Übungsblatt 11
Rechnerarchitektur im SoSe 2020

Zu den Modulen N

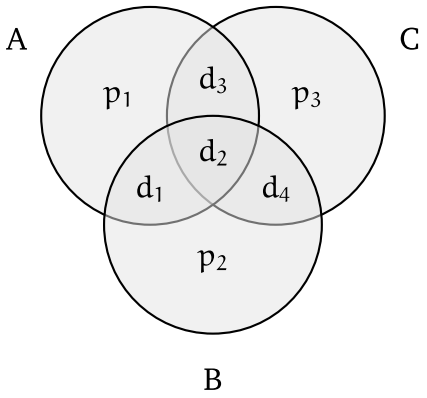
Besprechung: Besprechung der Übungsaufgaben in den Übungsgruppen vom 13. – 17. Juli 2020

Aufgabe Ü23: Fehlererkennung und -korrektur

(– Pkt.)

Zum Schutz vor Speicherfehlern werden sogenannte Codes zur Fehlererkennung und zur Fehlerkorrektur eingesetzt. Bearbeiten Sie die folgenden Teilaufgaben:

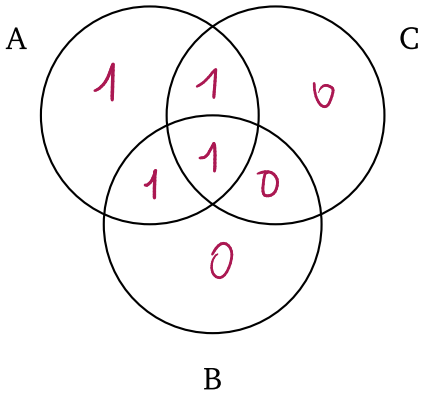
a. Wir gehen von folgender Struktur der Code-Wörter $d_1 d_2 d_3 d_4 p_1 p_2 p_3$ aus. Wobei $d_i (i \in \{1,2,3,4\})$ für das jeweilige Datenbit und $p_j (j \in \{1,2,3\})$ für das jeweilige Prüf- bzw. Paritätsbit steht. Die Paritätsbits zur Fehlererkennung bzw. Fehlerkorrektur für ein Datenwort $d_1 d_2 d_3 d_4$ können anschaulich mit Hilfe eines Venn-Diagramms berechnet werden, in welchem die Bits wie folgt angeordnet sind:



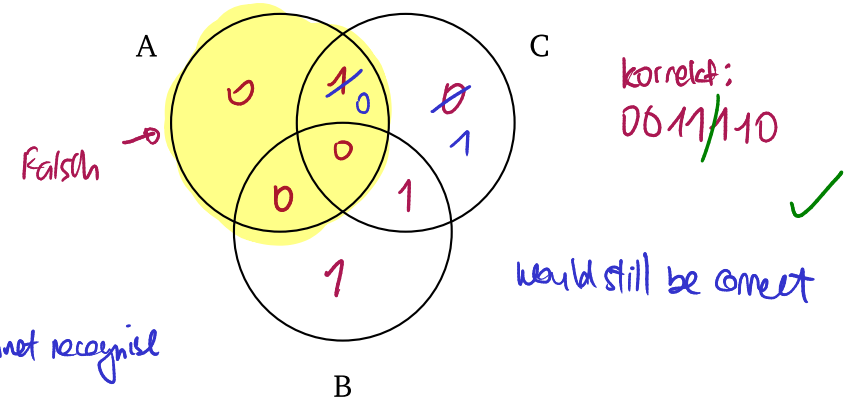
p -> paritätsbits
prüfbits

(i) Berechnen Sie unter Verwendung des folgenden Venn-Diagramms die Prüfbits für das Datenwort **1110**. Verwenden Sie dazu **gerade Parität**. Tragen Sie zunächst die Datenbits in die für die Berechnung sinnvollen (Schnitt-)Mengen ein.

1110100



(ii) Gehen Sie nun davon aus, dass Sie ein mit dem zuvor beschriebenen Code codiertes Code-Wort `0011010` empfangen haben. Es wurde **gerade Parität** verwendet. Handelt es sich um ein gültiges Codewort? Falls nein, treffen Sie eine Aussage darüber, an welcher/welchen Stelle/Stellen mutmaßlich (ein) Bitfehler aufgetreten ist/sind. Verwenden Sie zur Berechnung das folgende Venn-Diagramm. Korrigieren Sie (falls möglich/nötig) den/die Fehler **innerhalb** des Venn-Diagramms und **geben Sie das** (ggf. korrigierte) **4-Bit Datenwort an**.



if 2 Fehlerauftreten ⇒ we cannot recognise

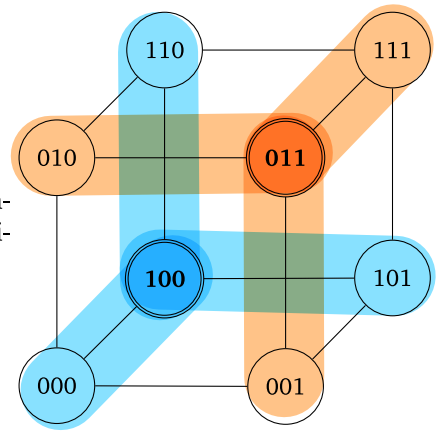
(iii) Wie ist beim vorliegenden Fehlererkennungs- bzw. Fehlerkorrekturcode der Overhead der Prüfbits bzgl. der Speicherbits in Prozent (%)?

$\frac{\#\text{Prüf}}{\#\text{Daten}} = \frac{3}{4} = 75\%$ ✓

b. In dieser Teilaufgabe sei ein Fehlerkorrekturcode gegeben, der nur aus **zwei gültigen Codewörtern** besteht:

- `100`
- `011`

Hamming-Abstand = 3



each kanten = 1 Bit change

In der Abbildung rechts sind alle möglichen Kombinationen von 3 Bit eingezeichnet und die gültigen Codewörter markiert.

Bearbeiten Sie dazu die folgenden Aufgaben:

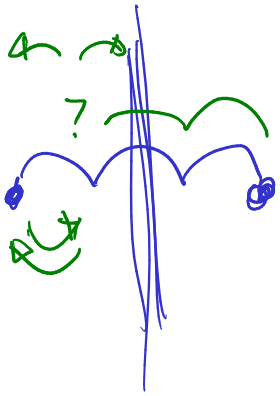
(i) Wie viele Einzelbitfehler können mit dem vorliegenden Code in jedem Fall **erkannt** werden? = e

(ii) Wie viele Einzelbitfehler können mit dem vorliegenden Code in jedem Fall **korrigiert** werden? = k

(i) 2 Einzelbitfehler (ii) 1 Einzelbitfehler

we think about how many bits then we cannot erkennen the wrongness

⇒ 3 Bits (will become another correct code)



(i) $d = e + 1$

(ii) $d = 2k + 1$

Aufgabe Ü24: Hamming Codes

(– Pkt.)

Übertragung von Daten über physische Kanäle (Kabel etc.) ist fehleranfällig. Als Schutz vor solchen Fehlern setzen die meisten Speicher Codes für die Fehlererkennung und ggf. auch zur Fehlerkorrektur ein. Lesen Sie sich im Vorlesungsskript das Kapitel 14.4 zur „Fehlererkennung und -korrektur“ (S. 173-176) aufmerksam durch und bearbeiten Sie die folgenden Aufgaben:

a. Kodieren Sie die folgenden 16-Bit Daten in 21-Bit Hamming Code. Verwenden Sie dazu gerade Parität.

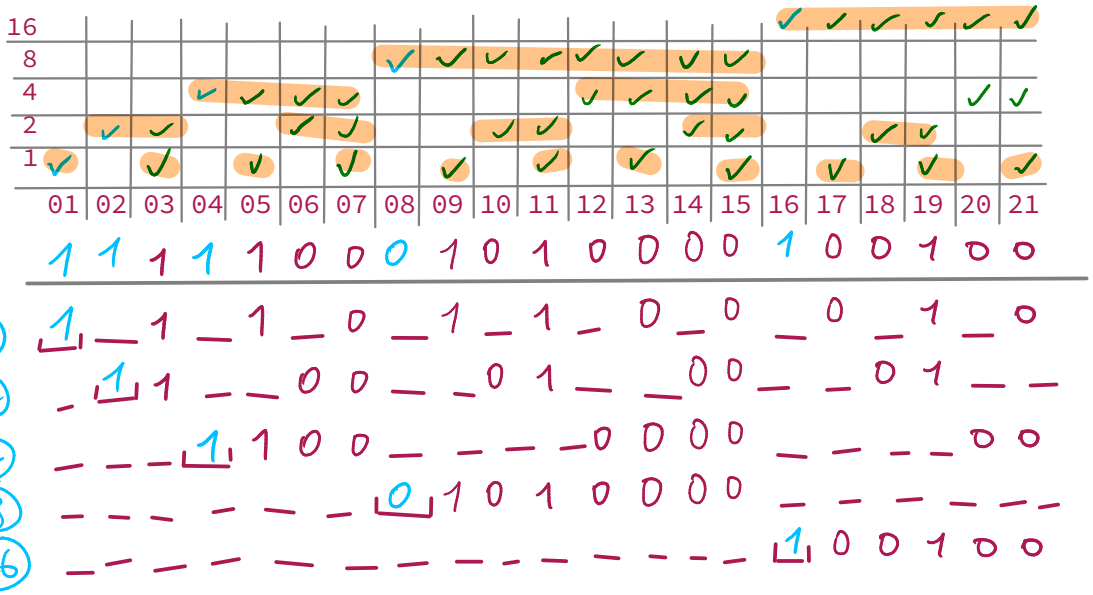
   (i) 1100 1010 0000 0100

b. Dekodieren Sie das folgende 21-Bit Codewort. Wenn Sie Fehler enthalten, identifizieren Sie das fehlerhafte Bit und korrigieren Sie den Fehler. Das Ergebnis muss ein 16-Bit Datenwort sein.

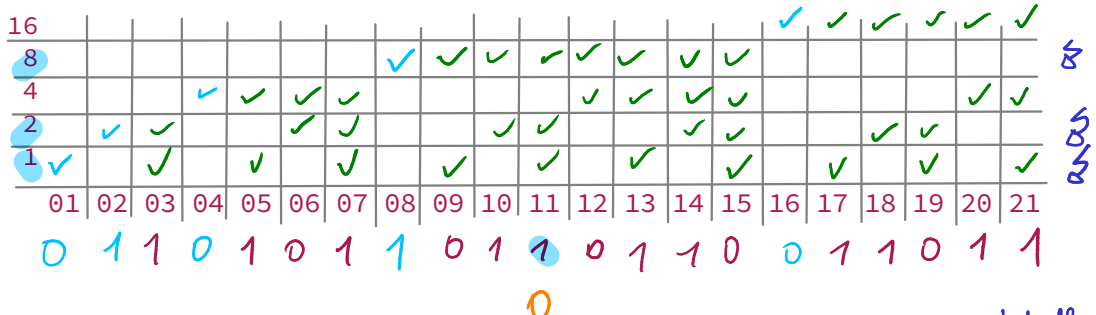
   (i) 0110 1011 0110 1100 1101 1

Prinzip: Jede Zahl kann eindeutig dargestellt werden ⇒ 2-er Potenzen sind Paritätsbits

(a)



(b)



Ergebnis: 1101 0100 1101 1011

⇒ Super skalierbar! ⇒ 10 Prüfbit für 512 Datenbits

2 betroffen → 1 betroffen

1 0 1 1

if we stopped early