Test und Verlässlichkeit Grosse Übung zu Foliensatz 3

Prof. G. Kemnitz

28. Juni 2018

Contents

1 Überwachung und Test

Aufgabe 3.1: Wiederholungsaufgaben

1. Wie lauten die drei Ebenen zur Sicherung der Verlässlichkeit?
2. Warum sind hohe Erkennungswahrscheinlichkeiten für potentieller Probleme (Fehler, Fehlfunktionen, ...) wichtiger für die Sicherung der Verlässlichkeit als hohe Beseitigungswahrscheinlichkeiten für erkannte Probleme?

Zur Kontrolle

1. Die drei Ebenen zur Sicherung der Verlässlichkeit:
   - Fehlervermeidung durch Beseitigung/Minderung der Entstehungsursachen für Fehler
   - Iteration aus Test und Fehlerbeseitigung.
   - Funktionsüberwachung und im Falle erkannter FF Übergang in einen gefahrenfreinen Zustand, Sicherung von Daten für die Fehlerlokalisierung und Fortsetzung der Arbeit, ... Maximale Ausbaustufe Fehlertoleranz (selbstständige Beseitigung der FF durch das System).
2. Die Problembeseitigung erfolgt in einer Iteration bis das Problem nicht mehr erkennbar ist. Geringe Beseitigungswahrscheinlichkeit erhöht die Anzahl der Beseitigungsversuche, aber nicht primär das Risiko der Nichtbeseitigung. Bei Beseitigungsversuchen können allerdings neue, möglicherweise nicht erkennbare Probleme entstehen.

Aufgabe 3.2: Statische und dynamische Tests

Einer Programmieraufgabe in ein Praktikum wird folgenden Test unterzogen:

1. Erstellen der Aufgabenstellung: Korrekturlesen durch den Lehrenden.
2. Beispielimplementierung durch den Lehrenden, Syntaxtest.
3. Beispiele ausprobieren.
4. Lösungssuche durch den Studierenden und Diskussion über die Lösung mit dem Praktikumspartner.
5. Programmieren und Syntaxtest,
6. Ausprobieren.
7. Praktikumsbetreuer schaut sich den Code an.
8. Praktikumsbetreuer lässt sich Beispiel vorführen.

Welche der Tests sind statisch und welche dynamisch?

Zur Kontrolle

1. Erstellen der Aufgabenstellung: Korrekturlesen durch den Lehrenden (S).
2. Beispielimplementierung durch den Lehrenden, Syntaxtest (S), Beispiele ausprobieren (D).
3. Lösungssuche durch den Studierenden und Diskussion über die Lösung mit dem Praktikumspartner (S).
4. Programmieren und Syntaxtest (S), Ausprobieren (D).
5. Praktikumsbetreuer schaut sich den Code an (S).
6. Praktikumsbetreuen lässt sich Beispiel vorführen (D).

(S – statisch, D – dynamisch).

Aufgabe 3.3: Scheinbare und tatsächliche Zuverlässigkeit, Erkennungs- und Phantom-FF-Wahrscheinlichkeit

Bei der Kontrolle von 10^5 SL sind 10^3 FF aufgetreten, von denen 600 FF erkannt wurden. Darüber hinaus wurden 10 SL als FF ausgewiesen, die in Wirklichkeit korrekt ausgeführt wurden. Welche Schätzwerte ergeben sich daraus für

1. die beobachtete Zuverlässigkeit,
2. die tatsächliche Zuverlässigkeit,
3. die Erkennungswahrscheinlichkeit der Kontrolle,
4. die Maskierungswahrscheinlichkeit der Kontrolle,
5. die Phantom-FF-Wahrscheinlichkeit?

Zur Kontrolle

1. Beobachtete Zuverlässigkeit:
$$Z_{\text{Beob}} \approx \frac{10^5\,\text{SL}}{610\,\text{FF}} = 164\,\frac{\text{SL}}{\text{FF}}$$
2. Tatsächliche Zuverlässigkeit:
$$Z \approx \frac{10^5\,\text{SL}}{10^3\,\text{FF}} = 100\,\frac{\text{SL}}{\text{FF}}$$
3. Erkennungswahrscheinlichkeit der Kontrolle:
$$p_{\text{E}} \approx \frac{600}{1000} = 60\%$$
4. Maskierungswahrscheinlichkeit der Kontrolle:
$$p_{\text{M}} = 1 - p_{\text{E}} \approx 40\%$$
5. Phantom-FF-Wahrscheinlichkeit:
$$p_{\text{Phan}} \approx \frac{10}{10^5} = 10^{-4}$$

Aufgabe 3.4: Sicherheitserhöhung durch Kontrollen

Bei einem IT-System mit einer mittleren Zeit zwischen zwei FF (Fehlfunktionen) von $MTBF_{\text{Z}} = 10^3$ h, Service-Dauer 1 h, gefährde abschätzungsweise jede hundertste FF die Betriebssicherheit. Um die Betriebssicherheit auf $10^6\,\frac{\text{SL}}{\text{GFF}}$ zu erhöhen, soll das System um eine Funktionsüberwachung erweitert werden, die es bei Erkennen einer Fehlfunktion in einen sicheren Zustand überführt.

1. Wie hoch muss die Erkennungswahrscheinlichkeit sein, wenn beim Überführen in den sicheren Zustand keine Fehlfunktionen auftreten?
2. Wie hoch muss die Erkennungswahrscheinlichkeit sein, wenn zu erwarten ist, dass jeder 20te Versuch, einen sicheren Zustand herzustellen, scheitert?
3. In welchem mittleren zeitlichen Abstand wird überschlagsweise ein sicherer Zustand hergestellt, ohne dass die Betriebssicherheit gefährdet ist?

Zur Kontrolle

Zur Erhöhung der Sicherheit von $Z_{\text{S}} \approx 10^5$ SL/GFF auf $Z_{\text{S}} \approx 10^6$ SL/GFF muss das System im Mittel bei 9 von 10 FF in den sicheren Zustand versetzt werden.

1. Wenn jeder Versuch erfolgreich ist, genügt es, 9 von 10 (sicherheitskritischen) Fehlfunktionen zu erkennen:
$$p_{\text{E}} = 90\%$$
2. Wenn jeder 20-te Versuch scheidert, dann müssen 19 von 20 (sicherheitskritischen) Fehlfunktionen erkannt werden:
$$p_{\text{E}} = 95\%$$
3. Ein sicherer Zustand wird etwa aller 1000 h hergestellt, in 99% der Fälle für eine ungefährliche FF:
$$\frac{1000\,\text{h}}{99\%} = 1010\,\text{h}$$

2 Informationsredundanz

2.1 Fehlererkennende Codes

Aufgabe 3.5: Arithmetischer Code

1. Bilden Sie für den Bitvektor
$$x = 1100100010000111012$$
das fehlererkennende Codewort durch Multiplikation seines Wertes als vorzeichenfreie ganze Binärzahl mit der Primzahl $c = 10313$ (Bestimmung des Dezimalwerts, Multiplikation und Konvertierung des Produkts in einen Binärvektor).

2. Mit welcher Wahrscheinlichkeit werden mit dem gewählten fehlererkennenden Code Datenverfälschungen des codierten Bitvektors $s = c \cdot x$ erkannt?
3. Werden mit dem gewählten Code Verfälschung von s erkannt, die die Bitstellen 3 und 14 invertieren?

Hinweis: Eine Verfälschung von s ist am Divisionsrest zu erkennen, wenn die Abweichung zum Sollwert $\Delta s = s - s_{\text{soll}}$ kein Vielefachses des Multiplikators c ist.

Zur Kontrolle

1. Codewort berechnen:
   - Eingabewert hexadezimal: 11.0010.0010.0001.1101 = 0x3221D
   - Mit Octave (Matlab) Produkt als hexadezimal:

```
>> printf('CW=0x%x\n',0x3221D*10313)
CW=0x7e394245

binär: 0b111.1110.0011.1001.0100.0010.0100.0101
```

2. Erkennungswahrscheinlichkeit:
$$p_{\text{E}} \approx 1 - \frac{1}{10313} = 99{,}990\%$$
3. Keine Maskierung, wenn Bit 3 und 14 invertiert ist:
$$\text{Rest}(\frac{0\text{b}100.0000.0000.1000}{10313}) \neq 0 \surd$$
   Für Differenzen ungleich null, die kleiner als der Quotient sind, immer erfüllt.

2.2 Prüfkennzeichen

Aufgabe 3.6: Prüfsummen

Bilden Sie für die Bytefolge

```
0x13, 0xF2, 0x33, 0xE6
```

die Prüfsumme:

1. durch byteweises Aufsummieren unter Vernachlässigung der Überträge und
2. durch bitweise EXOR-Verknüpfung der Bytes.

Welche der beiden Prüfsummen erkennt, dass die nachfolgenden Datenfolgen verfälscht sind?

```
F1: 0x13, 0x33, 0xF2, 0xE6
F2: 0x13, 0xF2, 0x37, 0xE6
F3: 0x13, 0xF1, 0x90, 0x56
```

Wert unverf.	(Teil-) Prüfsum.	binär
0x13		
0xF2		
0x33		
0xE6		
	EXOR:	

Wert F1	(Teil-) Prüfsum.	binär
0x13		
0x33		
0xF2		
0xE6		
	EXOR:	

Wert F2	(Teil-) Prüfsum.	binär
0x13		
0xF2		
0x37		
0xE6		
	EXOR:	

Wert F3	(Teil-) Prüfsum.	binär
0x13		
0xF1		
0x90		
0x56		
	EXOR:	

	erkennbar an Prüfsumme	erkennbar an EXOR-Summe
F1		
F2		
F3		

Zur Kontrolle

Wert	(Teil-) Prüfsum.	binär
0x13	0x13	0001 0011
0xF2	0x05	1111 0010
0x33	0x38	0011 0011
0xE6	0x1E	1110 0110
	EXOR:	0011 0100

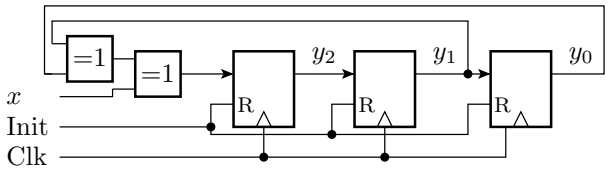
Wert	(Teil-) Prüfsum.	binär
0x13	0x13	0001 0011
0x33	0x46	0011 0011
0xF2	0x38	1111 0010
0xE6	0x1E	1110 0110
	EXOR:	0011 0100

Wert	(Teil-) Prüfsum.	binär
0x13	0x13	0001 0011
0xF2	0x05	1111 0010
0x37	0x3C	0011 0111
0xE6	0x22	1110 0110
	EXOR:	0011 0000

Wert	(Teil-) Prüfsum.	binär
0x13	0x13	0001 0011
0xF1	0x04	1111 0001
0x90	0x94	1001 0000
0x46	0xDA	0100 0110
	EXOR:	0011 0100

Aufgabe 3.7: Prüfkennzeichen mit LFSR

Gegeben ist folgendes linear rückgekoppelte Schieberegister:

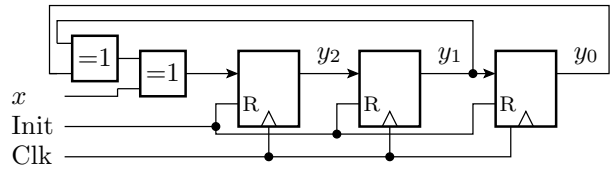


	x	y_2	y_1	y_0
0	1	0	0	0
1	0			
2	1			
3	1			
4	0			
5	0			
6	1			
7	1			
8	0			
9	1			
10	0			
11	0			
12	1			
13	0			
14	1			
15	0			
PKZ:				

1. Auf welches Prüfkennzeichen $\mathbf{y} = y_2 y_1 y_0$ wird die Datenfolge 1011 0011 0100 1010 beginnend mit dem linken Bit und Startwert 000 abgebildet? Füllen Sie dazu die Tabelle in der Abbildung aus.
2. Wie hoch ist Fehlererkennungswahrscheinlichkeit?

Zur Kontrolle

1. Prüfkennzeichen der Datenfolge 1011 0011 0100 1010:



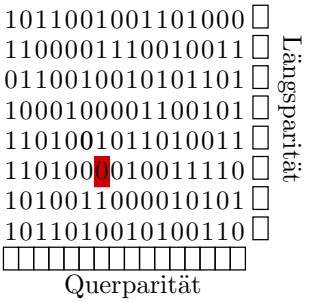
	x	y_2	y_1	y_0
0	1	0	0	0
1	0	1	0	0
2	1	0	1	0
3	1	0	0	1
4	0	0	0	0
5	0	0	0	0
6	1	0	0	0
7	1	1	0	0
8	0	1	1	0
9	1	1	1	1
10	0	1	1	1
11	0	0	1	1
12	1	0	0	1
13	0	0	0	0
14	1	0	0	0
15	0	1	0	0
PKZ:		0	1	0

1. Fehlererkennungswahrscheinlichkeit:

$$p_E \approx 1 - 2^{-3} = 87{,}5\%$$

2.3 Fehlerkorr. Codes

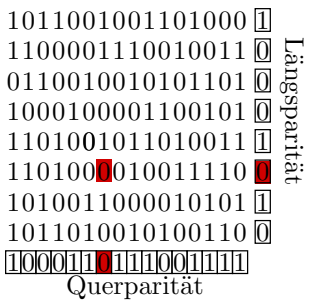
Aufgabe 3.8: Berechnung der Kreuzparität



1. Ergänzen Sie die Bitwerte für die Längs- und Querparität so, dass die Anzahl der Einsen in jeder Zeile und Spalte incl. Paritätsbit gerade ist.
2. Woran ist eine Invertierung des rot unterlegten Bits zu erkennen?

Zur Kontrolle

1. Ergänzte Bitwerte für die Längs- und Querparität:



2. Die Invertierung des rot unterlegten Bits ist an einem Paritätsfehler in Zeile 6 und in Spalte 7 zu erkennen.

2.4 Hamming-Codes

Aufgabe 3.9: (8,12)-Hamming-Code

b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0

$$
\begin{aligned}
q_0 &= x_0 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_6 \\
q_1 &= x_0 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \\
q_2 &= x_1 \oplus x_2 \oplus x_3 \oplus x_7 \\
q_3 &= x_4 \oplus x_5 \oplus x_6 \oplus x_7
\end{aligned}
$$

1. Bilden Sie die Codeworte für die darzustellenden Werte: $w_1 = 0x73$, $w_2 = 0x1D$ und $w_3 = 0xD6$.
2. Bestimmen Sie für die Codeworten $c_4 = 0xA24$, $c_5 = 0x5D6$ und $c_6 = 0x141$, ob zulässig oder korrigierbar und wenn zulässig oder korrigierbar, den Wert.

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0
Kontrollbits												
w_1= 0x73												
w_2= 0x1D												
w_3= 0xD6												
c_4= 0xA24												
c_5= 0x5D6												
c_6= 0x141												

Zur Kontrolle

Bitnummer	12	11	10	9	8	7	6	5	4	3	2	1	
Zuordnung	x_7	x_6	x_5	x_4	q_3	x_3	x_2	x_1	q_2	x_0	q_1	q_0	
Kontrollbits													
w_1= 0x73	0	1	1	1	1	0	0	1	1	1	1	0	c_1= 0x79E
w_2= 0x1D	0	0	0	1	1	1	1	0	0	1	1	1	c_2= 0x1E7
w_3= 0xD6	1	1	0	1	1	0	1	1	1	0	0	1	c_3= 0xDB9
c_4= 0xA24	1	0	1	0	0	0	1	0	0	1	0	0	$dq_4 = 3$
c_5= 0x5D6	0	1	0	1	1	1	0	1	0	1	1	0	$dq_5 = 9$
c_6= 0x141	0	0	0	1	0	1	0	0	0	0	0	1	$dq_6 = 15$

w_4 : Wert 0xA5 mit verfälschtem $x_0 \Rightarrow w_4$ =0xA4
w_5 : Wert 0x5B mit verfälschtem $x_4 \Rightarrow w_5$ =0x4B
w_6 : Wert 0x18 mit verfälschtem »Bit 15«, nicht korrigierbar.

3 Formatkontrolle

3.1 Syntaxtest

Aufgabe 3.10: Entwurf Kontrollautomat

Ein (vereinfachter) Rechnerbefehlssatz besteht aus vier verschiedenen Befehlstypen

```
add␣rr,rr;
addi␣rr,imm8;
sub␣rr,rr;
subi␣rr,imm8;
```

␣ – Leerzeichen; »rr« Bezeichner eines der 32 Register ("r0", "r1", ... "r31"); »imm8« für die Wert einer 8-Bit Hexzahl ("0x00", "0x01", ..., "0xFF"; "0x" gefolgt von zwei Hex.-Ziffern mit den Zifferenwerten '0' bis 'F').

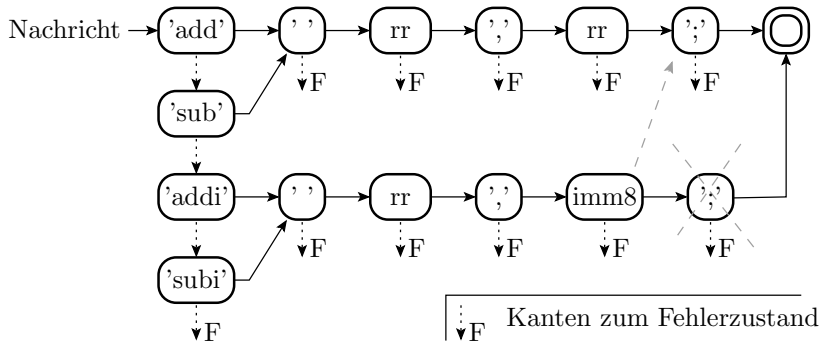
1. Beschreiben Sie das Befehlsformat in der EBNF mit den Ersetzungsregeln für Sequenz, Option, Wiederholung etc.
2. Entwerfen Sie einen deterministischen Kontrollautomaten auf Syntaxfehler als Graph für einen Moore-Automaten.

Zur Kontrolle

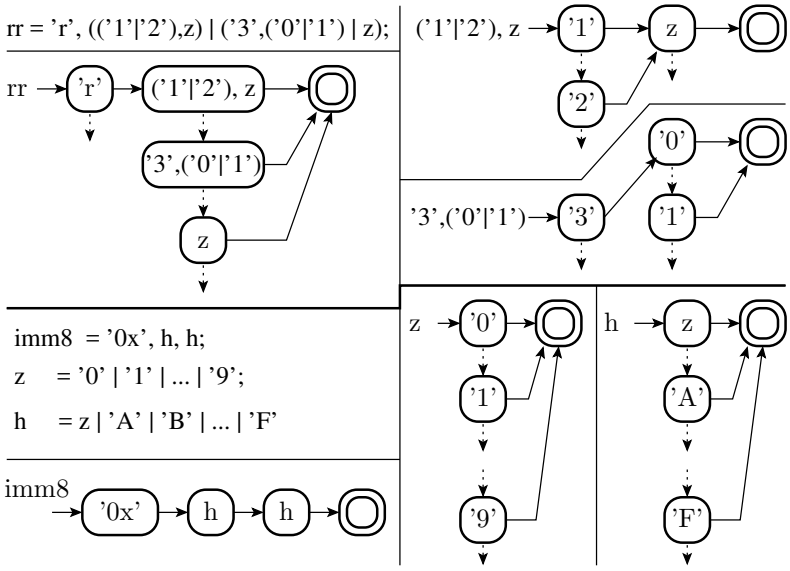
1. EBNF-Beschreiben:

```
Befehl = ('add' | 'sub', '␣', rr, ',', rr,   ';') |
         ('addi'| 'subi','␣', rr, ',', imm8, ';');
rr     = 'r', (('1'|'2'),z) | ('3',('0'|'1') | z);
imm8   = '0x', h, h;
z      = '0' | '1' | ... | '9';
h      = z | 'A' | 'B' | ... | 'F'
```

2. Moore-Automat für den Test auf Syntaxfehler:



Teilautomaten für den Test der Befehlsbestandteile:

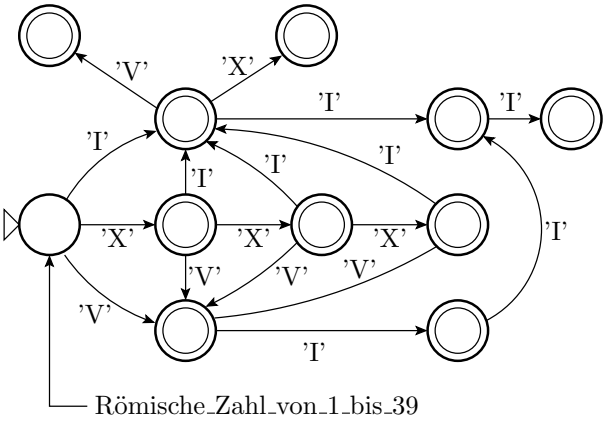


Aufgabe 3.11: Syntaxtest für römische Zahlen

Entwerfen Sie einen Mealy-Kontrollautomaten[1] für einen Syntaxtest für römische Zahlen mit einem Wert von 1 bis 39.

Wert		Wert		Wert		Wert	
1	I	11	XI	21	XXI	31	XXXI
2	II	12	XII	22	XXII	32	XXXII
3	III	13	XIII	23	XXIII	33	XXXIII
4	IV	14	XIV	24	XXIV	34	XXXIV
5	V	15	XV	25	XXV	35	XXXV
6	VI	16	XVI	26	XXVI	36	XXXVI
7	VII	17	XVII	27	XXVII	37	XXXVII
8	VIII	18	XVIII	28	XXVIII	38	XXXVIII
9	IX	19	XIX	29	XXIX	39	XXXIX
10	X	20	XX	30	XXX		

Zur Kontrolle



I	XI	XXI	XXXI
II	XII	XXII	XXXII
III	XIII	XXII	XXXII
IV	XIV	XXIV	XXXIV
V	XV	XXV	XXXV
VI	XVI	XXVI	XXXVI
VII	XVII	XXVII	XXXVII
VIII	XVIII	XXVIII	XXXVIII
IX	XIX	XXIX	XXXIX
X	XX	XX	

(potentieller) Endzustand

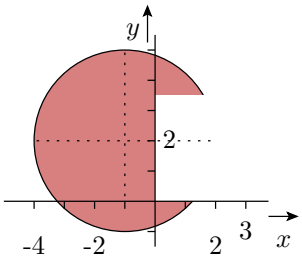
Bei allen Eingaben, für die keine Kante gezeichnet ist, Übergang in den Fehlerzustand.

[1] Ein Mealy-Automat, der die Zeichen an den Kanten abräumt.

3.2 Typ und Wertebereich

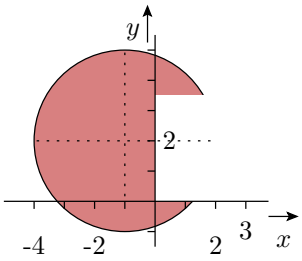
Aufgabe 3.12: Kontrollausdruck

Die Wertpaare (x, y) sollen Punkte der im nachfolgenden Bild eingezeichneten Kreisfläche mit dem Mittelpunkt $(-1, 2)$ und dem Radius 3 mit dem ausgeschnittenen rechteckigen Bereich sein.



Entwickeln Sie einen Kontrollausdruck für die Wertebereichskontrolle, der genau dann wahr ist, wenn ein Punkt (x, y) im zulässigen Bereich liegt.

Zur Kontrolle



Kontrollausdruck für die Wertebereichskontrolle:

$$((x < 0) \vee (y < 0) \vee (y > 3{,}5)) \wedge \left((x+1)^2 + (y-2)^2 < 3^2\right)$$

4 Wertekontrollen

Aufgabe 3.13: Quantisierungsfehler

1. Runden Sie die nachfolgenden Werte

   ```
   a = 123,6793; b = 16,7345;
   c =   5,9463; d = 24,7832;
   ```

   auf 4 Nachkommabits.

2. Führen Sie mit den Originalwerten und mit den gerundeten Werten[2] folgende Operationen durch:

   ```
   e = a - 7*b;
   f = 5*c - d;
   g = e*f;
   ```

Hinweis: Ergänzen Sie in der nachfolgenden Tabelle die gerundeten Werte, ihre Hex-Darstellung und den Rundungsfehler.

[2] Nach jeder Operation ist auf 4 Nachkommabits zu runden.

	w (Wert)	w_{4NKB}	hex(w_{4NKB})	$w - w_{4NKB}$
a	123,6793			
b	16,7345			
c	5,9463			
d	24,7832			
e=a-7*b				
f=5*c-d				
g=e*f				

w_{4NKB} Wert gerundet auf 4 Nachkommabits.

Zur Kontrolle

	w (Wert)	w_{4NKB}	hex(w_{4NKB})	$w - w_{4NKB}$
a	123,6793	123,6875	0x7B,B	0,0082
b	16,7345	16,7500	0x10,C	0,0155
c	5,9463	5,9375	0x05,F	0,0088
d	24,7832	24,8125	0x18,D	0,0293
e=a-7*b	6,5378	6,4375	0x6,7	0,1003
f=5*c-d	4,9483	4,8750	0x4,E	0,0733
g=e*f	32,3510	31,3750	0x1f,6	0,9706

Zum Vergleich, der Quantisierungsfehler von ±0,5 LSB beträgt $\pm 0,03125$.

Aufgabe 3.14: Loop-Back-Test

Scheiben Sie einen Testrahmen, den das nachfolgende fehlerhafte C-Programm für die Wurzelberechnung

```
uint8_t wurzel(uint16_t x){
 uint8_t w=0;
 uint16_t sum=0;
 while (sum<x){sum += (w<<1)+1;
 w++;}
 return w;
}
```

mit 1000 zufälligen Werten testet. Ergebniskontrolle mit der inversen Funktion und Fenstervergleich

$$y^2 \leq x < (y+1)^2$$

Protokollierung aller x und y , die die Ergebniskontrolle nicht bestehen. Nutzen Sie dafür »rand()« aus »std_lib.h«.

Zur Kontrolle

```
#include <std_lib.h>
#include <time.h>
#include <stdio.h>
int main(){
 uint16_t x, y, xmin, xmax;
 srand(time(NULL));    // Init. Pseudozufallsg.*
 for (idx=0; idx<1000; idx++){
  x = rand() & 0xFF;   // Begrenzung auf 8 Bit
  y = wurzel(x);       // Testobjekt
  xmin = y*y;          // inversen Fkt.
  xmax = (y+1)*(y+1); // zu Kontrolle
  if ((x<xmin)||(x>xmax)){
   printf("x=%d, y=%d, y^2=%d, (y+1)^2=%d\n",
       x, y, xmin, xmax);
  }
 }
}
```

*time(NULL) liefert Sekunden seit dem 01.01.1970.