TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Implementierung von Weavesort in VHDL

Dresden, 08.10.2015

Gliederung

Der Algorithmus
- Burrows-Wheeler-Transformation (BWT)
- Anwendung des Weavesort-Algorithmus
- Aufbau
- Ablauf

Die Implementierung
- Schnittstellen
- Ablauf
- Details
- Synthese
- Komplexität
- Zusammenfassung

DER ALGORITHMUS

Burrows-Wheeler-Transformation (BWT)

- Transformiert natürlichsprachlichen String in besser komprimierbare Zeichenfolge
- Gleiche Zeichen häufen sich
- Effektiv durch häufig vorkommende Zeichenfolgen in Sprachen, im Deutschen bspw.: „en“, „ie“

Anwendung des Weavesort-Algorithmus

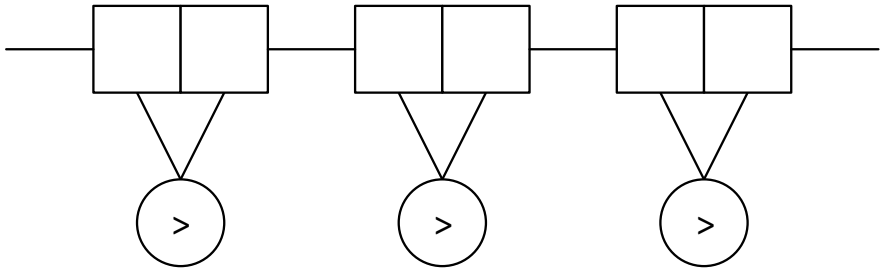
- Erzeugung sämtlicher, durch Rotation entstehende, zyklische Permutationen der Eingabe
- Anschließend lexikographische Sortierung der Permutationen

Anwendung - Erzeugung und Sortierung der Permutationen

Erzeugung	Sortierung
1. W – E – A – V – E – $	6. $ – W – E – A – V – E
2. E – A – V – E – $ – W	3. A – V – E – $ – W – E
3. A – V – E – $ – W – E	5. E – $ – W – E – A – V
4. V - E – $ – W – E – A	2. E – A – V – E – $ – W
5. E – $ – W – E – A – V	4. V – E – $ – W – E – A
6. $ – W – E – A – V - E	1. W – E – A – V – E – $

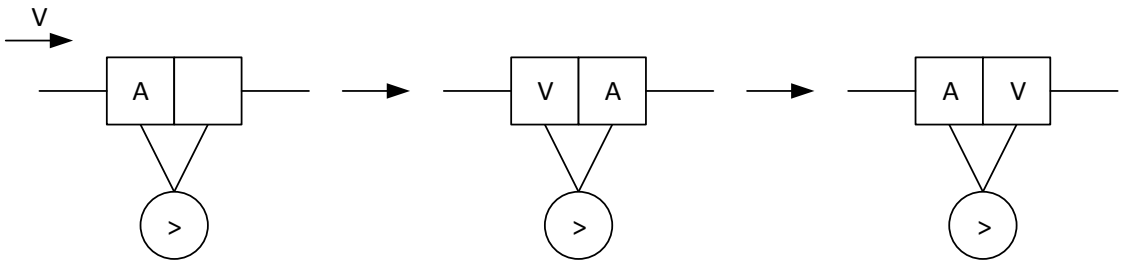
Aufbau – Zellen

- Erzeugung von Registern entsprechend der Anzahl der Zeichen
- Verbindung zweier Register jeweils durch einen Vergleicher (Zelle)



Aufbau - Funktionsweise der Zellen

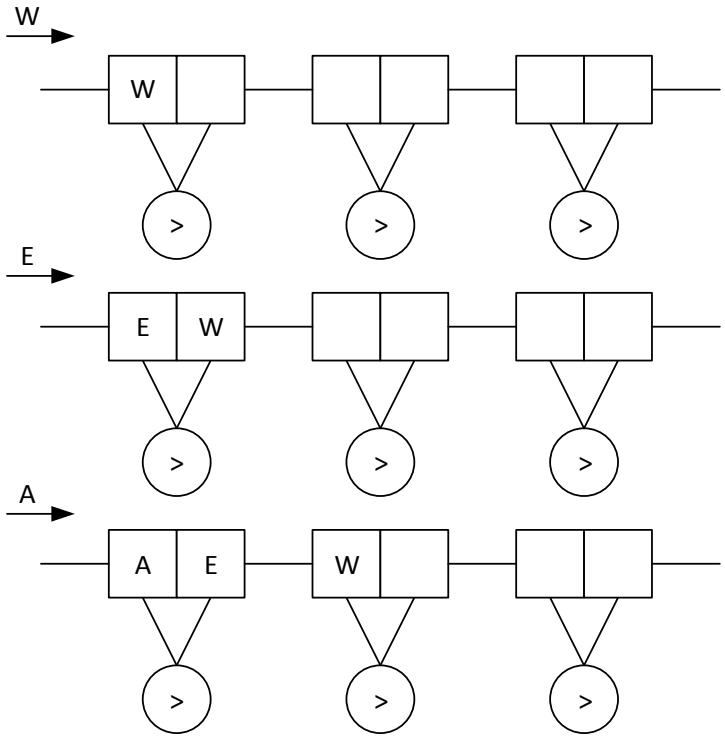
- Kleinerer Wert soll in linkem Register stehen, größerer im rechten
- Zu sortierende Zeichen werden nacheinander von einer Seite hineingeschoben
- Nach jedem Zeichen vergleichen Zellen ihre Register, tauschen gegebenenfalls



Aufbau - Hineinschiebevorgang, Teil 1

Zeichenfolge: „WEAVE$“

Schritt 1 – 3: Nur hineinschieben, keine Tauschvorgänge nötig

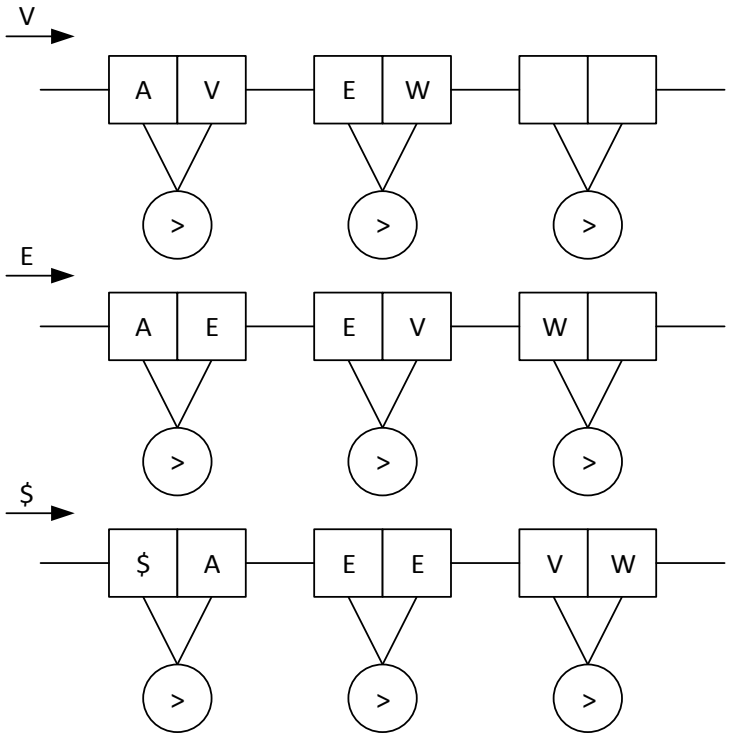


Aufbau - Hineinschiebevorgang, Teil 2

Schritt 4: V wird hineingeschoben, mit A getauscht

Schritt 5: E wird hineingeschoben, mit A getauscht, V mit E getauscht

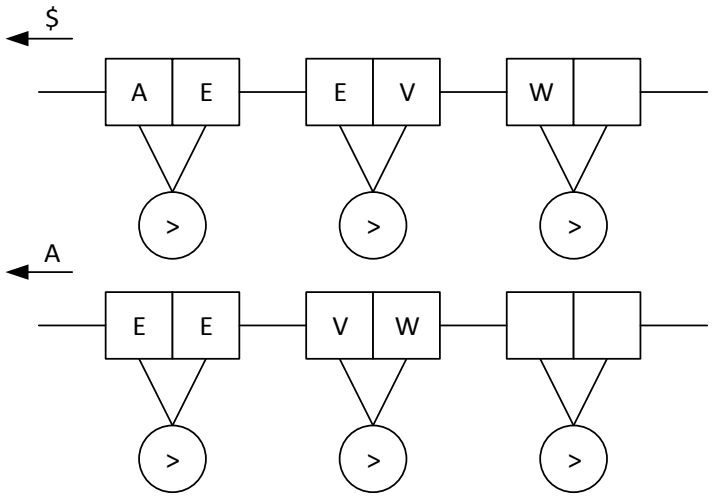
Schritt 6: ASC wird hineingeschoben



Aufbau - Herausschieben

Funktionsweise wie Hineinschieben, Schieberichtung umgekehrt

Reihenfolge der herausgeschobenen Zeichen:
$ – A – E – E – V – W



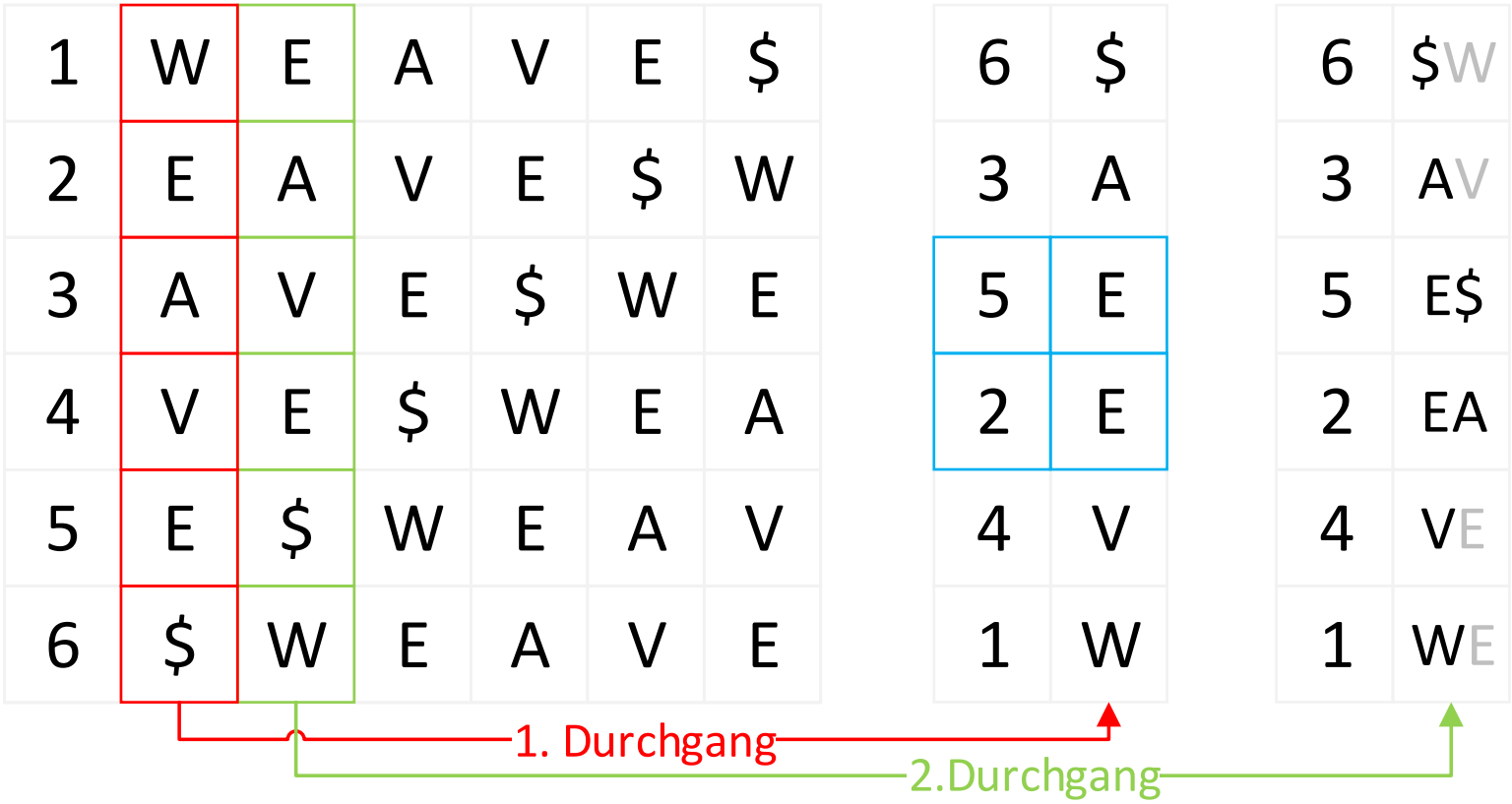
Aufbau - Korrektheit

- Beim Hineinschieben bleibt kleinstes Zeichen im linksten Register, wird zuerst hinausgeschoben
- Zweitkleinstes Zeichen im rechten Register der ersten Zelle oder im linken Register der zweiten Zelle
- Wird beim Herausschieben des kleinsten Zeichens in linkstes Register geschoben
- Aktuell kleinstes Zeichen immer im linksten Register

Ablauf

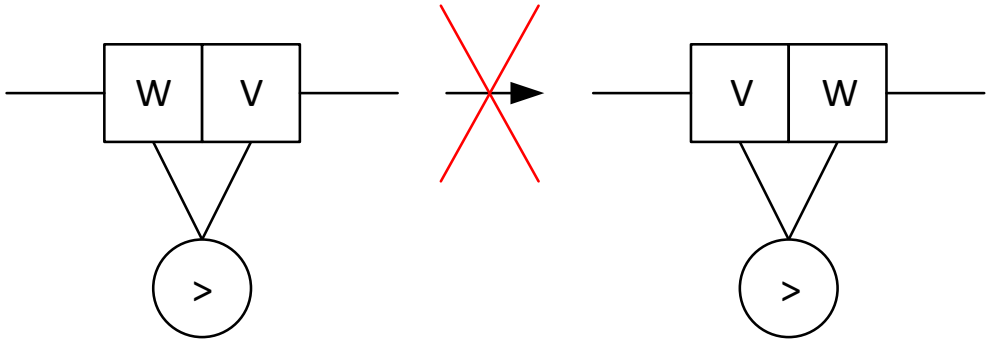
- 1. Durchgang mit erstem Zeichen jedes Strings
- Überprüfen, ob eindeutige Reihenfolge hergestellt
- Falls nicht, zweiten Durchgang mit zweitem Zeichen jedes Strings
- Fortsetzen, bis eindeutige Reihenfolge hergestellt

Ablauf - Beispiel



Ablauf - Deaktivierung der Vergleicher

- Gruppe: Strings, die noch nicht eindeutig sortiert sind
- Durch Sortierung bilden sich kleinere Gruppen
- Befinden sich Zeichen aus verschiedenen Gruppen in Zelle, kein Tausch → Lediglich Betrachtung der noch nicht eindeutig sortierten Strings



Ablauf - Optimierung

- Hereinschieben von rechts und herausschieben nach rechts auch möglich → Größtes Zeichen zuerst herausgeschoben, dann zweitgrößtes usw.

- Während aktuellen Zeichen herausgeschoben werden, Hineinschieben der Nachfolger → Höhere Effizienz

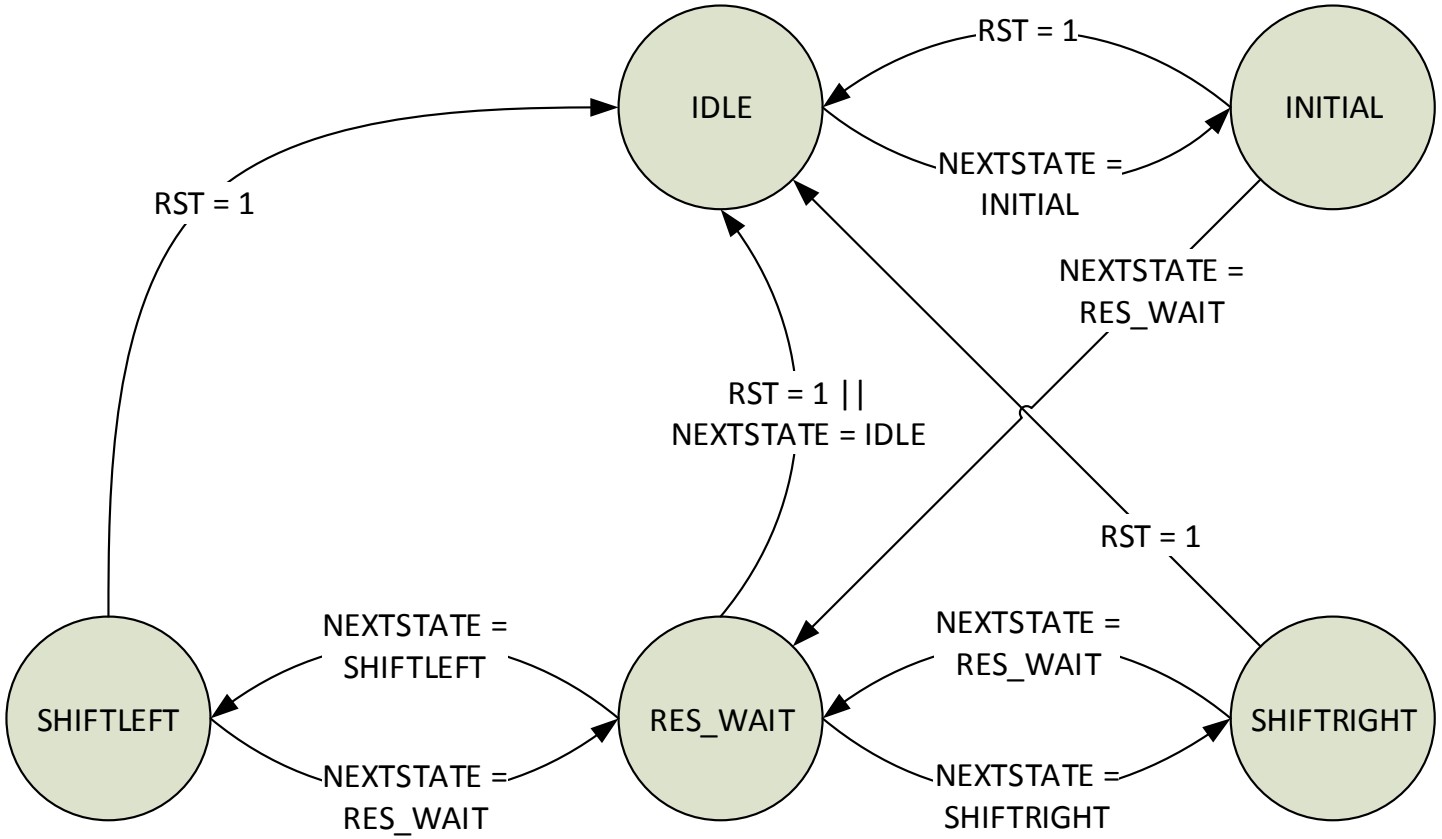
DIE IMPLEMENTIERUNG

Schnittstellen

Eingabe:
- STR_SLV: Bitcodierte Zeichen des Strings
- NUM_ELEM: Anzahl Zeichen
- ADR_WIDTH: Nötige Adressbreite für die Zeichen + 1
- CHAR_WIDTH: Bitbreite der Zeichen
- INIT: Signal zum initiieren des Algorithmus
- RST, CLK: Reset und Takt

Ausgabe:
- OUTPUT: Reihenfolge der Adressen

Ablauf


IDLE
INITIAL
SHIFTLEFT
RES_WAIT
SHIFTRIGHT
RST = 1
NEXTSTATE = INITIAL
NEXTSTATE = RES_WAIT
RST = 1
RST = 1 || NEXTSTATE = IDLE
RST = 1
NEXTSTATE = SHIFTLEFT
NEXTSTATE = RES_WAIT
NEXTSTATE = RES_WAIT
NEXTSTATE = SHIFTRIGHT

Details - Arrays, Teil 1

- INSERT_CHARS: Enthält den jeweiligen Nachfolger des aktuell herausgeschobenen Zeichens

- START_CHARS: Enthält die Adresse des Strings vom herausgeschobenen Zeichen

- ORDER: Enthält die aktuell (teil-)sortierte Reihenfolge + das Trennungsbit, OUTPUT enthält zum Schluss die Einträge von ORDER ohne Trennungsbit

Details - Arrays, Teil 2

Originalstring

0000	0001	0010	0011	0100	1111
W	E	A	V	E	$

INSERT_CHARS

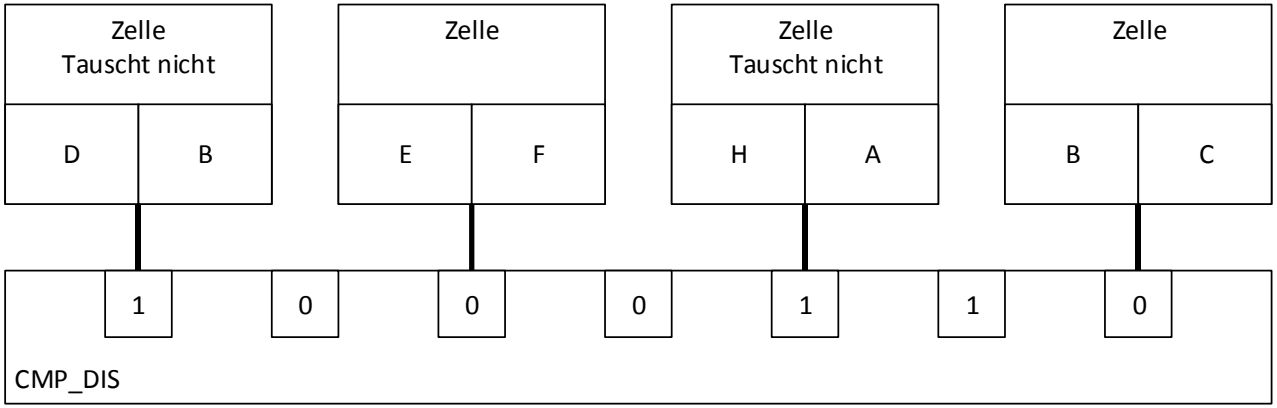
0001	0010	0011	0100	1111	0000
E	A	V	E	$	W

START_CHARS →

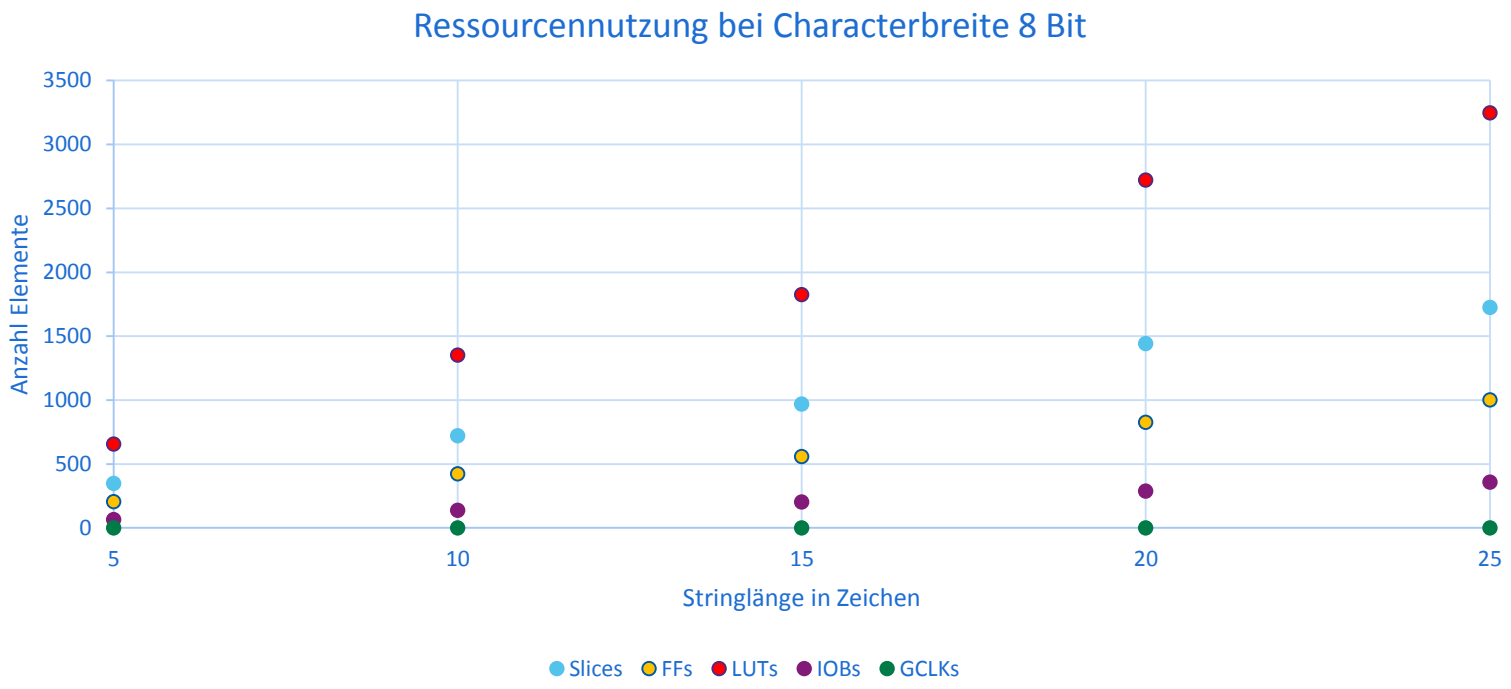
0000	0001	0010	0011	0100	1111

Details - Deaktivierung der Vergleicher

- CMP_DIS: Schieberegister, wird bei jedem Schiebevorgang in entsprechende Richtung mitgeschoben
- „1“ deaktiviert Vergleicher in entsprechender Zelle



Synthese - Ressourcenverbrauch



Komplexitätsklasse, Worst Case

- 2*n Schritte pro Durchlauf, durch Optimierung n Schritte (n = Länge des Strings)

- Komplexitätsklasse: n bis n^2

- Worst Case: Zwei oder mehrere Strings unterscheiden sich nur in letztem Zeichen: n Durchläufe notwendig

Zusammenfassung

- Sortieralgorithmus für Permutationen von Strings
- Komplexitätsklasse meist n (n = Länge des Strings)
- Durch Optimierung nahezu 100 % Nutzung der Register
- Nutzung von Arrays → Vermeidung der Speicherung aller Permutationen

Quelle

A. Mukherjee, N. Motgi, J. Becker, A. Friebe,

C. Habermann, M. Glesner:

„Prototyping of Efficient Hardware Algorithms for Data Compression in Future Communication Systems“, 2001