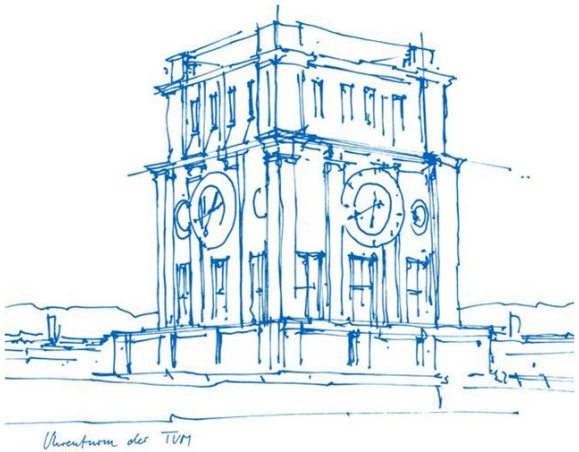
Technische Universität München

Grundlagenpraktikum: Rechnerarchitektur

CRC32 (A400)

25.08.2023



Gruppe 146 | Gregorius Nico Toreh | Rafael Buala Lahagu | Raymond King Setia

Einleitung

- Was ist CRC32 Algorithmus?

- Wo wird der Algorithmus verwendet?

Prüfsumme

$$N = K_{l-1}K_{l-2}\dots K_1K_0 \quad \hat{=} \sum_{i=0}^{l-1} K_i \cdot x^i =: N(x)$$

$$(CRC32(N, G) = P(x) \equiv N(x) \mod G(x))$$

$$N' = (N \parallel P)$$

Ergebnis = 0 ??

Lösungsansätze

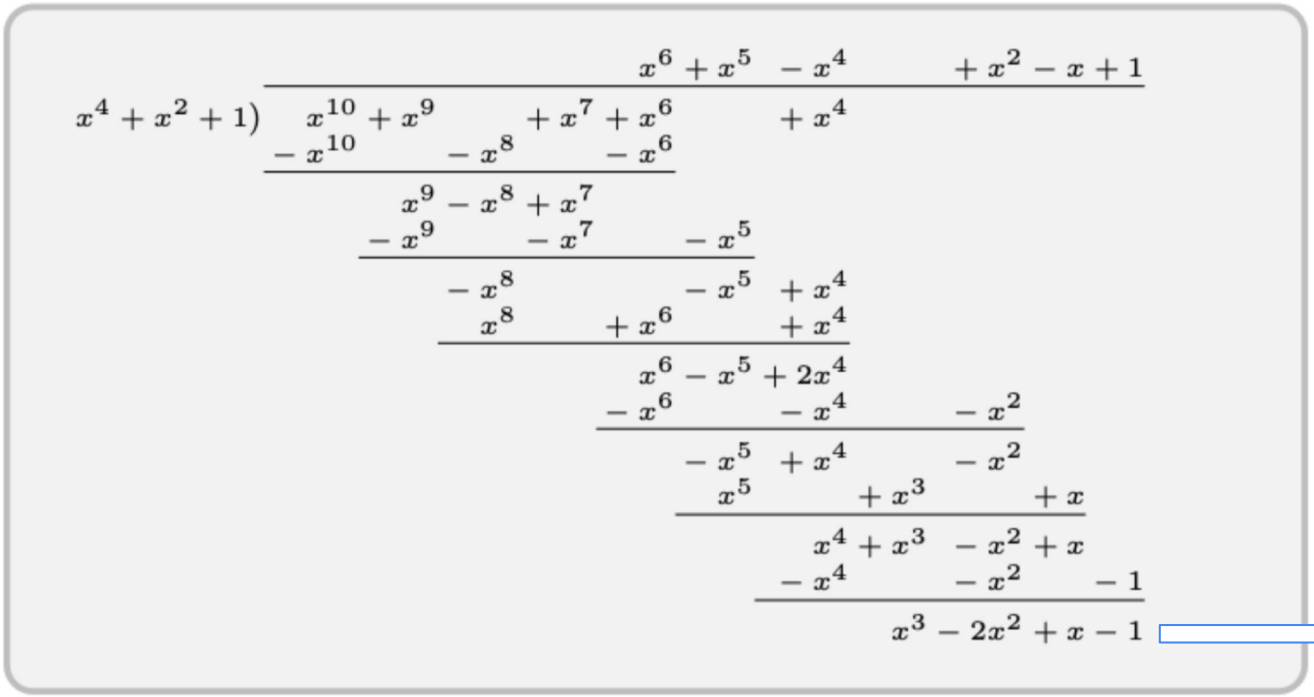
- Der algebraische Ansatz
- Der bitorientierte Ansatz (als Implementierung)
- Der tabelgesteuerten Ansatz (als Implementierung)

Algebraischer Ansatz

Technische Universität München

Beispiel mit CRC4 :
Nachricht = 1 1 0 1 1 0 1
Generatorpolynom = 1 0 1 0 1 = $x^4 + x^2 + 1$
Der Divisor hat 5 Bits (daher ist dies ein CRC-4-Polynom), also werden 4 Nullbits an das Eingabemuster angehängt.
N mit angehängten Nullen = N + 0 = 1 1 0 1 1 0 1 0 0 0 0 = $x^{10} + x^9 + x^7 + x^6 + x^4$

Algebraischer Ansatz

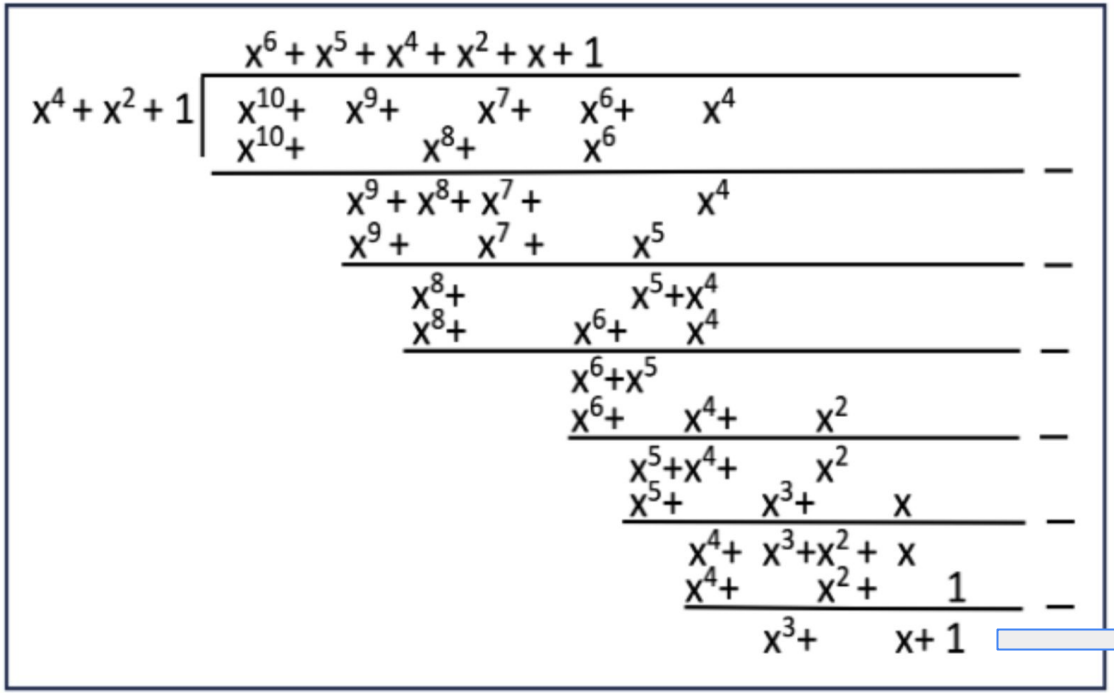


nicht als Bit repräsentierbar !

Assoziativität	$: a + (b + c) = (a + b) + c$	$\mid a \cdot (b \cdot c) = (a \cdot b) \cdot c$
Kommutativität	$: a + b = b + a$	$\mid a \cdot b = b \cdot a$
Identität	$: a + 0 = a$	$\mid a \cdot 1 = a$
Additive Inverse	$: a + (-a) = 0$	
Multiplikative Inverse	$: a \cdot \frac{1}{a} = 1$	
Distributivität	$: a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	

$0+0=0$	$0-0=0$	$0\cdot 0=0$	$\frac{0}{1}=0$
$0+1=1$	$0-1=1$	$0\cdot 1=0$	$\frac{1}{1}=1$
$1+0=1$	$1-0=1$	$1\cdot 0=0$	$\frac{1}{0}=\infty$
$1+1=0$	$1-1=0$	$1\cdot 1=1$	$\frac{0}{0}=\textit{undefiniert}$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



die Prüfsumme 1 0 1 1 darstellt

Naive (bitorientierter Ansatz)

Technische Universität München

Die CRC-Prüfsumme wird bitweise innerhalb jedes Bytes berechnet. Das Ergebnis wird Schritt für Schritt aktualisiert, indem die XOR-Operation auf die entsprechenden Bits angewendet wird

Beispiel : (CRC5)

Technische Universität München

Generatorpolynom: `110101` (zuvor festgelegt)

Rahmen: `11011` (Nutzdaten)

Rahmen mit Anhang: `1101100000` (das Generatorpolynom hat r Stellen, also werden $r - 1 = n$ Nullen ergänzt; hier ist $n = 5$)

↓ immer mit der ersten gemeinsamen 1 anfangen

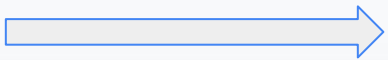
```
1101100000
110101
------
0000110000
    110101
    ------
     00101 (Rest)
```

Übertragener Rahmen: `1101100101`

Korrekte Übertragung der Nachricht: `1101100101`

Das Generatorpolynom (wie oben): `110101`

```
1101100101
110101
------
    110101
    110101
    ------
     00000
```



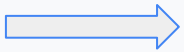
Der Rest der Division ist gleich null. Es ist also wahrscheinlich kein Fehler aufgetreten.

Fehlerhaft übertragene Nachricht (Beispiel): `1001100101`

Das Generatorpolynom (wie oben): `110101`

Technische Universität München

```
1001100101
110101
------
 100110
 110101
 ------
  100111
  110101
  ------
   100100
   110101
   ------
    100011
    110101
    ------
     10110
```



Der Rest der Division (`10110`) ist ungleich null. Also ist ein Fehler aufgetreten.

Quelle : https://de.wikipedia.org/wiki/Zyklische_Redundanzpr%C3%BCfung

Technische Universität München

Vier Fälle bei der Überprüfung auf Richtigkeit :

Gruppe 146 | Gregorius Nico Toreh | Rafael Buala Lahagu | Raymond King Setia

Lookup-Table

Technische Universität München

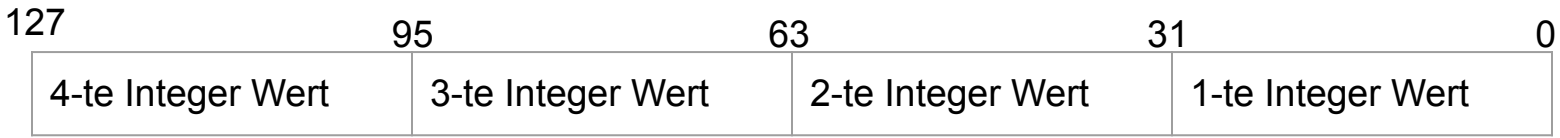
Die Nachricht wird byteweise verarbeitet, indem die vorberechneten Werte (0x00-0xFF) aus einer Lookup-Tabelle verwendet werden.

- Die Größe der Tabelle beträgt 256, da die Größe eines Characters *(char)* 8 Bit beträgt und es daher 256 mögliche Eingaben für das Character gibt.

- keine redundanten Berechnungen, da mit LookupTableFill alle Werte nur einmal berechnet werden

Lookup-Table + SIMD

Die Lookup-Tabelle wird parallel mit SIMD (Single Instruction, Multiple Data) verarbeitet, wobei gleichzeitig 4 Werte verarbeitet werden.



Korrektheit

Ein Fall wo der Algorithmus den Fehler nicht erkennen kann:

Generatorpolynom G = 1 1 0 1 0 1 und eine Nachricht N′

Fallunterscheidung :

$$\textbf{Fall 1}: N' = 1\mathbf{1}0110010\,1 = x^9 + x^8 + x^6 + x^5 + x^2 + 1 \; mod\, G = 0$$

$$\textbf{Fall 2}: N'_{Fehler} = 1\mathbf{0}0110010\,1 = x^9 + \quad x^6 + x^5 + x^2 + 1 \; mod\, G = x^4 + x^2 + x$$

$$\textbf{Fall 3}: N'_{Fehler} = 1\mathbf{0}\mathbf{1}0100\mathbf{1}1 = x^9 + \; x^7 + x^5 + x^2 + x + 1 \; mod\, G = 0$$

Korrektheit

Technische Universität München

Wünschenswerte Eigenschaften des Generatorpolynoms G :

- G soll kein Monom sein.
- G soll möglichst hohen Grad haben.
- G soll möglichst viele Polynome der Form $x^k + 1$ nicht teilen.

Generatorpolynom, das bei Ethernet verwendet wird : 0x104C11DB7 (IEEE 802.3)

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Korrektheit

In der Methode "test" wird die Prüfsumme an die ursprüngliche Nachricht angehängt und die neue Nachricht berechnet. Wenn das Ergebnis der Berechnung 0 ist, kann mit Gewissheit festgestellt werden, dass während der Verarbeitung der Nachricht kein Fehler aufgetreten ist und die Daten nicht beschädigt wurden.

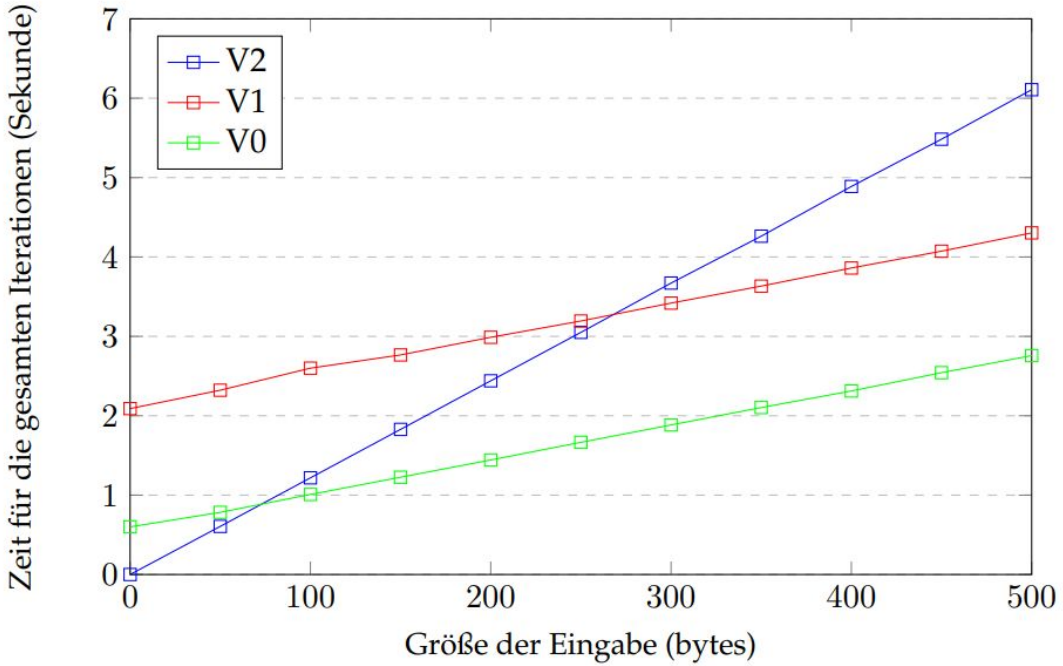
Performanzanalyze (kleine Eingabe)

Technische Universität München

Implementierungen :

Anzahl der Iterationen : 2,000,000

- Lookup-Table mit SIMD (V0)
- Lookup-Table (V1)
- Naive Methode (V2)



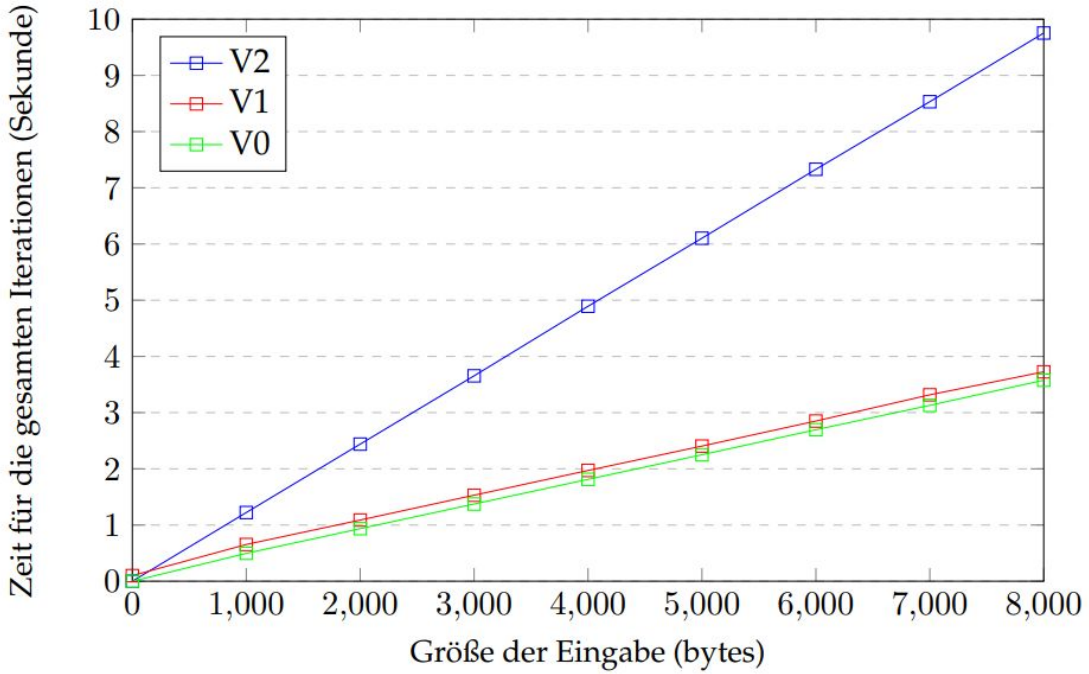
Performanzanalyze (große Eingabe)

Technische Universität München

Implementierungen :

Anzahl der Iterationen : 200,000

- Lookup-Table mit SIMD (V0)
- Lookup-Table (V1)
- Naive Methode (V2)



Zusammenfassung

Technische Universität München

- Ein gutes Generatorpolynom erfüllt einige wünschenswerte Eigenschaften, und sollte eine hohe Wahrscheinlichkeit haben, Fehler zu erkennen

- Die Implementierung mit der Lookup-Tabelle und SIMD-Optimierung zeigt die beste Performanz und ist effizienter als die anderen Implementierungen.

- Andere Ansätze zur Optimierung des CRC32 Algorithmus können untersucht werden, um die Geschwindigkeit und die Fähigkeit zur Fehlererkennung des Programms zu verbessern.

Literatur

[1] Martin Stigge, Henryk Plötz, Wolf Müller, and Jens-Peter Redlich. Reversing crc-theory and practice. *Berlin : Humboldt University Berlin*, 17, 2006.https://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2006-05/SAR-PR-2006-05_.pdf, visited 2023-06-16.

[2] Ross Williams et al. A painless guide to crc error detection algorithms. *Internet publication, August*, 75, 1993. http://www.ross.net/crc/download/crc_v3.txt, visited 2023-06-16.