5. Darstellung und Implementierung von Fehlertoleranzverfahren

- ***Informationsredundanz:***

  Der Rückgriff auf die **Informationsredundanz** erfordert die Verwendung von Fehlererkennungs- und Fehlerkorrektur-codes, deren Decodierung (in Grenzen, z. B. 2 Bitfehler erkennen, jedoch nur 1 Bitfehler korrigierbar) unmittelbar eine Fehlerkorrektur erlaubt.

- ***Grundbegriffe:***

  Im folgenden wollen wir uns auf die einfachste Form eines Codes beschränken, nämlich auf **lineare Binärcodes**.

  Bezeichnung: $\Rightarrow$ (**n**, **m**, **d**)-Codes

5. Darstellung und Implementierung von Fehlertoleranzverfahren

- ***(n, m, d)-Codes:***

  n = Länge der Codewörter
  (Codewörter enthalten Informationsbits und Prüfbits)

  m = Anzahl der Informationsstellen

  d = Mindest-Hammingdistanz (eigentlich: $\mathbf{d_{min}}$)

  Es gilt:

$$\boxed{n = m + k}$$

  wobei

  k = Anzahl der Prüfbits

- ***Codierung:***

  Abbildungsvorschrift, durch die Elemente eines Zeichenvorrats (hier als **Nachricht oder Information** bezeichnet) in solche eines anderen Zeichenvorrats (hier als redundanter Code bzw. redundante **Codewörter** bezeichnet) umgewandelt werden.

  Hinweis: Die Elemente der beiden Zeichenvorräte können dabei durchaus identisch sein, z. B. **A : B | {0, 1} → {0, 1}**

- ***Decodierung:***

  Dient der eindeutigen Bildung der ursprünglichen Nachricht aus eventuell gestörten Codewörtern.

- ***Beispiel 1:***

  Codierung:

  $X \rightarrow Y \mid Y = C \oplus X$; mit z. B. C = beliebige Zufallsfolge

  und $\oplus$ = **XOR** (Addition mod 2)

  Decodierung:

  $Y \rightarrow X \mid X = C \oplus Y$; mit C = identische Zufallsfolge

- ***Beispiel 2:***

  ASCII-Code:

  $\{A, B, C, ..., Z\} \leftrightarrow \{41_H, 42_H, 43_H, ..., 5A_H\}$

Annahme: Während Speicherung oder Übertragung der Codewörter treten Störungen auf, die einzelne Bitpositionen verfälschen.

- **Ohne Störung** erfolgt Decodierung einwandfrei.
- Bei **geringen Störungen** werden die Fehler eliminiert ($\rightarrow$ Nachricht läßt sich zurückgewinnen).
- Bei **stärkeren Störungen** können die Fehler allenfalls erkannt werden ($\rightarrow$ Rückgewinnung der Nachricht u. U. nicht möglich).
- Bei **sehr starken Störungen** besteht die Möglichkeit, dass durch die Decodierung die eigentliche Nachricht verfälscht wird ($\rightarrow$ erfordert geeignete Gegenmaßnahmen).

- **Binärcodes** sind Mengen aus mehrstelligen Codewörter (oder Zeichen), die aus Binärzeichen (0, 1) bestehen.
- Codes mit **fester** Wortlänge werden als **Blockcodes** bezeichnet.
- Ein Blockcode heißt **dicht**, wenn die Abbildungsfunktion $X \rightarrow Y$ surjektiv ist, d. h. alle $y \in Y$ Codewörter darstellen.
- Die Stellenzahl **n** gibt die **Wortlänge** aller vorhandenen **M** Codewörter an.
- Charakteristische Größen eines Codes sind:
  - die Hammingdistanz **d**,
  - die Wortanzahl **M** und
  - das Gewicht **w**.

Fehlererkennungscodes (FEC):

Nach **DIN 44 300** werden Codes, bei denen die Zeichen (Codewörter) so gebildet werden, dass durch Störungen verursachte Abweichungen erkennbar werden, Fehlererkennungscodes (FEC) genannt.

Fehlerkorrekturcodes (FKC):

Darüber hinaus sind Fehlerkorrekturcodes (FKC) spezielle Fehlererkennungscodes, bei denen die Teilmenge der gestörten Zeichen innerhalb bestimmter Grenzen ohne Rückfrage korrigiert werden kann.

Hammingdistanz:

Ist der Abstand zwischen zwei beliebigen Codewörtern X und Y eines vereinbarten Codes, wobei diese Größe **d** als arithmetische Summe aller unterschiedlichen Binärstellen ausgedrückt wird.

$$d = \sum_{i=1}^{n} (X_i \oplus Y_i) \; ; \quad \oplus = \textbf{XOR} \text{ (Addition mod 2)}$$

mit

$$X = (X_1, X_2, ..., X_n) \; ; \qquad Y = (Y_1, Y_2, ..., Y_n) \; .$$

Speziell gilt:

$$d_{min} = Min(d_i) \qquad \text{für } i = 1 \text{ bis } M*(M-1)/2$$

(alle Wortkombinationen)

mit

$$M = 2^m$$

wobei

$$m \leq n \; .$$

In Verbindung mit der Hammingdistanz d steht die Anzahl der erkennbaren bzw. korrigierbaren Fehler:

- erkennbar:

$$n_e = d - 1$$

- korrigierbar:

$$n_c = \begin{cases} (d-1)/2 & \text{für } d = \text{ungerade} \\ d/2 & \text{für } d = \text{gerade} \end{cases}$$

Gewicht:

Hierunter wird die Summe der **von Null verschiedenen** Bitstellen verstanden.

$$w = \sum_{i=1}^{n} X_i \quad ;$$

mit

$$X = (X_1, X_2, ..., X_n) .$$

Wortanzahl:

Anzahl möglicher unterschiedlicher Codewörter bei sonst gleichen Eigenschaften $\rightarrow$ Codeumfang.

$$M = 2^m \quad ;$$

mit $m = 1, 2, 3, \ldots$ und $m \leq n$.

M = Anzahl der Codewörter

m = Länge der Nachrichtenwörter (nur Informationsstellen)

n = Länge der Codewörter (Informationsstellen + Prüfstellen)
$(n = m + k)$

Prüfziffern zur Fehlererkennung:

Redundanz in den einzelnen Nachrichtenworten bedeutet, dass zusätzliche Bitpositionen vorgesehen werden, um einzelne Fehler erkennbar und/oder korrigierbar zu machen.

Einfachster Fall: **Ein** zusätzliches Bit als Redundanz hinzugefügt (Prüf- bzw. Paritätsbit) $\rightarrow$ **k = 1**

- Die Gesamtlänge der Codewörter ist demnach: $n = m + 1$
- Relativer Redundanzanteil: $= 1 / (n - 1)$

Relative Redundanz:

Mit **m** Informationsstellen kann man 2^m unterschiedliche Datenwörter darstellen, ohne dass dabei Redundanz vorliegt. Enthalten die Codewörter **ein** zusätzliches **Prüfbit**, so ergibt sich der Gesamtaufwand zu:

$$\frac{n}{m} = \frac{n}{n-1} = 1 + \frac{1}{n-1}$$

- Ohne das Prüfbit hat die Redundanz dagegen den Wert **Null**.
- Das zusätzliche Prüfbit bewirkt also eine Einzelfehlererkennung; dagegen werden zwei Fehler nicht erkannt.

Relative Redundanz:

Eine typische Regel für die Bildung des zusätzlichen Prüfbits ist, dass das

$$\text{Prüfbit} = \left(\sum_{\forall} \text{Informationsbits} \right) \bmod 2$$

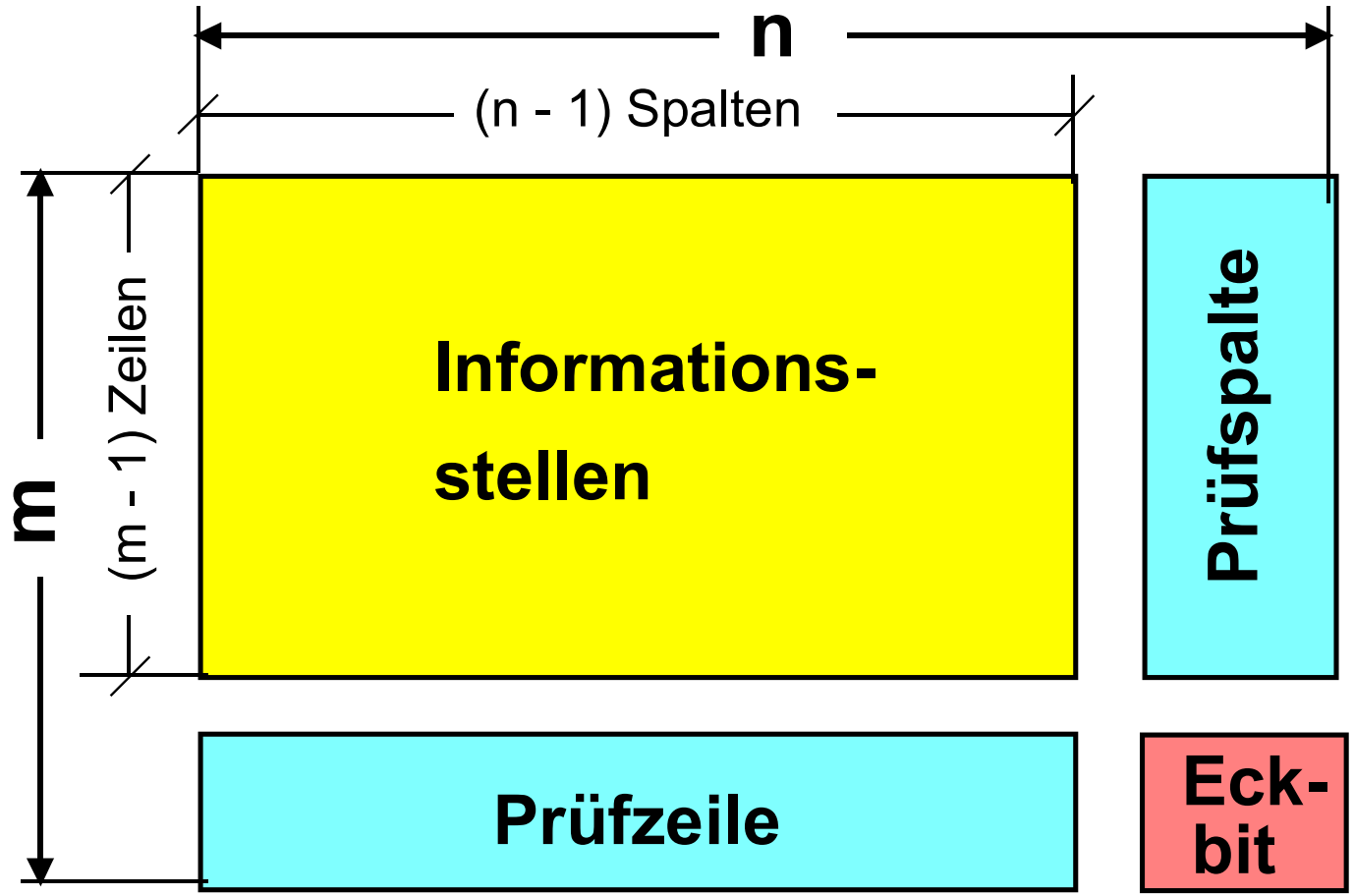
gesetzt wird (sogenannte **gerade** Parität).

Rechteckcodes:

Zur Korrektur einzelner Fehler ist mehr Redundanz erforderlich als zur einfachen Fehlererkennung. Eine häufig verwendete Möglichkeit besteht in den sogenannten **Rechteckcodes**, die zusätzliche Prüfbits in einer Prüfspalte bzw. Prüfzeile enthalten.

Bezeichnung: Binärer Rechteckcode mit **m** · **n** Stellen.

- **Prüfspalte** (gebildet aus Zeilenquersumme mod 2)
- **Prüfzeile** (gebildet aus Spaltenquersumme mod 2)
- **Eckbit** (z. B. gebildet aus Prüfzeilensumme mod 2 oder unbenutzt)


n
(n - 1) Spalten
m
(m - 1) Zeilen
Informations-
stellen
Prüfspalte
Prüfzeile
Eck-
bit

Rechteckcode:

Durch die gemeinsame Betrachtung von insgesamt **m** Codewörtern der Länge **n** ergeben sich

- **(m – 1)·(n – 1)** Informationsstellen und
- insgesamt ein Aufwand von **m** · **n** Stellen.

Die Redundanz wird dann durch folgende Gleichung berechenbar:

$$\frac{n \cdot m}{(m-1) \cdot (n-1)} = 1 + \frac{1}{n-1} + \frac{1}{m-1} + \frac{1}{(m-1) \cdot (n-1)}$$

Redundanz:

$$\frac{1}{n-1} = Prüf\,zeile \qquad \frac{1}{m-1} = Prüf\,spalte$$

$$\frac{1}{(m-1)\cdot(n-1)} = Eckbit$$

- Jeder **Einzelfehler** verändert dabei **zwei** Prüfbits.
- Damit wird ein Einzelfehler nicht nur erkennbar, sondern auch korrigierbar.
- Ein ähnlicher Weg führt zu den sogenannten **Hammingcodes**, die für eine Einfehlerkorrektur mit minimaler Redundanz verwendet werden.

Beispiel: 10 x 7 (beim Sender)

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 1: Code mit einer fehlerhaften Stelle x

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
1	0	1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 1: fehlerhafte Stelle x ist erkennbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
1	0	1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 1: fehlerhafte Stelle x ist auch korrigierbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 2: Code mit zwei fehlerhaften Stellen x

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 2: fehlerhafter Frame ist zwar erkennbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 2: Frame ist aber nicht korrigierbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	1	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 3: Code mit drei fehlerhaften Stellen x

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 3: fehlerhafte Frame ist zwar erkennbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Beispiel 3: Frame ist aber nicht korrigierbar

1	1	1	0	1	1	0	0	1	0
0	1	1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0	1	1
1	0	1	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	0	1	0	0	0	1	0

Ergebnis:

- 1-bit Fehler sind erkennbar und korrigierbar
- 2- und 3-bit Fehler (fehlerhafter Frame) sind zwar erkennbar aber **nicht** mehr korrigierbar
- bei einem 3-bit Fehler würde die falsche Stelle korrigiert werden
- Grund hierfür ist die Hammingdistanz, die offensichtlich d = 2 ist
- die Hammingdistanz erlaubt im vorliegenden Fall nur das sichere Erkennen und Korrigieren von 1-bit-Fehlern
- bis 3-bit Fehler würden aber sicher erkannt werden

Voraussetzungen:

1. Sieht man im Codewort **k** Prüfstellen vor, so lassen sich mit den Prüfbits $\mathbf{2^k}$ verschiedene Ereignisse beschreiben.
2. Umgekehrt müssen **n** Einzelfehler und der **fehlerfreie** Fall zum Zweck der Korrektur unterschieden werden können. Um **alle** möglichen Fehlerarten sowie den **fehlerfreien** Fall angeben zu können, ist folglich die Beziehung zu erfüllen:

Informationsstellen	Prüfstellen	fehlerfreie Fall
m	k	1

$$\mathbf{2^k \geq n + 1 = m + k + 1}$$

Realisierungsschema:

1. Nachricht [$\mathbf{m_1}$, $\mathbf{m_2}$, $\mathbf{m_3}$, ...] so spreizen, dass Stellenpositionen $\mathbf{2^i}$ für die i-te Quersumme $\mathbf{QS_i}$ bzw. (i+1)-te Prüfstelle $\mathbf{K_{i+1}}$ frei bleiben ($\mathbf{i}$ = 0, 1, 2, ...).
2. Prüfstellen $\mathbf{K_1}$, $\mathbf{K_2}$, $\mathbf{K_3}$, ... so ergänzen, dass die Quersummen $\mathbf{QS_i}$ mod 2 Null ergeben.
3. Ansatz geht von **Dualzahlen** mit den Wertigkeiten 1, 2, 4, 8, ... aus, wobei die i-te Quersumme $\mathbf{QS_i}$ über alle Positionen gebildet wird, die die Potenz $\mathbf{2^i}$ in ihrer Ziffernwertigkeit enthalten.

Berechnung der QS_i:

K_1 durch QS_0 über 1, 3, 5, 7, 9, ...

K_2 durch QS_1 über 2, 3, 6, 7, 10, 11, 14, ...

K_3 durch QS_2 über 4, 5, 6, 7, 12, 13, 14, 15, 20, ...

K_4 durch QS_3 über 8, 9, 10, 11, 12, 13, 14, 15, 24, 25, ...

u.s.w.

Position	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	12	13	14	15	**16**
i	0	1		2				3								4
m_i / K_{i+1}	K_1	K_2	m_1	K_3	m_2	m_3	m_4	K_4	m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	K_5

m_i = Nachrichtenbit K_i = Prüfbit

Fehlersyndrom:

- Treten nun Stellenfehler auf, so erlaubt das **erneute** Bilden der Quersummen deren **Lokalisierung** und **Korrektur**.
- Dabei bilden die Veränderungen das sogenannte **Syndrom**, z. B. für $n \leq 7$:

$$\Delta QS_2 \quad \Delta QS_1 \quad \Delta QS_0$$

d. h. eine Angabe der fehlerhaften Stelle in Form einer **Dualzahl** (beachte die Stellenposition beim Syndrom!).

Fehlersyndrom:

Dabei ist:

$$\Delta QS_i = \begin{cases} \mathbf{0}, & \text{wenn } \mathbf{keine} \text{ Änderung in } \mathbf{QS_i} \\ \mathbf{1}, & \text{sonst} \end{cases}$$

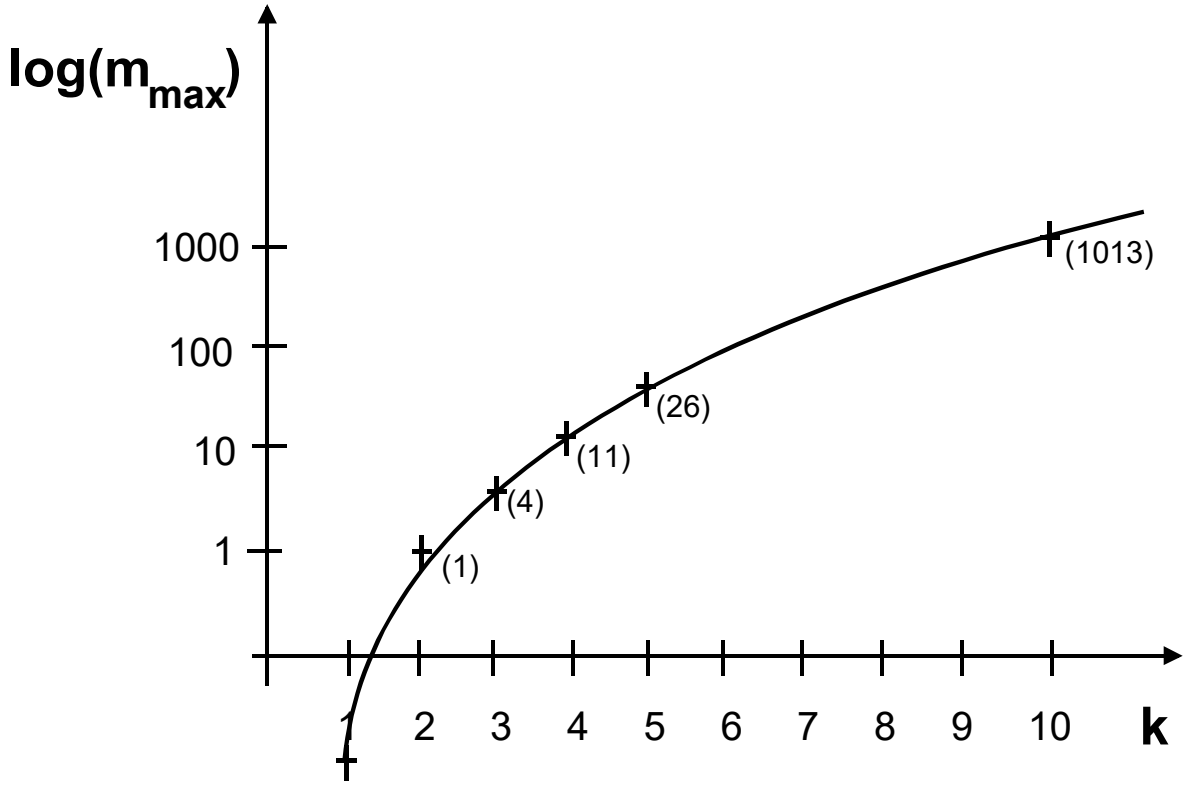
Natürlich muss sich für den **fehlerfreien** Fall das Syndrom **0 0 0** ergeben.

- Code liefert gleichartigen Schutz für die **m** Informationsstellen und die **k** Prüfstellen.
- Hamming-Code gehört ferner zu den systematischen Codes.
- Bezüglich der Korrektur wird kein Unterschied zwischen Informations- und Prüfstellen gemacht.
- Die Redundanz des Codes wächst mit **log(m)**.
- Zur Absicherung von **m** Informationsstellen und **k** Prüfstellen (einschließlich des fehlerfreien Falles) gilt die Relation:

$$2^k \geq m + k + 1$$

Beispiel:

$k = 10 \Rightarrow m = m_{max} = 1013$



- Unterklasse der Gruppencodes
- damit auch Unterklasse der linearen Codes
- jede zyklische Verschiebung eines Codeworts ergibt wieder ein anderes Codewort

⇒ **können leicht mit Hilfe von linearen rück-gekoppelten Schieberegistern erzeugt werden**

Beispiel:

0	***1***	0	1	1	0	0
1	0	1	1	0	0	**0**
0	1	1	0	0	**0**	***1***

Benutzt die Darstellung der Codewörter durch Polynome in **endlichen Körpern**; hier: Galois-Feld **GF(2)**

Es sei:

n = Länge der Codewörter

m = Anzahl der Informationsstellen
z. B. **ein** Prüfbit $\Rightarrow$ **n = m + 1**

d = minimale Hammingdistanz (hier: d := d_{min})

$\Rightarrow$ **(n, m, d)-Code**
z. B. zyklischer **Code (7, 4, 3)**

Nachrichten- und Codewörter werden als **Polynome** interpretiert.
Dabei ist **x** eine Hilfsgröße zur Feststellung der Stellenposition.

Es sei:

c(x) = Nachrichten- bzw. Informationsstellen
v(x) = Codewörter (Informations- und Prüfstellen)
g(x) = Generatorpolynom (z. B.: $1 + \mathbf{x} + \mathbf{x}^3$)

mit

Grad[c(x)] = m – 1 ; $\Rightarrow$ m Informationsstellen
Grad[v(x)] = n – 1 ; $\Rightarrow$ n Codewortstellen
Grad[g(x)] = n – m ;

Codewörter v(x):

$$v = v(x) = v_{n-1}\cdot\mathbf{x}^{n-1} + v_{n-2}\cdot\mathbf{x}^{n-2} + \cdots + v_2\cdot\mathbf{x}^2 + v_1\cdot\mathbf{x} + v_0$$
$$= (v_{n-1}, v_{n-2}, \cdots, v_2, v_1, v_0) \text{ mit } v_i \in \{0, 1\} \text{ und } \forall \bmod 2 \quad (1)$$

Information c(x):

$$c = c(x) = c_{m-1}\cdot\mathbf{x}^{m-1} + c_{m-2}\cdot\mathbf{x}^{m-2} + \cdots + c_2\cdot\mathbf{x}^2 + c_1\cdot\mathbf{x} + c_0$$
$$= (c_{m-1}, c_{m-2}, \cdots, c_2, c_1, c_0) \text{ mit } c_i \in \{0, 1\} \text{ und } \forall \bmod 2 \quad (2)$$

- Eine Multiplikation mit **x** schiebt alle Stellen um eine Position nach links.
- Um zu erreichen, dass die höchstwertige Stelle (hier v_{n-1} bzw. c_{m-1}) rechts angefügt wird ($\rightarrow$ **zyklisch**), erfolgt die Multiplikation im **endlichen Körper mod $(x^n + 1)$**.

Beispiel:

$$(v_{n-1}\cdot\mathbf{x}^{n-1} + v_{n-2}\cdot\mathbf{x}^{n-2} + \cdots + v_2\cdot\mathbf{x}^2 + v_1\cdot\mathbf{x} + v_0) * \mathbf{x}$$
$$= \; v_{n-1}\cdot\mathbf{x}^{n} + v_{n-2}\cdot\mathbf{x}^{n-1} + \cdots + v_2\cdot\mathbf{x}^3 + v_1\cdot\mathbf{x}^2 + v_0\cdot\mathbf{x}$$

$$(v_{n-1}\cdot x^n + v_{n-2}\cdot x^{n-1} + \cdots + v_2\cdot x^3 + v_1\cdot x^2 + v_0\cdot x) : (x^n + 1) = v_{n-1}$$
$$\pm\ (v_{n-1}\cdot x^n + \qquad\qquad\qquad\qquad + v_{n-1})$$

$$(v_{n-2}\cdot x^{n-1} + v_{n-3}\cdot x^{n-2} + \cdots + v_2\cdot x^3 + v_1\cdot x^2 + v_0\cdot x + v_{n-1}) \;:=\; \mathbf{r(x)}$$

Die Codewörter, d. h. der Code **v = v(x)**, läßt sich mit Hilfe eines **Generatorpolynoms g(x)** bilden. Ein Wort heißt Codewort genau dann, wenn es durch das Generatorpolynom **ohne Rest** teilbar ist.

Satz: Ein zyklischer Code ist durch das Generatorpolynom **g(x)** und **n** vollständig bestimmt.

Generatorpolynome

... sind ihrerseits Faktoren des Polynoms $\mathbf{x^n + 1}$, d. h.

$$x^n + 1 = g(x) \cdot h(x) \qquad (3)$$

wobei h(x) = **Prüfpolynom**, so dass die Division von $x^n + 1$ durch g(x) ebenfalls **ohne Rest** durchgeführt werden kann.

Letzteres liefert:

$$(x^n + 1) : g(x) = h(x) \qquad (4)$$

Bezeichnet **c(x)** die zu codierende **Nachricht**, so erhält man das **Codewort v(x)** nach dem oben gesagten aus der Vorschrift:

$$v(x) = c(x) \cdot g(x) \mod (x^n + 1) \qquad (5)$$

Grad: [n - 1] [m - 1] [n - m]

- g(x) ist vom Grad [n – m]
- v(x) ist durch g(x) ohne Rest teilbar (vgl. Gl(5))

$$\Rightarrow v(x) : g(x) = c(x)$$

- $x^n + 1$ ist durch g(x) ebenfalls teilbar (vgl. Gl(3))

$$\Rightarrow x^n + 1 : g(x) = h(x)$$

Beispiel:

Zyklischer Code (7, 4, 3) mit $g(x) = 1 + x + x^3$

- Generatorpolynom g(x) ist vom Grad [n – m]: $g(x) = 1 + \mathbf{x} + \mathbf{x}^3$
- Nachricht c(x) sei: $c(x) = c_0 + c_1 \cdot \mathbf{x} + c_2 \cdot \mathbf{x}^2 + c_3 \cdot \mathbf{x}^3$ mit $c_i \in \{0, 1\}$
- Codewörter v(x) ergeben sich aus Gl(5) – $v(x) = c(x) \cdot g(x)$ – zu:

$$v(x) = c_0 + (c_0 + c_1) \cdot \mathbf{x} + (c_1 + c_2) \cdot \mathbf{x}^2 + (c_0 + c_2 + c_3) \cdot \mathbf{x}^3 + (c_1 + c_3) \cdot \mathbf{x}^4 + c_2 \cdot \mathbf{x}^5 + c_3 \cdot \mathbf{x}^6 \quad (6)$$

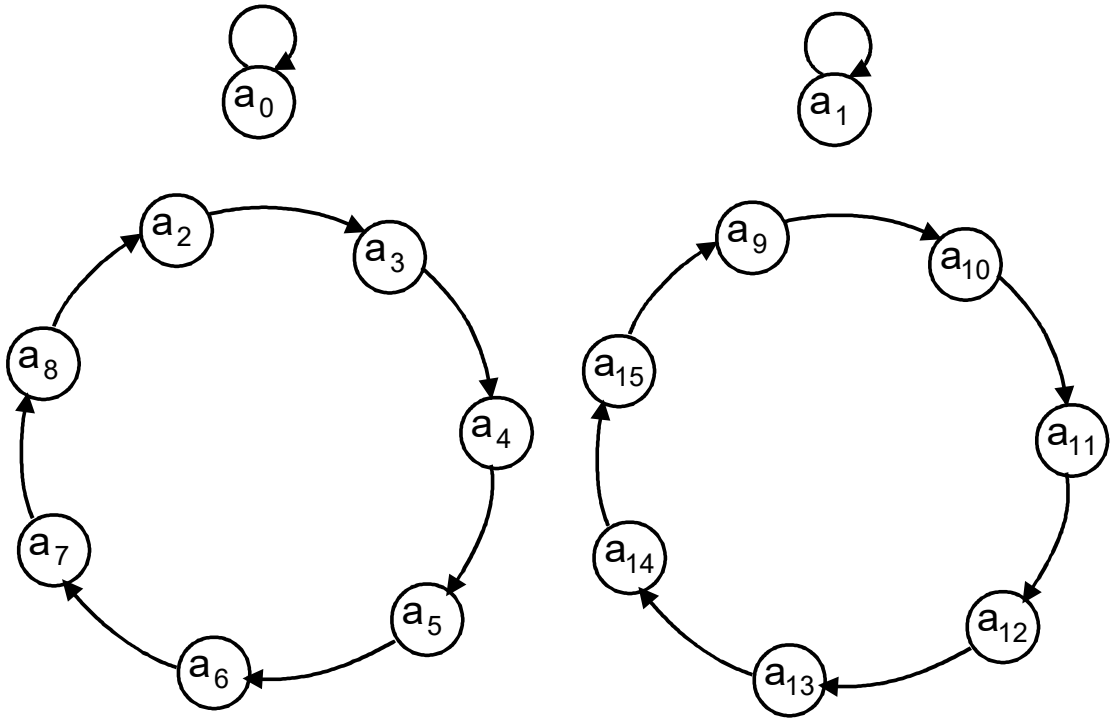
$\Rightarrow$ nur c_0, c_2 und c_3 direkt als Informationsstellen in v(x), dagegen ist c_1 nur implizit enthalten.

$\Rightarrow$ Code ist nicht systematisch!

	Nachrichten c(x)				→	Codewörer gem. GI (6) v(x)							
	c_3	c_2	c_1	c_0		v_6	v_5	v_4	v_3	v_2	v_1	v_0	
c^0 :=	0	0	0	0		0	0	0	0	0	0	0	:= a^0
c^1 :=	0	0	0	1		0	0	0	1	0	1	1	:= a^2
c^2 :=	0	0	1	0		0	0	1	0	1	1	0	:= a^3
c^3 :=	0	1	0	0		0	1	0	1	1	0	0	:= a^4
c^4 :=	1	0	0	0		1	0	1	1	0	0	0	:= a^5
c^5 :=	0	1	1	1		0	1	1	0	0	0	1	:= a^6
c^6 :=	1	1	1	0		1	1	0	0	0	1	0	:= a^7
c^7 :=	1	0	1	1		1	0	0	0	1	0	1	:= a^8

Fortsetzung:

	c(x)				→	v(x)							
	c_3	c_2	c_1	c_0		v_6	v_5	v_4	v_3	v_2	v_1	v_0	
c^8 :=	1	1	0	1		1	1	1	1	1	1	1	:= a^1
c^9 :=	0	0	1	1		0	0	1	1	1	0	1	:= a^9
c^{10} :=	0	1	1	0		0	1	1	1	0	1	0	:= a^{10}
c^{11} :=	1	1	0	0		1	1	1	0	1	0	0	:= a^{11}
c^{12} :=	1	1	1	1		1	1	0	1	0	0	1	:= a^{12}
c^{13} :=	1	0	0	1		1	0	1	0	0	1	1	:= a^{13}
c^{14} :=	0	1	0	1		0	1	0	0	1	1	1	:= a^{14}
c^{15} :=	1	0	1	0		1	0	0	1	1	1	0	:= a^{15}



Man erkennt, daß die unterschiedlichen 16 Nachrichtenwörter (c^0 bis c^{15}) einen **zyklischen Code** erzeugen.

Bildungsvorschrift:

1. Multipliziere c(x) mit x^{n-m}.
2. Dividiere $x^{n-m} \cdot c(x)$ durch g(x) und bestimme den Rest r(x) für den gilt:

$$\frac{x^{n-m} \cdot c(x)}{g(x)} = q(x) \text{ Rest } r(x) \qquad (7)$$

3. Das gesuchte Codewort v(x) entsteht aus:

$$v(x) = x^{n-m} \cdot c(x) \oplus r(x) = q(x) \cdot g(x) \qquad (8)$$

Beispiel:

- Nachricht sei: $c(x) = c_0 + c_1 \cdot \mathbf{x} + c_2 \cdot \mathbf{x}^2 + c_3 \cdot \mathbf{x}^3$ mit $c_i \in \{0, 1\}$
- Generatorpolynom sei: $g(x) = 1 + \mathbf{x} + \mathbf{x}^3$
- Mit GI(7) ergibt sich das Restpolynom **r(x)** zu:

$$\mathbf{x}^3 \cdot (c_0 + c_1 \cdot \mathbf{x} + c_2 \cdot \mathbf{x}^2 + c_3 \cdot \mathbf{x}^3) \ : \ (1 + \mathbf{x} + \mathbf{x}^3) \ =$$

$$c_3 \cdot \mathbf{x}^3 + c_2 \cdot \mathbf{x}^2 + (c_1 + c_3) \cdot \mathbf{x} + (c_0 + c_2 + c_3)$$

$$\text{Rest} \ \ (c_1 + c_2 + c_3) \cdot \mathbf{x}^2 + (c_0 + c_1 + c_2) \cdot \mathbf{x} + (c_0 + c_2 + c_3)$$

d. h. $\mathbf{r(x)} = (c_1 + c_2 + c_3) \cdot \mathbf{x}^2 + (c_0 + c_1 + c_2) \cdot \mathbf{x} + (c_0 + c_2 + c_3)$ (9)

- Mit dem so gewonnenen **r(x)** ergibt sich **v(x)** aus Gl(8) zu:

$$v(x) = c_3 \cdot \mathbf{x}^6 + c_2 \cdot \mathbf{x}^5 + c_1 \cdot \mathbf{x}^4 + c_0 \cdot \mathbf{x}^3 + (c_1 + c_2 + c_3) \cdot \mathbf{x}^2 + (c_0 + c_1 + c_2) \cdot \mathbf{x} + (c_0 + c_2 + c_3) \quad (10)$$

Man erkennt, dass die höherwertigen Stellen in **v(x)** die Informationsstellen des Polynoms **c(x)** darstellen.

$\Rightarrow$ **systematischer Code!**

Prüfungsvorschrift:

- Können Fehler auftreten, so ist durch folgende Prüfbedingung das Vorhandensein von Fehlern festzustellen:

$$\frac{v(x)}{g(x)} = \frac{v(x) \cdot h(x)}{x^n + 1} = q(x) \quad \underline{\textbf{ohne}} \quad \text{Rest ? (11)}$$

d. h. in beiden Fällen ist der Rest nur dann gleich Null, wenn ein fehlerfreies Codewort vorliegt. Jeder Fehler erzeugt einen von Null verschiedenen Rest, sofern der Fehler nicht gerade ein Vielfaches des Generatorpolynoms ist.

Syndromberechnung:

Wie bei den Hamming-Codes lassen sich die möglichen Fehler auch hier interpretieren. Man kann sich vorstellen, dass ein **fehlerhaftes Codewort** $\mathbf{v_{err}(x)}$ durch Addition eines **Fehlerpolynoms** e(x)

$$e(x) = e_0 + e_1 \cdot \mathbf{x} + e_2 \cdot \mathbf{x}^2 + \cdots + e_{n-1} \cdot \mathbf{x}^{n-1} \qquad (12)$$

entstanden ist, d. h.

$$v_{err}(x) := v(x) + e(x) \qquad (13)$$

Die Prüfung ergibt dann:

Syndromberechnung:

$$\frac{V_{err}(x) \cdot h(x)}{x^n + 1} = \frac{\big(v(x) + e(x)\big) \cdot h(x)}{g(x) \cdot h(x)}$$

$$= \frac{v(x) + e(x)}{g(x)} = q(x) \text{ Rest } s(x) \quad (14)$$

Wegen v(x) = q(x) · g(x) – vgl. Gl(8) – kann der Rest s(x) nur aus dem Term

$$\frac{e(x)}{g(x)}$$

entstanden sein. Dieser Rest **s(x)** läßt sich ebenfalls als Polynom ausdrücken, das höchstens den Grad **m – 2** haben kann.

$$s(x) = s_0 + s_1 \cdot \mathbf{x} + s_2 \cdot \mathbf{x}^2 + \cdots + s_{m-2} \cdot \mathbf{x}^{m-2} \qquad (15)$$

Dieser Rest wird als **Syndrom** bezeichnet, das nur durch den Fehler bestimmt wird und ihn andererseits erkennbar bzw. ggf. korrigierbar macht.

Beispiel:

- Nachricht sei: $c(x) = c_0 + c_1 \cdot \mathbf{x} + c_2 \cdot \mathbf{x}^2 + c_3 \cdot \mathbf{x}^3$ mit $c_i \in \{0, 1\}$
- Generatorpolynom sei: $g(x) = 1 + \mathbf{x} + \mathbf{x}^3$

$\Rightarrow$ $v(x) = c(x) \cdot g(x)$ gem. GI(5)

$\Rightarrow$ $v_{err}(x) = v(x) + e(x)$ gem. GI(13)

$\Rightarrow$ $v_{err}(x) : g(x) = q(x)$ Rest $\mathbf{s(x)}$ gem. GI(14)

Das Syndrom **s(x)** ergibt sich zu:

$$\mathbf{s(x)} = (e_2 + e_4 + e_5 + e_6) \cdot \mathbf{x}^2 + (e_1 + e_3 + e_4 + e_5) \cdot \mathbf{x} + (e_0 + e_3 + e_5 + e_6) \quad (16)$$

Ergebnis:

- Es entsteht ein Rest mit drei Termen.
- Allein die Fehlerkoeffizienten bestimmen den Rest s(x), der nur dann einem nichterkennbaren Fehler entspricht, wenn er den Wert Null aufweist.
- Wie beim Hamming-Code bilden die verschiedenen Fehlermöglichkeiten die 7 unterschiedlichen Werte des Syndroms, während kein Fehler das Syndrom 0 erzeugt.
- Alle Einzelfehler an den Polynomstellen e_i (i = 0, 1, ..., 6) sind eindeutig erkennbar und anschließend korrigierbar.

- Man beachte allerdings, daß das Syndrom nicht direkt als Dualzahl interpretiert werden kann.
- Doppelfehler erzeugen eine falsche Korrektur.
- Auch für die Signaturanalyse sind derartige Codes von Wichtigkeit. Ein Vorteil besteht darin, daß Multiplikationen oder Divisionen im endlichen Körper leicht mittels Schieberegister-schaltungen ausgeführt werden können.
- Es lassen sich mit diesen Codes eine Fehlererkennung und gfs. eine Fehlerkorrektur durchführen. Beachte jedoch, dass durch den Code nur eine beschränkte Anzahl von Fehlern toleriert werden kann.

Gem. Gl(16) ergibt sich:

Fehlerart	s_2	s_1	s_0	Syndrom s(x)
nur $e_0 = 1$	0	0	1	1
nur $e_1 = 1$	0	1	0	x
nur $e_2 = 1$	1	0	0	x^2
nur $e_3 = 1$	0	1	1	1 + x
nur $e_4 = 1$	1	1	0	$x + x^2$
nur $e_5 = 1$	1	1	1	$1 + x + x^2$
nur $e_6 = 1$	1	0	1	$1 + x^2$
$\forall\ e_i = 0$	0	0	0	Null

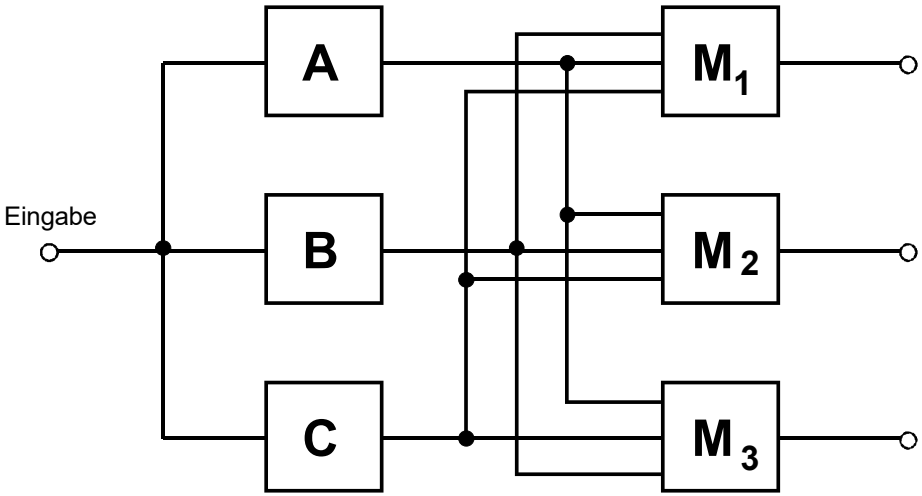
5. Darstellung und Implementierung von Fehlertoleranzverfahren

- Fehlerkorrektur erfordert selbst bei längeren Codes nur geringen Zusatzaufwand;
- Syndromberechnung ist dagegen komplexer, insbesondere wenn auch Mehrfachfehler erkennbar bzw. korrigierbar sein sollen;
- Insgesamt ist Codierung eine effektive Redundanzmaßnahme auf den unteren Schichten (Rechnerprotokolle, TK-Anlagen, Fertigungsprozesse, Übertragungsnetze etc.);
- In praktischen Anwendungen werden der Coderedundanz oft noch weitere Redundanzmaßnahmen überlagert (z. B. Strukturredundanz in Form einer Vervielfältigung bestimmter Systemkomponenten);

  $\Rightarrow$ durchgängige Absicherung auf unterschiedlichen Systemebenen!

  Beispiel: (2-von-3)-System in Verbindung mit Mehrheitsbildner

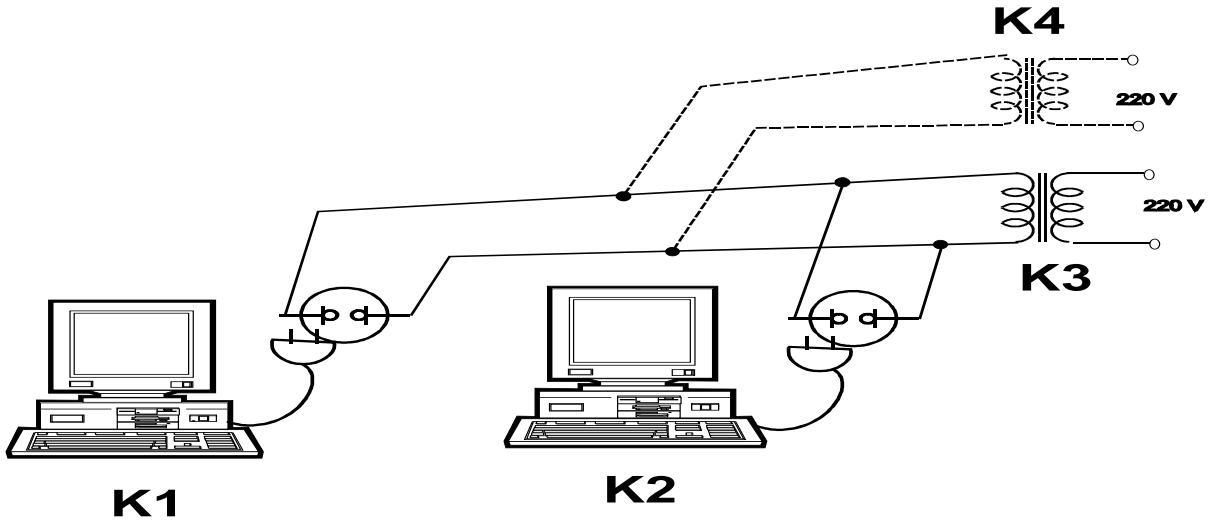
2-von-3-Systemanordnung mit redundantem Mehrheitselement:



- Auch ohne explizite Fehlerkorrektur am Ausgang des Mehrheitsbildners das korrekte Ergebnis
- Zusätzliches Signalisieren, wenn Abweichung vorliegt (Fehleranzeige)
- Zusätzliches Fehlerdiagnoseelement identifiziert ausgefallene Komponente (Fehlerbehandlung)

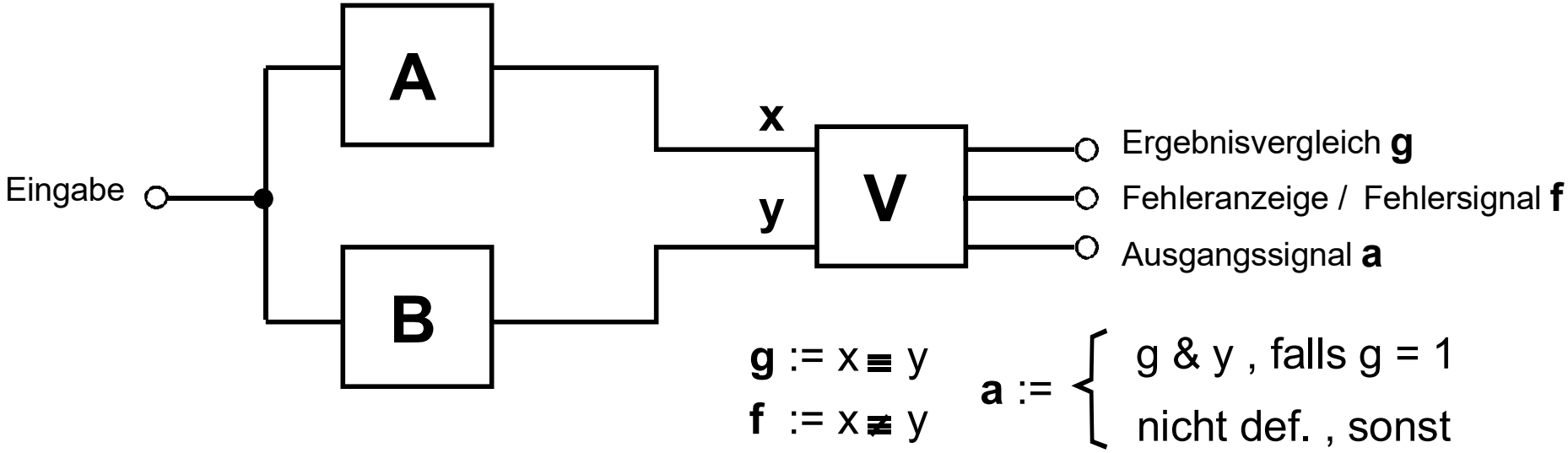
5. Darstellung und Implementierung von Fehlertoleranzverfahren

Sicherheitsanwendungen:



- Funktionsunterbrechung, Stillstand oder gezielte Abschaltung oft besser als fehlerhafte Betriebsfortsetzung
- Nur Abweichung zum Ausdruck bringen und dann gezielte Fehlerbehandlung einleiten.

2-von-2-Systemanordnung mit Vergleichselement:



Durch Kombination der beiden Prinzipien **Mehrheitsbildner** und **Ergebnisvergleich** kann sowohl die Sicherheit als auch die Verfügbarkeit erhöht werden.

5. Darstellung und Implementierung von Fehlertoleranzverfahren

- In Rechnersystemen bieten sich Realisierungsalternativen durch Hard- und Software an;
- Wichtig hierbei die Unterscheidung bzgl. der Effektivität und Optimalität der Fehlertoleranz bei gegebenem Aufwand:
  - Menge der tolerierbaren Fehler
  - Auslastung im fehlerfreien Fall
  - Abschätzung der Fehlerwahrscheinlichkeiten
  - Fehlerfortpflanzung etc.
- Zu beachten sind auch die Randbedingungen bezüglich Transparenz, Latenz, Performance, Komplexität, Kosten etc.

Lassen sich Fehler nicht verhindern, so ist die Schaffung einer gemäßigten Fehlerauswirkung ein gangbarer Weg!

Maßnahmen zur Isolierung, Eindämmung und Fehlerentkopplung betreffen die Fehlertolerierung im **laufenden** Betrieb;

- spezielle Entkopplungseinrichtungen können etwaige Fehlerfortpflanzungen unterdrücken;
- die Fehlereindämmung (engl. fault containment) verhindert, dass sich ein Fehler über gewisse Grenzen ausbreitet;
- spezielle Trenneinheiten können die Isolierung der Komponenten voneinander vornehmen;
- obige Maßnahmen erleichtern die Fehlererkennung (Identifikation) Fehlerlokalisierung und Reparatur.

- Rückwirkungsfreiheit durch hochohmige Entkopplung oder galvanische Trennung (z. B. Optokoppler, FET, OP);
- Störung am Ausgang einer Komponente darf den eigenen Eingang nicht beeinflussen;
- räumliche Trennung oder Abschirmung zur Vermeidung von eingestreuten Störungen (EMV);
- Prüfmarken und Prüfsummen bzw. Signaturen zur Integritätswahrung und Unversehrtheit im laufenden Betrieb.

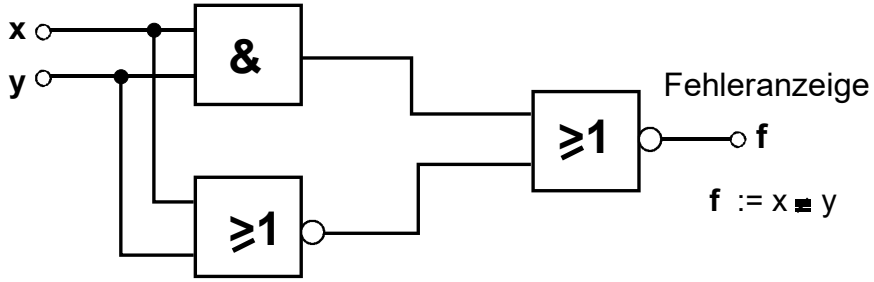
Hardware-Implementierung:

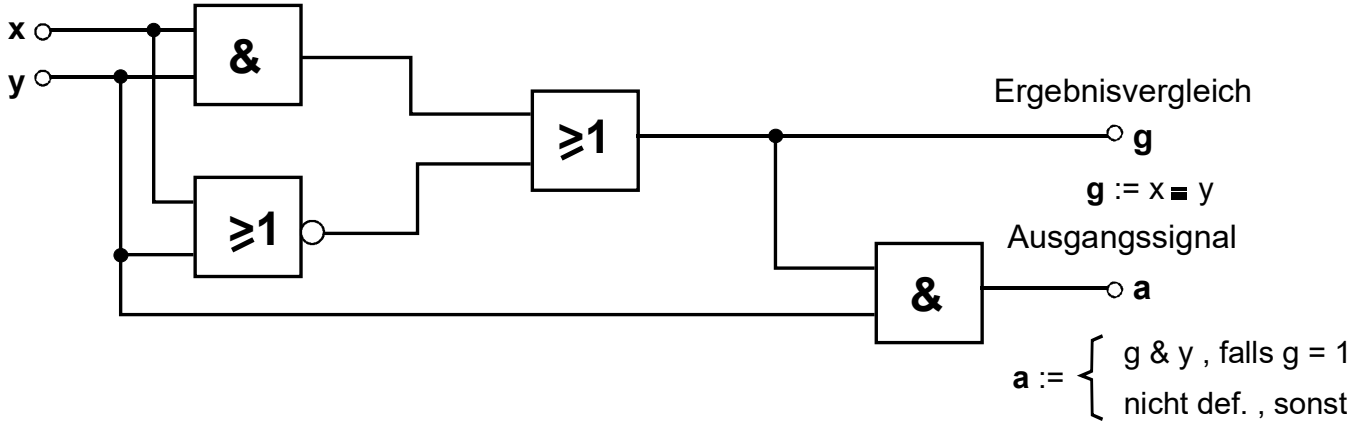
- Zuvor diskutierte Prinzipien Mehrheitsbildung und Ergebnis-vergleich sollen nun im Hinblick auf deren Implementierung vertieft werden.
- Dabei wird von logischen (booleschen) Signalen mit den Werten 0 und 1 ausgegangen.

Lösungsmöglichkeiten:

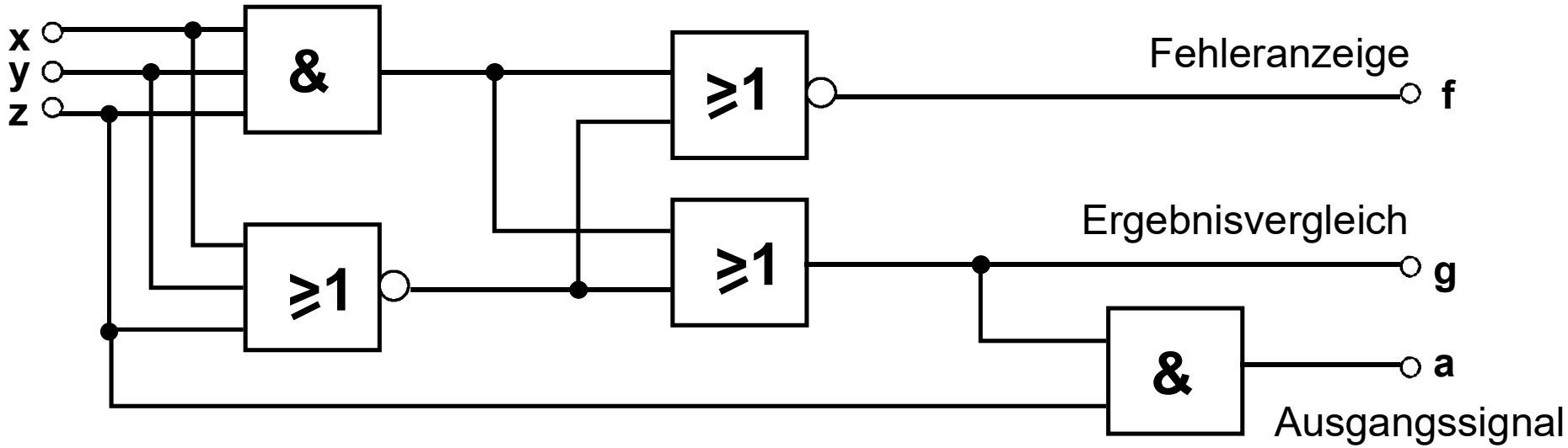
- 2-von-2-Vergleichssystem (Überwachungssystem)
- 3-von-3-Vergleichssystem in Kombination mit Mehrheitsbildung
- gegenseitige Funktionsüberwachung
- selbsttestbare Schaltungen

Schaltungsrealisierung:





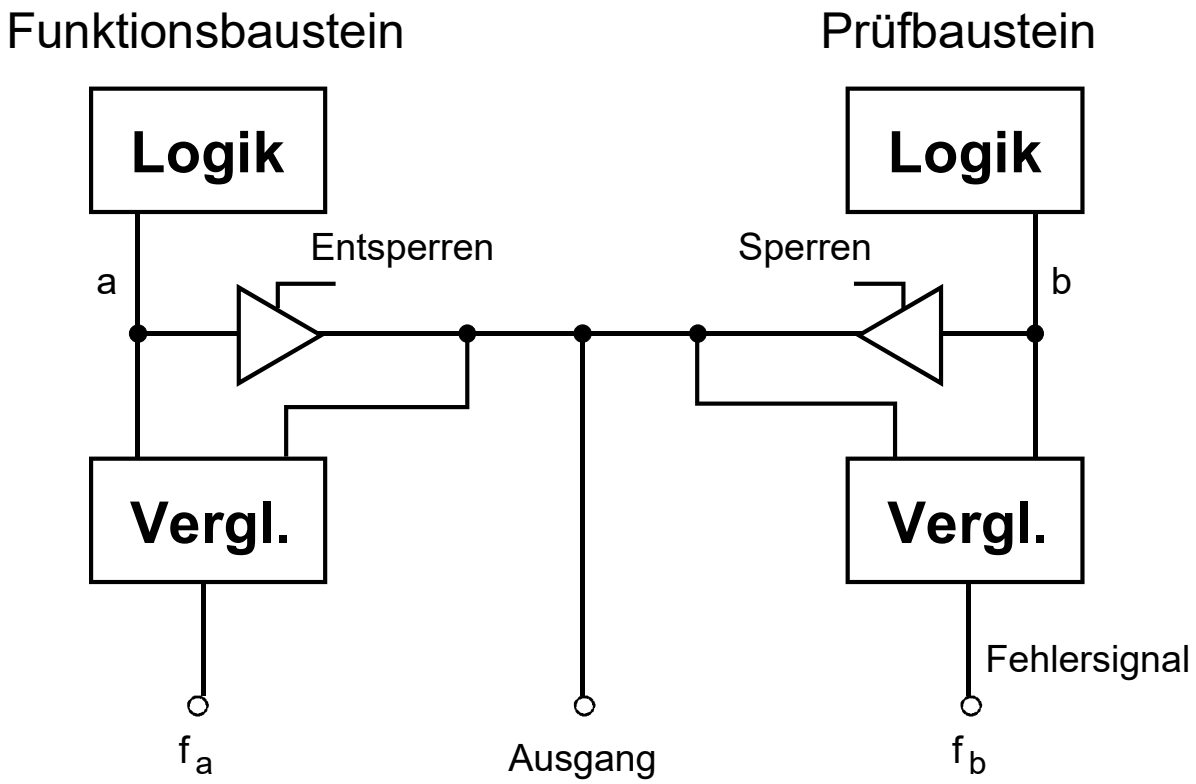
Schaltungsrealisierung:



Mikroprozessorsystem iAPX 432 der Fa. Intel:

- jeweils zwei Grundelemente: **Funktionsbaustein und Prüfbaustein**
- Funktionsbaustein und Prüfbaustein bilden die gleiche logische Funktion;
- ferner je ein Vergleicher und sperrbarer Buspuffer;
- Funktionsbaustein zunächst entsperrt & Prüfbaustein gesperrt
  $\Rightarrow$ Funktionsbaustein vergleicht sein eigenes Ergebnis!
  $\Rightarrow$ Im fehlerfreien Fall keine Fehleranzeige.
- Tritt Fehler auf, so reagiert Baustein mit einer Fehleranzeige. Dabei nicht erkennbar, welcher Baustein defekt ist!

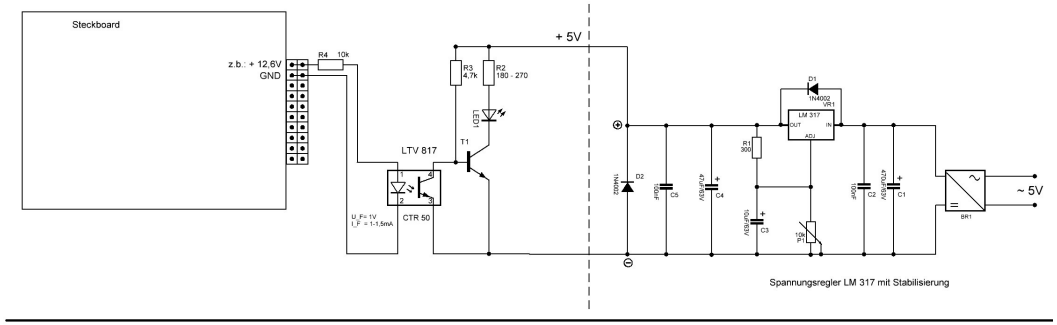
Schaltungsprinzip:

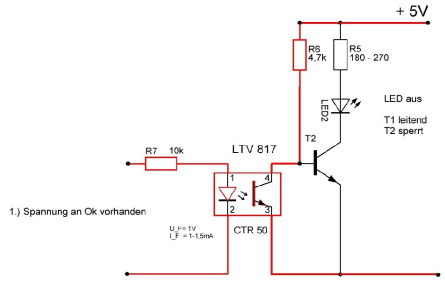


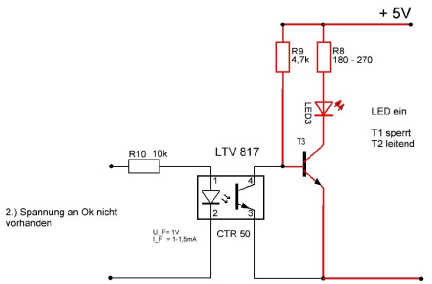
Schaltungsrealisierung:

Zwei Betriebszustände einer Steckkarte mit Spannungsüberwachung.

- Spannung am Optokoppler-Eingang vorhanden (Fototransistor ist leitend – die Anzeige-**LED ist aus**)
- Spannung am Optokoppler-Eingang nicht vorhanden bzw. fällt aus (Fototransistor sperrt, die Anzeige-**LED leuchtet**)





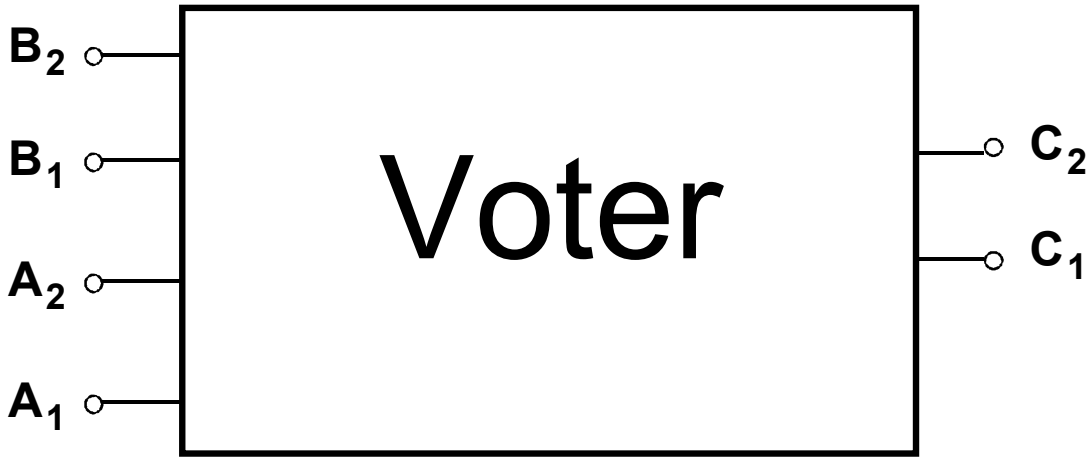


Fehleranzeige umfasst nicht nur die fehlerhaften Eingabedaten, sondern auch die Funktionsfähigkeit der Schaltung.

- Selbsttestbarkeit: bei mindestens einem gültigen Eingabewert wird ein ungültiger (**nicht** fehlerhafter) Ausgabewert erzeugt.
- Fehlersicherheit: trotz fehlerhafter Eingabewerte werden niemals falsche Ausgabewörter erzeugt.
- Totale Selbsttestbarkeit: liegt vor, wenn beide Eigenschaften, d. h. Selbsttestbarkeit und Fehlersicherheit, erfüllt sind.
- Partielle Selbsttestbarkeit: liegt vor, wenn die Fehlersicherheit nur für eine Untermenge von gültigen Eingabewörtern erfüllt wird.

Beispiel:

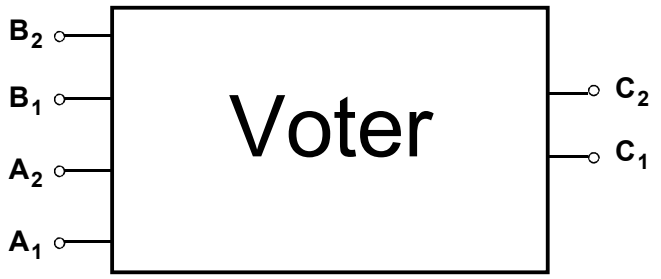
- Zweikanaliger Vergleicher (cp steht für compare) $\boldsymbol{A}\ cp\ \boldsymbol{B} \rightarrow \boldsymbol{C}$ mit $\boldsymbol{X} = (\mathbf{X}_1, \mathbf{X}_2)$ für $\boldsymbol{X} = \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$.



Mit den beiden Eingabewörtern **A** und **B** sowie dem Ausgabewort **C**.

Voter-Funktionalität:

$$\boldsymbol{A}\ cp\ \boldsymbol{B} \rightarrow \boldsymbol{C}$$



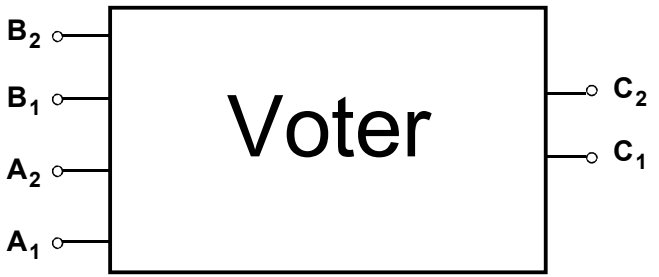
- Dabei werden für die drei Binärsignale **A**, **B** und **C** vereinbart:
  - falls **A**, **B**, **C** = 0 1 oder 1 0 $\Rightarrow$ der Wert ist gültig
  - falls **A**, **B**, **C** = 0 0 oder 1 1 $\Rightarrow$ der Wert ist ungültig

und demnach:

$$A\ cp\ B \rightarrow C : \left\{ \begin{matrix} a_1 = b_1 \\ a_1 \neq b_1 \end{matrix} \right\} und \left\{ \begin{matrix} a_2 = b_2 \\ a_2 \neq b_2 \end{matrix} \right\} \rightarrow \left\{ \begin{matrix} 0\,1 \\ 1\,0 \end{matrix} \right\}$$

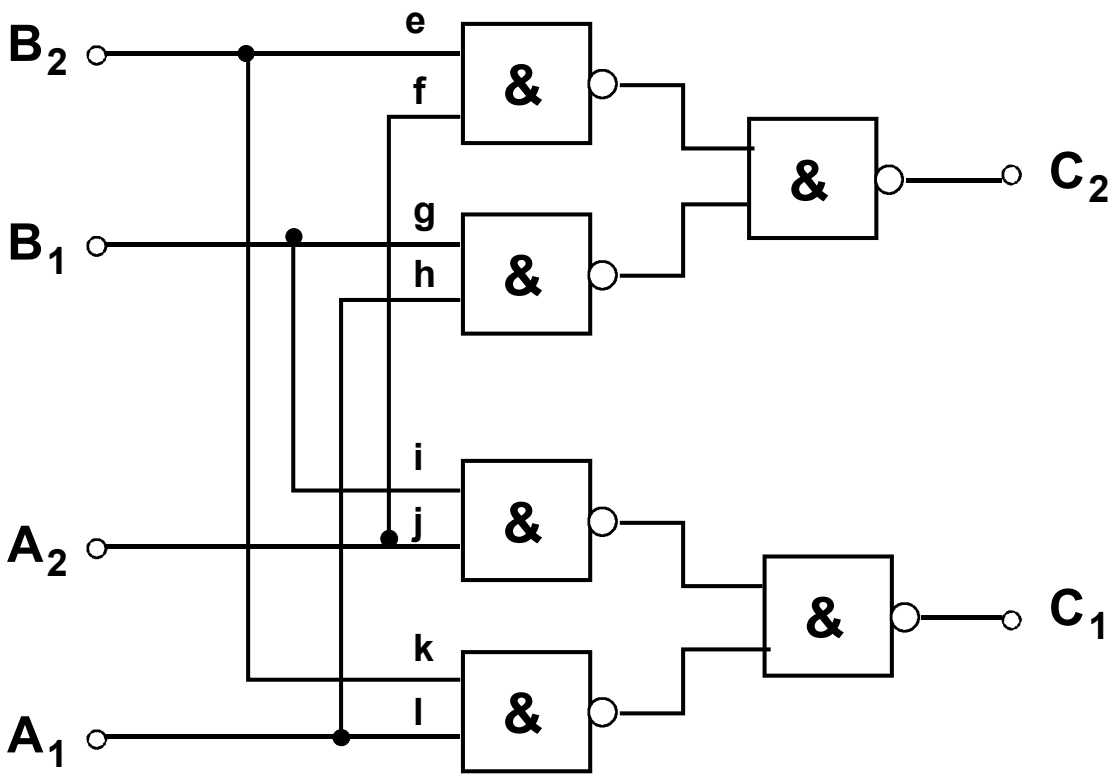
Voter-Funktionalität:

$$\boldsymbol{A}\ cp\ \boldsymbol{B} \rightarrow \boldsymbol{C}$$



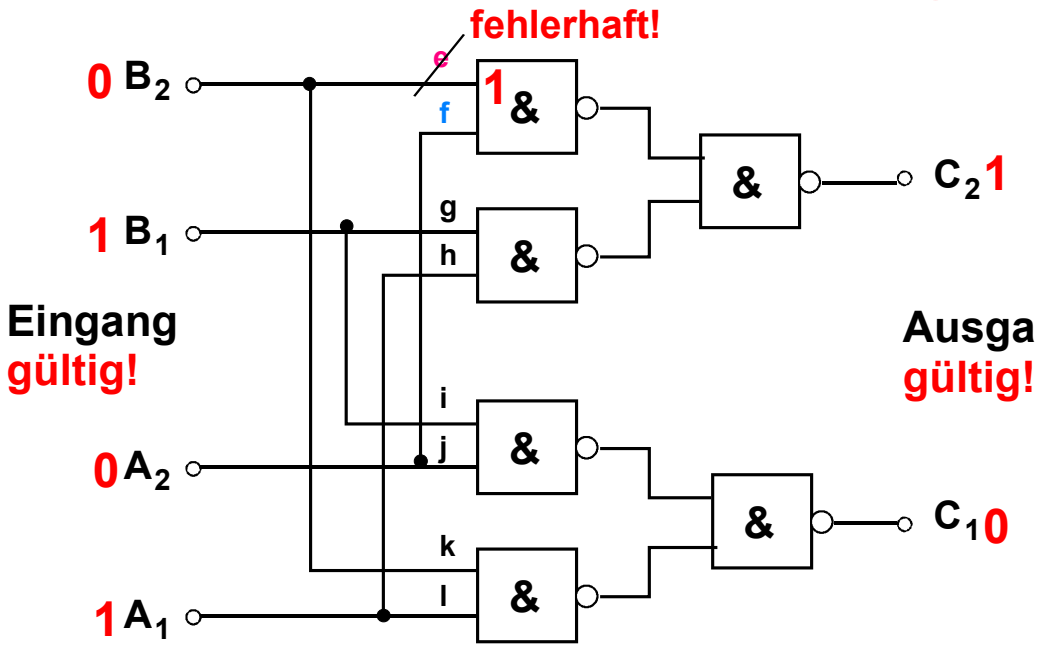
- Die Funktion der Schaltung wird nach der folgenden Beziehung gebildet:
  - $\mathbf{C}_1 = \mathbf{A}_2 \cdot \mathbf{B}_1 \vee \mathbf{A}_1 \cdot \mathbf{B}_2$
  - $\mathbf{C}_2 = \mathbf{A}_1 \cdot \mathbf{B}_1 \vee \mathbf{A}_2 \cdot \mathbf{B}_2$
- Die Schaltung lässt sich einheitlich aus **NAND**-Verknüpfungsglieder aufbauen.

Schaltungsrealisierung:



Fehlerbetrachtung:

Fehlerquelle: „**Signal e**“ fehlerhaft!

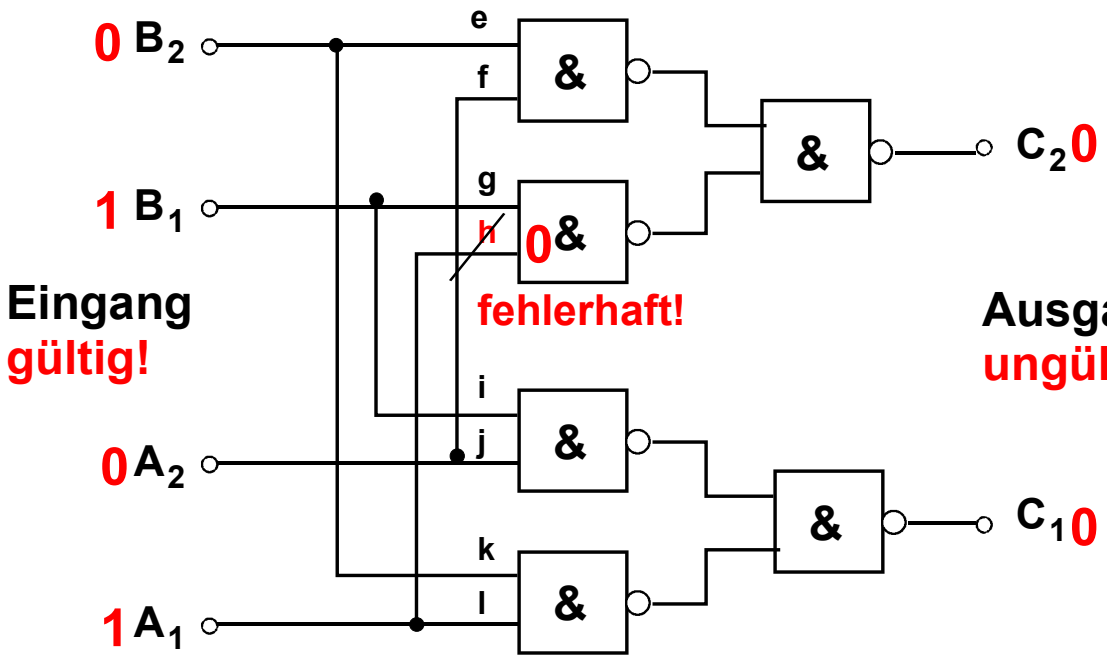


Egal, ob **Signal e** 0 oder 1, da **Signal f** 0 ist (NAND-Verknüpfung → Ausgang immer 1)

Ergebnis: Unerheblicher Fehler (ist a priori isoliert)

Fehlerbetrachtung:

Fehlerquelle: „**Signal h**“ fehlerhaft!

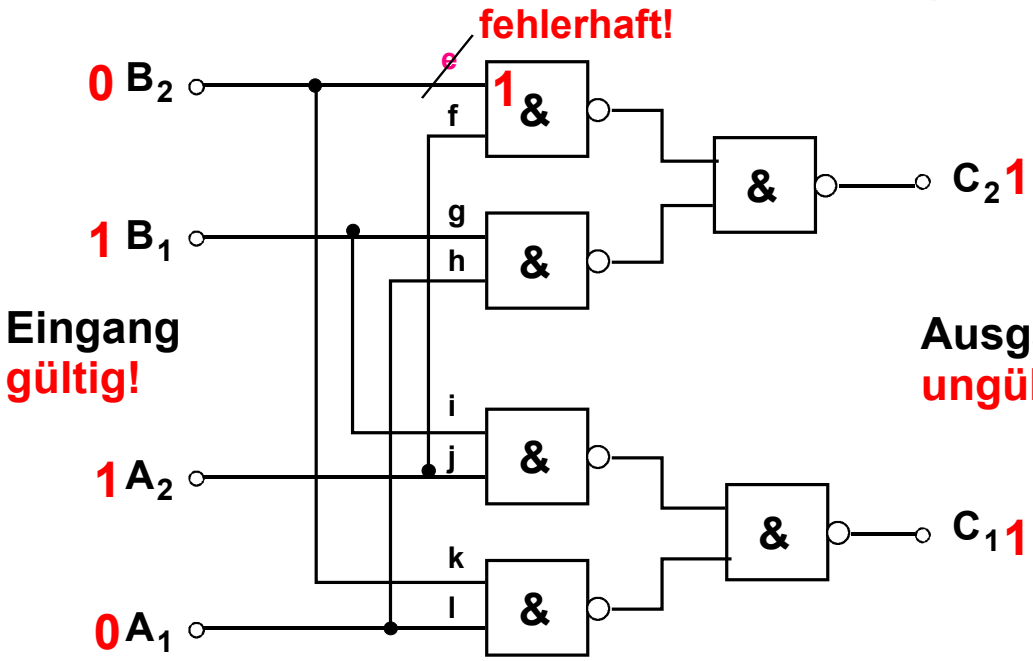


Ein fehlerhaftes **Signal h** ist jetzt signifikant und wird erkannt.

Ergebnis: Unzulässiger Ausgang

Fehlerbetrachtung:

Fehlerquelle: „**Signal e**“ fehlerhaft!



Ein fehlerhaftes **Signal e** ist jetzt signifikant und wird erkannt.

Ergebnis: Unzulässiger Ausgang

Wertetabelle (Fehlerhaftigkeit):

A_1	A_2	B_1	B_2		C_1	C_2		
0	1	0	1		0	1	**A = B**	**gültige Ein- und Ausgänge!!!**
0	1	1	0		1	0	**A ≠ B**	
1	0	0	1		1	0	**A ≠ B**	
1	0	1	0		0	1	**A = B**	
0	0	0	0		0	0	**A fehlerhaft** 00, B beliebig	f oder j oder h oder l ständig 0
0	0	0	1		0	0		
0	0	1	0		0	0		
0	0	1	1		0	0		
0	1	0	0		0	0	**A gültig** **B fehlerhaft** 00 oder 11	g oder i oder e oder k ständig 0 oder ständig 1
0	1	1	1		1	1		
1	0	0	0		0	0		
1	0	1	1		1	1		
1	1	0	0		0	0	**A fehlerhaft** 11 B beliebig	f oder j oder h oder l ständig 1
1	1	0	1		1	1		
1	1	1	0		1	1		
1	1	1	1		1	1		

Ergebnis:

- Schaltung reagiert auf fehlerhafte Eingabewörter mit einer ungültigen (jedoch nicht fehlerhaften) Ausgabe. Gültige Eingaben führen auch zu gültigen Ausgabewerten.
- Ist eine Variable e bis l **ständig 0** (sog. Haftfehler), so folgt: **C** = 0 0 $\Rightarrow$ fehlerhafter Ausgangszustand wird signalisiert.
- Ist eine Variable e bis l **ständig 1**, so folgt: **C** = 1 1 $\Rightarrow$ entsprechender Eingangsfehler kann sich nicht auswirken, d. h. auch hier wird fehlerhafter Ausgangszustand signalisiert.

Da **alle** Fehler (hier: Einzelschaltungsfehler) angezeigt werden bzw. erkennbar sind, ist die Schaltung **selbsttestbar**. Da **alle** Fehler einen ungültigen Eingang vortäuschen, ist die Schaltung auch **fehlersicher** und damit insgesamt **total selbsttestbar**.