

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Florian Franzmann Martin Hoffmann Tobias Klaus

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<http://www4.cs.fau.de>

4. Juni 2014



Überblick

- 1 Organisatorisches
- 2 C-Quiz Teil IV
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



Table of Contents

- 1 Organisatorisches
- 2 C-Quiz Teil IV
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



Annahmen

- C99
- x86 bzw. x86-64, d. h.
 - vorzeichenbehaftete Integer als Zweierkomplement implementiert
 - char hat 8 Bit
 - short hat 16 Bit
 - int hat 32 Bit
 - long hat 32 Bit auf x86 und 64 Bit auf x86-64



Frage 10

Angenommen x hat Typ `int` und ist positiv. Ist $x \ll 1 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Es darf nicht in das Vorzeichenbit hineinverschoben werden
⇒ nicht definiert für große Werte von x



Frage 11

Angenommen x hat Typ `int`. Ist $x \ll 31 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Es darf nicht in das Vorzeichenbit hineinverschoben werden
⇒ funktioniert hier nur mit $x == 0$



Frage 12

Angenommen x hat Typ `int`. Ist $x \ll 32 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Verschiebung um Bitbreite eines Datentyps nicht zulässig



Frage 13

Angenommen x hat Typ `short`. Ist $x \ll 29 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- Vor der Verschiebeoperation wird nach `int` umgewandelt
- Verschiebung um mehr als die Bitbreite ist also kein Problem



Frage 14

Angenommen x hat Typ `unsigned`. Ist $x \ll 31 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- jeder Wert, dessen *promoted type* `unsigned` ist kann um nichtnegativen Wert verschoben werden
- solange die Bitbreite nicht erreicht wird



Frage 15

Angenommen x hat Typ `unsigned short`. Ist $x \ll 31 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- `unsigned short` wird nach `int` umgewandelt
- eine 1 darf nicht in das Vorzeichenbit hineinverschoben werden
- Verschiebung um bis zu 15 wäre immer in Ordnung

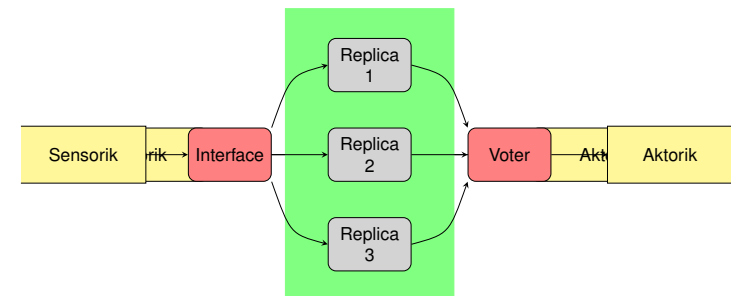


Table of Contents

- 1 Organisatorisches
- 2 C-Quiz Teil IV
- 3 Wiederholung Software-TMR
- 4 Eliminierung von Bruchstellen in TMR
- 5 Aufgabenstellung



Klassische "Triple Modular Redundancy" (TMR)



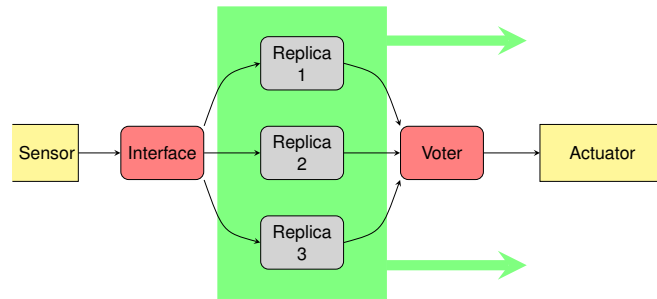
- Schnittstelle sammelt Eingangsdaten (Replikdeterminismus)
- Verteilt Daten und aktiviert Replikate
- Mehrheitsentscheider (Voter) wählt Ergebnis
- Ergebnis wird an Aktuator versendet

Redundanzbereich

Ausschließlich Replikatausführung.



Erweiterung I – codierte Ausgangswerte



- Erweiterung der Ausgangsseite mit Informationsredundanz
- Mehrheitsentscheid über codierte Prüfsumme



Table of Contents

- 1 Organisatorisches
- 2 C-Quiz Teil IV
- 3 Wiederholung Software-TMR
- 4 **Eliminierung von Bruchstellen in TMR**
- 5 Aufgabenstellung



Erweiterte arithmetische Codierung

nach Forin 1989: "Vital coded microprocessor principles and application for various transit systems" [1]

- Arithmetisch codierter Wert V_C
- Ausgangswert

$$V_C = V * A + B_V + D$$

Bitfehlererkennung (Restfehlerwahrscheinlichkeit $P = 1/A$)
 Adressengliederung
 Fehlererkennung
 Variable Daten

- Schlüssel
- Variablenspezifische Signatur
- Zeitstempel

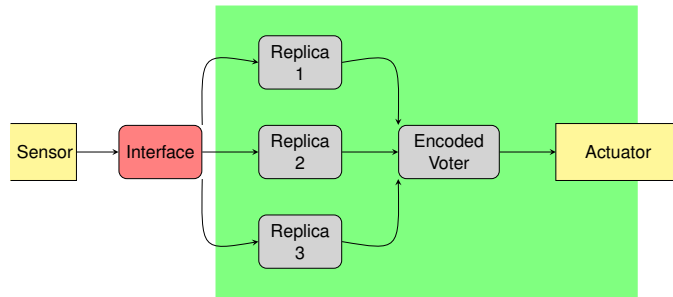


Wertebereichseinschränkungen

- Schlüssel A sollte so groß wie möglich sein:
 \rightsquigarrow Möglichst geringe Restfehlerwahrscheinlichkeit ($P = 1/A$)
- Wertebereich des dynamischen Zeitstempels
 - $D = \{x | x \in \mathbb{N}_0 \wedge x \leq D_{max}\}$
 - Zeitstempel darf überlaufen: $D_{max} + 1 = 0$
- Für jede Signatur B_* muss dann gelten
 - $B_* + D_{max} < A$
 - Die minimale Distanz zwischen jeweils zwei Signaturen im System muss kleiner D_{max} sein: $\forall i, j : |B_i - B_j| < D_{max}$



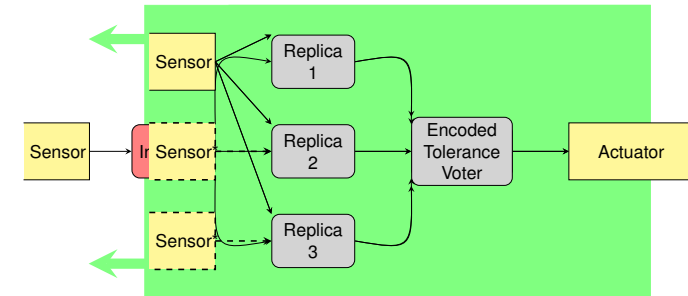
Erweiterung I – codierte Ausgangswerte



- Replikate liefern arithmetisch codierte Ergebnisse
- Mehrheitsentscheid auf codierten Prüfsummen
- Übertragung codierter Ergebnisse



Erweiterung II – Datendiversität



- Replikate ermitteln Eingangsdaten selbständig
- Diversitäre Eingangsdaten
 - Unterschiedliche Messzeitpunkte (zeitliche Redundanz)
 - Redundante Sensoren (physikalische Redundanz)
- Mehrheitsentscheid mittels Toleranzbereich (Tolerance Voter)



Vereinfachung für diese Übung

Für diese Übungsaufgabe:

- Keine Datendiversität am Eingang
- Kein Zeitstempel
- Nur Absicherung der Ausgangsseite!



EAN Vergleichsoperator

- Voting basiert auf codierter Vergleichsoperation:

$$\sim X_C = Y_C \Rightarrow X * A + B_X = Y * A + B_Y$$

- Im fehlerfreien Fall gilt:

$$X = Y, A = A \text{ aber } B_X \neq B_Y !$$

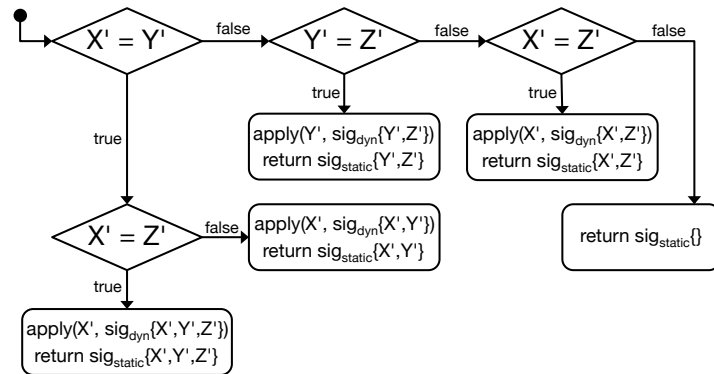
- Rohwerte sind identisch
- Schlüssel ist per Definition identisch
- Signaturen sind unterschiedlich (aber konstant!)

Bestimmung der Gleichheit durch Differenzbildung:

$$\sim X_C - Y_C = B_X - B_Y = \text{const.}$$



Codierter Mehrheitsentscheid



- Bestimmung von dynamischer und statischer Signatur:

$$\sim sig_{dyn}(X', Y') : X' = Y' \Rightarrow X' - Y'$$

$$\sim sig_{static}(X', Y') : X' = Y' \Rightarrow B_X - B_Y$$



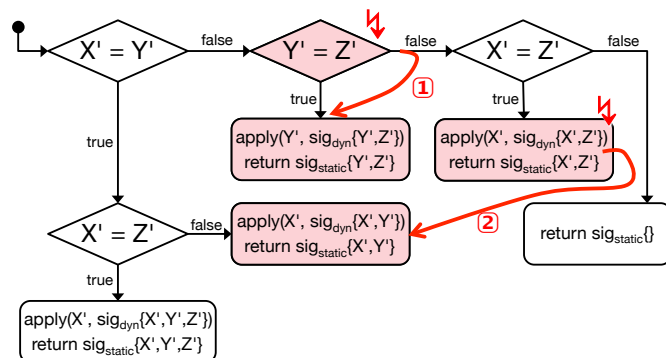
Codierter Mehrheitsentscheid (Forts.)

- Vergleichsoperation wird durchgeführt (z. B. $X' = Y' \wedge X' = Z'$)
 - Berechnung von sig_{dyn}
 - Vergleich mit sig_{static}
- Verzweigungsentscheidung wird nachberechnet:
 - Wiederholte (redundante) Berechnung von sig_{dyn}
 - Addiere sig_{dyn} ($apply$) zum gewählten Ergebnis
- Konstante Signatur des durchlaufenen Zweiges identifiziert Gewinner (Rückgabewert: sig_{static})
 - Aktor wählt entsprechendes Replikatergebnisse
 - führt inverse Operation zu $apply$ durch

Im Voter wurde die *dynamisch berechnete Signatur der Verzweigungsentscheidung* hinzuaddiert. Im Aktor wird mit der entsprechenden *konstanten Signatur zurückgerechnet*.



Codierter Mehrheitsentscheid - Fehlerfall



- Falsche Verzweigungsentscheidung: ($Y' \neq Z'$)
 - Y' wird als korrekt angenommen, sig_{dyn} wird erneut berechnet
 - allerdings ist sig_{dyn} tatsächlich $\neq sig_{static}$
 - Fehler wird bei der inversen Operation zu $apply$ erkannt
- Falscher (plötzlicher) Sprung
 - X' wird als korrekt erkannt, sig_{dyn} wird erneut berechnet
 - Ein fehlerhafter Sprung in einen anderen Block führt zu einem inkonsistenten Rückgabewert $sig_{static}\{X', Z'\}$



Table of Contents

- Organisatorisches
- C-Quiz Teil IV
- Wiederholung Software-TMR
- Eliminierung von Bruchstellen in TMR
- Aufgabenstellung



Aufgabenstellung

Aufgabe

Erweitern Sie Ihre Software-TMR Implementierung um einen EAN-codierten Voter Filterimplementierung aus der vorherigen TMR-Aufgabe.

- Jedes Replikat hat genau einen Ausgabewert (integer \mapsto enc_t): Eine kodierte Prüfsumme des Ergebnisses
 - ↪ Legen Sie für jede der drei Ausgabewerte (X' , Y' , Z') jeweils *unterschiedliche* aber *konstante* Signaturen (SIG_X , SIG_Y , SIG_Z) fest
- Nutzen Sie für X' den nächstgrößeren Datentyp zu X
 - ↪ Wählen Sie eine Zahl A mit möglichst großem Hamming-Abstand, *vermeiden Sie* dabei mögliche *Überläufe bei der Codierung*



Hinweise

- In dieser Aufgabe betrachten wir nur die Ausgangsseite
- Die Eingangsseite bleibt vorerst „ungeschützt“
- die besten Kandidaten findet ihr hier:
www4.cs.fau.de/Research/CoRed/experiments
 - 8 Bit-Schlüssel: 185 und 233 ($d_h = 4$)
 - 16 Bit-Schlüssel: 58659, 59665, 63157, 63859 und 63877 ($d_h = 6$)
- Holen Sie die Vorgabe mittels: `git pull vorgabe master`

Für jede Operation zwischen zwei codierten Werten

ist eine eigene Funktion mit konstanten Signaturwerten notwendig!



Literatur

- [1] Forin.
Vital coded microprocessor principles and application for various transit systems.
IFA-GCCT, pages 79–84, 1989.



Fragen?

