technische universität
dortmund

fakultät für informatik
informatik 12

Rechnerarchitektur (RA)

Sommersemester 2015

Foliensatz 3: **Netzwerkprozessoren und CRC-Zeichen**

Michael Engel
Informatik 12
michael.engel@tu-..
http://ls12-www.cs.tu-dortmund.de/daes/
Tel.: 0231 755 6121

2015/04/14

2.3 Befehle von Netzwerkprozessoren

- Viele aktive Komponenten in heutigen LAN/WANs
- Extrem hohe Anforderungen an die Geschwindigkeit bei begrenztem Energieaufwand
- Aktive Komponente = Spezialrechner (mit Netzwerkprozessor)

Netzwerkprozessoren: Aufgaben (1)

- Klassifikation/Filterung von Netzwerkpaketen
  (z.B. Prüfung auf Übertragungsfehler [CRC], *Firewall*, ...)
- Paketweiterleitung (*IP forwarding*)
  (für *Routing* zwischen verschiedenen Teilnetzen)
- Adressübersetzung zwischen globalen und lokalen/
  privaten Adressbereichen (z.B. *IP masquerading*,
  virtuelle Web-Services, ...)

Netzwerkprozessoren: Aufgaben (2)

- *Virtual Private Networks*, d.h. gesicherte private Verbindungen über öffentliches, ungesichertes Netz (Kryptographie [DES] und Authentisierung [MD5] erforderlich)
- *Intrusion Detection*: Erkennung von Angriffsszenarien auf Paketebene durch Signaturvergleich
- *Deep packet inspection:* Netzwerkkomponenten analysieren Informationen jenseits des *Headers*
- Daten-Umkodierung: Formatumwandlung für Multimediadaten innerhalb des Netzwerks

Netzwerkprozessoren: CRC (1)

Problem: Datenübertragung über physikalisches Medium kann fehlerbehaftet sein

- Auf Ebene der Netzwerkpakete:

  Durch Netzwerkprotokoll detektiert / behandelt

- Auf Ebene der Paketinhalte
  (fehlerhafte Übertragung einzelner oder mehrerer Bits):

  Möglichkeit zur Validierung empfangener Daten erforderlich

Netzwerkprozessoren: CRC (2)

Prinzip: Redundante Codierung der Daten
(d.h. es werden mehr Bits übertragen als Nutzdaten)

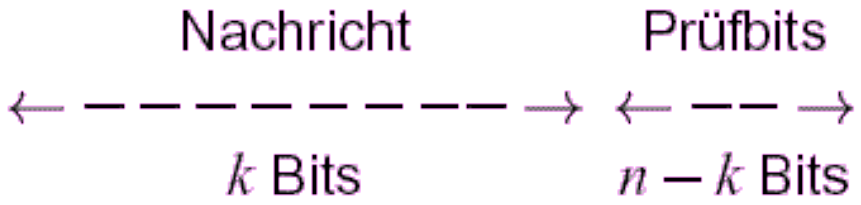
Verbreitete Methode: *Cyclic Redundancy Check* (CRC)

- Erkennt Vielzahl von Fehlertypen
  (je nach Parametrisierung)
- Problem: Berechnung aufwendig
  ☞ Spezial-Hardware/-Befehle

Bildung von CRC-Zeichen: Sicherung von Daten mit zyklischen Codes

Zu übertragen: Nachricht, in k Bits kodiert. **Ziel:** durch Übertragung von n Bits ($n > k$) Schutz gegen Fehler.

Annahme: systematischer Code: Code stimmt in k Bits mit der Ausgangsnachricht überein:



Betrachtung der Bits a_i der Nachricht als Koeffizienten eines Polynoms G, des **Nachrichtenpolynoms**

(1) $$G = \sum_{i=0}^{k-1} a_i \cdot 2^i$$

Basis 2 ☞ Basis x:

Darstellung von Nachrichten

(2) $$G(x) = \sum_{i=0}^{k-1} a_i \cdot x^i$$

Der übertragene Code ist ($R(x)$= Prüfbits):

(3) $$F(x) = G(x) * x^{n-k} + R(x)$$

Forderung: $F(x)$ soll durch **Generator-Polynom** $P(x)$ teilbar sein

(4) $$F(x) = P(x) * Q(x) = G(x) * x^{n-k} + R(x)$$

Im Folgenden: alle Operationen modulo 2 durchführen.
Wirkung von + und – ist gleich.

Def.: Rest r von g/p ist $r = g - p*q$ mit $r < p$.

(5) $$R(x) = G(x) * x^{n-k} - P(x) * Q(x)$$

Beispiel zur Polynomdivision

$(1x^6+0x^5+1x^4+0x^3+1x^2+0x^1+1x^0)$ §: $(1x^3+1x^2+0x^1+1x^0) =$

$1x^6+1x^5+0x^4+1x^3$ $\quad 1x^3+$

$0x^6+1x^5+1x^4+1x^3+1x^2$

$1x^5+1x^4+0x^3+1x^2$ $\quad 1x^2+$ Quotient

$0x^5+0x^4+1x^3+0x^2+0x^1$

$0x^4+0x^3+0x^2+0x^1$ $\quad 0x^1+$

$1x^3+0x^2+0x^1+1x^0$

$1x^3+1x^2+0x^1+1x^0$ $\quad 1x^0$

$1x^2+0x^1+0x^0$ ☞ Rest 100

§: Nur Prinzip; bei Nachrichten wegen „$*x^{n-k}$“ Nullen am rechten Rand

Reduktion der Polynomdivision auf die Koeffizienten

```
(1 +0 +1 +0 +1 +0 +1 ):(1 +1 +0 +1 )=
 1 +1 +0 +1                 1 +
---------------
 0 +1 +1 +1 +1
    1 +1 +0 +1              1 +
    --------------
    0 +0 +1 +0 +0
       0 +0 +0 +0           0 +
       ----------------
          1 +0 +0 +1
          1 +1 +0 +1        1
          ---------------
             1 +0 +0   ☞ Rest 100
```

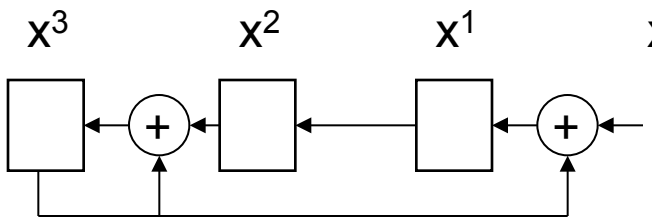
Signifikanteste Stelle der Zwischenergebnisse stets =’0’;

Stellen des Divisors (bis auf MSB) abziehen, falls gerade betrachtetes MSB des Dividenden = ’1’ ist.

Realisierung der Division mittels Schieberegister und XOR-Gattern

```
(1 +0 +1 +0 +1 +0 +1 ):(1 +1 +0 +1 )= 1+1+0+1
 1 +1 +0 +1
 ---------------
 0 +1 +1 +1 +1
    1 +1 +0 +1
    --------------
    0 +0 +1 +0 +0
         0 +0 +0 +0
         -----------------
             1 +0 +0 +1
             1 +1 +0 +1
             ------------------
                 1 +0 +0
             ☞ Rest 100
```

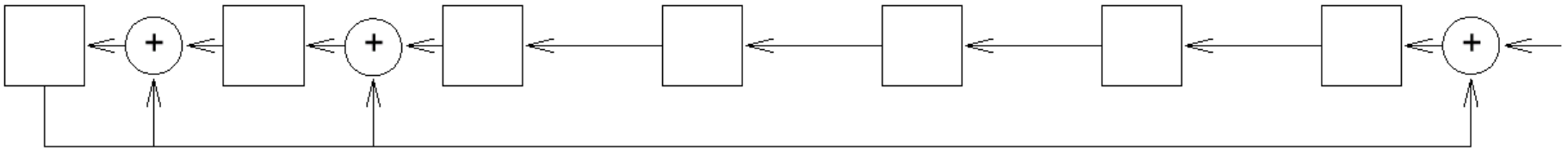
Schaltung:



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

Schaltung für Polynom höheren Grades

Schaltung für Polynom $x^7 + x^6 + x^5 + 1$



Eigenschaften von zyklischen Codes

Empfänger wird im Allgemeinen eine gestörte Nachricht $H(x)$ empfangen:

$$H(x)=F(x)+E(x)$$

$E(x)$ heißt **Fehlerpolynom**.

Jeder Term von $E(x)$ kennzeichnet 1 Fehlerstelle

Erkennung von Fehlern:

$H(x)$ nicht durch $P(x)$ teilbar ☞ Fehler erkannt.

$H(x)$ durch $P(x)$ teilbar ☞

- **Übertragung fehlerfrei oder**
- **nicht erkennbarer Fehler ($P(x)$ teilt $E(x)$).**

Einzelfehler

Satz 1: Ein zyklischer Code, der durch ein Polynom mit mehr als einem Term erzeugt wird, entdeckt alle Einzelfehler.

Beweis:

- Einzelfehler besitzen ein Fehlerpolynom der Form $E(x) = x^i$.
- Hat $P(x)$ mehr als einen Term, so teilt es x^i nicht.

Erkennung von ungeraden Fehlerzahlen

Satz 2: Jedes durch 1+ x teilbare Polynom hat eine gerade Termzahl.

Beweis: P durch (1+ x) teilbar

$$\rightarrow P = \sum a_i x^i * (1 + x)$$

$$\rightarrow P = \sum a_i * (x^i + x^{i+1})$$

Falls nie benachbarte a_i gleich 1 sind: gerade Termzahl

Falls benachbarte a_i gleich 1 sind:

Auslöschung einer geraden Termzahl:

$$P = \ldots + a_i (x^i + x^{i+1}) + a_{i+1} (x^{i+1} + x^{i+2})$$

$$P = \ldots + a_i (x^i + x^{i+2}) \qquad \text{(wegen mod 2)}$$

Es ergibt sich eine gerade Termzahl.

☞ Mit (1+ x) als Faktor findet man eine ungerade Anzahl von Fehlern (Parity-Prüfung).

Burstfehler

Definition: Ein Burstfehler der Länge b ist ein Fehler, bei dem die falschen Symbole den Abstand b haben.

Beispiel: $E(x) = x^7 + x^4 + x^3 = 0010011000$.
Per Definition zählt man dies als Abstand 5, d.h. man zählt die Randsymbole mit.

Satz 3: Ein durch ein Polynom vom Grad n-k erzeugter zyklischer Code entdeckt alle Burstfehler der Länge $b \leq n$-k, wenn $P(x)$ x nicht als Faktor enthält.

Erkennbarkeit von Burstfehlern

Satz 3: Ein durch ein Polynom vom Grad n-k erzeugter zyklischer Code entdeckt alle Burstfehler der Länge $b \leq n$-k, wenn $P(x)$ x nicht als Faktor enthält.

Beweis: Sei x^i der Term mit dem kleinsten Exponenten, der in $E(x)$ vorkommt: ☞ $E(x) = x^i * E_1(x)$

Nicht durch $P(x)$ teilbar, wenn beide Terme nicht teilbar sind.

1. x^i nicht teilbar, wenn $P(x)$ nicht x als Faktor enthält.
2. Grad von $E_1(x)$ ist $\leq b-1$, d.h. $\leq n-k-1$,
   Grad von $E_1(x) <$ Grad von $P(x)$
   $\rightarrow P(x)$ teilt $E_1(x)$ nicht.

* Wenn x^0 Term in $E(x)$: kein Ausklammern, aber alle Argumente gelten direkt bezüglich $E(x)$.

Anzahl nicht erkannter Burstfehler in Relation zu möglichen Burstfehlern

Satz 4: Die Anzahl der nicht erkannten Burstfehler der Länge $b > n\text{-}k$ *ist, bezogen auf die Anzahl möglicher Burstfehler:*

$$\begin{cases} 2^{-(n-k-1)}, & \text{wenn } b = n-k+1 \\ 2^{-(n-k)}, & \text{wenn } b > n-k+1 \end{cases}$$

Beispiel: $P(x)=(1+x^2+x^{11})(1+x^{11})$ erkennt:

1 Burst der Länge ≤ 22

Jede ungerade Anzahl von Fehlern (Faktor (x^q+1))

99,99996 % der Bursts der Länge 23

99,99998 % der Bursts der Länge > 23

In der Praxis sind viele der vorkommenden Fehler Burstfehler.

Sehr einfaches Beispiel

$P(x)=(1+x)$, $k=2$, $n=3$

Aufgrund o.a. Sätze:

- Erkennung aller Einzelfehler (Satz 1, 2 und 3)
- Für Burstfehler mit b =2:
  - Wahrscheinlichkeit, Fehler nicht zu erkennen: $2^{-(1-1)}=2^0=1$
- Für Burstfehler mit b =3:
  - Wahrscheinlichkeit, Fehler nicht zu erkennen: höchstens $2^{-(1)}=1/2$
  - Aber: ungerade Fehlerzahl, daher alle erkannt.

Interaktive Berechnung

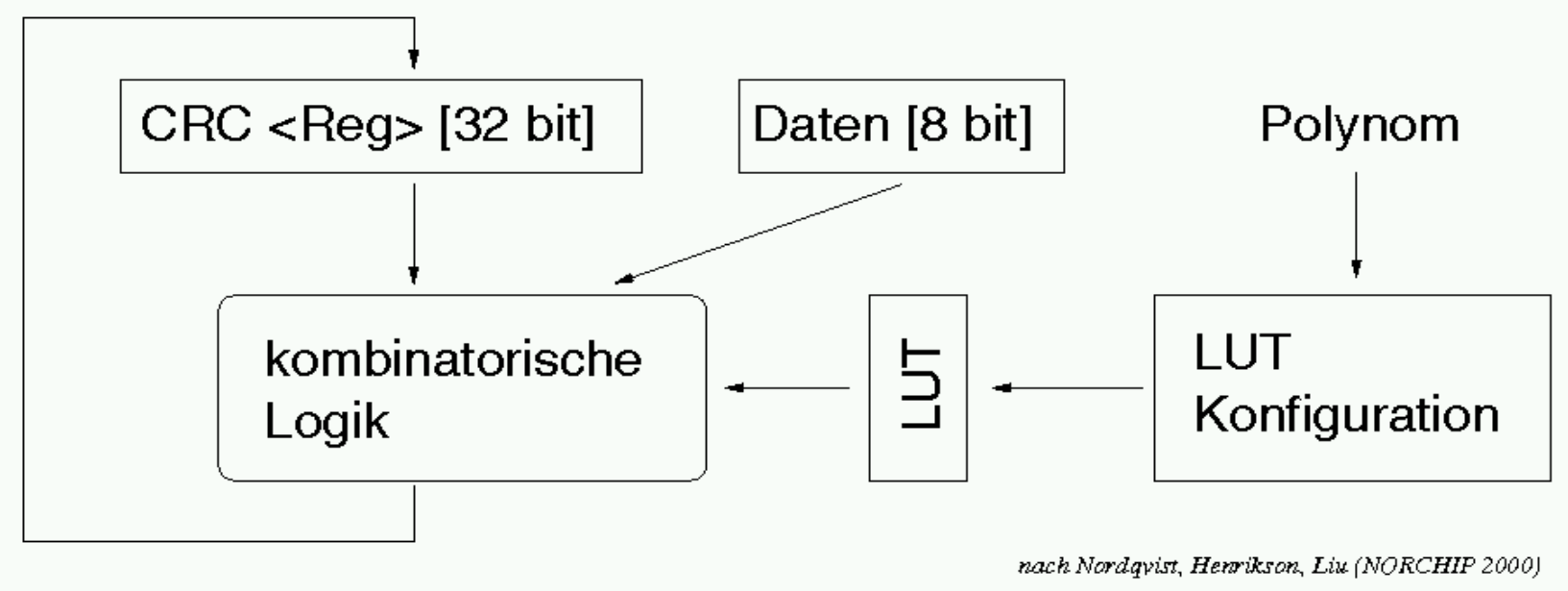
Unter http://einstein.informatik.uni-oldenburg.de/20911.html:

- Simulationsprogramm
- Anleitung
- Theorie von CRC-Zeichen

CRC-Befehl

Spezialbefehl für CRC-Berechnung in Paketen

- Pakete werden byteweise verarbeitet
- Über eine *Lookup*-Tabelle werden alle 256 möglichen Ergebnisse der bitweisen Verarbeitung von 8 Bit erzeugt. Beschleunigung gegenüber bitweiser Verarbeitung um den Faktor 8.



CRC: Anwendung

In der Praxis werden u.a. folgende Generatorpolynome verwendet:

- CRC-5 (USB) $x^5 + x^2 + 1$
- CRC-7 (SD/MMC-Card) $x^7 + x^3 + 1$
- CRC-16 (HDLC, X.25) $x^{16} + x^{12} + x^5 + 1$
- CRC-32 (Ethernet): $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Netzwerkprozessoren: Verarbeitungsprinzipien

Datenübertragung in Netzwerken erfolgt paketorientiert

- Paketlänge (üblich: 64 Byte) bestimmt bei maximaler Auslastung der Bandbreite die verfügbare Bearbeitungszeit
  Beispiel: GigaBit-Ethernet
  max. Bandbreite 1Gb/s, d.h. 64 Byte ☞ ca. 475 ns
- Trotzdem: Konstante Verzögerung (im Umfang der Paketbearbeitung) entsteht pro aktiver Komponente im Verbindungspfad

☞ Performanzsteigerung durch Parallelisierung (mehrere Verarbeitungseinheiten) und **nicht** durch *Pipelining* (Verschränkung der Verarbeitungsschritte)

Netzwerkprozessoren: Verarbeitungsprinzipien (2)

- Netzwerkrelevante Information kodiert in Bitfeldern
  z.B.: IP

4	4	8	16	
Version	*Header-Länge*	*Type of service*	Gesamtlänge	
Identification			*Flags*	*Fragment offset*
Time to live		*Protocol*	*Header checksum*	
Source address				
Destination address				
Option+Padding				
Data				

☞ Effiziente Manipulation von Daten auf Bit-Ebene notwendig

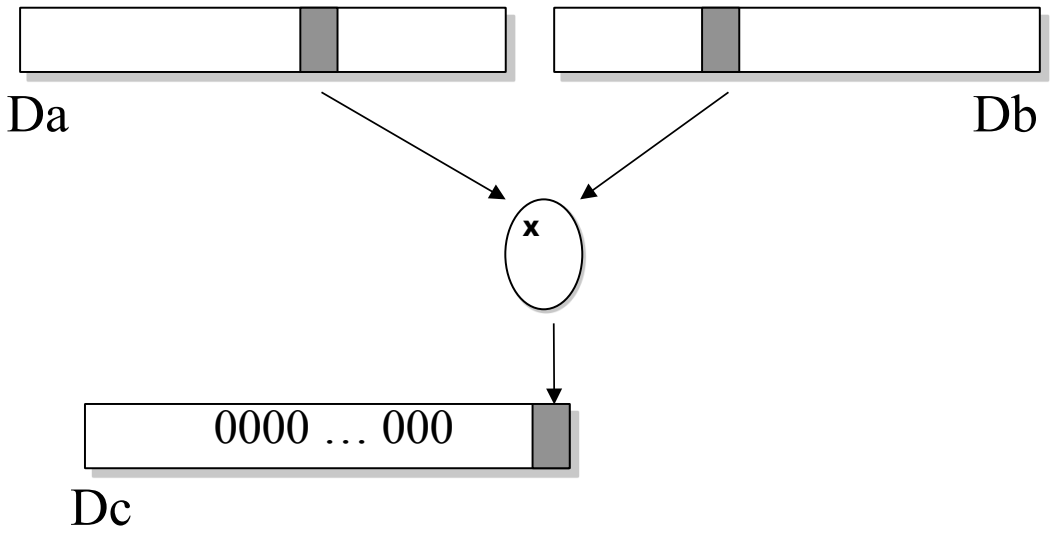
Netzwerkprozessoren: Verarbeitungsprinzipien (3)

- Adressierungsarten für Bits/Bitfelder
  - übliche Rechner/Speicher byte- oder wort-adressierbar
- ALU-Operationen für Bits/Bitfelder
  - übliche Architekturen unterstützen nur Registeroperanden
  - ggf. Halbwort- bzw. Byte-Op. möglich (manchmal gepackt)
- Operanden/Operationen deutlich fein-granularer als in „Universal“-Rechnern üblich

Netzwerkprozessoren: Bit-Operationen

Hier: Infineon TriCore

- Operationen, die einzelne Bits aus 2 Quellregistern (wahlfrei adressierbar) logisch verknüpfen und Ergebnis im niederwertigsten Bit des Zielregisters ablegen

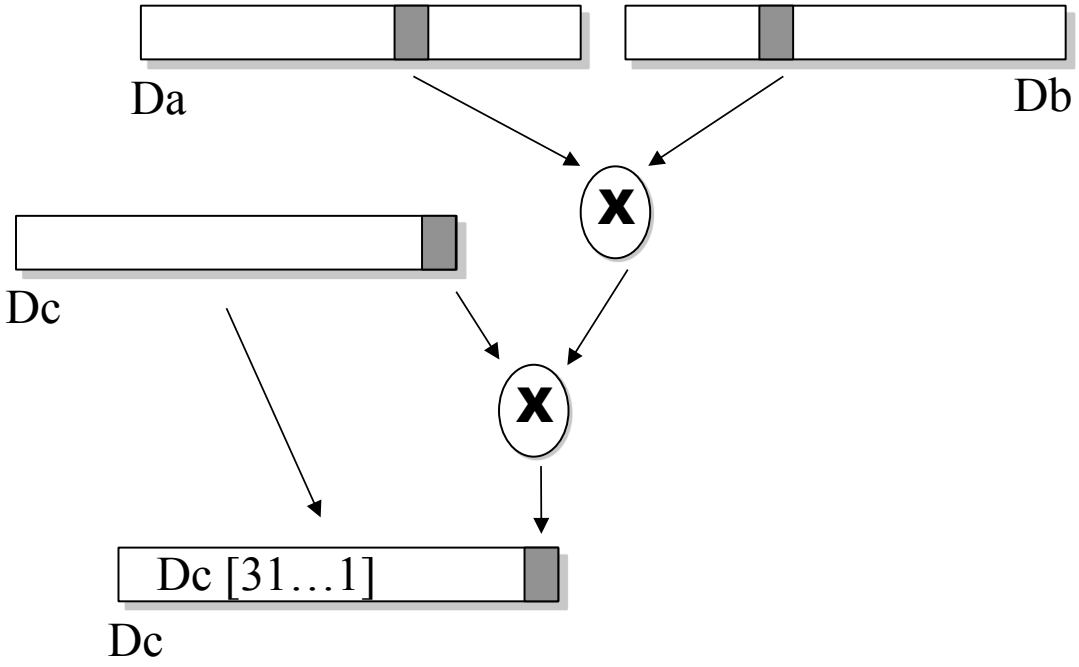


- **AND.T**
- **NAND.T**
- **NOR.T**
- **OR.T**
- **ORN.T**
- **XNOR.T**
- **XOR.T**

Wie viel Aufwand mit Standardprozessor?

Infineon TriCore: Bit-Operationen (2)

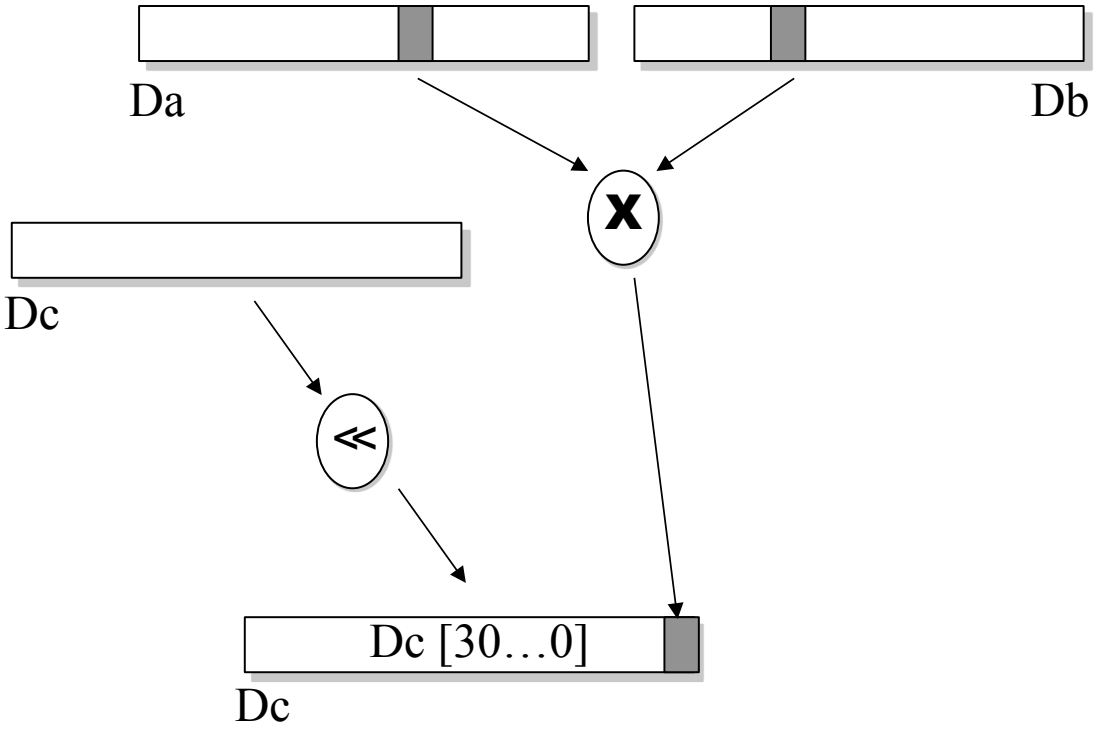
Optional: Weitere logische Verknüpfung mit unterstem Bit im Zielregister



- **AND.AND.T**
- **AND.ANDN.T**
- **AND.NOR.T**
- **AND.OR.T**
- **OR.AND.T**
- **OR.ANDN.T**
- **OR.NOR.T**
- **OR.OR.T**

Infineon TriCore: Bit-Operationen (3)

Optional: Schiebeoperation im Zielregister

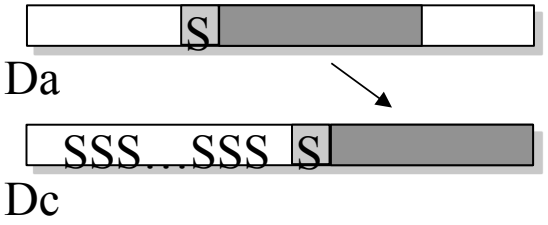


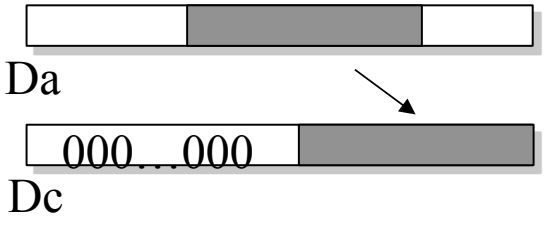
- **SH.AND.T**
- **SH.ANDN.T**
- **SH.NAND.T**
- **SH.NOR.T**
- **SH.OR.T**
- **SH.ORN.T**
- **SH.XNOR.T**
- **SH.XOR.T**

Wozu Schiebeoperationen im Zielregister?

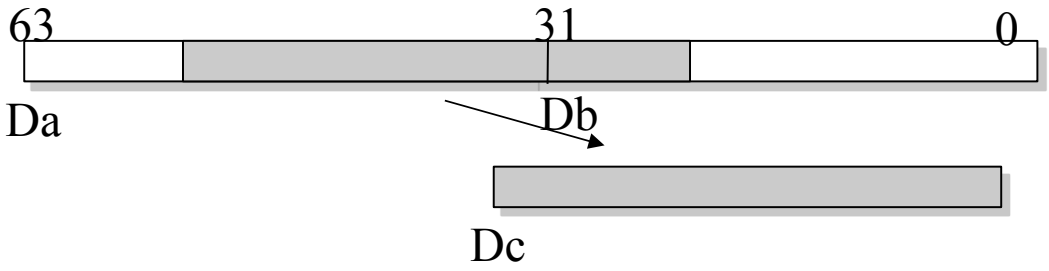
Infineon TriCore: Bit-Operationen (4)

Extraktion / Manipulation von Bitfeldern

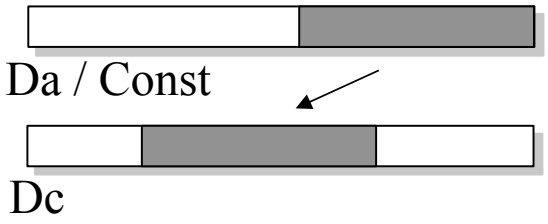




- **EXTR**
- **EXTR.U**



- **DEXTR**



- **INSERT**

Netzwerkprozessoren: Bit-Operationen

Bit-genaue Speicherzugriffe

- Analog zu Einzelbit-Manipulationen auf Registerebene
- Spezielle Adressdekodierungshardware erforderlich
- *Load-Modify-Store* Befehle (für Bits-/Bitfelder)
  - Prinzip wie ALU-Befehl (für Bits/Bitfelder) mit Speicheradresse als Quell und Zieloperanden
  - Unterschied: Operation ist atomar, d.h. kann nicht von anderem Befehl unterbrochen werden
    ☞ Realisierung von z.B. Semaphoren möglich

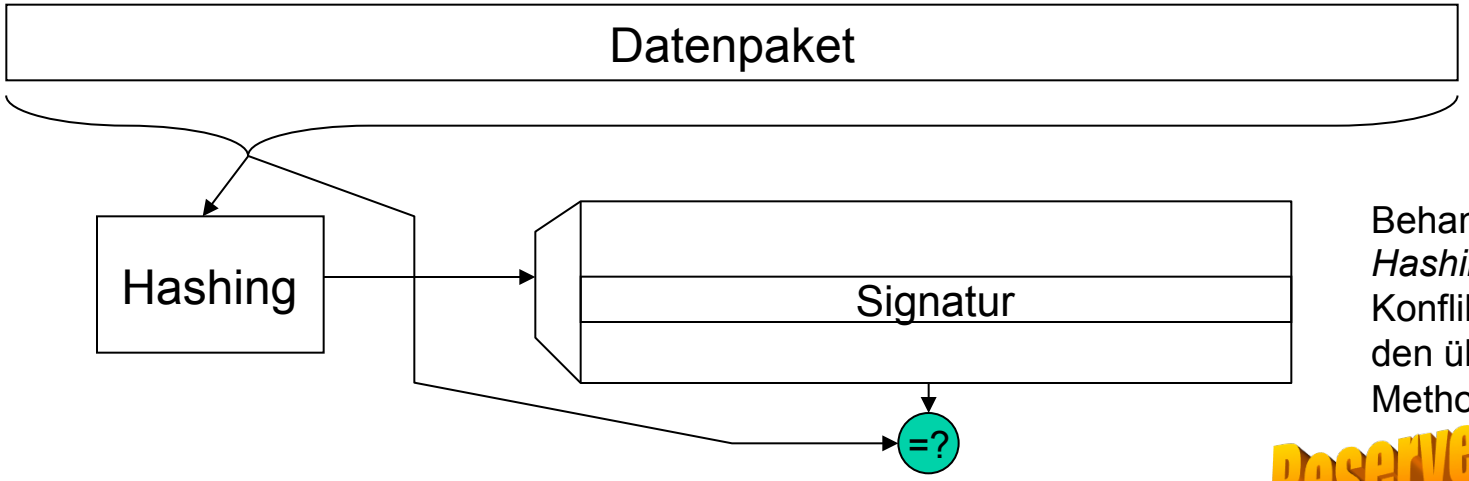
Hardware zum Signaturvergleich

- Detektion von Angriffsszenarien auf Netzwerkebene mit Hilfe von Signaturen = tatsächliche (Daten-)Pakete, die an Szenarien beteiligt sind (z.B. Codesequenz eines Virus, ggf. auch bestimmte Protokoll-*Header*)
- Problem: Jedes eingehende Paket muss (im Pakettakt!) mit allen(!) Signaturen verglichen werden
- Hardwarelösung: Assoziativspeicher
  ☞ bei weitem zu aufwendig / teuer

Beschleunigung des Signaturvergleichs

Lösung: Hashing:

- Datenraum = Menge der Signaturen / Pakete
- Berechne (schnell) “kurzen” Schlüssel pro Signatur
  - Schlüsselraum << Datenraum
  - Schlüssel sind nicht eindeutig!
- In Array mit Indexbereich der Schlüsselmenge speichere tatsächliche Signaturen

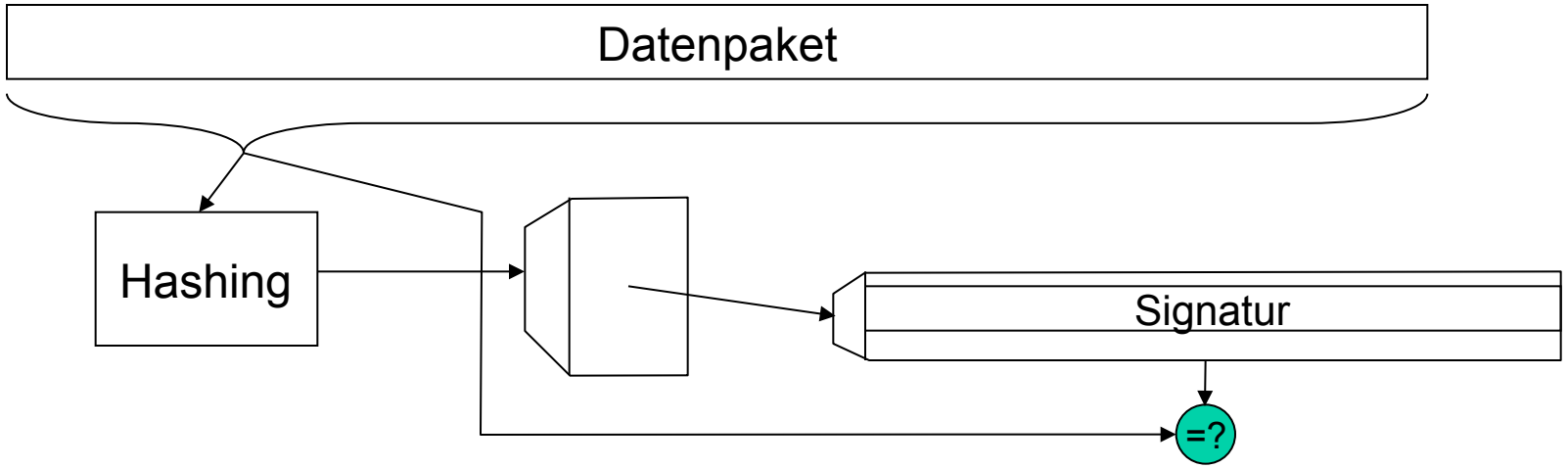


Behandlung von *Hashing*-Konflikten mit den üblichen Methoden

Beschleunigung des Signaturvergleichs (2)

Hash-Tabelle muss |Schlüssel| x Länge(Signaturen) groß sein ☞ zu viel Speicher- und Stromverbrauch!

Annahme: Es gibt wenige Signaturen (d.h. auch der Schlüsselraum ist nur dünn besetzt) ☞ indirekte Adressierung



Zusammenfassung

- Schnelle + sichere Verarbeitung im Netzwerk ist wichtig
- CRC-*Codes* zur Absicherung der Übertragung (und der Speicherung von Daten)
  - Günstig bei Burstfehlern der Länge b: für
    - $b \leq n\text{-}k$ vollständige Erkennung
    - $b > n\text{-}k$ Erkennung mit hoher Wahrscheinlichkeit
- Netzwerkprozessoren zur schnellen Verarbeitung von Netzwerkprotokollen
  - Bit-orientierte Befehle
  - Spezielle Unterstützung der CRC-Berechnung
  - Hardware-Unterstützung zum Signaturvergleich