Entwicklung logischer Schaltungen mit GALs

Die GALs 16V8 und 20V8 ersetzen Stadard-Logikbausteine, ideal für Entwicklung und Kleinserien

Franz Fiala DSK-361,362; LIT-95,97,98

Aufbauend auf Kenntnisse über grundlegende Schaltungstechnik mit logischen Schaltelementen, können Sie mit den folgenden Hinweisen selbst eine Logische Schaltung konstruieren, minimieren und in einem GAL-Baustein programmieren.

Dieser Streifzug durch die Vorgangsweise bei der Entwicklung logischer Schaltkreise stammt aus einer Laborübung, die mit einer Kolleg-Klasse in den zwei Teilen **kombinatorische Logik** und **sequentielle Logik** durchgeführt wurde. Zu jedem Teil wählt die Gruppe eine Aufgabe. Die Durchführung beziehungsweise Ausarbeitung unfaßt:

- den Aufbau der Schaltung sowohl in traditioneller TTL- oder CMOS-Logik als auch mit einem programmierbaren Baustein GAL20V8,
- die Minimierung der Wahrheitstabelle mit den Rechenregeln und mit Karnaugh-Diagramm,
- die Programmierung des GAL-Bausteins, sowohl direkt im JEDEC-File als auch mit dem Logik-Compiler ABEL.

Die vorliegende Arbeitsunterlage enthält alle Hinweise für die Vorgangsweise.

Weiters wird benötigt:

- Datenbücher TTL/CMOS
- Datenblatt GAL20V8
- 2 HD-Disketten der Shareware-Version von ABEL. DSK 361,362
- Dokumentation zu ABEL (on-line und ausgedruckt als LIT-94)
- Universal-Programmiergerät für GALs

Man benötigt einen Experimentiertkasten zum Aufbau der Gatterlogik, ein GAL 20V8 sowie einen Programmer zum Erstellen des programmierten GAL. Es wird der Programmer ALL02 verwendet.

Weiterführende Unterlagen

- Programmierbare Logik und ihre Entwicklung, PI-Seminar, LIT-94, Balog, Prasky, Thorwartl
- Systematisierung der Programmieralgorithmen für GAL-Bausteine und Aufbau eines GAL-Programmiergeräts, LIT-95, Pöschko

Bei der Übungsvorbereitung kann die gesamte Aufgabenstellung theoretisch und am Logik-Compiler im voraus erarbeitet werden. Bei der Übungsdurchführung wird die Gatterlogik aufgebaut und an Hand der Wahrheitstabelle überprüft. Beim GAL wird das JEDEC-File editiert und das GAL programmiert und ebenso wie bei der Gatterlogik getestet.

Für das Testen der sequentiellen Logik wird ein Logikanalysator verwendet, der bei der Übung erklärt wird. Die Ausarbeitung enthält für Kombinatorische Logik und Sequentielle Logik je:

- Aufgabenstellung
- Minimierung mit Rechenregeln
- Minimierung mir Karnaugh
- handeditiertes JEDEC-File
- Dokumentation als Ausgabe des ABEL-Kompilers
- JEDEC-File, Simulation

Kombinatorische Logik

Von der Idee zur Wahrheitstabelle

Diesen Schritt kann man dem Konstrukteur im allgemeinen nicht abnehmen. Die Aufgabe aber, aus der Wahrheitstabelle eine fehlerfreie Schaltung abzuleiten, wird mit vielen Methoden und Hilfsmitteln unterstützt. Wir kennen:

- Minimierung mit Rechenregeln
- Minimierung mit Karnaugh-Schema
- Programmierbare Logik mit Handprogrammierung
- Programmierbare Logik mit Logik-Compiler

Von der Wahrheitstabelle zur Realisierung

Das nachfolgende Beispiel für die kombinatorische Logik wird in allen Varianten konstruiert. Die Aufgabe der Gruppe ist ebenso auszuführen. Es gibt 4 Eingangsvariablen A, B, C, D und eine Ausgangsvariable X, um das Beispiel auch in Gatterlogik mit vertretbarem Aufwand realisieren zu können.

	D	C	B	A	X
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

Logische Gleichung

2 Alternativen:

a. Disjunktion (ODER) aller Minterme (Minterm: ergibt für eine einzige Eingangskombination 1)

b. Konjunktion (UND) aller Maxterme (Maxterm: ergibt für eine einzige Eingangskombination 0)

```
X = /D &  C & /B & /A  +  /D &  C &  B & /A +
     D & /C &  B & /A  +   D & /C &  B &  A +
     D &  C &  B & /A  +   D &  C &  B &  A
```

Bei weniger 1en als 0en sind Minterme ein vorteilhafterer Ansatz, bei mehr 1en als 0en könnte man man Maxterme wählen (ist aber selten). Wie man auch beginnt, nach der Minimierung ergeben beide Ansätze dasselbe vereinfachte Ergebnis mit jeweils etwas unterschiedlichem Rechenaufwand.

Diese Gleichung kann - so wie sie ist - mit Standard-Logikbausteinen aufgebaut werden. Man benötigt eine ODER-Verknüpfung mit 6 Eingängen, 6 UND-Verknüpfungen mit 4 Eingängen sowie 4 Inverter. Mit Standardbausteinen ein erheblicher Aufwand, daher versucht man durch Anwendung der Rechenregeln einfachere Formen zu finden.

Minimierung mit Rechenregeln

Mit den Rechenregeln für logische Gleichungen kann die Anzahl der benötigten Bauelemente reduziert werden. Man hebt aus ähnlich aufgebauten Ausdrücken die gleichbleibenden Terme heraus und kann die verbleibenden Klammern wegen X + /X = 1 und X & /X = 0 kürzen.

```
X = /D &  C & /A & (/B + B) +
    D & /C &  B & (/A + A) +
    D &  C &  B & (/A + A) =

  = /D &  C & /A +  D & B & (C + /C) =

  = /D &  C & /A +  D & B
```

Jetzt benötigt man nur mehr eine UND-Verknüpfung mit 2 und eine mit 3 Eingängen, ein ODER-Gatter mit 2 Eingängen und 2 Inverter.

Umwandlung in verfügbare Schaltkreise

Üblicherweise bleiben bei Konstruktionen Gatter über, die durch geeignete Umformung der Gleichungen verwendbar werden. Typische Beispiele sind Umformungen auf NAND- oder NOR-Logik.

Dabei benutzt man am häufigsten den **Satz von de-Morgan**, der besagt, daß eine Funktion unverändert bleibt, wenn man die Ein- und Ausgangsgrößen negiert und die Rechenoperationen umkehrt.

Beispiel:

```
I =  J & K = /(/J + /K)
I =  J + K = /(/J & /K)
```

Wenn wir das auf unsere Funktion anwenden:

```
X = /D &  C & /A +  D & B =
  = /(/( /D &  C & /A ) & /(D & B))
```

Jetzt besteht die Funktion lediglich aus NAND-Schaltkreisen.

Minimierung mit Karnaugh-Schema

Da die Minimierung mit den Rechenregeln fehleranfällig und prinzipiell unbeliebt ist, kam man auf die Idee, die Vereinfachung in einem grafischen Schema zu ermöglichen, welches implizit die Rechenregeln widerspiegelt. Dabei wird die Ausgangsvariable in einer quadratischen Anordnung in einer besonderen Reihenfolge angeschrieben. Eine Vereinfachung erkennt man an der Nachbarschaft von Einsern in diesem Schema, wobei auch Randeinser als Nachbarn gelten, wenn man sich das Schema als Kugel vorstellt.

```
                      <       B       >
  ^ +-------+-------+-------+-------+
    |8      |9      |11     |10     |
    |       |       |       |       |
    |   1000|   1001|   1011|   1010|
  D +-------+-------+-------+-------+ ^
    |12     |13     |15     |14     |
    |       |       |       |       |
    |   1100|   1101|   1111|   1110|
  v +-------+-------+-------+-------+ C
    |4      |5      |7      |6      |
    |       |       |       |       |
    |   0100|   0101|   0111|   0110|
    +-------+-------+-------+-------+ v
    |0      |1      |3      |2      |
    |       |       |       |       |
    |   0000|   0001|   0011|   0010|
    +-------+-------+-------+-------+
            <       A       >
```

In den angegebenen Bereichen werden die betreffenden Variablen 1. Zusammenhängende Felder von Einsen werden durch UND-Verknüpfungen realisiert.

```
                          B
    +-------+-------+-------+-------+
    |       |       |+-------------+|
    |   0   |   0   ||  1   |   1  ||
    |       |       ||      |      ||
  D +-------+-------+|------+------|+
    |       |       ||      |      ||
    |   0   |   0   ||  1   |   1  ||
    |       |       |+-------------+|
    +-------+-------+-------+-------+ C
   -------+ |       |       | +-------
    |   1 | |   0   |   0   | | 1   |
   -------+ |       |       | +-------
    +-------+-------+-------+-------+
    |       |       |       |       |
    |   0   |   0   |   0   |   0   |
    |       |       |       |       |
    +-------+-------+-------+-------+
                    A
```

Die zugehörige vereinfachte Gleichung kann sofort angeschrieben werden:

```
X = B & D   +   C & /A & /D
```

Dieses Schema funktioniert auf diese Weise am besten für 2, 3 oder 4 Variablen; für mehr Variablen ist es unübersichtlich, da man mehrere Quadrate zusammensetzen muß.

PLD (Programmable Logic Device)

Eine Programmierbare Logik versucht erst gar nicht mit der Minimierung zu beginnen (wenn die Logic-Compiler moderner Bauart das auch tun), sondern stellt von vornherein alle möglichen UND-Verknüpfungen der invertierten oder nicht-invertierten Eingangsvariablen zur Verfügung und verknüpft alle in einem ODER. Welche wirklich zum Zug kommen, bestimmt ein Programmiervorgang, der einmalig (maskenprogrammierbar oder kundenprogrammierbar) oder aber auch wiederholt (elektrisch löschbar) durchführbar ist.

PLA	Programmable Logic Array	maskenprogrammierbare
FPLA	Field Programmable Logic Array	kundenprogrammierbare

Da die Programmierung von PLAs ziemlich aufwendig ist, wurden vereinfachte Schaltungen PAL (Programmable Logic Array) geschaffen, deren ODER-Verknüpfung fest verdrahtet ist und nur die UND-Verknüpfungen programmierbar sind.

Die Grundschaltung ist:

```
          |  |  |  |  |  |...
x0  --+---x--|--|--x--|--|---    Jedes x ist eine UND-Verknüpfung
x0/   o---|--x--|--|--|--|---    Hier wurde folgende Gleichung
x1  --+---|--|--|--|--|--|---    realisiert:
x1/   o---x--x--|--|--|--|---    Y =  x0 & /x1 & /x2 & /x3 +
x2  --+---|--x--|--x--|--|---        /x0 & /x1 &  x2 &  x3 +
x2/   o---x--|--|--|--|--|---         x0       &  x2       +
x3  --+---|--x--|--|--|--|---                          /x3
x3/   o---x--|--|--|--x--|---
...       |  |  |  |  |  |
        +-+--+--+--+--+--+-+
        |        >=1       |
        +---------+--------+
                  | Y
```

Die Art des Ausgangs und die Anzahl der möglichen Eingänge bedingt eine Vielzahl von Schaltkreisen. Bei den Ausgängen unterscheidet man:

- Einfache Ausgänge,
- Ausgänge mit Registern,
- Bidirektionale Ausgänge und
- Tri-State Ausgänge

Diese Vielfalt an Bausteinen bedingt auch gewisse Probleme in der Lagerhaltung und Besorgung, sodaß Schaltkreise mit größtmöglicher Zahl von Möglichkeiten geschaffen wurden:

GAL Generic Array Logic

Ein GAL vereinigt die Möglichkeiten der meisten PALs bezüglich der Ausgangsstufe in einer sogenannten OLMC-Ausganszelle. Darüberhinaus lassen sich GALs auch wieder löschen, wodurch der Einsatz in Entwicklungslabors sehr praktisch ist. Da sie kompatibel mit den jeweiligen PAL-Familien sind, kann man während der Entwicklung die teureren GALs verwenden und im endgültigen Aufbau die billigeren PALs.

Ursprünglich wurden GALs vom Erzeuger Lattice hergestellt, sind aber jetzt von mehreren Herstellern erhältlich. Ein wichtiger Vorteil ist die elektrische Löschbarkeit des Inhalts, wodurch ein GAL ein ähnlich universeller Baustein ist wie ein EPROM. Durch die elektrische Löschbarkeit ist ein Programmierzyklus sogar kürzer.

GAL 20V8

20 ist die Anzahl der Eingänge zur der logischen Matrix; 8 ist die Anzahl der Ausgänge. Jede Ausgangsleitung wird durch eine OLMC-Zelle getrieben.

OLMC Output Logic Macro Cell

Die OLMC-Zelle ist programmierbar. Es können alle herkömmlichen PALs simuliert werden.

Wie würde nun unsere Funktion in einem GAL programmiert werden? Man beläßt sie am einfachsten in ihrem ursprünglichen Zustand, der sich durch die Wahrheitstabelle ergibt.

Wir benutzen nur einen einzigen Ausgang und vier Eingänge. Mit den Bits SYN und AC0 werden alle OLMCs gleichermaßen beeinflußt. Mit SYN=1 gibt es nur kombinatorische Ausgänge, mit SYN=0 gibt es nur Registerausgänge. SYN bestimmt, ob Pins 1 und 13 als Takteingänge oder Steuereingänge für den Tristate-Zustand verwendet werden (SYN=0) oder ob es gewöhnliche Eingänge sind (SYN=1). AC0 schaltet zwischen dem „Small-PAL“ (AC0=0) und dem „Medium/Registered-PAL“-Mode (AC0=1) um.

Im „Small-PAL“-Mode bestimmt das Bit AC1 jeder Zelle, ob sie sich als Register oder als einfacher Ausgang verhält. Das zusätzliche Bit AC1, das in jeder einzelnen Zelle definiert ist, bestimmt mit AC1=0, daß das Register verwendet wird und mit AC1=0, daß der Ausgang der eines gewöhnlichen Logikbausteins ist.

Im „Medium-PAL“-Mode müssen alle Bits AC1 auf 1 gesetzt sein.

Das XOR-Bit einer jeden OLMC-Zelle bestimmt, ob der Ausgang invertiert oder nicht-invertiert gebildet wird.

Die Bits PTD0..PTD7 bestimmen, ob die Ausgänge des Logischen Array in der ODER-Verknüpfung wirksam werden.

Die Numerierung dieses GAL erfolgt so:

Die logische Matrix besteht aus 40 Spalten (20 normale und 20 invertierte Eingänge) und 64 Zeilen. Die 64 Zeilen bilden 8 x 8 Eingänge einer ODER-Verknüpfung am Eingang einer OLMC-Zelle. Die jeweils erste Leitung wird entweder genauso behandelt wie die anderen (AC0.AC1=0) oder dient als Steuerung für den Tristate-Zustand der Ausgangszelle.

Die Spalten und ihre Bedeutung

```
Spalte 0  << Pin 2
Spalte 1  << Pin 2/
Spalte 2  << SYN=1: Pin1    SYN=0: Pin 23
Spalte 3  << SYN=1: Pin1/   SYN=0: Pin 23/
Spalte 4  << Pin 3
Spalte 5  << Pin 3/
Spalte 6  << vereinfacht: Pin 22
Spalte 7  << vereinfacht: Pin 22/
Spalte 8  << Pin 4
Spalte 9  << Pin 4/
Spalte 10 << vereinfacht: Pin 21
Spalte 11 << vereinfacht: Pin 21/
Spalte 12 << Pin 5
Spalte 13 << Pin 5/
...
```

Numerierung

Die Numerierung des Array beginnt links oben mit der Zelle 0000, jede Zeile entspricht den 40 Kreuzungspunkten im Array. Dieses Darstellungsformat heißt JEDEC. Man kann es in voller (alle Zellen) oder abgekürzter Schreibweise (nur jene Zellen, die man in einem bestimmten Zustand braucht) verwenden. Bei Vollschreibweise genügt die einmalige Angabe der Startnummer (L0000) gefolgt von den zu programmierenden Bits, gefolgt von einem "*". Die abgekürzte Schreibweise beendet einen zu programmierenden Bereich mit einem "*", um dann mit einer neuen Adreßangabe zu beginnen.

```
      00000000001111111111222222222233333333333
      01234567890123456789012345678901234567890

L0000 11111111111111111111111111111111111111* << ODER00
L0040 11111111111111111111111111111111111111* << ODER01
L0080 11111111111111111111111111111111111111* << ODER02
L0120 11111111111111111111111111111111111111* << ODER03
L0160 11111111111111111111111111111111111111* << ODER04
L0200 11111111111111111111111111111111111111* << ODER05
L0240 11111111111111111111111111111111111111* << ODER06
L0280 11111111111111111111111111111111111111* << ODER07
L0320 11111111111111111111111111111111111111* << ODER10
L0360 11111111111111111111111111111111111111* << ODER11
L0400 11111111111111111111111111111111111111* << ODER12
L0440 11111111111111111111111111111111111111* << ODER13
L0480 11111111111111111111111111111111111111* << ODER14
L0520 11111111111111111111111111111111111111* << ODER15
L0560 11111111111111111111111111111111111111* << ODER16
L0600 11111111111111111111111111111111111111* << ODER17
....
L2480 11111111111111111111111111111111111111* << ODER77
L2520 11111111111111111111111111111111111111* << Ende logisches Array
L2560 11111111*                               << XOR pro Zelle
L2568 11111111*                               << \ ab hier eine
L2576 11111111*                               <<  \ beliebige Signatur
L2584 11111111*                               <<   \ definierbar
L2592 11111111*                               <<    \
L2600 11111111*                               <<     \
L2608 11111111*                               <<      \
L2616 11111111*                               <<       \
L2624 11111111*                               <<        \
L2632 11111111*                               << Bit AC1 pro Zelle
L2640 11111111*                               << PTD für Zelle 0
L2648 11111111*                               << PTD für Zelle 1
L2656 11111111*                               << PTD für Zelle 2
L2664 11111111*                               << PTD für Zelle 3
L2672 11111111*                               << PTD für Zelle 4
L2680 11111111*                               << PTD für Zelle 5
L2688 11111111*                               << PTD für Zelle 6
L2696 11111111*                               << PTD für Zelle 7
L2704 1*                                      << SYN
L2705 1*                                      << AC0
```

Programmiertabelle eines GAL

Als ersten Schritt wollen wir festlegen, welche Betriebsart das GAL grundsätzlich hat:

Wir wählen den „Small-PAL“-Mode, da wir keine Tristate-Funktion brauchen: SYN=1, AC0=0

Alle Ausgänge sind kombinatorisch und immer aktiv.

Eingänge:

A Pin 2

B Pin 3

C Pin 4

D Pin 5

Ausgang X Pin 22

Da wir wollen, daß die erste OLMC-Zelle als Ausgang fungiert, wählen wir AC10=0, daher ist Pin 22 ein Ausgang.

Unsere Gleichung lautet:

```
X = /D &  C & /B & /A  +  /D &  C &  B & /A +
     D & /C &  B & /A  +   D & /C &  B &  A +
     D &  C &  B & /A  +   D &  C &  B &  A
```

```
     A   B   C   D
     2/  3/  4/  5/
     00000000001111111111222222222233333333333
     01234567890123456789012345678901234567890
L0000 10111011011110111111111111111111111111* << ODER00
L0040 10110111011110111111111111111111111111* << ODER01
L0080 10110111101101111111111111111111111111* << ODER02
L0120 01110111101101111111111111111111111111* << ODER03
L0160 10110111011101111111111111111111111111* << ODER04
L0200 01110111011101111111111111111111111111* << ODER05
L2560 01111111*                               << XOR pro Zelle
L2632 01111111*                               << Bit AC1 pro Zelle,
                                              << AC10=0
L2640 11111100*                               << PTD für Zelle 0
L2704 1*                                      << SYN
L2705 0*                                      << AC0
```

Je nach Bauart des Programmers kann auch diese abgekürzte Schreibweise verwendet werden; sie enthält nur die relevanten Zeilen, die nicht angegebenen Adressen bleiben in ihrem ursprünglichen Zustand. Wenn es der Programmer nicht zuläßt, verwendet man das etwas längere Voll-Format, das mit L0000 beginnt und erst nach der Adresse L2705 mit "*" endet.

Da diese Art der Programmierung sehr unübersichtlich zu werden droht, gibt es vereinfachte programmunterstützte Verfahren, um das Programmieren von PALs zu erleichtern.

Programmierung mit dem Logik-Compiler

Wir verwenden für diese Laborübung das Shareware-Programm EASY-ABEL. Die Sprache ist sehr einfach zu erlernen indem man sein Problem mit vorhandenen Beispielen vergleicht. Das obige Problem würde man in der ABEL-Sprache so formulieren:

Jedes ABEL-Programm ist ein Modul, der mit einem entsprechenden Namen eingeleitet und beendet wird:

```
module uebung1
....
end uebung1
```

Ein Titel, der auch in der generierten DOC-Datei enthalten ist, kennzeichnet den Inhalt dieses Programms:

```
title 'Laborvorbesprechung kombinatorische Logik
       Franz Fiala, 12.12.1993.'
```

Der Titel beginnt mit dem Zeichen <'> und endet auch damit.

Je nachdem, ob man die Auswahl des Logikbausteins selbst vornimmt oder diese Auswahl durch das Programm bestimmen läßt, folgt die Bezeichnung des Bausteins mit seiner zukünftigen Bezeichnung in der Schaltung:

```
IC1   device  'P20V8';
```

Achten Sie darauf, jede Zeile mit einem Strichpunkt abzuschließen! (Aber der Compiler ahndet ohnehin jede Verletzung der Regeln). Danach legt man fest, welche Signale auf welchen Pins zu suchen sind:

```
A,B,C,D pin 2,3,4,5;
X pin 22;
```

Die Pinbezeichnungen können durch Beistriche getrennt oder mit einem Pin pro Zeile angegeben werden.

Es folgen die logischen Gleichungen, die man zu programmieren wünscht (ODER #, UND &):

```
equations

X = (!D &  C & !B & !A) # (!D &  C &  B & !A) #
    ( D & !C &  B & !A) # ( D & !C &  B &  A) #
    ( D &  C &  B & !A) # ( D &  C &  B &  A);
```

Wenn man die Vorrangregeln nicht kennt, sollte man die Ausdrücke in Klammern setzen.

Es ist auch möglich, Testfälle `test_vectors` anzugeben, für die diese Gleichung gelten soll:

```
test_vectors  'X'
              ( [D,C,B,A] -> X )
                [ 0, 0, 0, 0] ->  0;
                [ 0, 0, 0, 1] ->  0;
                [ 0, 0, 1, 0] ->  0;
                [ 0, 0, 1, 1] ->  0;
                [ 0, 1, 0, 0] ->  1;
                [ 0, 1, 0, 1] ->  0;
                [ 0, 1, 1, 0] ->  1;
                [ 0, 1, 1, 1] ->  0;
                [ 1, 0, 0, 0] ->  0;
                [ 1, 0, 0, 1] ->  0;
                [ 1, 0, 1, 0] ->  1;
                [ 1, 0, 1, 1] ->  1;
                [ 1, 1, 0, 0] ->  0;
                [ 1, 1, 0, 1] ->  0;
                [ 1, 1, 1, 0] ->  1;
                [ 1, 1, 1, 1] ->  1;
```

Diese Datei erhält den Namen `UEBUNG1.ABL`. Der Logik-Compiler kompiliert diesen Text und erzeugt eine Vielzahl von Dateien, darunter auch das gewünschte JEDEC-Programmierformat `IC1.JED`.

Die Bedienung des Kompilers ist einfach und Gegenstand der Übungsdurchführung. Da der Compiler ein Shareware-Programm ist, kann man diese Vorgänge auch als Bestandteil der Übungsvorbereitung durchführen. Siehe Anhang: Installation und Bedienung des ABEL-Compilers.

Folgende Einstellungen werden gewählt:

```
Compile-Options:
   Expanded Listing (Für die Dokumentation)
PartMap-Options:
   Device-Feld sollte bei einer neuen Programmierung leer sein
   No Checksum          << unser Programmierer kann das nicht)
   Short/Long Document  << ausprobieren)
   X-Value 0            << Anfangswert jener Zellen, die nicht programmiert werden
   Brief JEDEC-Format   << Kurz- oder Langformat wählen
   User Signature       << hier kann man sogar elektronisch dokumentieren
```

Jetzt kann man mit Compile-Compile kompilieren und erhält die Dateien `.AOP, .DMC, .LST, .DAV, .TMV, .TT1`. Danach kann man sich das JEDEC-Format mit View-Jedec anschauen und erhält die Dateien `.DOC, .TT3, .TT2` und `IC1.JED`.

Simulation

Mit *View-Simulation-Results* kann man sich den Erfolg oder Mißerfolg der Test-Vektoren ansehen, das Ergebnis wird in einer Datei `.SM1` abgelegt. Die Einstellung, welche Art der Darstellung gewählt wird, kann man in *Compile-Trace-Options* festlegen.

```
             D    C    B    A         X
        V0001  |    |    |    |         |
        V0002  |    |    |    ~|        |
        V0003  |    |    ~|  |~         |
        V0004  |    |     |   ~|        |
        V0005  |    ~|  |~   |~         ~|
        V0006  |     |  |    ~|        |~
        V0007  |     |   ~|  |~         ~|
        V0008  |     |    |   ~|        |~
        V0009   ~|  |~   |~   |~         |
        V0010    |  |    |    ~|        |
        V0011    |  |    ~|  |~         ~|
        V0012    |  |     |   ~|         |
        V0013    |   ~|  |~   |~         |~
        V0014    |    |  |    ~|        |
        V0015    |    |   ~|  |~         ~|
        V0016    |    |    |   ~|         |
```

Es gibt ein abgekürztes und ein volles JEDEC-Format. Um Platz zu sparen, verwenden wir hier die abgekürzte Version, alle nicht dargestellten Zellen sind 0:

```
EZ-ABEL 4.32 Data I/O Corp. JEDEC file for: P20V8AS V9.0
Created on: Sun Dec 12 11:50:29 1993

Laborvorbesprechung kombinatorische Logik
       Franz Fiala, 12.12.1993.
*
QP24* QF2706* QV16* F0*
 X0*
NOTE Table of pin names and numbers*
NOTE PINS A:2 B:3 C:4 D:5 X:22*
L0000 11110111111101111111111111111111111111111*
L0040 10111111011110111111111111111111111111111*
L2560 10000000*
L2640
11111111111111111111111111111111111111111111111111111111111111111111
11*
L2704 1*
N X*
C11AD*
```

Vergleichen wir mit der Belegung der Leitungen, welche Gleichung das ergibt:

```
      A   B   C   D
      2/  3/  4/  5/
      00000000001111111111222222222333333333
      01234567890123456789012345678901234567 89

L0000 11110111111101111111111111111111111111111*
L0040 10111111011110111111111111111111111111111*

L000: B & D
L040: !A & C & !D
```

Vergleichen Sie das mit unserer händischen Minimierung im Karnaugh-Diagramm: genau unser Ergebnis! Sie sehen, der Compiler wendet die Rechenregeln genauso an wie wir es tun, nur irrt er sich halt weniger oft und zur Realisierung der verschiedensten Gleichungen benötigt man nur einen einzigen IC. Bedenken Sie, daß in unserem Beispiel noch 7 weitere Funktionen gebildet werden könnten und daß noch 16 Eingänge zur Beschaltung frei sind und mit diesen auch von unserem Beispiel völlig unabhängige Funktionen realisierbar sind.

Sequentielle Logik

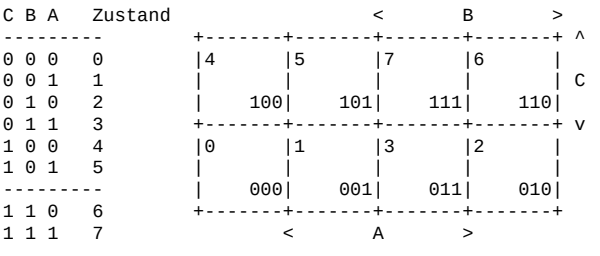
Bei zeitlich wechselnden logischen Beziehungen, deren aktueller Zustand von der Vorgeschichte abhängt, spricht man von sequentieller Logik. Das Grundelement sequentieller Logik ist ein JK-(oder D-)Flip-Flop, dessen Eingänge entsprechend der gewünschten Ausgangslage für jeden inneren Zustand des Schaltwerkes bestimmt werden. Jedem Eingang (D oder J oder K) wird eine Logik vorgelagert, deren Eingänge die Ausgänge der verwendeten Flip-Flops und zusätzliche Bedingungseingänge von außen sind. Die Flip-Flops speichern einen Zustand, der Wechsel von einem Zustand zum nächsten wird durch die Art der Logik festgelegt.

Die einfachste Verdrahtung ist die ohne zusätzliche Logik: ein Frequenzteiler, mit bewerteten Ausgängen auch Binärzähler genannt. Da die Ausgänge einer Stufe mit dem Takteingang der nächsten Stufe verbunden werden, handelt es sich um einen Asynchronzähler. Schaltwerke werden dagegen im allgemeinen mit einem gemeinsamen Takt betrieben, es sind synchrone Schaltwerke.

Binärzähler bis 5 mit Flip-Flops

Wie konstruiert man einen Zähler ganz allgemein? Wir probieren es für einen dreistufigen Binärzähler, der ausnahmsweise nur bis fünf statt bis sieben zählen soll und danach wieder bei Null beginnen soll. (Wie ein dreistufiger synchroner Binärzähler bis 7, also ohne vorzeitiges Ende, verdrahtet wird, entnehme man der Literatur.)

Wir nehmen als Bauelement ein J-K-Flip-Flop an.

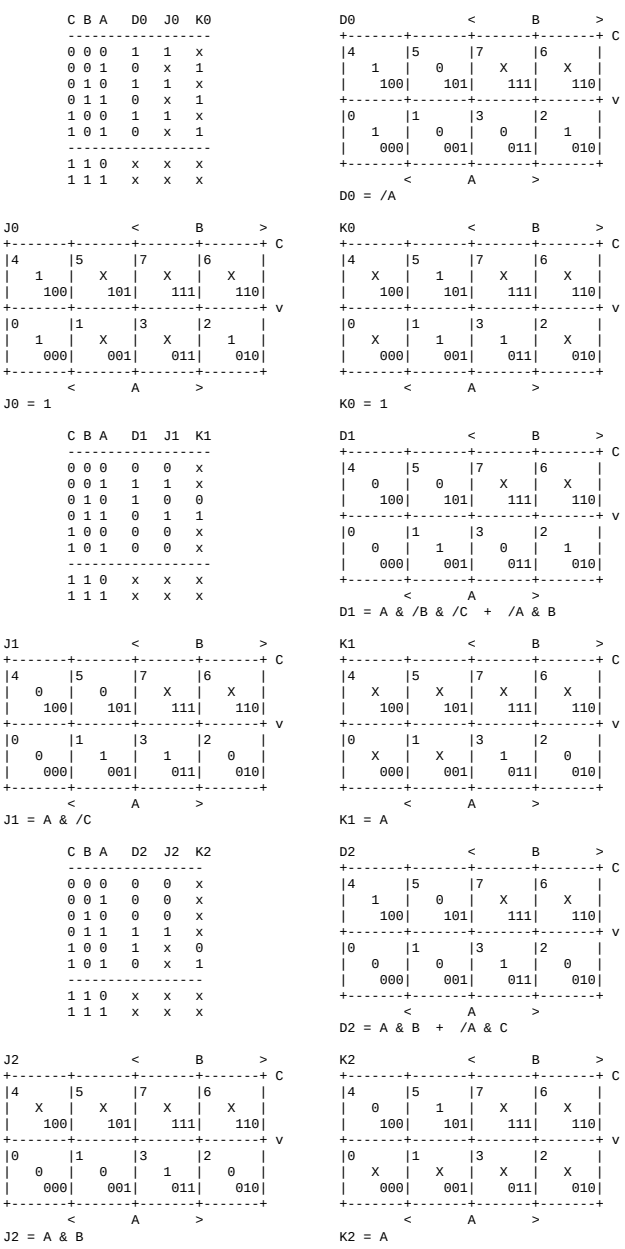


Da wir nur drei Variable haben, reduziert sich das Karnaugh-Diagramm auf die Hälfte.

Wir müssen in jedem Zustand die Logik für die Eingänge J und K (oder D) bestimmen. Dazu ist es hilfreich zu wissen, wie sich J und K (oder D) verhalten sollen:

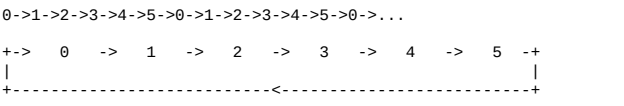
Qalt	Qneu	J	K	D
0	0	0	X	0
0	1	1	X	1
1	0	X	1	0
1	1	X	0	1

In der Vergleichstabelle wurde auch gleich die Bedingung für ein D-FF als Grundbauelement mitaufgenommen, da diese Bauform in den PALs oder GALs enthalten ist. Die Logik für ein D-FF ist etwas aufwendiger, da der zweite Steuereingang des JK-FF eine etwas flexiblere Programmierung zuläßt.

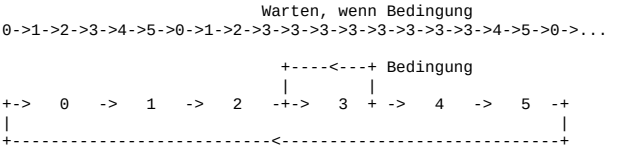


Zustandsdiagramm

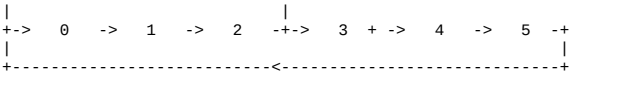
Sehr anschaulich kann der Ablauf auch in einem Zustandsdiagramm dargestellt werden.



In unserem einfachen Beispiel gibt es als Variablen lediglich die Ausgänge der Flip-Flops selbst, man kann aber genausogut weitere Bedingungen dazunehmen, etwa einen zusätzlichen Schalter, der bewirken soll, daß ein bestimmter Zustand zum Wartezustand wird.



oder daß der normale Zählzyklus von 0->1->2->3->4->5->0... auf 0->1->2->0->1->2... geändert wird usw.



Das Zustandsdiagramm ist eine wertvolle Hilfe bei der Konstruktion sequentieller Schaltwerke mit einem PAL.

Binärzähler bis 5 mit einem GAL

Die Betriebsart des GAL muß „Registered-PAL"-Mode sein, d.h. SYN=0, AC0=1. Wir benutzen nur die ersten drei Zellen, daher AC10=0, AC11=0 und AC12=0, die restlichen AC1i sind beliebig.

Unser Zähler einfacher Art hat gar keine Eingänge, lediglich die Ausgänge der Zählstufen wirken sich an den Bedingungseingängen D der Flip-Flops aus.

Unsere Gleichungen lauten:

```
D0 = /A
D1 = A & /B & /C  +  /A & B
D2 = A & B  +  /A & C
```

Die Pinbelegung:

```
A Pin 22
B Pin 21
C Pin 20
```

```
       AA  BB  CC
        /   /   /
     00000000001111111112222222223333333333
     01234567890123456789012345678901234567890123456789

L0000 11101111111111111111111111111111111111111111  << ODER00
...
L0320 11011110111011111111111111111111111111111111  << ODER10
L0360 11011101111111111111111111111111111111111111  << ODER11
...
L0640 11011101111111111111111111111111111111111111  << ODER20
L0680 11101111110111111111111111111111111111111111  << ODER21
...
L2560 000xxxxx                                       << XOR pro Zelle
L2568 11111111                                       << \ ab hier kann man eine
L2576 11111111                                       <<  \ beliebige Signatur
L2584 11111111                                       <<   \ definieren
L2592 11111111                                       <<    \
L2600 11111111                                       <<     \
L2608 11111111                                       <<      \
L2616 11111111                                       <<       \
L2624 11111111                                       <<        \
L2632 000xxxxx                                       << Bit AC1 pro Zelle
L2640 10000000                                       << PTD für Zelle 0
L2648 11000000                                       << PTD für Zelle 1
L2656 11000000                                       << PTD für Zelle 2
L2664 xxxxxxxx                                       << PTD für Zelle 3
L2672 xxxxxxxx                                       << PTD für Zelle 4
L2680 xxxxxxxx                                       << PTD für Zelle 5
L2688 xxxxxxxx                                       << PTD für Zelle 6
L2696 xxxxxxxx                                       << PTD für Zelle 7
L2704 0                                              << SYN
L2705 1                                              << AC0
```

Binärzähler bis 5 mit dem Logik-Compiler

Um einen Binärzähler zu entwerfen, benötigen wir einige sprachliche Erweiterungen:

Wir haben nur Ausgangspins, man muß aber dem Logik-Compiler mitteilen, daß es sich um Register-Ausgänge handelt:

```
A,B,C pin 22,21,20 istype 'reg_D';
```

Der Takteingang ist Pin1.

```
Clk pin 1;
```

Wir definieren als Zustände des Zählers die Zahlen 0..7.

```
Z0 = ^b000;
Z1 = ^b001;
Z2 = ^b010;
Z3 = ^b011;
Z4 = ^b100;
Z5 = ^b101;
Z6 = ^b110;
Z7 = ^b111;
```

Die Ausgänge A,B,C werden getaktet:

```
[A,B,C].c = Clk;
```

Anders als bei der kombinatorischen Logik haben wir ein Zustanddiagramm zu erfüllen:

```
state_diagram [C,B,A]
   State Z0: GOTO Z1;
   State Z1: GOTO Z2;
   State Z2: GOTO Z3;
   State Z3: GOTO Z4;
   State Z4: GOTO Z5;
   State Z5: GOTO Z0;
" Verbotene Zustände lassen den
" Zähler von vorne beginnen
   State Z6: GOTO Z0;
   State Z7: GOTO Z0;

test_vectors  '5er-Zähler'
   ( [Clk] -> [C,B,A] )
     [ .C. ] ->  Z1;
     [ .C. ] ->  Z2;
     [ .C. ] ->  Z3;
     [ .C. ] ->  Z4;
     [ .C. ] ->  Z5;
     [ .C. ] ->  Z0;
```

Wenn wir wieder kompilieren, erhalten wir als Endergebnis die Datei `IC2.JED`.

```
QP24* QF2706* QV6* F0*
 X0*
NOTE Table of pin names and numbers*
NOTE PINS A:22 B:21 C:20 Clk:1*
L0000 11111101111111111111111111111111111111111*
L0040 11111111110111011111111111111111111111111*
L0320 11111101111011101111111111111111111111111*
L0360 11111110110111101111111111111111111111111*
L0640 11111101110111101111111111111111111111111*
L0680 11111110111011011111111111111111111111111*
L2560 01100000*
L2640
11111111111111111111111111111111111111111111111111111111111
1111111111*
L2705 1*
```

Aufgabe

Zwei Motoren X und Y haben drei Geschwindigkeitsstufen. Diese und der Stillstand werden mit je zwei Bit x1,x2 und y1,y2 signalisiert

```
Motor X Motor Y
x2 x1   y2 y2   Drehzahl
-------------------------
0  0    0  0    Stillstand
0  1    0  1    Stufe 1
1  0    1  0    Stufe 2 > Stufe 1
1  1    1  1    Stufe 3 > Stufe 2
```

Bestimmen Sie jene logische Funktion Z, die mit log.1 angibt, daß der Motor X schneller oder gleich schnell wie Motor Y läuft.

Realisieren Sie diese Schaltung sowohl aus Standard-Logikbausteinen als auch mit einem GAL. □