktisch nicht mehr änderbar	alle Verbindungen sind vor-gefertigt, die nicht benötigten werden beim Programmieren getrennt
Aufschmelzprinzip (Antifuse)	beim Anwender*)	im einzelnen Schaltkreis praktisch nicht mehr änderbar	alle Verbindungsstellen sind zunächst getrennt, benötigte Verbindungen werden beim Programmieren hergestellt

Programmier-verfahren	Ausführung	Änderbarkeit	Bemerkungen
Ladungsspeiche-rung mit UV- Löschung	beim Anwender*)	durch Löschen und Neuprogrammieren	Löschen durch UV-Licht; erfordert Quarzglasfenster im Schaltkreis (es gibt auch preisgünstige Ausführungen ohne Fenster; diese kann man nicht mehr löschen)**)
Ladungsspeiche-rung mit elektri-scher Lö-schung (EEPROM, Flash)	beim Anwender*)	durch Löschen und Neuprogrammieren (auch: in der Anwendungsschaltung (In System Programming)	Löschen durch elektrische Impulse
RAM-Zellen	beim Anwender*) bzw. während des Betriebs	durch Umladen; auch während des normalen Betriebs beliebig oft mög-lich	Halten der Information in Flipflops bzw. Latches; nach jedem Einschalten ist erneu-tes Laden erforderlich

*): Anwender = Hersteller des Gerätes bzw. der Funktionseinheit; **) Bezeichnung: OTP (One Time Programmable)

Tabelle 4 Programmierverfahren (Übersicht)

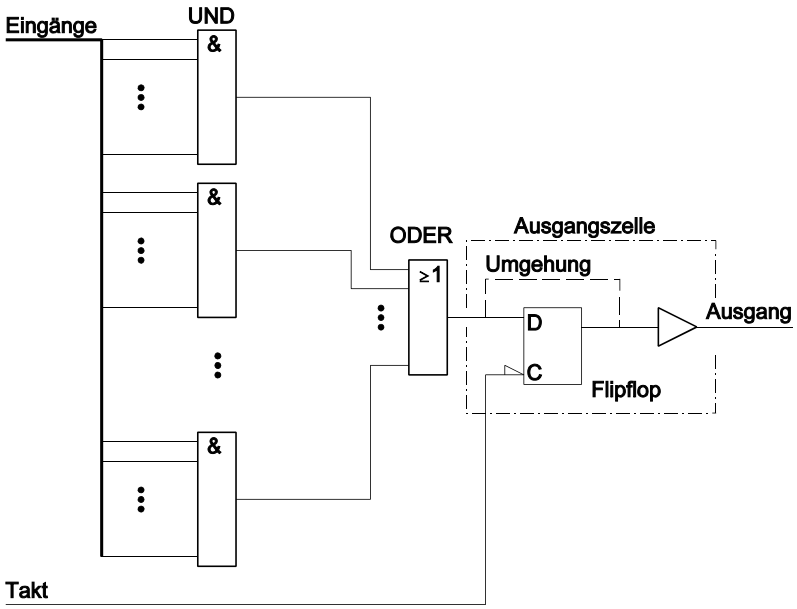


Abb. 1 Eine typische Makrozelle – eine dicke UND-ODER-Struktur mit nachgeschaltetem Flipflop. Viele Eingänge (etwa 20... über 50), die in mehreren UND-Gattern konjunktiv verknüpft werden können

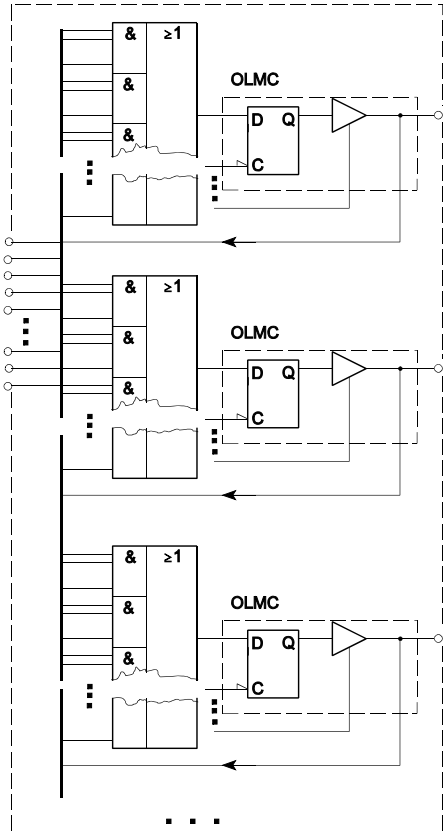


Abb. 2 Mehrere von außen zugängliche Makrozellen bilden einen GAL-Schaltkreis

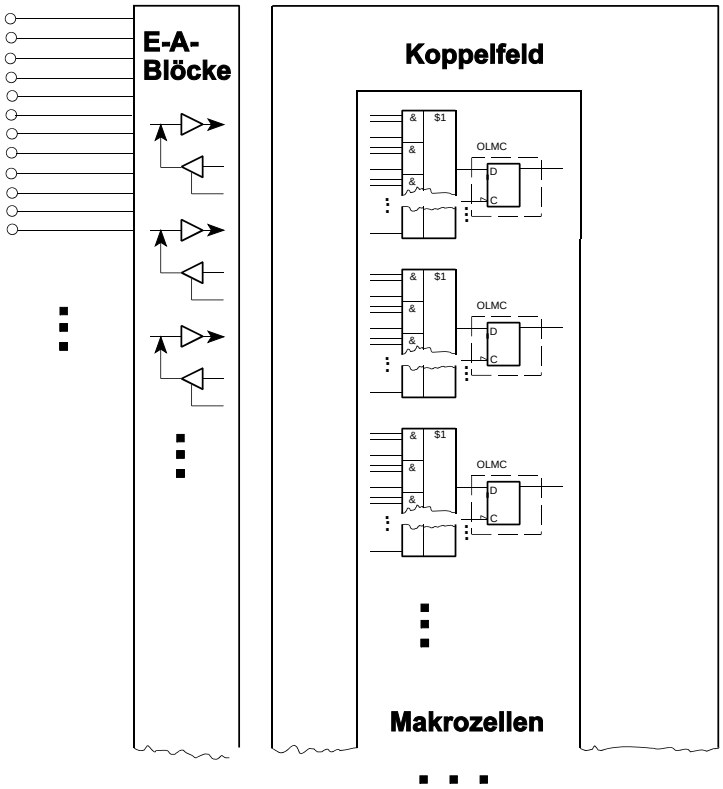


Abb. 3 CPLDs haben viele Makrozellen, die im Innern des Schaltkreises untereinander verbunden

werden können

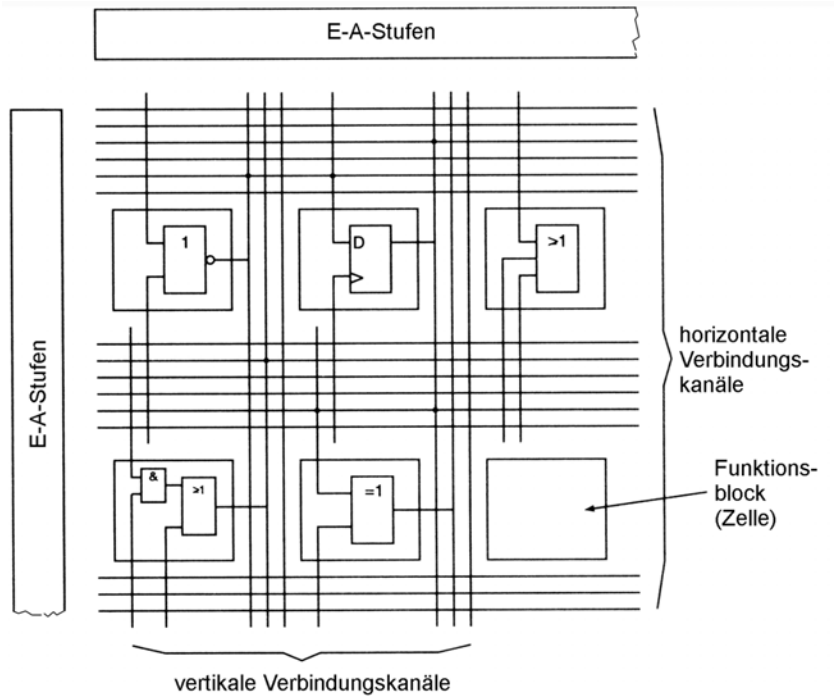


Abb. 4 FPGAs enthalten viele kleine Zellen, die in ein Netzwerk programmierbarer Verbindungskanäle eingebettet sind. Eine FPGA-Zelle entspricht einer kombinatorischen Verknüpfung von wenigen Eingängen (z. B. 4 bis 8) und ggf. einem nachgeschalteten Flipflop

Taktfrequenz	Technologie	Entwurfsregeln
bis ca. 8...10 MHz*)	(noch) Zweiebenen-Leiterplatten	gesunder Menschenverstand**), Faustformeln
bis ca. 33 MHz	Mehrebenen-Leiterplatten	Faustformeln noch anwendbar, Leiterplattenentwicklung (Placieren, Routen) noch auf empirischer Grundlage möglich
> 33 MHz	Mehrebenen-Leiterplatten	Faustformeln nicht mehr ausreichend, statt dessen Analogsimulation von Bauelementen und Leiterplatte auf Grundlage möglichst genauer Ersatzschaltungen

*): bei mäßiger Flankensteilheit und Treibfähigkeit (bei steilen Flanken und hoher Treibfähigkeit gilt die nächste Zeile); **): mit brauchbaren Kenntnissen der Impulstechnik angereichert – Impulse sind nicht einfach zerhackter Gleichstrom ...

Tabelle 5 Realisierungsprobleme in Abhängigkeit von der Taktfrequenz

Praxisübliche Projekte

Wir reden grundsätzlich über vergleichsweise kleine Vorhaben. Zumeist wird man alles, was irgendwie kompliziert oder aufwendig aussieht, einer passenden Computer-Plattform (vom Mikrocontroller bis zum Industrie-PC) überlassen und dort mit Software erledigen. Bei der Bauelementeauswahl sind heutzutage nicht nur die Kosten von Bedeutung, sondern auch Fragen der praktischen Handhabbarkeit:

- wie aufwendig ist ggf. der rechnergestützte Schaltungsentwurf (Software, Computer, Programmiergeräte usw.)?
- kann man mit einfachen Mitteln einen Laboraufbau zustande bringen?
- kann man die Leiterplatte mit gleichsam hausbackenen Mitteln entwerfen?[1)]
- läßt sich die Leiterplatte bei jedem x-beliebigen Dienstleister preisgünstig fertigen?
- passen Speisespannung und Signalpegel zum Rest des Systems oder sind in dieser Hinsicht Sonderaufwendungen erforderlich (z. B. zur Pegelwandlung)?

Es geht also nicht nur um die eigentliche Funktion, sondern auch um Gehäusebauformen, um Anschlußabstände und Lötverfahren, um die Ebenenzahl der Leiterplatte, um Speisespannung und Signalpegel (Stichworte: 5-V-Toleranz, 3,3-V-Toleranz) sowie um die anzuwendenden Entwurfs- und Programmierverfahren. Unter diesen Gesichtspunkten wird es sich ab und zu herausstellen, daß mancher – eigentlich sehr attraktive – Schaltkreis für unsere Zwecke nicht in Frage kommt (das betrifft vor allem ganz moderne Typen, die vor allem zum Einsatz in Mobiltelefonen u. dergl. entwickelt wurden).

Herkömmliche Schaltkreise (Off the Shelf)
Naheliegend, wenn es um kleinere Vorhaben geht. Sind aber 20 Stück[2)] und mehr erforderlich, sollte man sich nach Alternativen umsehen.

Programmierbare Logikschaltkreise
Das Mittel der Wahl für alle auch nur halbwegs komplizierten und/oder umfangreichen Digitalschaltungen (Richtwert: zu erwägen, wenn der Entwurf ansonsten mehr als 5 DIL-Gehäuse (TTL/CMOS) erfordern würde). Der Einstieg erfordert aber einigen Aufwand, nämlich eine passende Entwicklungssoftware, einen hinreichend leistungsfähigen PC und ggf. ein Programmiergerät.

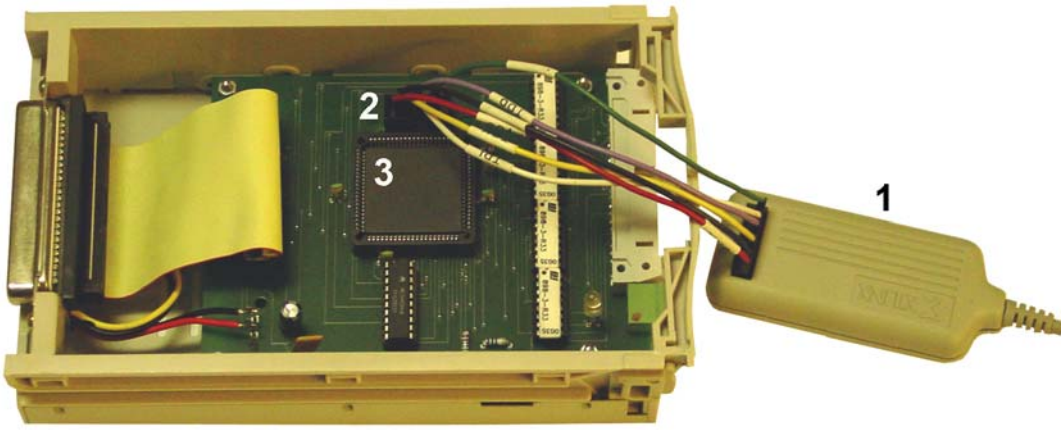
Praxistip:
Eine CPLD- oder FPGA-Familie mit Flash-Programmierung, die in der Anwendungsschaltung programmiert werden kann (In-System Programming (ISP)). Die Programmiervorkehrungen beschränken sich darauf, die zum Programmieren erforderlichen Signale auf geeignete Anschlüsse zu führen (Abb. 5). Die Preise der Entwicklungsumgebungen bzw. Starterkits liegen in erschwinglicher Größenordnung (Mindestausstattung: Entwurfssoftware + Download-Kabel). Einige Hersteller bieten kostenlose Entwurfssoftware an. Die einfachste Form der Entwurfseingabe: über Schaltplan. Typische Vorteile:

- viele Projekte kommen mit einem einzigen Schaltkreis aus (vgl. Abb. 5),
- man kann immer wieder ändern (= löschen und neu programmieren),
- man kann oft gleichsam naiv entwerfen, ohne sich um die Schaltungsoptimierung, um das Heraussuchen passender 74er oder 4000er Schaltkreise, um die Aufteilung der Funktionen auf die Schaltkreise usw. kümmern zu müssen (mit den DeMorganschen Regeln, mit Karnaugh-Plänen usw. müssen wir uns nicht mehr abgeben – das erledigt die Entwurfssoftware).

---

1): soll heißen: Placieren und Entflechten nach gängigen Faustregeln und gesundem Menschenverstand.

2): Richtwert. Entspricht typischerweise einer mit DIL-Gehäusen voll bestückten Europakarte 100 ▯ 160 mm.



1 - Download-Kabel. Z. B. an die Parallelschnittstelle des Entwicklungs-PCs angeschlossen. 2 - Programmier-Steckverbinder auf der Leiterplatte; 3 - der zu programmierende Schaltkreis (enthält einen Interfaceadapter mit 4 universellen E-A-Ports).

Abb. 5 Ein CPLD-Schaltkreis wird programmiert

IP: fertige Entwürfe nutzen
IP = Intellectual Property. Es werden Schaltungsentwürfe angeboten, die man in eigene CPLDs und FPGA einbringen kann. Das ist zum einen eine kommerzielle Angelegenheit (kostet richtig Geld). Zum anderen gibt es eine „freie" bzw. „offene" Szene. Von derartigen Schaltungen kann man aber keineswegs erwarten, daß sie unter allen Umständen ohne eigenes Zutun auf Anhieb funktionieren – es kommt u. a. darauf an, für welche Schaltkreistypen der Entwurf ursprünglich vorgesehen war, in welcher Form (VHDL, Verilog, Netzliste, Programmierdaten usw.) er übergeben wird und welche weitere Unterstützung (Dokumentation, Testdaten usw.) es gibt.

Mikrocontroller oder CPLD?
Mikrocontroller sind gelegentlich eine Alternative zum programmierbaren Logikschaltkreis. Ein kleiner CPLD-Schaltkreis mit typischerweise 32...36 Makrozellen (= Flipflops) kostet ungefähr dasselbe wie ein PIC oder AVR oder 8051 usw. mit vergleichbarer Anzahl an Signalanschlüssen. Man hat also gelegentlich die Qual der Wahl. Ein naheliegendes Entscheidungskriterium ist das Realzeitraster an den zu steuernden Schnittstellen:

- wenn es auf Mikrosekunden oder gar Nanosekunden ankommt: CPLD,
- wenn die Millisekunde kaum eine Rolle spielt: Mikrocontroller.

Hinweise:

1. Der Mikrocontroller bietet mehr fürs Geld, nämlich nahezu unbeschränkte Funktionsvielfalt (freie Programmierbarkeit), die CPLD hingegen nur das, was sich mit den wenigen Flipflops und einigen hundert Gattern realisieren läßt. Zudem haben viele Mikrocontroller eingebaute Schnittstellen (die man andernfalls selbst entwerfen müßte).
2. Die CPLD ist viel schneller. 100 MHz Takt heißt 10 ns Reaktionszeit vom Eingang zum Ausgang. Ein 100-MHz-Mikrocontroller hätte da bestenfalls einen einzigen Befehl ausgeführt – und der leistet nicht eben viel ...
3. Extrem schnelle Mikrocontroller (von 25 MHz an aufwärts) werden angeboten. Sie sind deutlich teurer als die Massenfabrikate (mit typischerweise 5...20 MHz). Trotz der hohen Taktfrequenz sind die Reaktionszeiten in nichttrivialen Anwendungen enttäuschend, ja manchmal geradezu lausig.