

# Zahlendarstellung

- Wandle -23 in eine binäre Festpunktzahl im Zweierkomplement
- Rechne binär:
  - a)  $-23 * 4$
  - b)  $-23 * -4$

# Grey-Code

Der Grey-Code ist ein Binär-code, bei dem sich zwei aufeinanderfolgende Zahlen nur um 1 unterscheiden.

0	0000		8	1100
1	0001		9	1101
2	0011		10	1111
3	0010		11	1110
4	0110		12	1010
5	0111		13	1011
6	0101		14	1001
7	0100		15	1000

Entwickle eine Schaltfunktion  $f : B^4 \rightarrow B^4$ , die zu jeder Binärzahl im Grey-Code jeweils den Nachfolger (modulo 16) liefert.

- Stelle die disjunktive Form der Funktion mittels Karnaugh-Diagrammen auf.
- Implementiere die Funktion mittels PLA.

# Dekrementer-Schaltwerk

Es soll ein Schaltwerk entwickelt werden, das eine gegebene 4-stellige Binärzahl bei jedem Takt um 1 herunterzählt.

1. Entwickle zunächst ein Schaltnetz für  $f : B^5 \rightarrow B^4$  mit einer (positiven) 4-Bit-Zahl und einer Steuerleitung  $D$  als Eingabe. Bei  $D=1$  soll die Zahl um 1 erniedrigt werden, bei  $D=0$  soll sie unverändert ausgegeben werden. Das Schaltnetz soll modular aufgebaut sein, so dass für jede Stelle der Binärzahl die gleichen Bausteine verwendet werden.
2. Der vollständige Baustein soll Eingänge  $x_3, x_2, x_1, x_0$  für den Startwert des Zählers und  $S$  zum Setzen des Zählers haben, wobei bei  $S = 1$  die an den Eingängen anliegende Binärzahl in ein 4-stelliges (aus D-Flipflops aufgebautes) Register  $y_3, y_2, y_1, y_0$  übernommen wird; bei  $S = 0$  beginnt der Zähler rückwärts zu zählen. Der Ausgang  $A$  soll den Wert 1 haben, solange der Zähler einen Wert  $> 0$  enthält, danach den Wert 0.

# Bubble-Sort in Pascal

```
bubblesort(L,S):- myappend(X,[A,B|Y],L), %% instanziiere A,B mit allen
                                                    %% aufeinanderfolgenden
                                                    %% Elementen aus L
B < A,!, %% wenn B kleiner als A
myappend(X,[B,A|Y],M), %% vertausche B und A
bubblesort(M,S). %% sortiere Resultat.
bubblesort(L,L).
```

Zur Implementierung des Bubblesort-Algorithmus' in Pascal sei ein Array vom Typ Integer wie folgt deklariert:

```
type sa: array[0..100] of integer
```

Implementiere eine (nichtrekursive) Prozedur `bubblesort` mit einem Parameter vom Typ `sa`, die die darin übergebenen Werte sortiert zurückgibt.

# Zugverbindungen in Prolog

Gegeben seien Zugpläne der Art `zug(Zugnr, Bahnhof, An, Ab)`, z.B.

`zug(1,do,838,840) . zug(1,e,900,902) . zug(1,du,925,927) .`

`zug(2,e,905,907) . zug(2,ha,930,932) .`

`zug(3,ha,940,942) . zug(3,w,1000,1002) . zug(3,k,1100,1102) .`

`zug(4,du,940,942) . zug(4,d,1000,1002) . zug(4,k,1030,1032) .`

Erstelle ein rekursives Prädikat `verbindung(Startort, Zielort, Ab, An)`, das Verbindungen mit beliebig häufigem Umsteigen berechnet. Dabei soll jeweils mindestens 1 Minute Zeit zum Umsteigen sein.

## Listen in Prolog

Gegeben sei eine Liste mit Integer-Werten. Schreibe ein Prolog-Prädikat  $\text{Imax}(L,M)$ , das den maximalen Wert aus der Liste  $L$  in  $M$  zurückliefert.

# Listen in OCaml

```
append([],Liste,Liste).
```

```
append([Kopf | AlteListe], Liste, [Kopf | NeueListe]):-  
    append(AlteListe,Liste,NeueListe).
```

```
invert([],[]).
```

```
invert([Kopf | Rest], Liste) :-  
    invert(Rest,Rest1),           %% invertiere den Rest  
    append(Rest1,[Kopf],Liste)   %% haenge Kopf hinten an  
                                %% das Ergebnis.
```

Für die vorstehenden Prolog-Prädikate `append` und `invert` sollen entsprechende Funktionen in OCaml geschrieben werden.