Entwurf eines booleschen Sortiernetzes mit der Struktur eines n– dimensionalen Würfels

Alexander Gamkrelidze

Diplomarbeit, angefertigt nach einem
Thema von Prof. Dr. Günter Hotz
im Fachbereich 14.1, Informatik,
Universität des Saarlandes

Dankansagungen

Als erstes möchte ich meinen Dank an die Leute äußern, die mir in verschiedener Art und Weise verholfen haben, diese Arbeit zu schaffen.

Zwei Professoren der Universität des Saarlandes gilt mein besonderer Dank:

Herrn Prof. Dr. Dr. h.c. mult Günter Hotz, der mich auf die Probleme dieser Arbeit aufmerksam machte und während meines ganzen Aufenthaltes an der Universität des Saarlandes stets mit Rat und Tat beiseite stand;

Herrn Prof. Dr. Dr. h.c. mult Gert Hummel, der mich — im Rahmen der Kooperation mit der Universität Tbilissi— nicht nur finanziell unterstützt hat und mein Studium an der Universität des Saarlandes ermöglichte.

Mein großer Dank geht an Herrn Dr. Thomas Burch — dem Betreuer dieser Arbeit, der mit mir verschiedene Probleme diskutiert und die Korrekturen sorgfältig durchgelesen hat.

Einleitung

Der rasche Wachstum der Industrie und Technik in den letzten Jahrzehnten verlangt immer schnellere Datenverarbeitung von den Rechensystemen. Das Problem der schnellen Datenverarbeitung hat man zu überwältigen, wenn man etwa 2^{25} Zahlen sortieren will. Mit einem sequentiellen System wäre das eine ziemlich zeitraubende Aufgabe, ganz zu schweigen vom Sortieren größerer Zahlenmengen. Aus diesem Grund versucht man, die Algorithmen zu parallelisieren, d.h., die Gesamtaufgabe (in unserem Beispiel Sortieren von Zahlenmengen) in Teilaufgaben zu zerlegen, diese einzeln parallel zu lösen und die Ergebnisse zusammenzufügen (oft ist es nötig, diese Prozesse rekursiv zu wiederholen).

Unter verschiedenen parallelen Systemen spielt der $n-$ dimensionale Würfel aufgrund seiner einfachen Struktur und logarithmischen Diameters (Anzahl der Verbindungen, die für die Kommunikation zweier Prozessoren durchgelaufen sein müssen) eine wichtige Rolle.
Für einen $n-$ dimensionalen Würfel sprechen unter anderem gute Eweiterbarkeit und Skalierbarkeit (Beibehaltung wesentlicher Eigenschaften bei beliebiger Erhöhung der Prozessorenzahl), sowie gute Einbettungsmöglichkeiten vieler anderer Netzwerkstrukturen, wie z. B. binärerer Bäume, Gitter u.s.w.

Als einige Beispiele der Würfelimplementierung kann man die Systeme iPSC/1 und iPSC/2 von Intel, sowie die nCUBE Systeme von NCUBE Corporation nennen.
Bei allen Vorteilen hat das System jedoch auch Nachteile, nämlich exponentielles Wachstum und keine kompakte darstellungsmöglichkeiten des Würfels. Das verlangt von dem Systementwickler die Teilung des Würfels in mehrere Teile, was später in dieser Arbeit erläutert wird.

Ein klassisches Problem der Informatik besteht in der Konstruktion effizienter paralleler Sortiersysteme, die ein wesentlicher Bestandteil paralleler Rechner sind und eines von denen in der hier vorliegenden Diplomarbeit konstruiert wurde. Das gewählte Sortiersystem implementiert das Verfahren Columnsort ([Lei 85]) auf einem $n-$ dimensionalen Würfel. In jedem Knoten des Würfels, der eine eindeutige Adresse enthält, wird ein und dieselbe Prozessorschaltung eingesetzt, die einen Speicher der Größe 2^{2n+1} enthält, in dem die zu sortierenden Zahlen in beliebiger Reihenfolge abgelegt werden (es wird also ein sog. Hybridsystem entwickelt, wo insgesamt 2^{3n+1} Zahlen sortiert werden).

Wir sagen, daß die Zahlen im Würfel sortiert sind, falls die Knoten mit kleineren Adressen die kleineren Zahlen enthalten und die Daten in jedem Knotenspeichert in aufsteigender Reihenfolge abgelegt sind.

Außer der Schaltungsentwicklung wird die Korrektheit des entworfenen Systems bewiesen, die oberen Schranken für Zeit– und Oberflächenwachstumfunktionen berechnet und die Verteilungsmethode der Schaltung angegeben.

Die Arbeit selbst ist wie folgt aufgeteilt:

Im Kapitel 1 wird die Würfelkonstruktionsmethode erläutert und deren Korrektheit bewiesen. Außerdem wird das Netz mit allen sog. Steuerleitungen konstruiert (siehe unten).

Das Verfahren Columnsort wird im Kapitel 2 beschrieben und die theoretischen Grundlagen für die Übertragung des Verfahrens auf den n– dimensionalen Würfel dargestellt.

Kapitel 3 befaßt sich mit der Übertragung des Verfahrens auf den n– dimensionalen Würfel. Außerdem wird hier der Algorithmus für die Datenein– und Ausgabe erläutert.

Die Knotenschaltung selbst ist im Kapitel 4 beschrieben, wo auch die Probleme der Synchronisation einzelner Teile besprochen werden.

Auf den Kapitel 4 gründet sich die Berechnung der oberen Schranken für Zeit– und Oberflächenfunktionen, die im Kapitel 5 berechnet werden.

Das letzte Kapitel dieser Arbeit befaßt sich mit der Verteilung der Würfelschaltung in mehrere Teile, was die Möglichkeit ergibt, große Schaltungen, die auf einem einzelnen Kristall nicht realisierbar sind, aus mehreren Chips zusammenzustellen.

Inhaltsverzeichnis

Kapitel 1

Beschreibung der Netzwerkstruktur

1.1 Beschreibung des n- dimensionalen Würfels

Die Verbindungsstruktur des $n-$dimensionalen Würfels ist durch einen Graphen $G_n = (V_n, E_n)$ gegeben, wobei V_n genau 2^n Elemente enthält. Jedem Knoten $v \in V_n$ wird eine eindeutige binäre Adresse zugeordnet, gemäß

$$adr : V_n \longrightarrow \{0,1\}^n.$$

Jeder Knoten hat Grad n, wobei jede von einem Knoten v ausgehende Kante eindeutig einer Dimension zugeordnet wird. Zwischen zwei Knoten $v, v' \in V_n$ mit $adr(v) = (a_{n-1}, ..., a_0)$ und $adr(v') = (a'_{n-1}, ..., a'_0)$ existiert genau dann eine Kante $e = (v, v')$, falls gilt:

$$\exists! i \in \{0, ..., n-1\} : a_i \neq a'_i.$$

Es sind also zwei Knoten über eine Kante der Dimension i genau dann miteinander verbunden, wenn sich ihre Adressen genau an der $i-$ten Stelle unterscheiden.
Die parametrisierte Beschreibung dieser Netzwerkstruktur wird im folgenden unabhängig von den Elementarknoten angegeben. Für die Verbindungsleitungen wird festgelegt, daß sie von einem zunächst unbestimmten Typ t sind, der erst durch die Realisierung des Basisknotens bestimmt wird. Eine Leitung vom Typ t kann beispielsweise eine Verbindung darstellen, die jeweils nur in einer Richtung benutzt werden darf, oder aber eine bidirektionale Verbindung repräsentieren. Einfachheitshalber wird angenommen, daß alle Verbindungsleitungen vom gleichen Typ t sind, was im allgemeinen der Realisierung eines $n-$dimensionalen Würfels entspricht.
Um eine kompakte topologische Beschreibung zu erhalten und dadurch die Verbindungsleitungen zwischen den einzelnen Knoten des Würfels möglichst kurz halten zu können, wird der k- dimensionale Würfel durch das Neben– oder Untereinandersetzen zweier $k-1$ –dimensionaler Teilwürfel und der Verbindung durch entsprechende Dimensionsleitungen konstruiert: bei den ungeraden k setzt man die Teilwürfel nebeneinander, bei den geraden dagegen übereinander.

Dies ist am Beispiel der Konstruktion eines dreidimensionalen Würfels in Abb. 1.1 dargestellt.

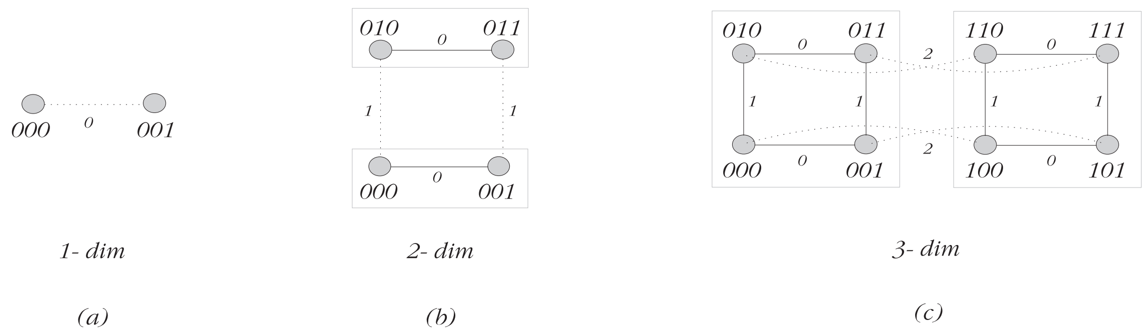


Abbildung 1.1: Konstruktionsschritte eines dreidimensionalen Würfels

Nach diesem Schema kann man jeden beliebigen $n-$ dimensionalen Würfel rekursiv beschreiben. Den Rekursionsabschluß bildet der Würfel der Dimension 0, der im wesentlichen aus dem Elementarknoten besteht.

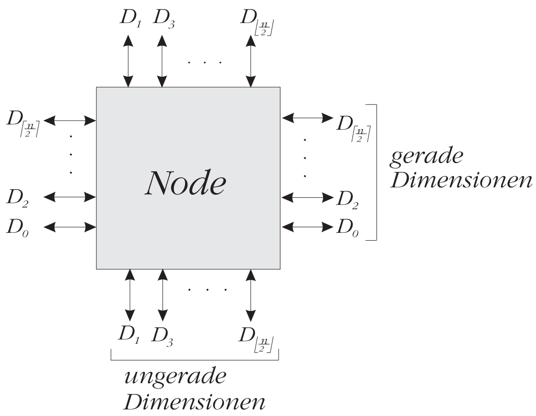


Abbildung 1.2: Schnittstelle des Elementarknotens

Da diese Beschreibung für alle n gelten soll, muß die Schnittstelle des 0– dimensionalen Würfels (Knotens) so beschrieben werden, daß die Verbindungsleitungen für alle höheren Dimensionen vorgesehen sind. Außerdem werden aus Symmetriegründen die Dimensionsleitungen wechselweise auf Ost/West (O/W) bzw. Nord/Süd (N/S) Seiten ausgeführt. Dabei werden auf Ost– und Westseiten (bzw. Nord und Südseiten) dieselben Leitungen durchgezogen (siehe Abb. 1.2).
Die Behandlung der Dimensionsleitungen während der Konstruktion eines $n-$ dimensionalen Würfels wird an folgendem Beispiel erläutert:

Beispiel 1.1: Konstruktion des dreidimensionalen Würfels

- Der 1– dimensionale Würfel wird durch die Nebeneinandersetzung zweier 0– dimensionaler Würfel (Knoten) und durch die Verbindung der 0– dimensionalen Kanten dargestellt. Die entsprechenden Dimensionsleitungen auf Ost– bzw. Westseiten werden abgeschnitten (projeziert):

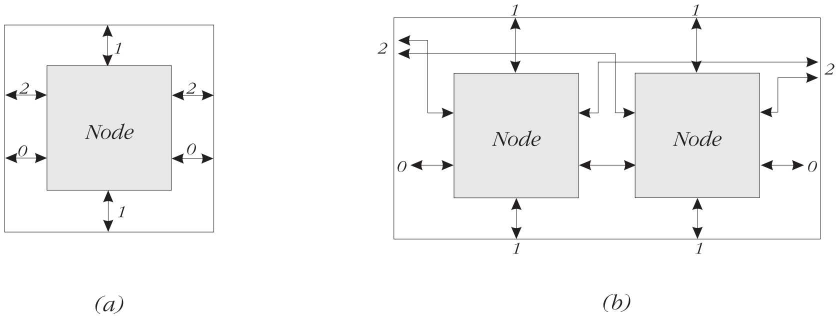


Abbildung 1.3: Der Knoten *(a)* und der 1– dimensionale Teilwürfel *(b)*

An diesem Beispiel kann man auch sehen, wie die Leitungen der 2. Dimension an den Knoten vorbeigeführt werden, um die Verbindungen auf höheren Stufen zu ermöglichen (die projezierten Leitungen kann man dadurch erkennen, daß sie nicht an die Ränder angeschlossen sind).

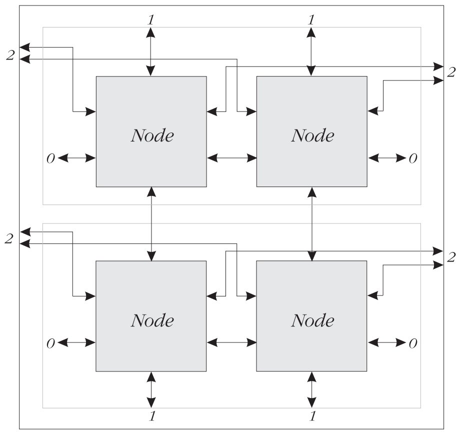


Abbildung 1.4: Konstruktion eines zweidimensionalen Teilwürfels

- Bei dem Entwurf eines 2– dimensionalen Würfels werden zwei 1– dimensionale Teilwürfel, die nach dem oberen Schema zusammengestellt wurden, übereinander gesetzt und durch die Kanten der 1. Dimension verbunden (Abb. 1.4).

  Auch hier werden die entsprechenden Dimensionsleitungen auf Nord– bzw. Südseiten der Teilwürfel projeziert. Dabei muß man beachten, daß die Leitungen der 0. Dimension schon im ersten Schritt projeziert wurden und auf der zweiten Hierarchieebene nicht mehr erscheinen.

- Betrachtet man nun das Konstruktionsschema eines dreidimensionalen Würfels, so sieht man, daß im Gegenteil zu den oben beschriebenen Schritten keine direkte Verbindung zwischen den Knoten bezüglich der Verbindungsleitungen der 2. Dimension gibt:

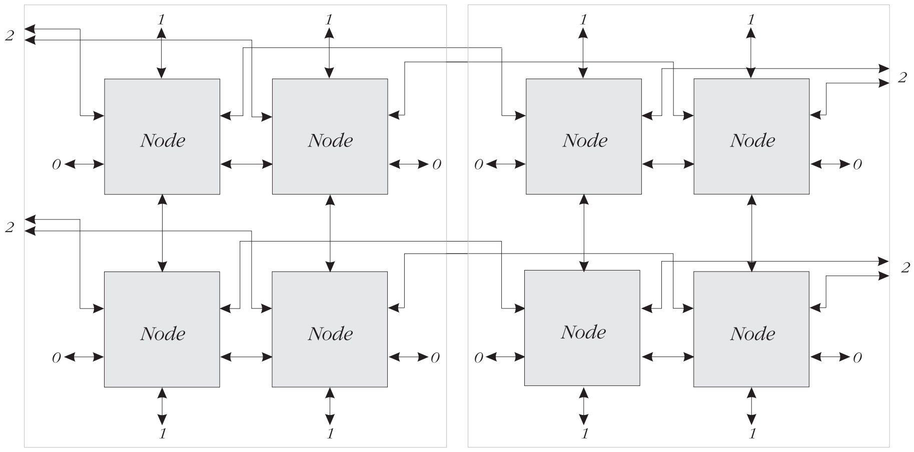


Abbildung 1.5: Konstruktion eines dreidimensionalen Würfels

  In diesem Schritt kann man die Leitungen auf der Ost– bzw. Westseite des linken bzw. rechten Teilwürfels direkt miteinander verbinden, da sie in den vorigen Schritten auf die richtigen Positionen geführt wurden.

Wie man am obigen Beispiel sehen kann, müssen in jedem Konstruktionsschritt einige Dimensionsleitungen an den Knoten vorbeigeführt und an die O/W bzw. N/S Seiten der Teilwürfel ausgegeben werden. Bisher wurde das ganze "manuell" gemacht, d.h., in jedem Schritt wurde solche Leitung extra umgeleitet und an den Nachbarknoten vorbeigeführt.
Nimmt man aber die Schaltung aus Abb. 1.6a mit ihren vorverarbeiteten Dimensionsleitungen als Basis für die zukünftige Konstruktion, so könnte man den Würfel durch das einfache Neben– bzw. Übereinandersetzen der Teilkomponenten und Verknüpfen derer Dimensionsleitungen auf den O/W bzw. N/S Seiten realisieren, da hier die "Umwege"

für alle Dimensionsleitungen (auch die, die in nächsten Schritten dazukommen) vorgesehen sind. Abb. 1.6b zeigt einen Zwischenschritt der Würfelkonstruktion, nämlich Zusammenstellung eines zweidimensionalen Teilwürfels aus zwei nebeneinander gesetzten Grundknoten (es bezeichne D_i das Leitungsbündel der i –ten Dimension).

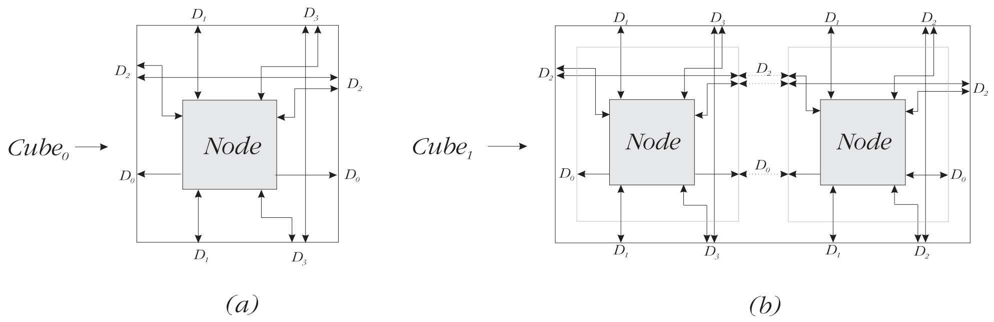


Abbildung 1.6: Vorverarbeitung der Leitungen (a) und der eindimensionale Teilwürfel (b)

Hier kann man deutlich sehen, wie die Leitungen der höheren Dimension dank der vorverarbeiteten Leitungen an

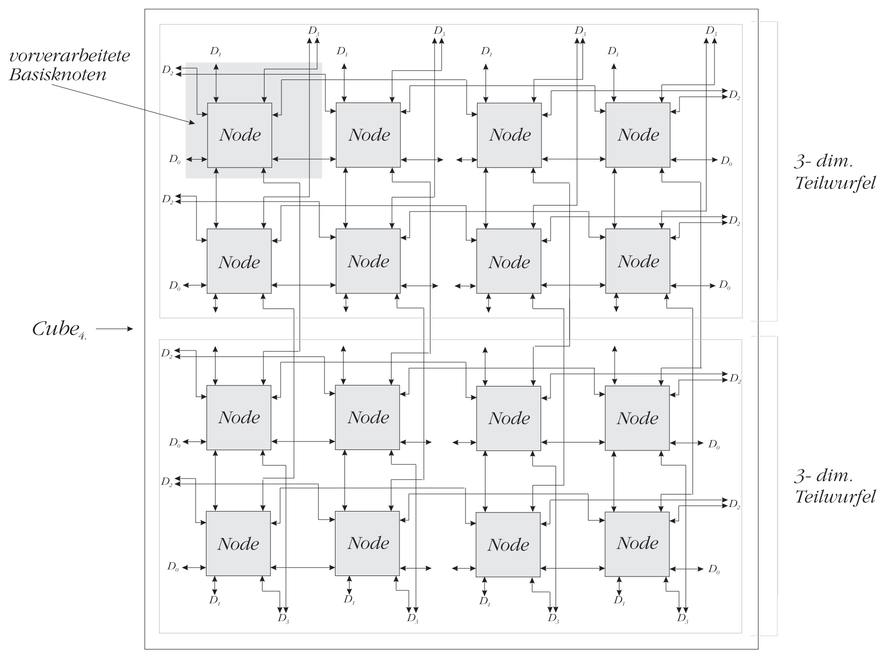


Abbildung 1.7: Der vierdimensionale Würfel

beiden Knoten vorbeigeführt werden. In diesem Beispiel sind die Leitungen des Grundknotens zur Konstruktion eines vierdimensionalen Würfels vorbereitet.
Abb. 1.7 zeigt den Abschluß der Iteration — einen 4– dimensionalen Würfel, der aufgrund des in Abb. 1.6a gezeigten Basisknotens konstruiert wurde.
Anhand des obigen Beispiels kann man sehen, daß in jedem i– ten Schritt die $i-1$ –ten Nachbarknoten entlang der $i-1$ –ten Dimension verbunden und die Leitungen D_{i-1} an den O/W bzw. N/S Seiten der Teilwürfel projeziert werden (Abb. 1.8).

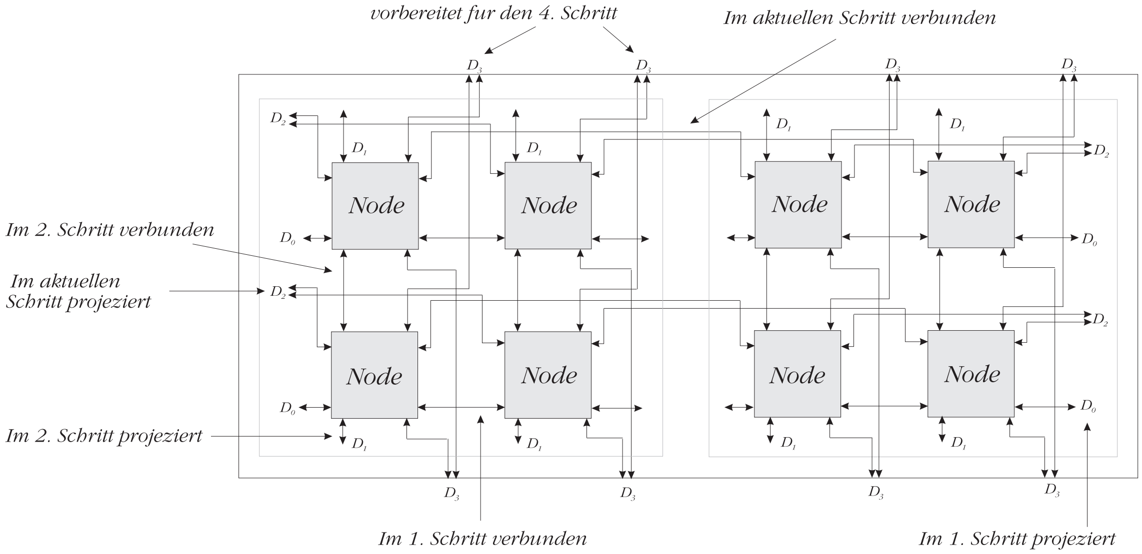


Abbildung 1.8: Verbundene und projezierte Leitungen

Bearbeitet man den Basisknoten wie in Abb. 1.6(a) gezeigt, so kann man den Würfel durch das einfache Neben– und Übereinandersetzen und direkte Verknüpfung der O/W– bzw. N/S Seiten der Teilwürfel erreichen.

Die Projektion und Durchziehung entsprechender Leitungen kann man mittels der Schaltungen $CutN_{n,m,j}$, $CutS_{n,m,j}$, $CutO_{n,m,j}$ und $CutW_{n,m,j}$ erreichen.
Die Schaltung $CutN_{n,m,j}$ ist in Abb. 1.9 definiert.
Dabei muß man bemerken, daß bei den 1– Bit Zahlen in jedem Schritt m je $2^{\lfloor\frac{m-1}{2}\rfloor}$ Leitungen projeziert werden sollen (bei den k– Bit Zahlen werden die Leitungen k–mal breiter); die Zahl der durchgezogenen Leitungen $@NC_{n,m}$ ergibt sich dann durch folgende Formel: $@NC_{n,m} = @S_{n,m} - 2^{\lfloor\frac{m-1}{2}\rfloor}$, wo $@S_{n,m}$ die Zahl der an die entsprechende Seite angeschlossenen Leitungen ist (hier und weiter zeigt das Zeichen @, daß es sich um eine Variable handelt).
Die Schaltung $CutN_{n,m,p}$ ergibt sich durch das 2^p –fache Nebeneinandersetzen von $CutN_{n,m,0}$ (Abb. 1.9b).

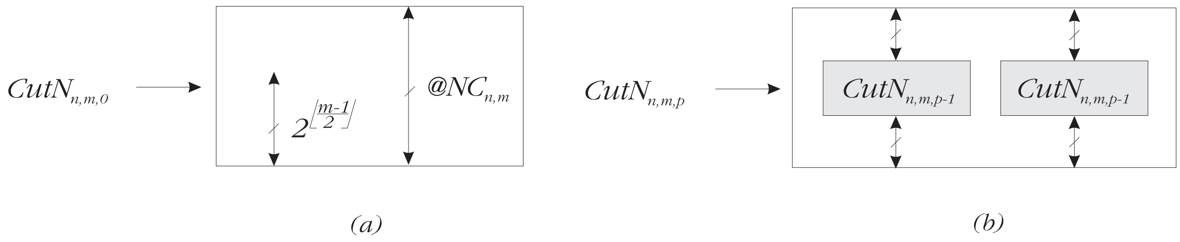


Abbildung 1.9: Die $Cut_{n,m,p}$ Schaltung

$CutS_{n,m,p}$, $CutW_{n,m,p}$ und $CutO_{n,m,p}$ dienen zur Projektion der Leitungen auf Süd–, West– und Ostseiten und sind in der Konstruktion zu $CutN_{n,m,p}$ analog.
Dabei ist $CutS_{n,m,p}$ eine Spiegelung an der horizontalen Achse, $CutW_{n,m,p}$ eine Drehung um 90^o der Schaltung $CutN_{n,m,p}$. $CutO_{n,m,p}$ dagegen ist eine Spiegelung der Schaltung $CutW_{n,m,p}$ an der vertikalen Achse.

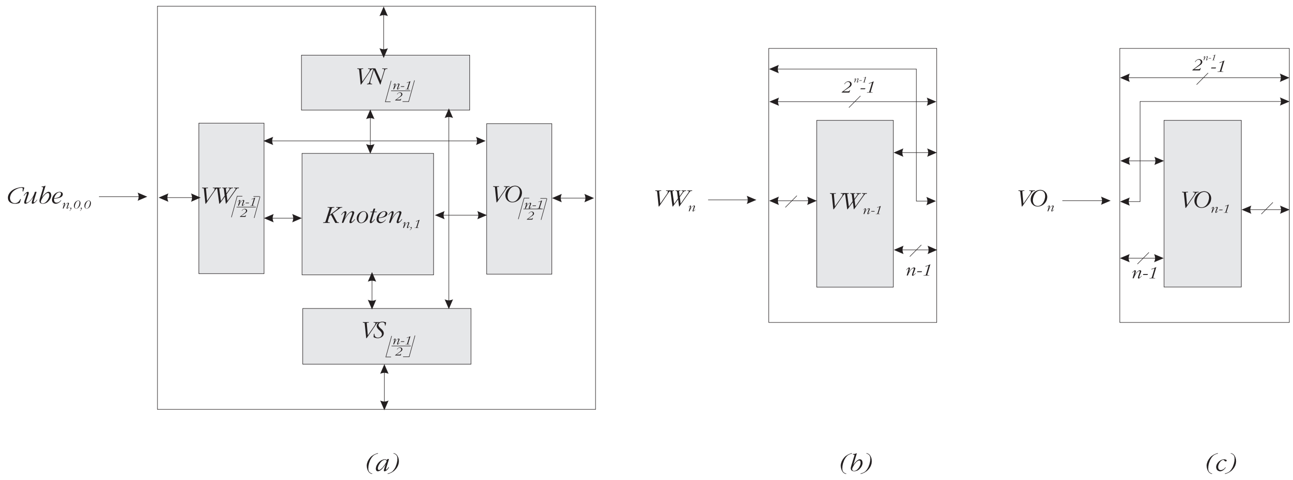


Abbildung 1.10: Die Vorverarbeitungsbausteine der Dimensionsleitungen

Den Vorverarbeitungsprozeß der Dimensionsleitungen kann man im allgemeinen wie in Abb. 1.10a beschreiben.
Die Schaltungen VN_k und VS_k stellen Drehungen um -90^o der Schaltungen VO_k bzw. VW_k dar, die in Abb. 1.10b und 1.10c beschrieben sind.

Benutzt man die bikategorielle Notation aus [KMO89], so ergibt sich folgende Würfelbeschreibung:

$$Cube_{n,j,k} = \begin{cases} CutW_{n,k,\lfloor\frac{k}{2}\rfloor} \ominus Cube_{n,0,k-1} \ominus Cube_{n,0,k-1} \ominus CutO_{n,k,\lfloor\frac{k}{2}\rfloor}, & \text{falls } k \text{ ungerade,} \\ CutN_{n,k,\lfloor\frac{k}{2}\rfloor} \oplus Cube_{n,1,k-1} \oplus Cube_{n,1,k-1} \oplus CutS_{n,k,\lfloor\frac{k}{2}\rfloor}, & \text{falls } k \text{ gerade.} \end{cases}$$

(hier und weiter in der Bezeichnung von $Cube_{n,j,k}$ gilt: $j = \lceil\frac{k}{2}\rceil - \lfloor\frac{k}{2}\rfloor$).

Folgender Satz beweist die Korrektheit des oben dargestellten Schemas:

Satz 1.1: *Für jede Schaltung $Cube_{n,1,2p+1}$ $(1 \leq 2p+1 \leq n)$ gilt:*

Es sind je zwei Teilkomponenten $Cube_{n,0,0}$ (Basisknoten) durch die Leitung der Dimension $0 \leq q \leq 2p+1$ miteinander verbunden, die in den früher ausgeführten Schritten projeziert wurden. Die restlichen Leitungen mit höheren geraden Dimensionen sind auf den O/W Seiten von $Cube_{n,1,2p+1}$ ausgeführt.

Dieser Satz besagt, daß in jedem ungeraden Konstruktionsschritt k die entsprechenden ($k-1$ –ten) Nachbarknoten durch die entsprechenden ($k-1$ –ten) Dimensionsleitungen verknüpft werden (als Beispiel siehe Abb.1.7). Da die Konstruktion in jedem geraden Schritt analog ist, kann man den Satz für beliebige (auch gerade) k erweitern.

Um den Beweis des Satzes besser zu verstehen, erläutern wir an dieser Stelle seine Idee. Betrachtet man die "flache“ Schaltung von $Cube_{n,j,k}$ (d.h. ihre Darstellung auf der niedrigsten Hierarchieebene durch die Elementarknoten), so kann man sehen, daß jede von diesen Schaltungen aus mehreren "Schichten“ der in gewisser Weise miteinander verbundenen Elementarknoten besteht. Diese Schichten, übereinander gesetzt und in gewisser Weise verbunden, bilden die Schaltung $Cube_{n,j,k}$ (siehe Beispiel in Abb. 1.11). Die Schichten mit 2^p Basisknoten bezeichnen wir mit $E_{n,p,0}$, die Schaltung, die durch Übereinandersetzen und Verknüpfungen 2^k solcher Schichte entsteht — mit $E_{n,p,k}$.
Es werden die Konstruktionsformeln von $E_{n,p,k}$ gegeben und gezeigt, daß $Cube_{n,j,k} = E_{n,k,k-j}$ gilt. Dies beweist die schichtweise Konstruktion von $Cube_{n,j,k}$.
Nachher wird die Aussage des Satzes für einzelne Schichten, also $E_{n,p,0}$ bewiesen, was gleichzeitig den Beweis des Satzes bedeutet, da die Konstruktion von $E_{n,k,k+j}$ aus zwei Teilen $E_{n,k,k+j-1}$ die O/W Seiten von den Schichten $E_{n,p,0}$ nicht beeinflußt.
Bevor der Satz bewiesen wird, betrachten wir erst einige Lemmata:

Lemma 1.1: Seien

$$E_{n,0,0} = Cube_{n,0,0}\ , \quad \begin{aligned} E_{n,p,0} &= CutW_{n,2p-1,0} \ominus E_{n,p-1,0} \ominus E_{n,p-1,0} \ominus CutO_{n,2p-1,0}; \\ E_{n,p,k} &= CutN_{n,2k,p} \oplus E_{n,p,k-1} \oplus E_{n,p,k-1} \oplus CutS_{n,2k,p}. \end{aligned}$$

Dann gilt für alle gerade k:

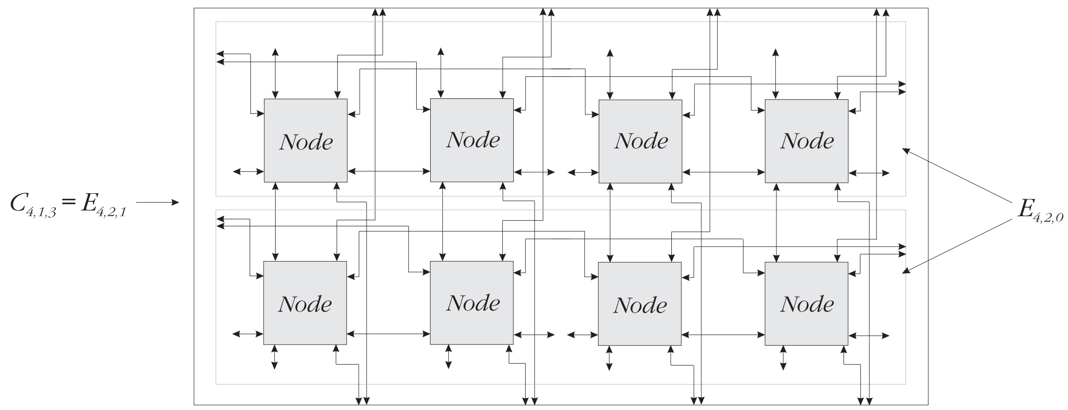


Abbildung 1.11: Die Schaltung $C_{4,1,3}$ und ihre Schichten

$$\begin{array}{lcl} E_{n,k+1,k} & = & Cube_{n,1,2k+1}, \\ E_{n,k,k} & = & Cube_{n,0,2k} \end{array}$$

Beweis:

$E_{n,p,k}$ hat folgende Eigenschaft:

$$E_{n,p,k} = CutW_{n,2p-1,k} \ominus E_{n,p-1,k} \ominus E_{n,p-1,k} \ominus CutO_{n,2p-1,k}.$$

(d.h., $E_{n,p,k}$ kann man, analog zu der "schichtweisen" Darstellung von $Cube_{n,j,k}$, auch spaltenweise darstellen).

Diese Eigenschaft folgt unmittelbar aus dem Distributivgesetz des bikategoriellen Kalküls (siehe auch Abb. 1.12).

Der Beweis des Lemmas wird mittels struktureller Induktion durchgeführt.
Für den Fall $k = 1$ gelten die Gleichungen offensichtlich:

$$\begin{array}{lcl} E_{n,1,0} & = & CutW_{n,1,0} \ominus Cube_{n,0,0} \ominus Cube_{n,0,0} \ominus CutO_{n,1,0} & = & Cube_{n,1,1}\ , \\ E_{n,1,1} & = & CutN_{n,2,1} \oplus E_{n,1,0} \oplus E_{n,1,0} \oplus CutS_{n,2,1} & = & \\ & = & CutN_{n,2,1} \oplus Cube_{n,1,1} \oplus Cube_{n,1,1} \oplus CutS_{n,2,1} & = & Cube_{n,0,2}. \end{array}$$

Gelte nun die Behauptung für alle gerade $k = 2p$: $Cube_{n,0,2p} = E_{n,p,p}$.

Wegen der oben erwähnten Eigenschaft von $E_{n,p,k}$ gilt:

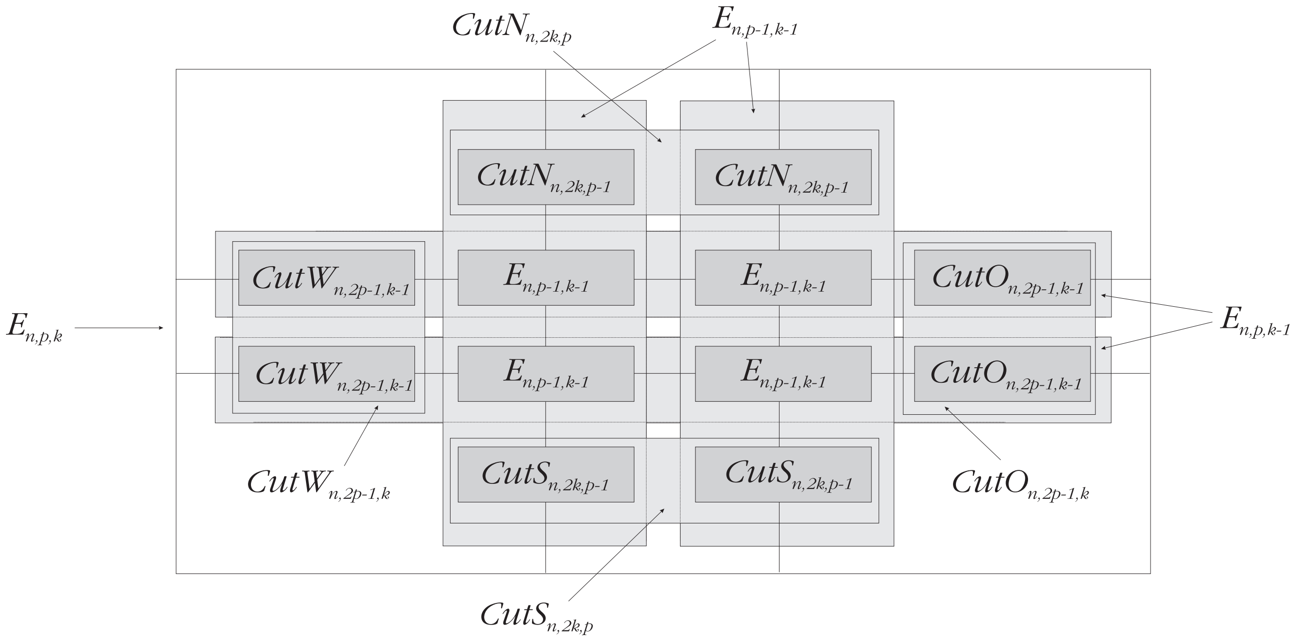


Abbildung 1.12: Zusammensetzung von $E_{n,p,k}$

$$\begin{aligned} E_{n,p+1,p} &= CutW_{n,2p+1,p} \ominus E_{n,p,p} \ominus E_{n,p,p} \ominus CutO_{n,2p+1,p} &= \\ &= CutW_{n,2p+1,p} \ominus Cube_{n,0,2p} \ominus Cube_{n,0,2p} \ominus CutO_{n,2p+1,p} &= \\ &= Cube_{n,1,2p+1}. \end{aligned}$$

Nehmen wir nun an, daß die Aussage für alle $k = 2p+1$ gilt, also $Cube_{n,1,2p+1} = E_{n,p+1,p}$, so kann man schreiben:

$$\begin{aligned} E_{n,p+1,p+1} &= CutN_{n,2p+2,p+1} \oplus E_{n,p+1,p} \oplus E_{n,p+1,p} \oplus CutS_{n,2p+2,p+1} &= \\ &= CutN_{n,2p+2,p+1} \oplus Cube_{n,1,2p+1} \oplus Cube_{n,1,2p+1} \oplus CutS_{n,2p+2,p+1} &= \\ &= Cube_{n,0,2p+2}. \end{aligned}$$

■

Lemma 1.2: Für $\forall n \in N$ gilt:

In jedem Konstruktionsschritt p von $E_{n,p,0}$ werden die Basisknoten $E_{n,0,0}$ in den Teilschaltungen $E_{n,p-1,0}$ paarweise durch die Leitungen der $2(p-1)$ – ten Dimension verbunden. Die restlichen Leitungen der geraden Dimensionen $\geq 2p$ werden auf die O/W Seiten ausgeführt.

Beweis:

O.B.d.A kann man annehmen, daß $n = 2k$ gerade ist (falls $n = 2k - 1$ ungerade, betrachten wir die Konstruktion von $E_{2k,p,0}$, die mit der Konstruktion von $E_{2k-1,p,0}$ übereinstimmt, was die Behandlung der geraden Dimensionsleitungen auf O/W Seiten betrifft).

Beweis des Satzes durch die Induktion über n:

I.A.: Gelte nun die Behauptung für $\forall q < n - 2$ (für $n \leq 4$ ist sie in obigen Konstruktionsbeispielen bewiesen).
Betrachten wir nun die Schaltung $E_{n,0,0}$, die man auch wie in Abb. 1.13 darstellen kann.

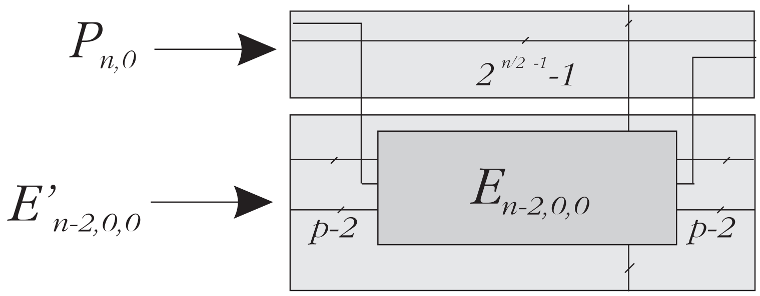


Abbildung 1.13: Darstellung von $E_{n,p,0}$

Es gilt also:

$$E_{n,0,0} = P_{n,0} \oplus E'_{n-2,0,0}.$$

Man beachte dabei, daß die beiden Schaltungen $E_{n,0,0}$ und $E'_{n,0,0}$ dieselben Schnittstellen auf O/W Seiten haben.

Seien ferner

$$\begin{array}{lcl} E'_{n,p,0} & = & CutW_{n,2p-1,0} \ominus E'_{n,p-1,0} \ominus E'_{n,p-1,0} \ominus CutO_{n,2p-1,0}; \\ P_{n,p} & = & P_{n,p-1} \ominus P_{n,p-1}. \end{array}$$

Es gilt für $\forall p \leq \frac{n}{2}$

$$E_{n,p,0} = P_{n,p} \oplus E'_{n-2,p,0}, \text{ denn:}$$

$$\begin{array}{rl} E_{n,p,0} = & (1^{2^{\frac{n}{2}-1}}_{\ominus} \oplus CutW_{n-2,2p-1,0}) \ominus E_{n,p-1,0} \ominus En,p-1,0 \ominus (1^{2^{\frac{n}{2}-1}}_{\ominus} \oplus CutO_{n-2,2p-1,0}) = \\ & (1^{2^{\frac{n}{2}-1}}_{\ominus} \oplus CutW_{n-2,2p-1,0}) \ominus (P_{n,p-1} \oplus E'_{n-2,p-1,0}) \ominus (P_{n,p-1} \oplus E'_{n-2,p-1,0}) \\ & (1^{2^{\frac{n}{2}-1}}_{\ominus} \oplus CutO_{n-2,2p-1,0}) = \\ & (P_{n,p-1} \ominus P_{n,p-1}) \oplus (CutW_{n-2,2p-1,0} \ominus E'_{n-2,p-1,0} \ominus E'_{n-2,p-1,0} \ominus CutO_{n-2,2p-1,0}) = \\ & P_{n,p} \oplus E'_{n-2,p,0} \end{array}$$

Da für $\forall p \leq \frac{n}{2}$ $E_{n,p,0}$ und $E'_{n-2,p,0}$ genau dieselben Schnittstellen auf den O/W Seiten haben, sollte nach der I.A. die Aussage des Lemmas auch für $E'_{n-2,p,0}$ gelten (siehe die Konstruktionsformel von $E'_{n-2,p,0}$).
Also gilt die Aussage auch für $E_{n,p,0}$, falls $p \leq \frac{n}{2}$ ist, da während der Konstruktion von $E_{n,p,0}$ $P_{n,p,0}$ keine Auswirkung auf die Dimensionsleitungen $2q$ $(0 \leq q \leq \frac{n}{2})$ hat.

Betrachten wir nun den letzten Schritt der Iteration, nämlich $p = \frac{n}{2}$. Die einzigen noch verbliebenen Leitungen auf den O/W Seiten der Teilschaltungen $E_{n,p-1,0}$ sind die von $P_{n,p-1}$.
Bezeichnen wir nun die Leitungen in $P_{n,p}$, die von den Ost– bzw. Westseiten auf die Südseite führen mit D, diejenigen, die durch die O/W Seiten durchgezogen werden — mit R (Abb. 1.14 zeigt diese Leitungen für $P_{n,0}$).
Wie man leicht sehen kann, bilden die Leitungen D auf der Westseite von $P_{n,p}$ das obere 2^{p-1} breite, die Leitungen R dagegen — das untere 2^{n-p-1} breite Leitungsbündel. Auf der Ostseite der Leitung ist das Gegenteil der Fall, da ist D über R plaziert (Abb. 1.14).

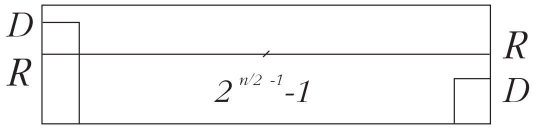


Abbildung 1.14: Leitungsbezeichnung in $P_{n,0}$

Im vorletzten Schritt der Iteration ($p = \frac{n}{2} - 1$) haben alle Leitungen den Typ D, d.h. sie führen in $E_{n,\frac{n}{2}-1,0}$ zu den Basiszellen $E_{n,0,0}$ undzwar zu den Leitungen der Dimension $\frac{n}{2}$. Nach der Verknüpfung zweier Teilschaltungen $E_{n,\frac{n}{2}-1,0}$ werden also je zwei Basiszellen entlang der Dimension $\frac{n}{2}$ verknüpft.
Was die Projektion diser Leitungen betrifft, so werden diese durch die *Cut*– Bausteine in $E_{n,\frac{n}{2},0}$ abgeschnitten.

■

Beweis des Satzes:

Der Beweis des Satzes folgt nun unmittelbar aus den obigen Lemmata. Im Lemma 1.2 wurde die Aussage des Satzes für jede "Schicht" von $Cube_{n,j,p}$ bewiesen. Da man nach Lemma 1.1 $Cube_{n,j,p}$ aus einzelnen "Schichten" $E_{n,p,0}$ konstruieren kann, ohne auf die O/W Seiten zu wirken, gilt die Behauptung für alle $Cube_{n,j,p}$.

■

1.2 Beschreibung des Schaltungnetzes

Im vorigen Abschnitt wurde die Struktur eines n– dimensionalen Würfels beschrieben, worauf sich die eigentliche Schaltung basiert ist. Außer der Dimensionsleitungen, durch die die Knoten miteinander verbunden sind, hat jeder Knoten auch sog. Steuer– und Adreßleitungen, die für die Zustandbestimmung eines jeden Knotens benötigt werden.

Wie es bereits erwähnt wurde, muß jedem Knoten eine eindeutige binäre Adresse zugeordnet werden. Diese wird in einem inneren Register des Knotens gespeichert.
Es gibt zwei Wege dies zu erreichen:

- Die Adresse wird während der Konstruktion der Knotenschaltung zugeordnet;
- Die Adresse wird während der Konstruktion des n– dimensionalen Würfels zugewiesen.

Aus Symmetriegründen fällt der erste Weg aus. Damit wird auch vermieden, für jeden Knoten eine neue Schaltung zu erzeugen.
Jeder Knoten besitzt gleiche Steuerleitungen, die *CK, Res* und andere Signalleitungen enthalten, die für den korrekten Ablauf des Verfahrens notwendig sind. Die genaue Beschreibung dieser Leitungen folgt im Kap. 4.
Da es für jeden Knoten die Steuerleitungen gleich sind, kann man sie in jedem Schritt der Würfelkonstruktion zusammenknüpfen (Abb. 1.16).

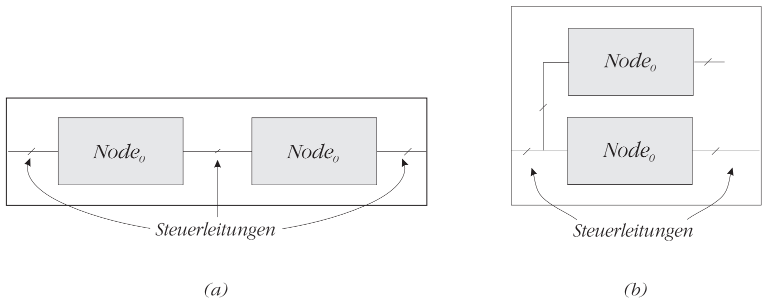


Abbildung 1.15: Behandlung der Steuerleitungen

Damit wird auch die relative Lage der Steuer– und Dimensionsleitungen nicht geändert. Außer der Steuerleitungen besitzt jeder Knoten auch $2n$ sog. Adreßleitungen. Diese sind paarweise an die Set– bzw. *Res*– Signale der entsprechenden Registerzelle angeschlossen (Abb. 1.17).
Sobald das k– te Bit des Adressregisters auf **'H'** gesetzt werden soll, wird die obere Leitung an die *Res*– Leitung angeknüpft. Die untere Leitung wird dann an die *LO*– Leitung, die auch in den Steuerleitungen enthalten ist, angeknüpft. Auf die *LO*– Leitung

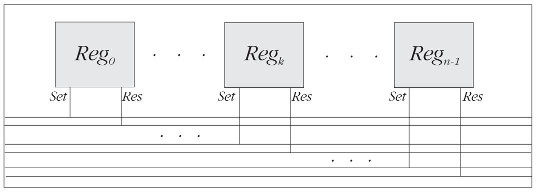


Abbildung 1.16: Initialisierungssignale des Adressregisters

wird später eine dem **'L'**– Signal entsprechende Spannung angelegt. Damit wird die dort verbundene Leitung deaktiviert. Falls das k– te Adressregisterbit auf **'L'** gesetzt sein soll, wird die untere Leitung an Res, die obere dagegen— an LO angeknüpft. Damit wird es erreicht, daß nach der L/H– Flanke von Res die entsprechende Adresszelle entweder auf **'H'** oder auf **'L'** gesetzt wird.

1.2.1 Die Adresszuweisung

Da die k– ten Nachbarknoten sich genau im k– ten Adressbit unterscheiden sollen, wird das in Abb. 1.18 dargestellte Zuweisungsschema vorgeschlagen (einfachheitshalber wird nur die relative Lage der Teilschaltungen ohne die Dimensions– und Steuerleitungen gezeigt).

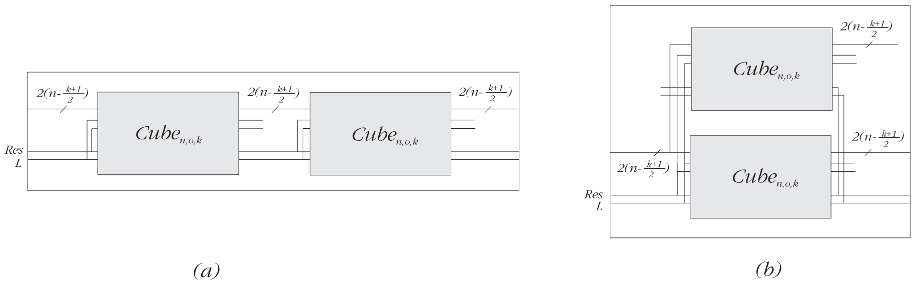


Abbildung 1.17: Die Adresszuweisung

Auf jeder Hierarchiestufe "verschwinden" je zwei Adressleitungen, so daß nach dem Ablauf des Würfelkonstruktionsverfahrens keine Adressleitung übrig bleibt. Wie man den Abbildungen 1.17a und 1.17b entnehmen kann, wird jedem Knoten des linken (bzw. oberen) k– dimensionalen Teilwürfels im k– ten Bit ein **'H'**– Signal zugeordnet. Den Knoten im rechten (bzw. unterem) k– dim. Teilwürfel wird in k– tem Bit ein **'L'**– Signal zugeordnet. Dies entspricht einer Adresszuweisung der Knoten in einem n–dimensionalen Würfel.

Hiermit ist die Konstruktion der $n-$ dim. Würfelschaltung komplett beschrieben. Abb. 1.19 zeigt ein vollständiges Layout eines vierdimensionalen Würfels, der nach dem obigen Schema konstruiert wurde.

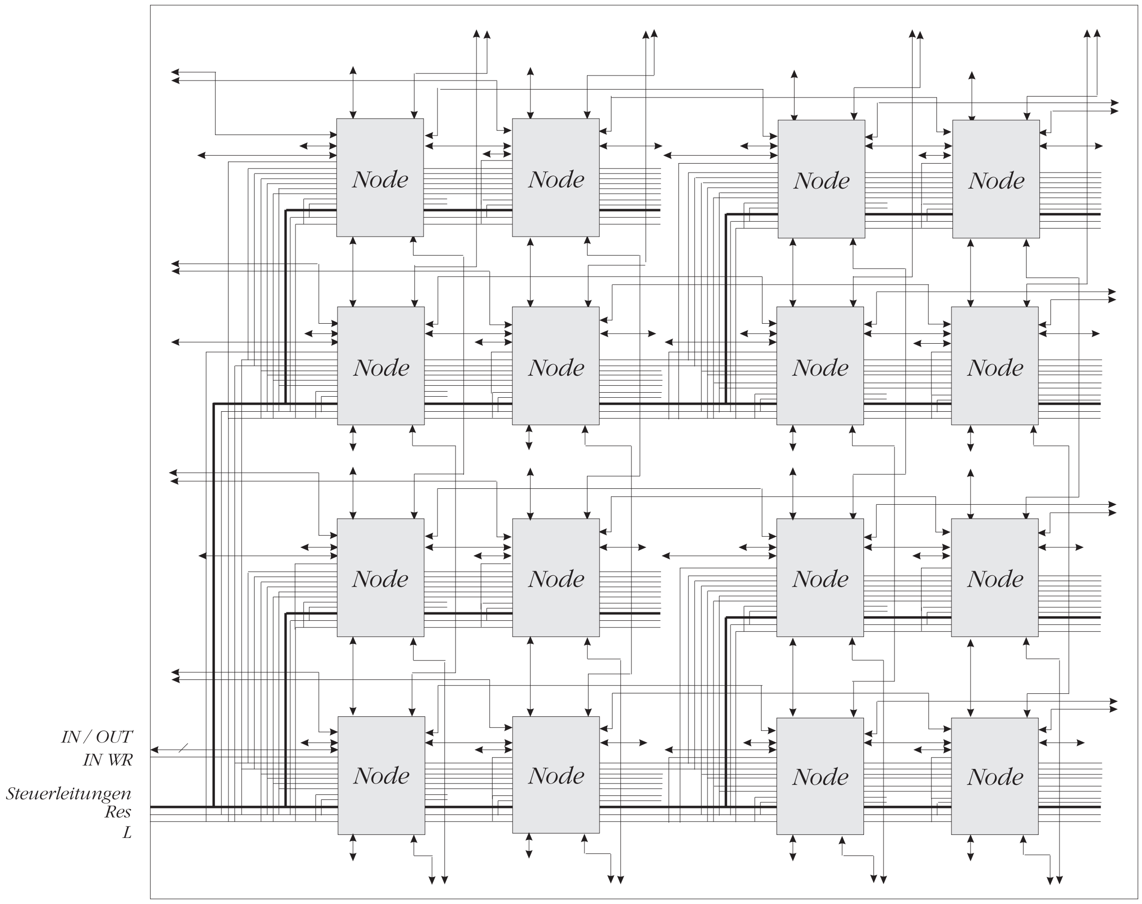


Abbildung 1.18: Der vierdimensionale Würfel

Kapitel 2

Theoretische Grundlagen

2.1 Column–Sort

In diesem Abschnitt wird der Elementarknoten des $n-$dimensionalen Würfels so konfiguriert, daß die gesamte Struktur einen parallelen Sortieralgorithmus beschreibt. Dazu wird das Verfahren Columnsort verwendet [Lei85].
das sich aufgrund seiner Struktur besonders einfach auf den $n-$dimensionalen Würfel übertragen läßt.
Columnsort ist eine Verallgemeinerung des Odd-Even-Mergesort Prinzips [Knu 73], dessen Beschreibung und Korrektheitsbeweis in **Anhang A** folgt. Am anschaulichsten läßt sich die Vorgehensweise von Columnsort anhand von Operationen auf einer Matrixdarstellung der zu sortierenden Elemente erklären. Sei dazu Q eine $r \times s$ Matrix von Elementen eines bestimmten Typs @t (z.B. $k-$bit Integer), d.h. es werden $N = r \cdot s$ Elemente sortiert, wobei $s|r$ und $r \geq 2(s-1)^2$ gelten soll:

$$Q = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,s-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,s-1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ a_{r-1,0} & a_{r-1,1} & \dots & a_{r-1,s-1} \end{pmatrix}$$

Nach dem Ablauf des Algorithmus liegen die Elemente in aufsteigender Reihenfolge vor, so daß auf Position i,j $(0 \leq i \leq r-1,\ 0 \leq j \leq s-1)$ von Q das $p = i + j \cdot r$ kleinste Element steht.
Columnsort arbeitet in acht Schritten. In den Schritten 1,3,5 und 7 werden jeweils die Elemente der Spalte sortiert. In den Schritten 2,4,6 und 8 dagegen werden die Matrixelemente in geeigneter Weise permutiert:

Schritte 1, 3, 5, 7: Spalten sortieren

In diesen Schritten werden die Matrixelemente in jeder Spalte unabhängig von den anderen Spalten parallel sortiert

Schritte 2, 4: Matrix transponieren (bzw. rücktransponieren):

Im Schritt 2 wird die spaltenweise Anordnung von oben nach unten gelesen und zeilenweise von links nach rechts abgelegt. Schritt 4 ist die Umkehrung des 2. Schrittes.

$$\begin{pmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,s-1} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{r-1,0} & a_{r-1,1} & \cdots & a_{r-1,s-1} \end{pmatrix} \underset{4}{\overset{2}{\rightleftarrows}} \begin{pmatrix} a_{0,0} & a_{1,0} & \cdots & a_{s-1,0} \\ a_{s,0} & a_{s+1,0} & \cdots & a_{2(s-1),0} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{r-s,s-1} & a_{r-s+1,s-1} & \cdots & a_{r-1,s-1} \end{pmatrix}$$

Dies entspricht nur im Fall $s = r$ einer echten Transposition der Matrix, aber der Einfachheit halber wird dieser Begriff auch für den Fall $r > s$ benutzt.

Schritte 6, 8: Zyklischer $\lfloor \frac{r}{2} \rfloor$-Shift der Elemente nach unten (bzw. nach oben), wobei die unteren $\lfloor \frac{r}{2} \rfloor$ Elemente der letzten Spalte in eine zusätzliche Spalte eingefügt werden. Die durch die Elementverschiebung entstandenen freien Plätze in der ersten bzw. zusätzlichen Spalte werden durch $-\infty$ bzw. $+\infty$ aufgefüllt:

$$\left(\begin{array}{|cccc|} \hline a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{\lfloor \frac{r}{2} \rfloor -1,0} & a_{\lfloor \frac{r}{2} \rfloor -1,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor -1,s-1} \\ \hline a_{\lfloor \frac{r}{2} \rfloor,0} & a_{\lfloor \frac{r}{2} \rfloor,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor,s-1} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ a_{r-1,0} & a_{r-1,1} & \cdots & a_{r-1,s-1} \\ \hline \end{array}\right) \underset{8}{\overset{6}{\rightleftarrows}} \left(\begin{array}{|cccc||c|} \hline -\infty & a_{\lfloor \frac{r}{2} \rfloor,0} & \cdots & a_{\lfloor \frac{r}{2} \rfloor,s-2} & a_{\lfloor \frac{r}{2} \rfloor,s-1} \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ -\infty & a_{r-1,0} & \cdots & a_{r-1,s-2} & a_{r-1,s-1} \\ \hline a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} & +\infty \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ a_{\lfloor \frac{r}{2} \rfloor -1,0} & a_{\lfloor \frac{r}{2} \rfloor -1,1} & \cdots & a_{\lfloor \frac{r}{2} \rfloor -1,s-1} & +\infty \\ \hline \end{array}\right)$$

Als Beispiel wird der Ablauf von *Columnsort* anhand einer 9×3 Matrix erläutert:

$$\begin{pmatrix} 8 & 22 & 6 \\ 27 & 18 & 21 \\ 5 & 11 & 24 \\ 17 & 23 & 2 \\ 9 & 26 & 19 \\ 25 & 10 & 3 \\ 12 & 13 & 7 \\ 14 & 15 & 4 \\ 1 & 16 & 20 \end{pmatrix} \xrightarrow{\text{Sort.}} \begin{pmatrix} 1 & 10 & 2 \\ 5 & 11 & 3 \\ 8 & 13 & 4 \\ 9 & 15 & 6 \\ 12 & 16 & 7 \\ 14 & 18 & 19 \\ 17 & 22 & 20 \\ 25 & 23 & 21 \\ 27 & 26 & 24 \end{pmatrix} \xrightarrow{\text{Transp.}} \begin{pmatrix} 1 & 5 & 8 \\ 9 & 12 & 14 \\ 17 & 25 & 27 \\ 10 & 11 & 13 \\ 15 & 16 & 18 \\ 22 & 23 & 26 \\ 2 & 3 & 4 \\ 6 & 7 & 19 \\ 20 & 21 & 24 \end{pmatrix} \xrightarrow{\text{Sort.}}$$

$$\begin{pmatrix} 1 & 3 & 4 \\ 2 & 5 & 8 \\ 6 & 7 & 13 \\ 9 & 11 & 14 \\ 10 & 12 & 18 \\ 15 & 16 & 19 \\ 17 & 21 & 24 \\ 20 & 23 & 26 \\ 22 & 25 & 27 \end{pmatrix} \xrightarrow[\text{transp.}]{\text{Rück-}} \begin{pmatrix} 1 & 9 & 17 \\ 3 & 11 & 21 \\ 4 & 14 & 24 \\ 2 & 10 & 20 \\ 5 & 12 & 23 \\ 8 & 18 & 26 \\ 6 & 15 & 22 \\ 7 & 16 & 25 \\ 13 & 19 & 27 \end{pmatrix} \xrightarrow{\text{Sort.}} \begin{pmatrix} 1 & 9 & 17 \\ 2 & 10 & 20 \\ 3 & 11 & 21 \\ 4 & 12 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 18 & 26 \\ 13 & 19 & 27 \end{pmatrix} \xrightarrow{\text{Shiften}}$$

$$\begin{pmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 18 & 26 \\ -\infty & 13 & 19 & 27 \\ 1 & 9 & 17 & +\infty \\ 2 & 10 & 20 & +\infty \\ 3 & 11 & 21 & +\infty \\ 4 & 12 & 22 & +\infty \\ 5 & 14 & 23 & +\infty \end{pmatrix} \xrightarrow{\text{Sort.}} \begin{pmatrix} -\infty & 6 & 15 & 24 \\ -\infty & 7 & 16 & 25 \\ -\infty & 8 & 17 & 26 \\ -\infty & 9 & 18 & 27 \\ 1 & 10 & 19 & +\infty \\ 2 & 11 & 20 & +\infty \\ 3 & 12 & 21 & +\infty \\ 4 & 13 & 22 & +\infty \\ 5 & 14 & 23 & +\infty \end{pmatrix} \xrightarrow[\text{shiften}]{\text{Rück-}} \begin{pmatrix} 1 & 10 & 19 \\ 2 & 11 & 20 \\ 3 & 12 & 21 \\ 4 & 13 & 22 \\ 5 & 14 & 23 \\ 6 & 15 & 24 \\ 7 & 16 & 25 \\ 8 & 17 & 26 \\ 9 & 18 & 27 \end{pmatrix}$$

In nächsten Abschnitten werden Funktionen $\mathcal{S}, \mathcal{T}, \mathcal{RT}, \mathcal{SH}$ und $\mathcal{RSH}$ definiert, mit deren Hilfe der Ablauf von *Columnsort* formal beschrieben werden kann. Nachher werden die Funktionen $\mathcal{T}', \mathcal{RT}', \mathcal{SH}'$ und $\mathcal{RSH}'$ definiert, die einfacher auf einen n– dimensionalen Würfel zu realisieren sind. Es wird auch bewiesen, daß sie anstelle der oben definierten Funktionen eingesetzt werden können, ohne die Korrektheit des Algorithmus zu verletzen.

2.1.1 Transponieren und Rücktransponieren

Sei $Q_{(r,s)} = \{Q : Q \text{ ist eine } r \times s \text{ Matrix } (s|r)\}$.
Es gelte: $r = l \cdot s$.
Dann kann man jede Matrix $Q \in Q_{(r,s)}$ folgendermaßen darstellen:

$$
Q = \begin{pmatrix}
\begin{array}{|cccc|}
\hline
a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\
\vdots & \vdots & & \vdots \\
a_{s-1,0} & a_{s-1,1} & \cdots & a_{s-1,s-1} \\
\hline
\end{array} \\
\begin{array}{|cccc|}
\hline
a_{s,0} & a_{s,1} & \cdots & a_{s,s-1} \\
\vdots & \vdots & & \vdots \\
a_{2s-1,0} & a_{2s-1,1} & \cdots & a_{2s-1,s-1} \\
\hline
\end{array} \\
\vdots \\
\begin{array}{|cccc|}
\hline
a_{(l-1)s-1,0} & a_{(l-1)s-1,1} & \cdots & a_{(l-1)s-1,s-1} \\
\vdots & \vdots & & \vdots \\
a_{l\cdot s-1,1} & a_{l\cdot s-1,2} & \cdots & a_{l\cdot s-1,s} \\
\hline
\end{array}
\end{pmatrix}
$$

wobei Q aus l quadratischen "Teilmatrixen" der Größe $s \times s$ besteht.

Seien ferner $\mathcal{S}, \mathcal{T}, \mathcal{T}' : Q_{(r,s)} \longrightarrow Q_{(r,s)}$ Funktionen, die folgendermaßen definiert sind:

$$a_{k\cdot s+i,j} \xrightarrow{\mathcal{T}} a_{j\cdot l+k,i}$$

$$a_{k\cdot s+i,j} \xrightarrow{\mathcal{T}'} a_{k\cdot s+j,i}$$

$(k \in \{0, ..., l-1\},\ i, j \in \{0, ..., s-1\})$.

Die Funktion $\mathcal{S}(Q)$ dagegen sortiere jede Spalte der Matrix $Q \in Q_{(r,s)}$.
(Hier steht $\mathcal{S}$ für Sortieren, $\mathcal{T}$ dagegen — für Transponieren.)

Anhand der Definition von $\mathcal{T}'$ und $\mathcal{T}$ kann man sehen, daß jede Spalte von $\mathcal{T}(Q)$ eine Permutation der Zahlen in der entsprechenden Spalte von $\mathcal{T}'(Q)$ ist. Deswegen gilt:

$$\mathcal{S}(\mathcal{T}(Q)) = \mathcal{S}(\mathcal{T}'(Q)).$$

Über $\mathcal{T}'$ und $\mathcal{T}$ kann man folgendes aussagen:

Die Funktion $\mathcal{T}$ entspricht dem in vorigen Abschnitt beschriebenen "Transponieren" der Matrix; Die Funktion $\mathcal{T}'$ dagegen — dem schrittweisen transponieren jeder quadratischen "Teilmatrix" von Q.

Diese Aussage folgt unmittelbar aus den oben angegebenen Formeln für $\mathcal{T}$ und $\mathcal{T}'$.

Da einem Transpositionsschritt stets ein Sortierschritt folgt und $\mathcal{S}(\mathcal{T}(Q)) = \mathcal{S}(\mathcal{T}'(Q))$ ist, kann man ohne Verletzung der Korrektheit von *Columnsort* die Funktion $\mathcal{T}'$ in zweitem Schritt des Verfahrens implementieren.

Betrachten wir nun eine eine andere Darstellung der Matrix Q, nämlich ihre Zusammenstellung aus den "Teilmatrixen" der größe $l \times s$. Dann gilt für das Rücktransponieren der Matrix:

Die "Rücktransponierfunktion" $\mathcal{RT}$ sieht folgendermaßen aus:

$$a_{i \cdot l+k,j} \xrightarrow{\mathcal{RT}} a_{k \cdot s+j,i}.$$

Sei nun $\mathcal{RT}' : Q_{(r,s)} \longrightarrow Q_{(r,s)}$ eine Funktion mit

$$a_{i \cdot l+k,j} \xrightarrow{\mathcal{RT}'} a_{j \cdot l+k,i}$$

$(k \in \{0, ..., l-1\}, \quad i, j \in \{0, ..., s-1\})$.

Dann ist jede Spalte von $\mathcal{RT}(Q)$ eine Permutation der entsprechenden Spalte von $\mathcal{RT}'(Q)$.

Deswegen gilt:

$$\mathcal{S}(\mathcal{RT}(Q)) = \mathcal{S}(\mathcal{RT}'(Q)).$$

Also kann man als "Rücktransponieren" die Funktion $\mathcal{RT}'$ implementieren.
Man bemerke dabei, daß die Funktion $\mathcal{RT}'$ jede j– te "Teilmatrix" der Größe $l \times s$ von Q in die j– te Spalte überführt:

$$\begin{pmatrix} \boxed{\begin{array}{cccc} a_{0,0} & a_{0,1} & \cdots & a_{0,s-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{l-1,0} & a_{l-1,1} & \cdots & a_{l-1,s-1} \end{array}} \\ \boxed{\begin{array}{cccc} a_{l,0} & a_{l,1} & \cdots & a_{l,s-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{2l-1,0} & a_{2l-1,1} & \cdots & a_{2l-1,s-1} \end{array}} \\ \vdots \\ \boxed{\begin{array}{cccc} a_{(s-1)l-1,0} & a_{(s-1)l-1,1} & \cdots & a_{(s-1)l-1,s-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{l\cdot s-1,0} & a_{l\cdot s-1,1} & \cdots & a_{l\cdot s-1,s-1} \end{array}} \end{pmatrix} \xrightarrow{\mathcal{RT}'} \begin{pmatrix} \boxed{\begin{array}{cccc} a_{0,0} & a_{l,0} & \cdots & a_{l\cdot(s-1)-1,0} \\ \vdots & \vdots & \vdots & \vdots \\ a_{l-1,0} & a_{2l-1,0} & \cdots & a_{l\cdot s-1,0} \end{array}} \\ \boxed{\begin{array}{cccc} a_{0,r} & a_{l,1} & \cdots & a_{l(s-1)-1,1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{l-1,1} & a_{2l-1,0} & \cdots & a_{l\cdot s-1,1} \end{array}} \\ \vdots \\ \boxed{\begin{array}{cccc} a_{0,s-1} & a_{l,s-1} & \cdots & a_{l(s-1)-1,s-1} \\ \vdots & \vdots & \vdots & \vdots \\ a_{l-1,s-1} & a_{2l-1,s-1} & \cdots & a_{l\cdot s-1,s-1} \end{array}} \end{pmatrix}$$

2.1.2 Shiften und Rückshiften

Sei $\{Q^i\}_{i=1}^5$ eine Folge der Matrixen, die folgendermaßen definiert ist:

$$\begin{aligned} Q^1 &= \mathcal{S}(Q) \\ Q^2 &= \mathcal{T}'(Q^1) \\ Q^3 &= \mathcal{S}(Q^2) \\ Q^4 &= \mathcal{RT}'(Q^3) \\ Q^5 &= \mathcal{S}(Q^4) \end{aligned}$$

Sei ferner Q^5 folgendermaßen dargestellt:

$$Q^5 = \begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_0 & B_1 & \dots & B_{s-1} \end{pmatrix}$$

Dabei ist A_j die obere, B_j dagegen— die untere Hälfte der j– ten Spalte dieser Matrix[1].

[1] Einfachhetshalber wird hier und weiter r als gerade Zahl angenommen

Die Elemente in A_j bzw. B_j sind sortiert, da im 5. Schritt des Sortieralgorithmus jede Spalte von Q^4 für sich sortiert wird.

Seien $\mathcal{SH}'$ und $\mathcal{RSH}'$ zwei Funktionen, die folgendermaßen definiert sind:

$$\begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_0 & B_1 & \dots & B_{s-1} \end{pmatrix} \xrightarrow{\mathcal{SH}'} \begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_{s-1} & B_0 & \dots & B_{s-2} \end{pmatrix}$$

$$\begin{pmatrix} A'_0 & A'_1 & \dots & A'_{s-1} \\ B'_0 & B'_1 & \dots & B'_{s-1} \end{pmatrix} \xrightarrow{\mathcal{RSH}'} \begin{pmatrix} A'_0 & A'_2 & \dots & B'_0 \\ A'_1 & B'_1 & \dots & B'_{s-1} \end{pmatrix}$$

Dabei gelte:

$$Q^6 = \mathcal{SH}'(Q^5) = \begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_{s-1} & B_0 & \dots & B_{s-2} \end{pmatrix},$$

$$\mathcal{S}\left(\begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_{s-1} & B_0 & \dots & B_{s-2} \end{pmatrix}\right) = \begin{pmatrix} A'_0 & A'_1 & \dots & A'_{s-1} \\ B'_{s-1} & B'_0 & \dots & B'_{s-2} \end{pmatrix}.$$

Da nach *[Lei 85]* A_1 die $\frac{r}{2}$ minimalen, B_{s-1} dagegen — die $\frac{r}{2}$ maximalen Elemente von Q enthalten, und da A_0 und B_{s-1} vorsortiert sind, gilt:

$$Q^7 = \mathcal{S}\left(\begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_{s-1} & B_0 & \dots & B_{s-2} \end{pmatrix}\right) = \begin{pmatrix} A_0 & A'_1 & \dots & A'_{s-1} \\ B_{s-1} & B'_0 & \dots & B'_{s-2} \end{pmatrix}.$$

Dann gilt:

$$\mathcal{RSH}'(Q^7) = \begin{pmatrix} A_0 & A'_2 & \dots & B_{s-1} \\ A'_1 & B'_0 & \dots & B'_{s-1} \end{pmatrix}.$$

Seien ferner $\mathcal{SH} : Q_{(r,s)} \longrightarrow Q_{(r,s+1)}$ und $\mathcal{RSH} : Q_{(r,s+1)} \longrightarrow Q_{(r,s)}$ zwei Funktionen, die die oben beschriebenen Shift– bzw. Rückshiftschritte fon *Columnsort* realisieren. Für diese Funktionen gilt folgendes Schema:

$$\begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_0 & B_1 & \dots & B_{s-1} \end{pmatrix} \xrightarrow{\mathcal{SH}} \begin{pmatrix} -\infty & B_0 & \dots & B_{s-2} & B_{s-1} \\ A_0 & A_1 & \dots & A_{s-1} & +\infty \end{pmatrix}$$

$$\begin{pmatrix} -\infty & C_0 & \dots & C_{s-1} & C_{s-1} \\ D_0 & D_1 & \dots & D_{s-1} & +\infty \end{pmatrix} \xrightarrow{\mathcal{RSH}} \begin{pmatrix} D_0 & D_1 & \dots & D_{s-1} \\ C_0 & C_1 & \dots & C_{s-1} \end{pmatrix}$$

Nach *[Lei85]* und nach der Gleichung

$$\mathcal{S}\left(\begin{pmatrix} A_0 & A_1 & \dots & A_{s-1} \\ B_{s-1} & B_0 & \dots & B_{s-2} \end{pmatrix}\right) = \begin{pmatrix} A_0 & A'_1 & \dots & A'_{s-1} \\ B_{s-1} & B'_0 & \dots & B'_{s-2} \end{pmatrix}$$

gilt:

$$\mathcal{S}\left(\begin{pmatrix} -\infty & B_0 & \dots & B_{s-2} & B_{s-1} \\ A_0 & A_1 & \dots & A_{s-1} & +\infty \end{pmatrix}\right) = \begin{pmatrix} -\infty & A'_1 & \dots & A'_{s-1} & B_{s-1} \\ A_0 & B'_0 & \dots & B'_{s-2} & +\infty \end{pmatrix}.$$

Aus

$$\begin{pmatrix} -\infty & A'_1 & \dots & A'_{s-1} & B_{s-1} \\ A_0 & B'_0 & \dots & B'_{s-2} & +\infty \end{pmatrix} \xrightarrow{\mathcal{RSH}} \begin{pmatrix} A_0 & B'_0 & \dots & B'_{s-2} \\ A'_1 & A'_2 & \dots & B_{s-1} \end{pmatrix} = \overline{Q}$$

und

$$\begin{pmatrix} A_0 & A'_1 & \dots & A'_{s-1} \\ B_{s-1} & B'_0 & \dots & B'_{s-2} \end{pmatrix} \xrightarrow{\mathcal{RSH}'} \begin{pmatrix} A_0 & A'_2 & \dots & B_{s-1} \\ A'_1 & B'_0 & \dots & B'_{s-2} \end{pmatrix} = Q^8$$

kann man leicht sehen, daß jede Spalte von $\overline{Q}$ eine Permutation der entsprechenden Spalte von Q^8 ist.
Es gilt also:

$$\mathcal{S}(Q^8) = \mathcal{S}(\overline{Q}).$$

Deswegen könnte man in Schritten 6 und 8 des Sortierverfahrens die Funktionen $\mathcal{SH}$ bzw. $\mathcal{SH}'$ implementieren, ohne dessen Korrektheit zu verletzen.

Die entsprechende Permutation der Zahlen in Q^8 konnte man bei der Ausgabe der sortierten Elemente berücksichtigen, was in weiteren Kapiteln erläutert wird.

Kapitel 3

Übertragung des Verfahrens auf einen $n-$ dimensionalen Würfel

Wie man dem vorigen Kapitel entnehmen kann, besteht die Übertragung des *Columnsort*– Verfahrens auf einen n– dimensionalen Würfel in der Implementierung der Funktionen $\mathcal{S}, \mathcal{T}', \mathcal{RT}', \mathcal{SH}'$ und $\mathcal{RSH}'$ in gegebener Reihenfolge.
Dabei wird jeder Spalte von Q ein Knoten zugeordnet. Man geht also von einer Matrix mit 2^n Spalten aus. Wegen der Korrektheitsbedingung von *Columnsort* ($s|r$ und $r \geq 2(s-1)^2$) gilt: $r = 2^n \cdot 2^{n+1} = 2^{2n+1}$.
In jedem Knoten werden die Daten in einem inneren Speicher der Größe 2^{2n+1} abgelegt. Insgesamt werden also 2^{3n+1} Elemente sortiert.

3.1 Implementierung von $\mathcal{S}, \mathcal{T}'$ und $\mathcal{RT}'$

Als eine Implementierung der Funktion $\mathcal{S}$, die jede Spalte von Q (hier die Daten in jedem Knoten) sortiert, verwendet man den Odd– Even– Mergesortierer (kurz *OEM*), der im **Anhang A** beschrieben wird.

In diesem Abschnitt wird unter anderem die Methode der Transposition einer quadratischen Matrix $\overline{Q}$ dargestellt. Nach dieser Methode kann man die ganze Matrix Q nach dem Schema von $\mathcal{T}'$ "transponieren", indem man jede quadratische Teilmatrix von Q schrittweise transponiert (siehe Kap. 2).

Sei $\overline{Q}_n$ eine $2^n \times 2^n$ Matrix. Das Problem bei der Transposition der Matrix auf einem n– dimensionalen Würfel besteht darin, daß es keine Verbindung zwischen allen Knoten des Würfels gibt (abb. 3.1).

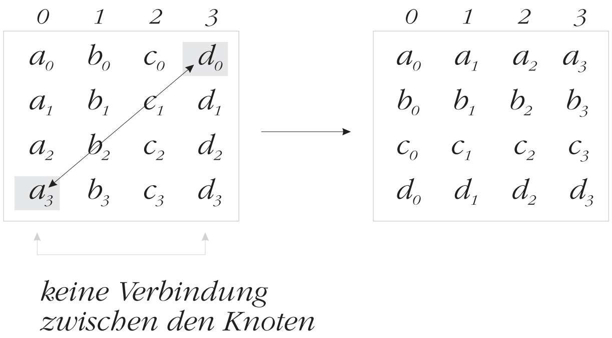


Abbildung 3.1: Probleme bei der Transposition einer Matrix

Wie man sieht, muß das Element a_3, was sich in der 0. Spalte befindet, nach der Transposition in die 3. Spalte gelangen. Es besteht aber keine direkte Verbindung zwischen den entsprechenden Knoten 0 und 3 in einem 2– dimensionalen Würfel. Deswegen muß dieses Element durch gewisse Umwege die richtige Position einnehmen. Dies geschieht in zwei Schritten:

Schritt 1: Die untere Hälfte der 0. bzw. 1. Spalte wird mit der oberen Hälfte der 2. bzw. 3. Spalte ausgetauscht:

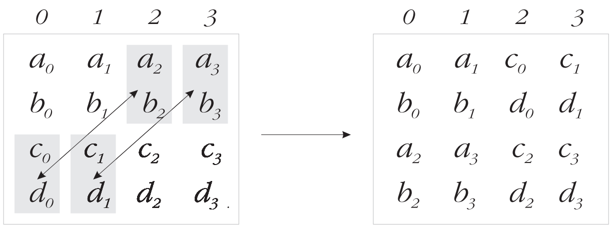


Abbildung 3.2: Schritt 1 des Transpositionsschemas

Schritt 2: Die Elemente der Spalten 0 bzw. 2, die auf ungeraden Stellen stehen, werden mit den gradstelligen Elementen der Spalten 1 bzw. 3 ausgetauscht (Abb. 3.3).

Allgemein gilt für die Transposition einer $2^n \times 2^n$ Matrix auf einem n– dimensionalen Würfel:

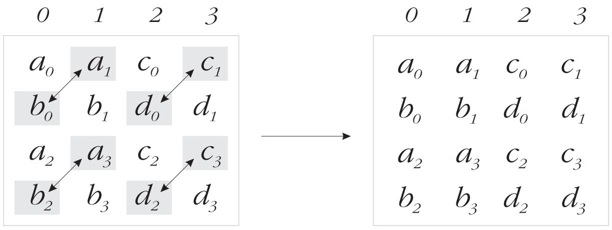


Abbildung 3.3: Schritt 2 des Transpositionsschemas

Der Austausch läuft in n Schritten ab. In jedem $k-$ ten Schritt findet der Elementaustausch zwischen zwei benachbarten Knoten entlang der $n-k$ – ten Dimensionsleitung (0. Spalte mit der 2. im ersten, 0. Spalte mit der 1. im zweiten Schritt des obigen Beispiels) statt. In jedem Schritt wird die Spalte der Matrix in 2^k gleichmäßige Teile der Größe 2^{n-k} verteilt. Der Knoten mit der niedrigeren Adresse tauscht die Daten der gradstelligen Blöcke mit den Daten der ungradstelligen Blöcken des Knotens mit der niedrigeren Adresse (die untere Hälfte der 0. Spalte mit der oberen Hälfte der 1. Spalte im Schritt 1 bzw. die ungradstelligen Zahlen der 0. Spalte mit den gradstelligen Zahlen der 1. Spalte im 2. Schritt des obigen Beispiels).

<u>*Beweis*</u> : Wie man leicht sehen kann, kann man jede $2^n \times 2^n$ Matrix $\overline{Q}$ folgendermaßen darstellen:

$$\overline{Q} = \left(\begin{array}{ccc|ccc} a_{0,0} & \cdots & a_{0,2^{n-1}-1} & a_{0,2^{n-1}} & \cdots & a_{0,2^n-1} \\ \vdots & & \vdots & \vdots & \vdots & \\ a_{2^{n-1}-1,0} & \cdots & a_{2^{n-1}-1,2^{n-1}-1} & a_{2^{n-1}-1,2^{n-1}} & \cdots & a_{2^{n-1}-1,2^n-1} \\ \hline a_{2^{n-1},0} & \cdots & a_{2^{n-1},2^{n-1}-1} & a_{2^{n-1},2^{n-1}} & \cdots & a_{2^{n-1},2^n-1} \\ \vdots & & \vdots & \vdots & \vdots & \\ a_{2^n-1,0} & \cdots & a_{2^n-1,2^{n-1}-1} & a_{2^n-1,2^{n-1}} & \cdots & a_{2^n-1,2^n-1} \end{array} \right) = \begin{pmatrix} \overline{Q}_1 & \overline{Q}_2 \\ \overline{Q}_3 & \overline{Q}_4 \end{pmatrix}$$

Hier sind $\overline{Q}_j \;\; (1 \leq j \leq 4)$ vier quadratische Teile der Matrix.

Dann ist $\overline{Q}^T$ (die Transposition der Matrix $\overline{Q}$):

$$\overline{Q}^T = \begin{pmatrix} \overline{Q}_1^T & \overline{Q}_3^T \\ \overline{Q}_2^T & \overline{Q}_4^T \end{pmatrix}$$

Diese Tatsache folgt unmittelbar aus der Betrachtung von $\overline{Q}$ auf der vorigen Seite.

Gelte nun die Behauptung für $\forall k < n$ (für $n = 2$ ist sie bereits bewiesen).

Sei $\overline{Q}$ eine $2^n \times 2^n$ Matrix. Im ersten Schritt des Transpositionsverfahrens werden die unteren Hälften der ersten 2^{n-1} Spalten mit den oberen Hälften der letzten 2^{n-1} Spalten ausgetauscht (Austausch entlang der $n-1$ – ten Dimension):

$$a_{i+2^{n-1},j} \longleftrightarrow a_{i,j+2^{n-1}} \; {\scriptstyle (i,j \in \{0,\dots,2^{n-1}-1\})}.$$

damit werden die zweite und dritte Teilkomponente der Matrix ihre Plätze tauschen:

$$\begin{pmatrix} \overline{Q}_1 & \overline{Q}_2 \\ \overline{Q}_3 & \overline{Q}_4 \end{pmatrix} \xrightarrow{\text{Schritt 1}} \begin{pmatrix} \overline{Q}_1 & \overline{Q}_3 \\ \overline{Q}_2 & \overline{Q}_4 \end{pmatrix}$$

Weiter wird jeder quadratische Teil der veränderten Matrix für sich transponiert (nach der Induktionsannahme und da die weitere Schrittfolge der Transposition einer $2^{n-1} \times 2^{n-1}$ Matrix entspricht), so daß letztendlich folgende Matrix entsteht:

$$\begin{pmatrix} \overline{Q}_1^T & \overline{Q}_3^T \\ \overline{Q}_2^T & \overline{Q}_4^T \end{pmatrix}$$

$\blacksquare$

Über die Implementierung der Rücktranspositionsfunktion $\mathcal{RT}'$ für eine $s \times r$ Matrix $(s = l \cdot r)$ mit

$$a_{i+k\cdot 2^{n+1},j} \xrightarrow{\mathcal{RT}'} a_{i+j\cdot 2^{n+1},k}$$

$(j,k\in\{0,...,2^{n-1}-1\}, i\in\{0,...,2^n-1\})$.

kann man folgendes aussagen:

Der Rücktranspositionsprozeß auf einem $n-$ dimensionalen Würfel läuft in n Teilschritten ab. In jedem Schritt k werden die Spalten der Matrix (bzw. der Knotenspeicher) in 2^k gleichmäßige Teile der Größe $\frac{2^{2n+1}}{2^k} = 2^{2n-k+1}$ verteilt. Jeder Austausch im $k-$ ten Schritt wird entlang der $n-k-$ ten Dimensionsleitung durchgeführt. Dabei werden die Daten in den Knoten mit niedrigeren Adressen, die in den gradstelligen Teilblöcken stehen, mit den Daten der Knoten mit den höheren Adressen, die sich in den ungradstelligen Blöcken befinden, ausgetauscht.

Man bemerke dabei, daß dieser Prozeß dem Transpositionsprozeß der quadratischen Matrix ähnlich ist, der Unterschied ist nur, daß der Prozeß nicht auf der quadratischen Teilmatrix, sondern auf der $r \times s$ Matrix angewandt wird. Dabei enthalten im letzten ($n-$ ten) Schritt der "Rücktransposition" die Blöcke nicht ein, sondern l Elemente.

Beweis der Aussage:

Sei

$$\overline{Q} = \begin{pmatrix} \overline{A}_{0,0} & \overline{A}_{0,1} & ... & \overline{A}_{0,2^n-1} \\ \overline{A}_{1,0} & \overline{A}_{1,1} & ... & \overline{A}_{1,2^n-1} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ \overline{A}_{2^n-1,0} & \overline{A}_{2^n-1,1} & ... & \overline{A}_{2^n-1,2^n-1} \end{pmatrix}$$

Die Elemente $\overline{A}_{i,j}$ $(i,j \in \{0,...,2^n-1\})$ der Matrix $\overline{Q}$ bestehen aus folgenden Blöcken der Matrix:

$$\overline{A}_{i,j} = \begin{pmatrix} a_{i\cdot 2^{n+1},j} \\ . \\ . \\ . \\ a_{2^{n+1}(i+1)-1,j} \end{pmatrix}$$

Wendet man $\mathcal{RT}'$ auf $\overline{Q}$ an, so kann man sehen:

$$\begin{pmatrix} \overline{A}_{0,0} & \overline{A}_{0,1} & \dots & \overline{A}_{0,2^n-1} \\ \overline{A}_{1,0} & \overline{A}_{1,1} & \dots & \overline{A}_{1,2^n-1} \\ . & & & \\ . & & & \\ . & & & \\ \overline{A}_{2^n-1,0} & \overline{A}_{2^n-1,1} & \dots & \overline{A}_{2^n-1,2^n-1} \end{pmatrix} \xrightarrow{\mathcal{RT}'} \begin{pmatrix} \overline{A}_{0,0} & \overline{A}_{1,0} & \dots & \overline{A}_{2^n-1,0} \\ \overline{A}_{0,1} & \overline{A}_{1,1} & \dots & \overline{A}_{2^n-1,0} \\ . & & & \\ . & & & \\ . & & & \\ \overline{A}_{0,2^n-1} & \overline{A}_{1,2^n-1} & \dots & \overline{A}_{2^n-1,2^n-1} \end{pmatrix}$$

Dies stimmt aber mit der Transposition einer quadratischen Matrix überein. Also kann man das oben angegebene Transpositionsschema an die gesamte Matrix $\overline{Q}$ anwenden, um den Rücktranspositionsprozeß zu realisieren.

■

3.2 Implementierung von $\mathcal{SH}'$ und $\mathcal{RSH}'$

Bei der Implementierung von $\mathcal{SH}'$ und $\mathcal{RSH}'$ treten dieselben Kommunikationsprobleme auf, die im vorigen Abschnitt erläutert wurden. Deswegen sind auch die Schritte *Shift* und *Rückshift* in n Teilschritten ausgeführt. Für die Implementierung von $\mathcal{SH}'$ gilt:

Der Shiftschritt wird in n Teilschritten ausgeführt. In jedem Teilschritt k werden die Elemente entlang der $k-1$ -ten Dimension ausgetauscht. Dabei tauschen nur die Knoten die Daten miteinander, für die gilt:

$$a_0 = \dots = a_{k-1} = 0$$

(Hier sind $a_0, \dots, a_{k-1}$ die entsprechenden Adressbits der Knoten.) Die Ausnahme ist der erste Schritt, wo alle Knoten die Daten mit entsprechenden Nachbarn tauschen.

Das obige Schema wird an einem Beispiel erläutert:

$$\begin{pmatrix} A_0 & A_1 & A_2 & A_3 \\ B_0 & B_1 & B_2 & B_3 \end{pmatrix} \xrightarrow{\text{S. 1}} \begin{pmatrix} A_0 & A_1 & A_2 & A_3 \\ B_1 & B_0 & B_3 & B_2 \end{pmatrix} \xrightarrow{\text{S. 2}} \begin{pmatrix} A_0 & A_1 & A_2 & A_3 \\ B_3 & B_0 & B_1 & B_2 \end{pmatrix}$$

Wie man an diesem Beispiel sieht, tauschen die Spalten 1 und 3 in zweitem Schritt keine Daten miteinander, da deren Adressen ungrade sind ($a_0 = 1$).

<u>*Beweis der Aussage:*</u> Gelte die Aussage für $\forall\, 2^q \times 2^p$ Matrix mit $p < n$, $i \in N$ (für die $q \times 4$ Matrix ist die Korrektheit bereits gezeigt).

Sei

$$\overline{Q} = \begin{pmatrix} A_0 & \ldots & A_{2^{n-1}-1} & A_{2^{n-1}} & \ldots & A_{2^n-1} \\ B_0 & \ldots & B_{2^{n-1}-1} & B_{2^{n-1}} & \ldots & B_{2^n-1} \end{pmatrix}$$

eine Matrix, auf die der Shiftalgorithmus angewandt werden soll. Wie man leicht sehen kann, werden nach der Anwendung der Schritte $1, \ldots, n-1$ des Shiftalgorithmus auf $\overline{Q}$ deren linke und rechte "Teilmatrixen" für sich geshiftet:

$$\overline{Q} \quad = \quad \begin{pmatrix} A_0 & \ldots & A_{2^{n-1}-1} & A_{2^{n-1}} & \ldots & A_{2^n-1} \\ B_0 & \ldots & B_{2^{n-1}-1} & B_{2^{n-1}} & \ldots & B_{2^n-1} \end{pmatrix} \xrightarrow{\text{S. 1,...,n-1}}$$

$$\begin{pmatrix} A_0 & \ldots & A_{2^{n-1}-1} & A_{2^{n-1}} & \ldots & A_{2^n-1} \\ B_{2^{n-1}-1} & \ldots & B_{2^{n-1}-2} & B_{2^n-1} & \ldots & B_{2^n-2} \end{pmatrix}$$

Im $n-$ ten Schritt des Verfahrens tauschen nur die Spalten mit den Adressen 0 und 2^{n-1} die Daten miteinander, da nur deren Adressen folgende binäre Darstellung haben:

$[\,0, \underbrace{0, \ldots\ldots\ldots\ldots, 0}_{n-1 \text{ untere Bits}}\,]\ ;\ [\,1, \underbrace{0, \ldots\ldots\ldots\ldots, 0}_{n-1 \text{ untere Bits}}\,]$

Deswegen gilt:

$$\begin{pmatrix} A_0 & \dots & A_{2^{n-1}-1} & A_{2^{n-1}} & \dots & A_{2^n-1} \\ B_{2^{n-1}-1} & \dots & B_{2^{n-1}-2} & B_{2^n-1} & \dots & B_{2^n-2} \end{pmatrix} \xrightarrow{\text{S. } n} \begin{pmatrix} A_0 & \dots & A_{2^{n-1}-1} & A_{2^{n-1}} & \dots & A_{2^n-1} \\ B_{2^n-1} & \dots & B_{2^{n-1}-2} & B_{2^{n-1}-1} & \dots & B_{2^n-2} \end{pmatrix}$$

■

Das Rückshiften verläuft analog, nur werden die oberen Hälften der Matrixspalten ausgetauscht (mit der Ausnahme der 0. Spalte, die immer noch die untere Hälfte austauscht) und in jedem j– ten Schritt tauschen nur die Knoten die Daten miteinander, für deren Adressen $[a_{n-1}, ..., a_0]$ gilt: $a_0 = ... = a_{j-1} = 1$.
Der Beweis der Korrektheit wird analog durchgeführt.

3.3 Ein– und Ausgabe

Bis zu diesem Zeitpunkt haben wir uns ausschließlich mit dem Sortieren der im Würfel gespeicherten Zahlen beschäftigt, mit der Annahme, daß die Daten schon im inneren Speicher der Knoten eingelesen sind.
Nun wird in diesem Abschnitt der Ein– und Ausgabeprozeß der Daten beschrieben.
Um die Anzahl der Ein– und Ausgabeleitungen in Grenzen zu halten, werden die Daten in den Knoten mit der Adresse 0 eingelesen und durch diesen an die restlichen Knoten weitergegeben. Bei der Ausgabe leiten alle Knoten ihre Daten (in der aufsteigender Adressreihenfolge) in den Knoten 0, der diese an die Außenwelt ausgibt (also kommuniziert nur der Knoten 0 mit der Aussenwelt).
Einfachheitshalber wird in diesem Abschnitt nur das Schema der Datenausgabe präsentiert, das für die Eingabe ist im Grunde genommen analog, nur werden die Daten vom Knoten 0 nicht ausgegeben, sondern eingelesen.
Grob kann man das Ausgabeschema für einen n– dimensionalen Würfel folgendermaßen darstellen:

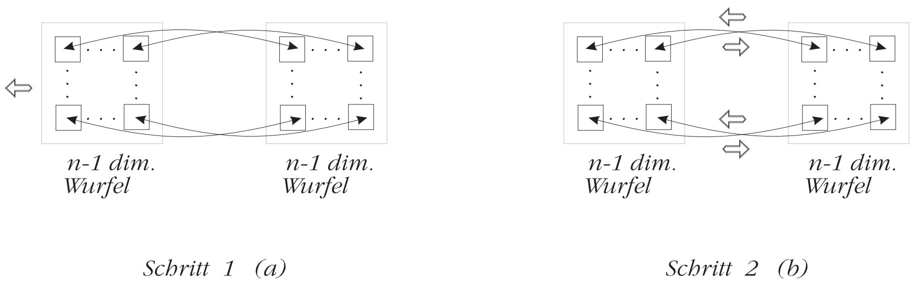


Abbildung 3.4: Das Datenausgabeschema für einen n –dim. Würfel

Schritt 1: Daten des unteren Teilwürfels ausgeben (Abb. 3.4 a);

Schritt 2: Daten zwischen den beiden $n-1$ dimensionalen Teilwürfeln entlang der $n-1$ –ten Dimension tauschen (Abb. 3.4 b);

Schritt 3: Daten des unteren Teilwürfels ausgeben (Abb 3.4 a).

Der oben definierte Datenausgabeprozeß wird auf einem Beispiel erläutert (siehe auch Abb. 3.5):

Beispiel: Datenausgabe aus einem dreidimensionalen Würfel.

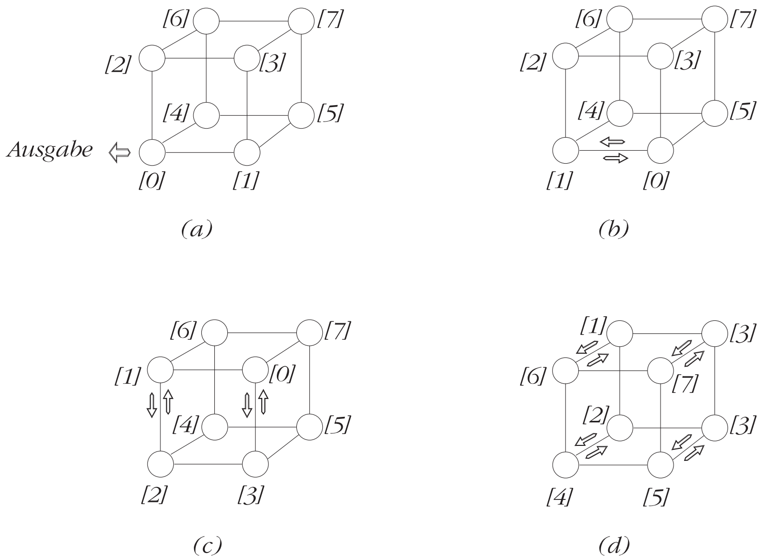


Abbildung 3.5: Die Datenausgabe

1. Daten aus dem Knoten 0 ausgeben (a);
2. Daten zwischen den Knoten 0 und 1 (entlang der 0. Dimension) austauschen (b);

   [Damit bekommt der Knoten 0 die Daten aus dem Knoten 1]
3. Daten aus dem Knoten 0 ausgeben (a);

   [Damit werden die Daten aus dem Knoten 1 ausgegeben]
4. Daten zwischen den Knoten 0 und 2 bzw. 1 und 3 (entlang der 1. Dimension) austauschen (c);

   [Damit bekommt der Knoten 0 die Daten aus dem Knoten 2]

5. Daten aus dem Knoten 0 ausgeben (a);

   [Damit werden die Daten aus dem Knoten 2 ausgegeben]

6. Daten zwischen den Knoten 0 und 1 (entlang der 0. Dimension) austauschen (b);

   [Damit bekommt der Knoten 0 die Daten aus dem Knoten 3]

7. Daten aus dem Knoten 0 ausgeben (a);

   [Damit werden die Daten aus dem Knoten 3 ausgegeben]

8. Daten zwischen allen möglichen Knoten entlang der 2. Dimension austauschen (d);

   [Damit bekommt jeder Knoten $j \in \{0, ..., 3\}$ die Daten aus dem Knoten $j + 4$]

9. Schritte 1 bis 7 wiederholen;

   [Damit werden die Daten der Knoten 4 bis 8 in aufsteigender Reihenfolge ausgegeben]

10. Ende.

Bemerkung: In jedem geraden Schritt $k = 2 \cdot p$ $(1 \leq p \leq 2)$ erfolgt der Austausch entlang der Dimension $p - 1$. Dabei tauschen nur die Knoten die Daten miteinander, für deren Adressen $(a_{n-1}, ..., a_0)$ gilt: $a_{n-1} = a_{n-2} = ... = a_p = 0$.

Damit tauschen nur die Knoten in unterem $p-1$ dim. Teilwürfel die Daten miteinander. Die Reihenfolge der Daten in oberem Teilwürfel bleibt unverändert.

Wie man leicht sehen kann, kann dieser Prozeß beliebig lang fortgesetzt werden, indem man das untere Schema anwendet (Abb. 3.6):

- Für einen eindimensionalen Würfel:

  1. Gebe den Inhalt des 0. Knotens aus;
  2. Tausche die Daten entlang der 0. Dimension;
  3. Gebe den Inhalt des 0. Knotens aus (Abb. 3.6 a).

- Sei nun der Ausgabeprozeß für einen $n - 1$ dimensionalen Würfel definiert. Für einen n dimensionalen Würfel gilt dann:

  1. Gebe den Inhalt des unteren $n - 1$ –dim. Teilwürfels aus;
  2. Tausche die Daten entlang der $n - 1$ –ten Dimension;
  3. Gebe den Inhalt des unteren $n - 1$ –dim. Teilwürfels aus (Abb. 3.6 b).

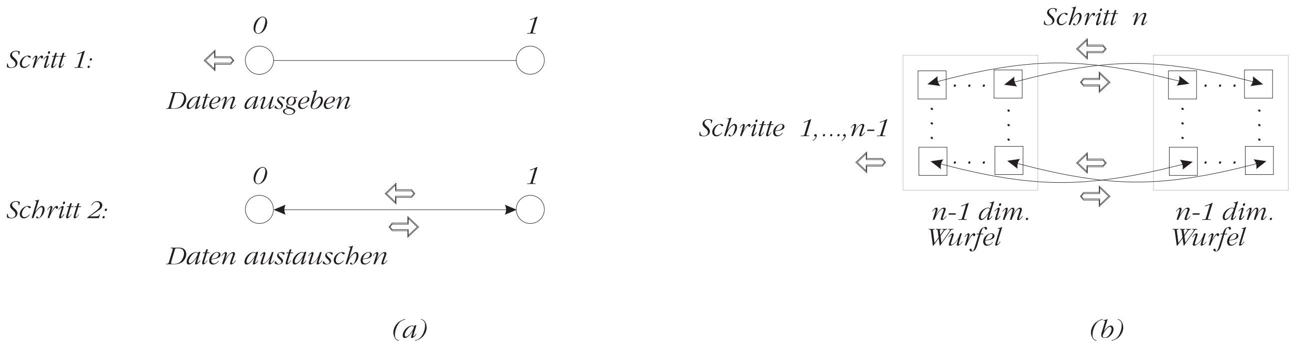


Abbildung 3.6: Das Schema der Datenausgabe

(Vergleiche auch das am Anfang dieses Kapitels vorgestellte Schema).
Auch hier kann man sehen, daß nur die Knoten die Daten im k –ten Schritt des Verfahrens miteinander tauschen, für deren Adressen $(a_{n-1}, ..., a_0)$ gilt: $a_{k-1} = 0$. Die restlichen Knoten bleiben blockiert, was die korrekte Reihenfolge sortierter Daten nicht ändert.

Was die Eingabe der Daten betrifft, so wird sie nach demselben Schema realisiert, der Unterschied ist nur, daß anstelle der Ausgabe aus dem 0. Knoten in jedem ungeraden Schritt die Eingabe erfolgen muß.

3.4 Binäre Darstellung der Adressen

Wie in diesem Kapitel schon gezeigt wurde, werden die Schritte "Transponieren", "Rücktransponieren", "Shiften" und "Rückshiften" in n Teilschritten realisiert, wobei in jedem Teilschritt gewisse "Datenblöcke" zwischen den Knoten ausgetauscht werden. Dabei enthalten diese "Blöcke" insgesamt 2^n Elemente. Betrachten wir nun ein Transpositionsbeispiel einer 4×4 Matrix aus dem Abschnitt 3.1 (Abb. 3.7).
Sei $T'^{j}_{a} : \{1, ..., 2^{n-1}\} \longrightarrow \{0; 1\}^n$ eine Funktion, die binäre Adressen der zu austauschenden Daten in jedem Unterschritt j berechnet (hier ist a die Knotenadresse). In unserem Beispiel gilt:

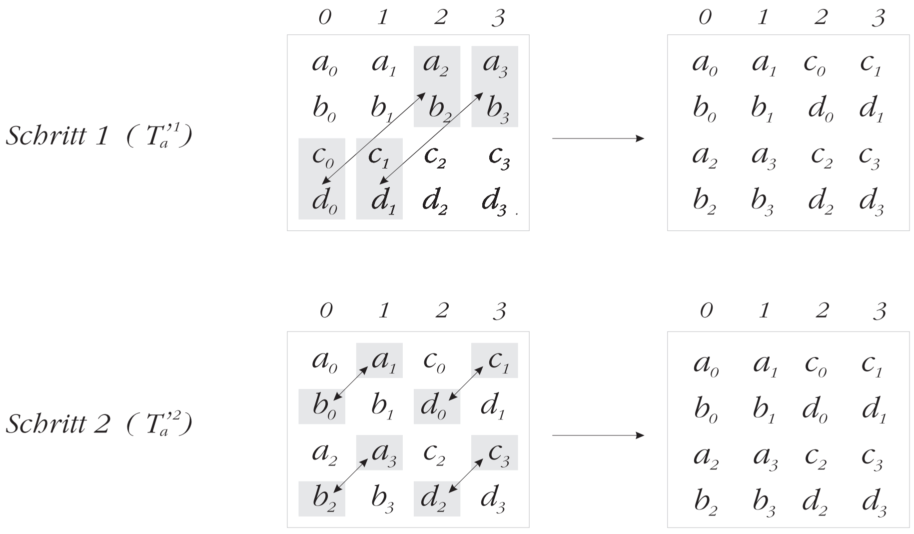


Abbildung 3.7: Transpositionsschritte und T'^{j}_{a}

Schritt j	Adresse a	Unterschritt k	$T'^{j}_{a}(k)$
1	0	1	10
1	0	2	11
1	2	1	00
1	2	2	01
2	0	1	01
2	0	2	11
2	1	1	00
2	1	2	10

Für die in der Tabelle nicht angegebenen Werte von T'^{j}_{a} gilt:

$$T'^{1}_{1}(k) = T'^{1}_{0}(k); \quad T'^{1}_{3}(k) = T'^{1}_{2}(k)$$
$$T'^{2}_{2}(k) = T'^{2}_{0}(k); \quad T'^{2}_{3}(k) = T'^{2}_{1}(k)$$

Seien ferner $T^{j}_{a}, RT^{j}_{a}, SH^{j}_{a}, RSH^{j}_{a} : \{0; ...; 2^{2n} - 1\} \longrightarrow \{0; 1\}^{2n+1}$ Funktionen, die die binären Adressen der im j –ten Schritt auszutauschenden Daten der Knoten mit den Adressen $a \in \{0; ...; 2^{n}-1\}$ berechnen (das Funktionsprinzip sei zu dem von T'^{j}_{a} analog).

Bezeichne nun $Bild(f)$ die Bildmenge der Funktion f.

Dann gilt:

$$Bild(T'^{j}_{a}) = \left\{ p \cdot 2^{n-j} + 2^{n-j-1} \cdot \overline{a}_j + p' \mid \; p \in \{0, ..., 2^{j} - 1\}, \; p' \in \{0, ..., 2^{n-j-1} - 1\} \right\}$$

Da für die "Transposition" der Matrix jede quadratische Teilmatrix einzeln transponiert sein soll, gilt:

$$\begin{aligned} Bild(T_a^j) &= \{2^n \cdot p'' + p \cdot 2^{n-j} + 2^{n-j-1} \cdot \overline{a_j} + p' \\ &\quad \mid p, p' \;\; wie\ oben,\ p'' \in \{0, ..., 2^{n+1} - 1\}\} \end{aligned}$$

Da $\mathcal{RT}'$ die Übertragung von $\mathcal{T}'$ auf die gesamte Matrix ist, kann man $Bild(RT_a^j)$ nach dem selben Schema wie $Bild({T'}_a^j)$ berechnen:

$$\begin{aligned} Bild(RT_a^j) &= \{p \cdot 2^{2n-j+1} + 2^{2n-j} \cdot \overline{a_j} + p' \mid \; p \in \{0, ..., 2^j - 1\}, \\ &\quad p' \in \{0, ..., 2^{2n-j} - 1\}\} \\ &= \{p \cdot 2^{2n-j+1} + 2^{2n-j} \cdot \overline{a_j} + p' \cdot 2^{n+1} + p'' \mid \\ &\quad p\ \ wie\ oben,\ p' \in \{0, ..., 2^{n-j-1} - 1\},\ p'' \in \{0, ..., 2^{n+1} - 1\}\} \end{aligned}$$

Für die Funktionen SH_a^j und RSH_a^j gilt:

$$Bild(SH_a^j) = \{p \mid p \in \{2^{2n}, ..., 2^{2n+1} - 1\}\},$$

$$Bild(RSH_a^j) = \begin{cases} \{p' \mid p' \in \{2^{2n}, ..., 2^{2n+1} - 1\} & , \text{ falls } a = 0 \\ \{p' \mid p' \in \{0, ..., 2^{2n} - 1\}\} & , \text{ falls } a > 0 \end{cases}$$

(siehe auch Abschnitt 3.2)

Die oben definierten Funktionen kann man auch folgendermaßen darstellen:

$$
\begin{aligned}
{T'}^0_a &: [\,\overline{a_0},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\,],\\
{T'}^j_a &: [\,\overbrace{p_{j-1},\dots,p_0}^{p},\ \overline{a_j},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\,],\\
T^0_a &: [\,\overbrace{{p''}_n,\dots,{p''}_0}^{p''},\ \overline{a_0},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\,],\\
T^j_a &: [\,\overbrace{{p''}_n,\dots,{p''}_0}^{p''},\ \overbrace{p_{j-1},\dots,p_0}^{p},\ \overline{a_j},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\,],\\
F^0_a &: [\,\overline{a_0},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\ \overbrace{{p''}_n,\dots,{p''}_0}^{p''}\,],\\
F^j_a &: [\,\overbrace{p_{j-1},\dots,p_0}^{p},\ \overline{a_j},\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\ \overbrace{{p''}_n,\dots,{p''}_0}^{p''},\ ],\\
P^j_a &: [\,0,\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\ \overbrace{{p''}_n,\dots,{p''}_0}^{p''}\,],\\
{P'}^j_a &: [\,0,\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\ \overbrace{{p''}_n,\dots,{p''}_0}^{p''}\,]\text{ , falls } a=0,\\
{P'}^j_a &: [\,1,\ \overbrace{{p'}_{n-j-2},\dots,{p'}_0}^{p'}\ \overbrace{{p''}_n,\dots,{p''}_0}^{p''}\,]\text{ , falls } a>0
\end{aligned}
$$

Seien nun

$$
\begin{aligned}
k &= [\,p''_n,\dots,p''_0,p_{j-1},\dots,p_0,p'_{n-j-2},\dots,p'_0\,]\\
&= [\,k_{2n-1},\dots,k_{n-1},k_{n-2},\dots,k_{n-j-1},k_{n-j-2},\dots,k_0\,]
\end{aligned}
$$

und

$$k' = [\,k_{n-2},\dots,k_{n-j-1},k_{n-j-2},\dots,k_0\,].$$

Dann gilt:

$$
\begin{aligned}
T^j_a(k) &= [\,k_{2n-1},\dots,k_{n-1},k_{n-2},\dots,k_{n-j-1},\overline{a_j},k_{n-j-2},\dots,k_0\,]\\
&= [\,k_{n-2},\dots,k_{n-j-1},{T'}^j_a(k')\,],\\
RT^j_a(k) &= [\,k_{n-2},\dots,k_{n-j-1},\overline{a_j},k_{n-j-2},\dots,k_0,k_{2n-1},\dots,k_{n-1}\,]\\
&= [\,{T'}^j_a(k'),k_{2n-1},\dots,k_{n-1}\,],\\
SH^j_a(k) &= [\,1,k_{n-2},\dots,k_0,k_{2n-1},\dots,k_{n-1}\,]\\
&= [\,{T'}^0_0(k'),k_{2n-1},\dots,k_{n-1}\,],\\
RSH^j_0(k) &= [\,1,k_{n-2},\dots,k_0,k_{2n-1},\dots,k_{n-1}\,]\\
&= [\,{T'}^0_0j(k'),k_{2n-1},\dots,k_{n-1}\,],\\
RSH^j_a(k) &= [\,0,k_{n-2},\dots,k_0,k_{2n-1},\dots,k_{n-1}\,]\\
&= [\,{T'}^0_{2^n-1}(k'),k_{2n-1},\dots,k_{n-1}\,]\ \ (a>0).
\end{aligned}
$$

Wir definieren nun die sog. "Erweiterte Adresse" EA, die in weiteren Kapiteln ziemlich oft benutzt wird:

Definition 3.4.1: Die "Erweiterte Adresse" EA wird in den Schritten "Transponieren", "Rücktransponieren", "Shiften" und "Rückshiften" folgendermaßen berechnet:
EA stimmt in jedem Schritt mit der entsprechenden Adressberechnungsfunktion (T_a^j, RT_a^j, SH_a^j oder RSH_a^j) bis auf das Bit $\overline{a_j}$ überein. Anstelle von $\overline{a_j}$ wird 0 eingesetzt.

Z.B. ist EA während der "Transposition" der Matrix:

$$[\ k_{2n-1}, ..., k_{n-1}, k_{n-2}, ..., k_{n-j-1}, \underset{\overline{a_j}}{\underset{\updownarrow}{0}}, k_{n-j-2}, ..., k_0\ ]$$

(Hier wird $\overline{a_j}$ mit 0 ausgetauscht, ansonsten stimmt alles mit T_a^j überein.)

Darstellung der Adressen bei der Daten Ein– und Ausgabe

Wie man in forigem Abschnitt sehen konnte, werden die Daten während des Shift– Vorganges in die obere , während *Rückshiftens*– in die untere Hälfte des inneren Speichers eingeschrieben. Um alle Daten zweier Knoten zu tauschen konnte man also erst die obere, danach die untere Hälfte austauschen, d.h., man könnte erst das *shift*–, nachher das *Rückshift*– Schema anwenden.
Es gilt also:

$$\begin{array}{l} I_a^j(k) = O_a^j(k) = SH_a^j(k) \\ I_a^j(k+2^{n-1}) = O_a^j(k+2^{n-1}) = RSH_a^j(k) \end{array}$$

Dabei muß man beachten, daß bei der allerersten Datenausgabe aus dem 0. Knoten (im Gegenteil zu den nächsten Ausgabeschritten) das Schema

$$\begin{array}{l} O_a^j(k) = RSH_a^j(k) \\ O_a^j(k+2^{n-1}) = SH_a^j(k) \end{array}$$

angewandt sein soll (siehe Abschnitt 3.2).

Kapitel 4

Die Schaltung

4.1 Beschreibung der Zustandsvariablen

Auf der obersten Hierarchieebene sieht die Schaltung folgendermaßen aus:

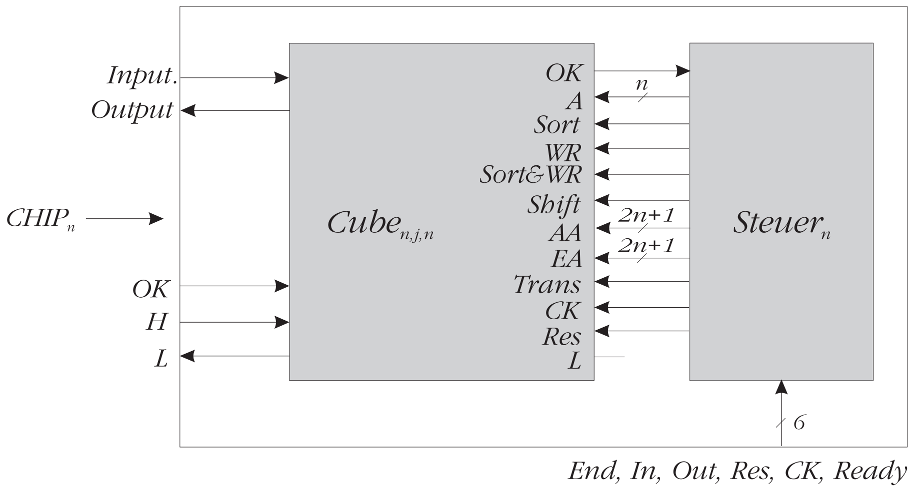


Abbildung 4.1: Die Sortierschaltung auf der obersten Hierarchieebene

Die Schaltung $Cube_{n,j,m}$ ($j = \lceil \frac{n}{2} \rceil - \lfloor \frac{n}{2} \rfloor$) ist der im Kap.1 beschriebene Würfel, der als Anschlüsse auf der Ostseite die Steuerleitungen hat. Die *In/Out* Leitungen dienen für die Ein– bzw. Ausgabe der zu sortierenden (bzw. sortierten) Zahlen.
Die Schaltung $Cube_{n,j,n}$ kann man als einen endlichen Automaten betrachten, dessen Zustände durch die Steuerleitungen (Zustandvariablen) bestimmt sind. Diese Variablen werden durch die Steuereinheit ($Steuer_n$) berechnet und an den n –dimensionalen Würfel weitergegeben.
Die Schaltung funktioniert folgendermaßen:
Als erstes wird In auf 'H' gesetzt und die Schaltung in den Eingabezustand versetzt.

Dabei werden die Daten von dem Knoten 0 durch die Eingabeleitung *Input* eingelesen (siehe Kap. 3). Wannimmer die Schaltung bereit ist die Daten aufzunehmen, wird die Leitung *Ready* auf 'H' gesetzt. Dabei übergeht die Schaltung in den Wartezustand, bis das Signal *OK* von außen kommt und damit die Daten von *In* eingelesen werden.
Nachdem alle Daten eingespeichert sind, werden sie sortiert. Der Sortiervorgang wird weiter in diesem Kapitel genauer beschrieben.
Nach dem Sortierprozeß folgt der Datenausgabeprozeß, der im wesentlichen dem Dateneingabeprozeß ähnlich ist, nur werden die Daten hier durch die Leitungen *Out* ausgegeben.
Auch hier gilt der *Ready/OK*– Vorgang, nur werdend bei der Datenaushabe die Signale *In* und *Out* beide auf 'H' gesetzt.
Nachdem alle Daten ausgegeben wurden, beendet die Schaltung ihre Arbeit, indem sie das Signal *End* auf 'H' setzt und blockiert.

Da die Funktionsweise bei der Datenein– und Ausgabe auf die vom Sortieren basiert (siehe Kap. 3), wird zunächst nur die Implementierung des Sortieralgorithmus beschrieben, als wäre keine Ein– oder Ausgabe der Daten nötig.

Betrachten wir nun die Arbeitsweise eines Knotens mit der Adresse $(a_{n-1}, ..., a_0)$.
Die acht Schritte des Sortierverfahrens, sowie die Ein– und Ausgabe (siehe Kap. 2) werden durch folgende Variablenkombinationen bestimmt:

	In	*Out*	*Sort*	*Trans*	*Shift*
Dateneingabe	'H'	'L'	'L'	$*$	$*$
Datenausgabe	'H'	'H'	'L'	$*$	$*$
Sortieren (4 mal)	'L'	$*$	'H'	$*$	$*$
Transponieren	'L'	$*$	'L'	'H'	'L'
Rücktransponieren	'L'	$*$	'L'	'L'	'L'
Shiften	'L'	$*$	'L'	'H'	'H'
Rückshiften	'L'	$*$	'L'	'L'	'H'

Wie in Abschnitt 3.4 bereits erwähnt wurde, wird anhand dieser Variablenbelegungen und mittels der Funktionen $\mathcal{T}', \mathcal{RT}', \mathcal{SH}'$ oder $\mathcal{RSH}'$ in jedem k –ten Schritt die sog. "erweiterte Adresse" $EA = (EA_{2n}, ..., EA_0)$ berechnet, wo für $\overline{a}_j$ eine Stelle EA_p "reserviert" ist. Diese Stelle wird durch die Variable $AA = (AA_{2n}, ..., AA_0)$ bestimt, wobei gilt:

$$AA_j = \begin{cases} 1, & \text{falls } j = p, \\ 0 & \text{sonst} \end{cases}$$

Ferner bestimmt die Variable A, entlang welcher Dimension der Datenaustausch erfolgen soll.

Es gilt also für $A = (A_{n-1}, ..., A_0)$: $A_j = 1 \longleftrightarrow$ der Austausch soll entlang der j –ten Dimension erfolgen.
Für A gilt folgende Belegung (siehe auch Abb. 4.2):

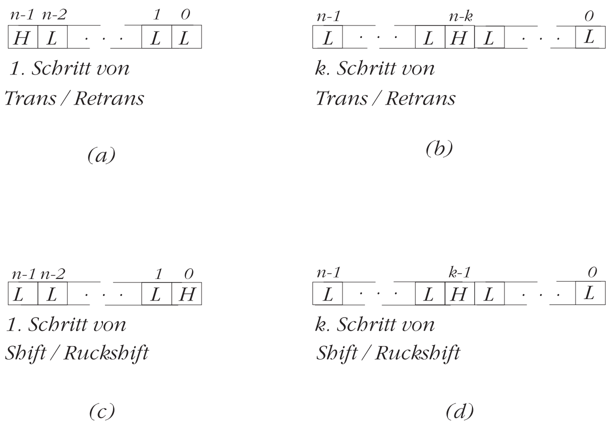


Abbildung 4.2: Die Belegung von A

Während "Transponieren" und "Rücktransponieren":

$$A_j = \begin{cases} 1, & \text{falls } j = n - k \\ 0 & \text{sonst} \end{cases}$$

(Absteigende Reihenfolge)

Während "Shiften" und "Rückshiften":

$$A_j = \begin{cases} 1, & \text{falls } j = k - 1 \\ 0 & \text{sonst} \end{cases}$$

(Aufsteigende Reihenfolge)

4.2 Die Knotenkonstruktion

4.2.1 Allgemeine Beschreibung

In Abb. 4.3 ist der Knoten auf der obersten Hierarchieebene gezeigt.

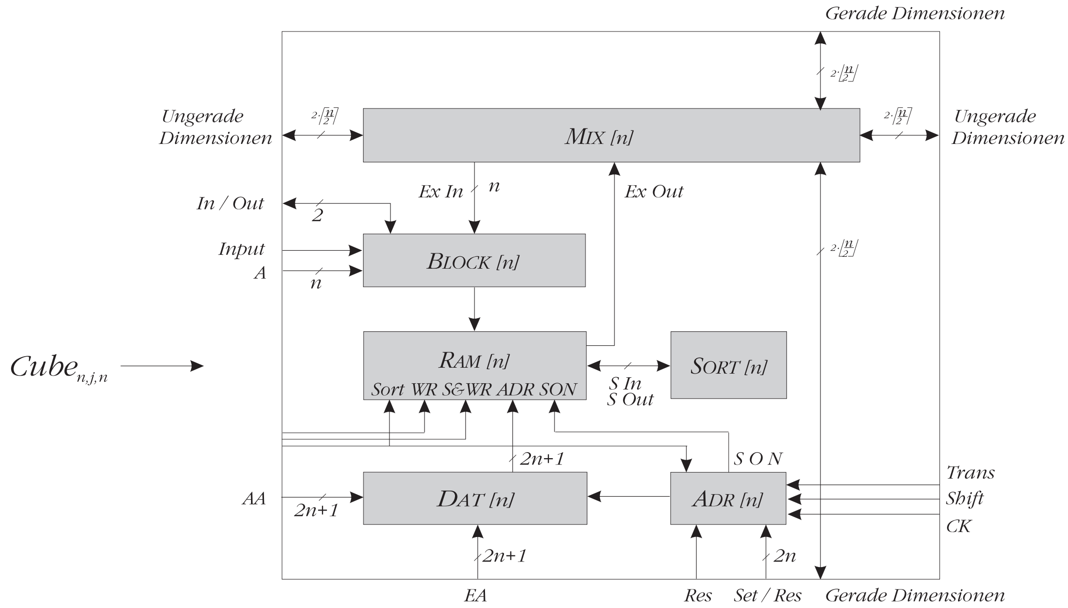


Abbildung 4.3: Schaltung $Cube_{n,0,0}$

Die Schaltung *MIX[n]* führt die geraden Dimensionsleitungen zusammen mit *ExOut* auf die O/W, die ungeraden (auch mit *ExOut* zusammen) dagegen — auf N/S. Dabei gilt: Es werden je $\lceil \frac{n}{2} \rceil$ bzw. $\lfloor \frac{n}{2} \rfloor$ Leitungen auf die O/W bzw. N/S Seiten von *MIX[n]* ausgeführt.
Das Schema dieser Schaltung ist in Abb. 4.4 gezeigt. Dort kann man sehen, daß jedes Leitungsbündel D_j aus einem Paar *Dimensionsleitung/ExOut* (d_j/*ExOut*) besteht. Dabei muß man bemerken, daß *ExOut* auf der Ostseite über, auf der Westseite dagegen — unter der entsprechenden Dimensionsleitung d_j ausgeführt wird. Dadurch wird erreicht, daß nach der Verknüpfung dieser Bausteine auf den O/W Seiten *ExOut* immer mit dem entsprechenden d_j verknüpft wird (es wird also je ein Ausgang mit einem Eingang verknüpft).

Grob kann man die Arbeitsweise des Verfahrens in zwei Schritte unterteilen: *Daten zwischen den Knoten austauschen* (kurz — *Datenaustausch*) und *Daten innerhalb des Knotens sortieren* (kurz — *sortieren*).
Diese Einteilung ist möglich, weil der einzige Unterschied zwischen den einzelnen Daten-

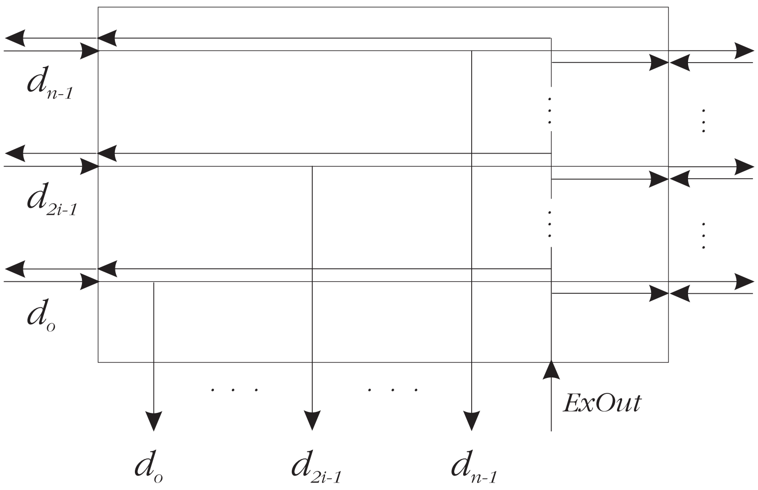


Abbildung 4.4: Das Konstruktionsschema der *MIX[n]* Schaltung

austauschschritten in der Adressberechnung besteht, die ausserhalb der Knoten durchgeführt wird.
Der Speicher *RAM[n]* besitzt zwei Arten von Ein– bzw. Ausgängen: *ExIn/ExOut* (externe Inp./Outp.) und *SIn/SOut* (Inp./Outp. für "Sortieren"). Während des Datenaustausches werden die Leitungen *ExIn/ExOut* aktiviert (d.h., die werden freigegeben), *SIn/SOut* dagegen — blockiert (im hochohmigen Zustand).

Während des Datenaustausches "aktiviert" die Adresse je eine Speicherzelle, d.h., es wird nur eine Zelle ausgewählt, deren Daten an die Leitung *ExOut* angelegt werden; außerdem werden die Daten vom *ExIn* auf den Eingang dieser Zelle angelegt.

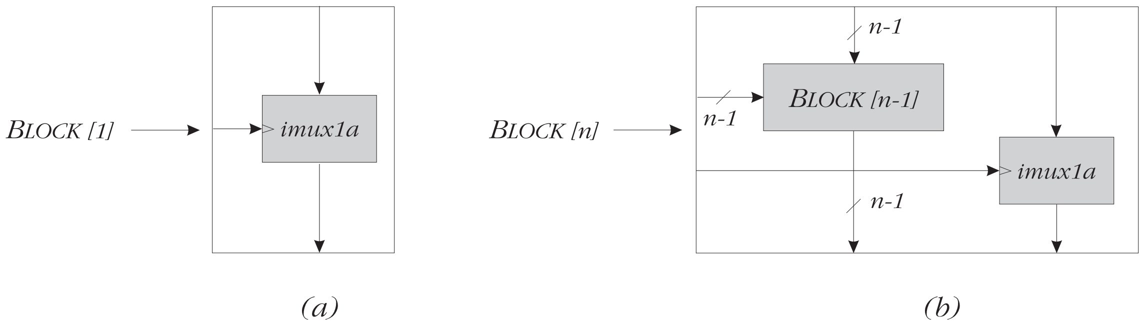


Abbildung 4.5: Realisierung der *BLOCK[n]* Schaltung

Durch die Leitung *ExOut* werden die Daten an jeden Nachbarknoten weitergeleitet, also bekommt jeder Knoten je eine Zahl von jedem Nachbarn. Da der Austausch aber nur

mit einem im gegebenen Schritt durch die Variable A bestimmten Nachbarn erfolgen soll, werden die Dimensionsleitungen durch den Baustein *Block[n]* (Abb. 4.5) "selektiert", d.h., es werden nur die Daten von der durch A bestimmten Dimensionsleitung an *ExIn* weitergeleitet, die restlichen werden blockiert.
Ein Beispiel dieses Prozesses (Austausch im 3. Schritt) ist in Abb. 4.6 gezeigt.

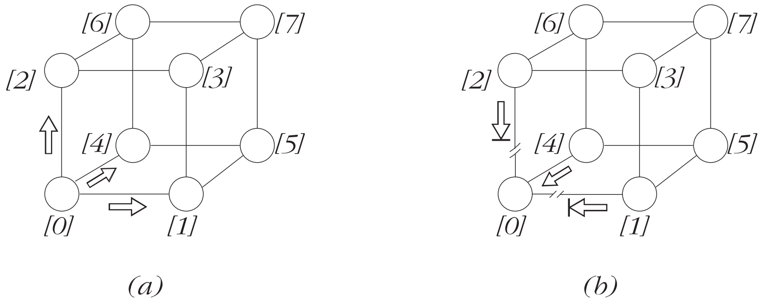


Abbildung 4.6: Datenfluß vom *(a)* und zum *(b)* Knoten 0 im 3. Schritt

Anhand dieses Beispiels kann man sehen, wie die Daten vom Knoten 0 zu jedem Nachbarn weitergegeben werden und wie nur die Daten von dem Nachbarn der 2. Dimension durchgeleitet sind (Austausch entlang der 2. Dimension).

Nachdem die Daten an den Eingang der aktivierten Speicherzelle angelegt werden, werden die mit der L/H Flanke von *WR* gespeichert.

Während der ungeraden Schritte des Algorihmus (Daten sortieren) werden die Daten durch die Leitungen *Sout* parallel an die innere Sortierschaltung weitergeleitet, innerhalb jedes Knotens sortiert, durch die Leitungen *SIn* an die Eingänge der Zellen angelegt und mit der L/H Flanke von *Sort&WR* abgespeichert. Damit werden die Daten im Speicher in aufsteigender Reihenfolge vorliegen (die genaue Beschreibung dieser Prozesse folgt in weiteren Abschnitten dieses Kapitels).

4.2.2 Speicher, Datenfluß und Adressberechnung

Wie man dem Kap. 3 entnehmen kann, werden die Daten zwischen den Nachbarknoten sequentiell ausgetauscht. Im Gegenteil zu diesem Vorgang werden die Daten während des Sortiervorganges parallel an den inneren Sortierer weitergegeben bzw. von ihm empfangen. Dies ermöglicht es, die sortierten Zahlen in einem Schritt zu speichern.
In Abb. 4.7 wird der Datenfluß während des Datenaustauschprozesses gezeigt. Die Daten aus der Zelle mit der Adresse *[i]* (die an die Adressleitungen angelegt wird) werden

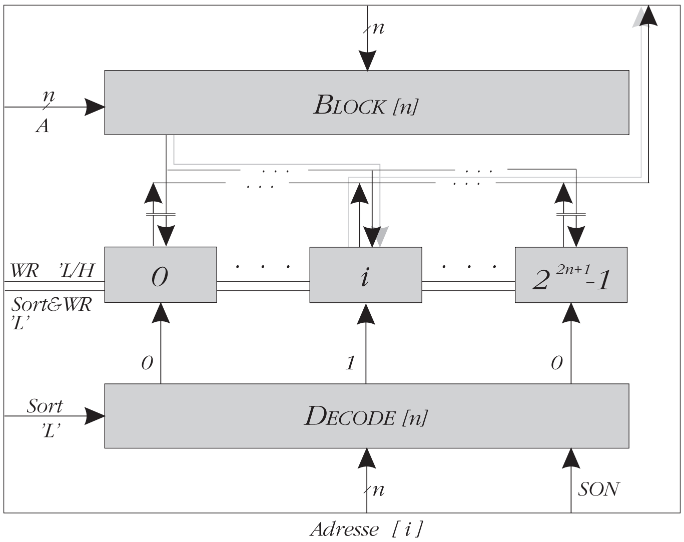


Abbildung 4.7: Der Speicherdatenfluß während des Datenaustausches

durch *ExOut* an jeden Nachbarknoten weitergeleitet. Andererseits bekommt jeder Knoten je ein Element von dem Nachbarn. Durch die Schaltung *Block[n]* und die Variable A wird, wie es bereits gesagt wurde, nur die Element vom aktuellen Nachbarknoten weitergeleitet, die Daten von den restlichen Nachbarn werden blockiert.
Durch $BLOCK[n]$ werden also nur die Daten vom aktuellen Nachbarn an die Eingänge aller Speicherzellen durchgeleitet. Jedoch werden die durch die L/H Flanke von WR nur in die Zelle mit der Adresse $[i]$ gespeichert (in Abb. 4.7 ist der Datenfluß mit punktierten Pfeilen gezeigt).
Dabei bleiben *SIn* und *SOut* blockiert. Letztendlich werden die aktuellen Daten durch die L/H Flanke von *WR* in die *[i]* –te Speicherzelle eingelesen (während dieses Vorganges muß *Sort&WR* auf $'L'$ bleiben).
Etwas anders sieht es während des *"Sortierens"* aus (Abb.4.8). Der erste Unterschied zum "Datenaustausch" ist, daß *SIn/SOut* aktiviert, *ExIn/ExOut* dagegen — blockiert werden.
Außerdem ist der Ausgang jeder Speicherzelle mit dem Eingang der entsprechenden Zelle von *Sort[n]* verbunden und umgekehrt: der Eingang jeder Speicherzelle ist mit dem Ausgang der entsprechender Sortzelle verknüpft. Dadurch wird erreicht, daß nach der L/H Flanke von *Sort&WR* (*WR='L', Adresse* — beliebig) die Daten im Speicher in aufsteigender Reihenfolge vorliegen, d.h., $R[i] \leq R[i+1]$ $(0 \leq i \leq 2^{2n+1} - 2)$, wo $R[i]$ der Inhalt der i –ten Speicherzelle ist.

Bemerkung: In beiden Fällen, falls der Eingang *SON*='L' ist, wird jeder Ausgang von *DECODE[n]* auf 'L' gesetzt und dadurch nichts gespeichert.

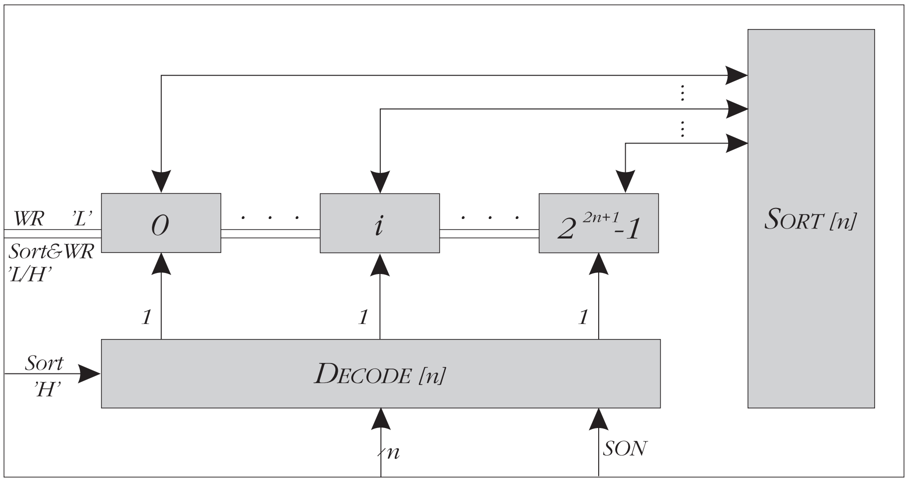


Abbildung 4.8: Datenfluß während "Sortieren"

Die Schaltung $ADR[n]$ besteht aus dem n –Bit langen Adressregister, was die Knotenadresse $a = (a_{n-1}, ..., a_0)$ beinhaltet. Im k –ten Schritt von $Transponieren$ bzw. *Rücktransponieren* wird auf der Westseite der Schaltung $\overline{a}_{k-1}$ ausgegeben. Während *Shift* bzw. *Rückshiftens* dagegen — 0 bzw. 1 (falls die Adresse gleich Null ist, bleibt der Ausgang auch beim *Rückshiften* auf 0).
Außerdem setzt $ADR[n]$ den SON– Ausganag auf'L', falls der Knoten während des gegebenen Shift– bzw. Rückshiftschrittes keine Daten austauschen soll (siehe Kap. 2). Dies wird durch die Variablen $Trans$, $Shift$ und a_{l-2} $(2 \leq l \leq n)$ bestimmt. Mit der L/H Flanke von CK werden die Ausgaben von $ADR[n]$ entsprechend geändert, falls $Sort$ ='L' ist. Ansonsten wird kein Zustansdwechsel stattfinden (während des Sortiervorganges werden die Registern in ihren Anfangzustang versetzt).

Wie oben erwähnt, muß die Adresse der aktuellen Zelle durch die Variablen $\overline{a}_j$, AA und EA ausgerechnet werden (hier stimmt j mit der Dimension überein, entlang welcher der Datenaustausch ablaufen soll). Das verwirklicht die Schaltung *DAT[n]*.
Schematisch kann man den Prozeß der Adresszusammenstellung wie in Abb. 4.9 darstellen.

4.3 Die Steuereinheit

Wie bereits am Anfang dieses Kapitels erwähnt wurde, gibt die Steuereinheit die Zustands– und Speichersignale, sowie die aktuellen Adressen an den n– dimensionalen

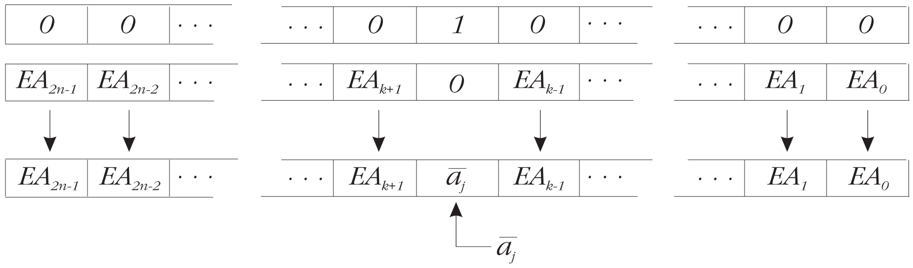


Abbildung 4.9: Das Schema der Adreßberechnung

Würfel weiter. Für den korrekten Ablauf des Verfahrens ist die richtige Reihenfolge der Steuersignale *Sort*, *Trans* und *Shift* entscheidend. Folgende Tabelle zeigt die korrekte Belegung dieser Signale:

	$Sort$	$\neg Trans$	$Shift$
Schritt 1:	H	(L)	(L)
Schritt 2:	L	L	L
Schritt 3:	H	(H)	(L)
Schritt 4:	L	H	L
Schritt 5:	H	(L)	(H)
Schritt 6:	L	L	H
Schritt 7:	H	(H)	(H)
Schritt 8:	L	H	H

Da in den ungeraden Schritten des Verfahrens (Sortieren) die Belegung von $Trans$ und *Sort* beliebig sein kann (siehe auch Abschnitt 4.1), werden diesen Variablen die in Klammern angezeigten Werte zugewiesen (hier in der Tabelle ist statt $Trans$ $\neg Trans$ dargestellt). Wie man leicht sehen kann, stimmen diese Werte mit denen eines 3– Bit Zählers überein. Also kann man $Sort$, $\neg Trans$ und $Shift$ als Ausgänge eines Zählers darstellen.
Der Zähler wird immer dann getaktet, wenn alle Daten von den Nachbarknoten (insgesamt $n \cdot 2^{2n}$) bzw. von dem Ausgang des OEM Sortierers gespeichert sind.

Die Variable A, die bestimmt, mit welchem Nachbarknoten der Datenaustausch erfolgen soll, ist durch ein n– Bit Shiftregister realisiert (siehe Abb. 4.10).

Nachdem der Austausch mit dem aktuellen Nachbarn durchgeführt ist (2^{2n} Zahlen ausgatauscht sind) wird bei $Trans/Retrans$ der Inhalt des Registers zyklisch um eine Position nach rechts geshiftet (absteigende Dimensionsfolge, vergleiche Abschnitt 4.1). Nach jedem durchgeführten Shift– bzw Rückshiftvorgang wird der Inhalt zyklisch um

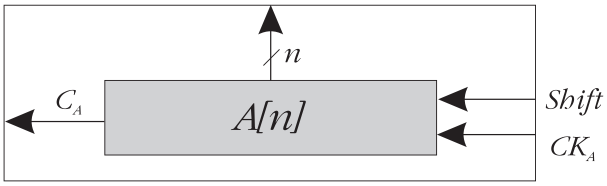


Abbildung 4.10: Realisierung von A

eine Position nach links geshiftet (aufsteigende Dimensionsreihenfolge).

An dieser Stelle wollen wir erneut die Belegung des Registers A zeigen (Abb. 4.11).

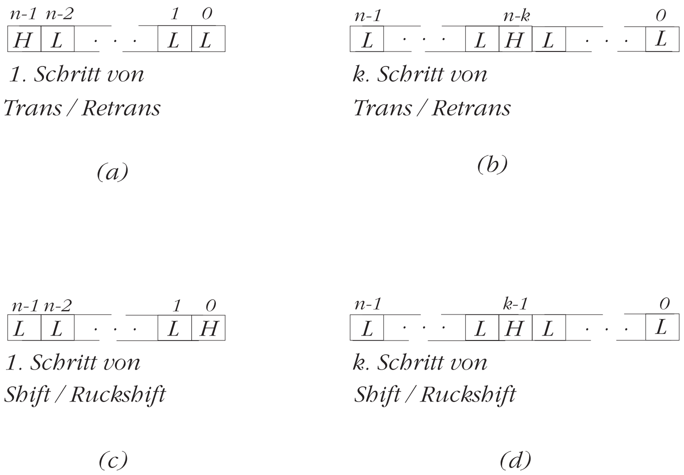


Abbildung 4.11: Die Belegung von A

Dieses Register wird mittels CK_A immer dann getaktet, wenn 2^{2n} Elemente ausgetauscht sind. Es ist so konstruiert, daß es mit den CK_A und Shiftsignalen korrekt geshiftet wird.
Das Signal C_A wechselt seinen Zustand von 'L' auf 'H' immer, wenn der Datenaustauschprozeß fertig ist, d.h., wenn $Shift\&A_{n-1} \vee \overline{Shift}\&A_0$ ='H' und das letzte Element gespeichert wurde. Wichtig ist dabei der $2n-$ Bit Zähler $Z[n]$, der für den Austausch der Daten innerhalb des gegebenen Datenaustauschschrittes zuständig ist (Abb. 4.12).

C_Z ist hier ein Parity– Bit von dem Zähler, der seinen Zustand von 'L' auf 'H' genau dann wechselt, wenn $Z[n]$ überfüllt ist, d.h., wenn alle 2^{2n} Elemente ausgetauscht wurden. Deswegen konnte man einen Teil der Steuereinheit wie in Abb. 4.13 konstruieren.

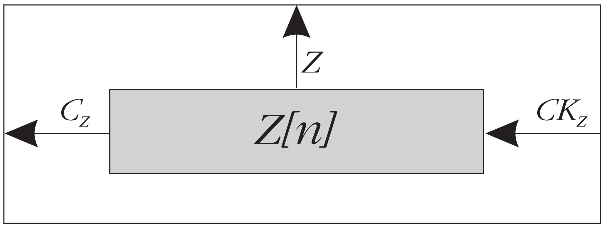


Abbildung 4.12: Der Zähler

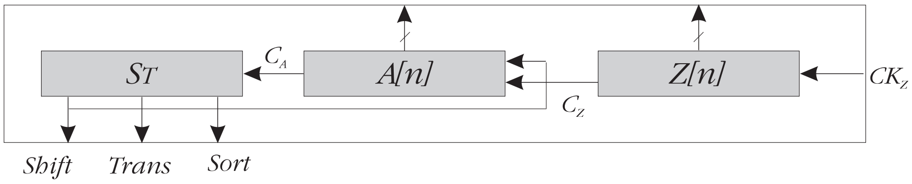


Abbildung 4.13: Verbindung einzelner Bausteine

4.4 Die Adressberechnung

Eine der wichtigsten Aufgaben von der Steuereinheit ist, die Adressen der zu sortierenden Zahlen zu berechnen und an die Würfelschaltung weiterzuleiten. Wie man dem Abschnitt 3.4 entnehmen kann, kann man die aktuelle Adresse wie in Abb. 4.14 berechnen.

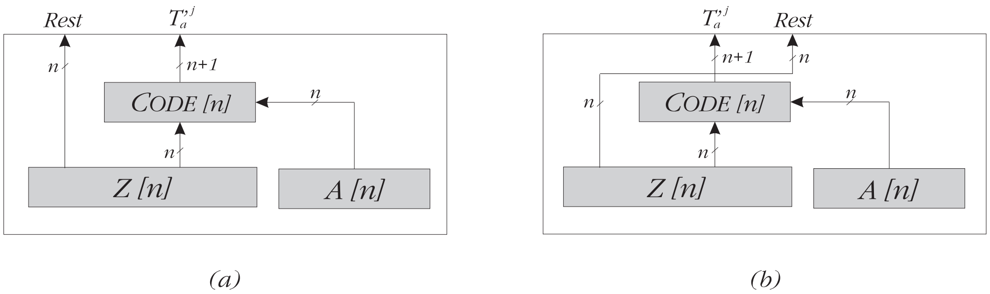


Abbildung 4.14: Das Adressberechnungsschema

Im ersten Fall (Transposition) wird T'^{j}_{a} rechts, in dem zweiten — links von dem oberen Teil des Zählers ausgeführt (Abb. 4.14(a)), in allen anderen Fällen ist das Gegenteil der Fall, Abb. 4.14(b). Die korrekte Plazierung der Leitungen wird durch die Schaltung $LR[n]$ realisiert (Abb. 4.15).

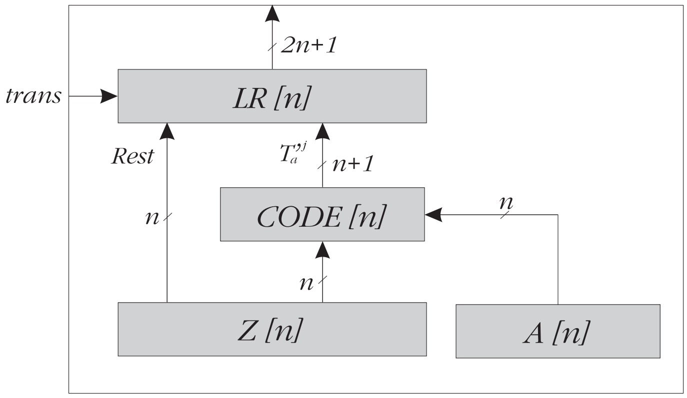


Abbildung 4.15: Schaltung für die Adressberechnung

Außer der oben beschriebenen Variablen gibt die Steuereinheit die Variable AA aus, die wie in Abb. 4.16 definiert ist.

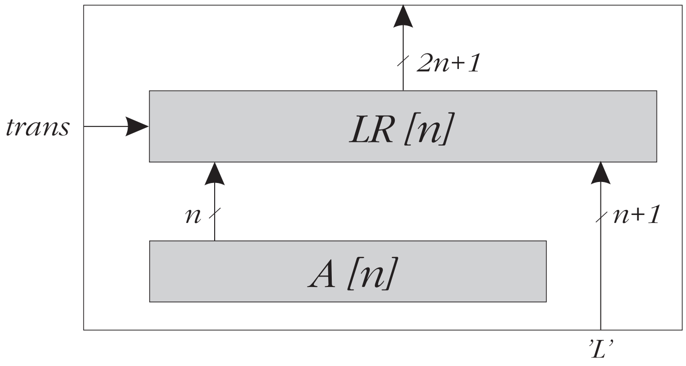


Abbildung 4.16: Realisierung von A

4.5 Realisierung der Ein– und Ausgabe

Wie schon im vorigen Kapitel gesagt wurde, kann man die Daten Ein– und Ausgabe nach der folgenden Prozedur realisieren:

begin{Ausgabe}
$j = 1$;
do

for $(i = -1, ++i, i > j)$
{
if $(a_j = 1)$
{
Versetze das System in *Shift*, tausche die Daten entlang der
i –ten Dimension;
Versetze das System in *Rückshift*, tausche die Daten entlang der
i –ten Dimension;
}
}
$j++;$
while $\{j \leq n-1\}$
end$\{Ausgabe\}$

Also ist der einzige Unterschied bei der Ein– und Ausgabe gegenüber dem *Shiften* bzw. *Rückshiften* in der Berechnung der Dimension, entlang welcher der Austausch stattfinden soll.
Sei $ZP[n]$ ein Zähler, der bei jeder Belegung $(k_{n-1}; ...; k_0)$ ihrer Zellen nicht $(k_{n-1}; ...; k_0)$, sondern deren Parity– Bits P_i ausgibt (Abb. 4.17).

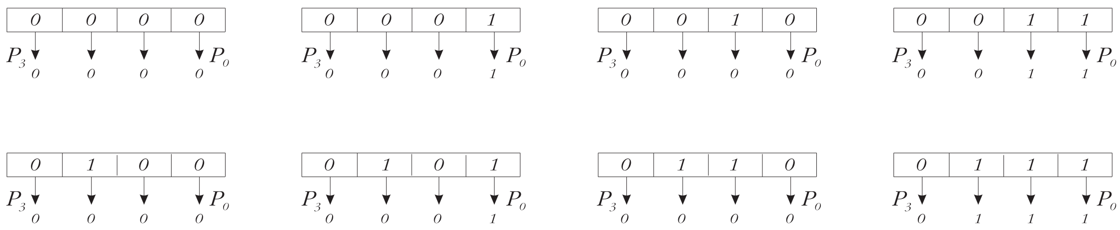


Abbildung 4.17: Zusammenhang der Ausgaben von $ZP[n]$ mit der inneren Belegung

Sei nun $C_n(k) : \{0; 1\}^n \longrightarrow N$ eine Funktion, die folgendermaßen definiert ist:

$$C_n(k) = \sum_{i=0}^{n-1} P_i(k).$$

Dann stimmt $C_n(k)$ mit der Dimension überein, entlang welcher der Datenaustausch in k. Schritt der Ein– bzw. Ausgabe erfolgen soll. Diese Tatsache folgt unmittelbar aus der Konstruktion von $Z[n]$ und dem Datenein– und Ausgabealgorithmus (Kap. 3).
Also kann man die Dimensionsausrechnung wie in Abb. 4.18 realisieren.

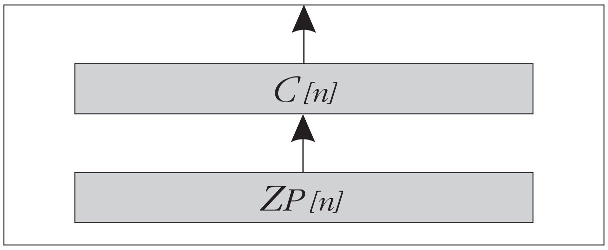


Abbildung 4.18: Realisierung der Adressberechnung

4.6 Synchronisation des Gesamtsystems

Unter der "Synchronisation des Gesamtsystems" wird die Steuerung einzelner Teilschritte des Sortieralgorithmus verstanden, d. h. die Teilschritte werden sequentiell, in gewisser Reihenfolge und zum gewissen Zeitpunkt ausgeführt, sodaß es zu keinem Datenverlust kommen kann.

Berechnen wir als erstes die Laufzeit des Datenaustausch– und Sortiervorganges in einzelnem.

- Datenaustausch

Bei dem Datenaustauschvorgang sollen folgende Schritte implementiert werden:

1. Registerzustände der Steuervariablen (Zähler Z, A, AA und Adressregister) ändern;
2. Die im gegebenen Schritt aktuelle Dimensionsleitung deblockieren;
3. Die 'erweiterte Adresse' EA berechnen;
4. Durch die Schaltung $DAT[n]$ die aktuelle Zellenadresse (mittels EA und AA) berechnen und an RAM weiterleiten;
5. Zellenadresse dekodieren;
6. Die Daten der aktuellen Zelle an $ExOut$ anlegen;
7. Daten speichern.

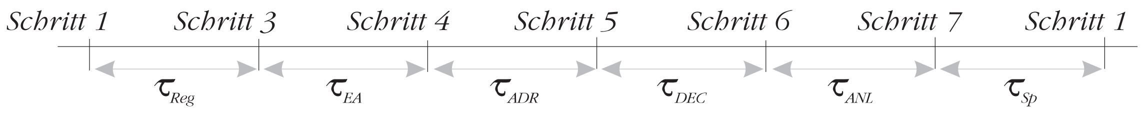


Abbildung 4.19: Reihenfolge der Unterschritte beim Datenaustausch

Das Schema der Ausführungsreihenfolge dieser Schritte ist in Abb. 4.19 gezeigt. (Schritt 2 kann parallel zu Schritt 1 ausgeführt werden und ist deswegen nicht in der obigen Reihenfolge angezeigt).
Da die Verzögerungszeit bei Datenspeichern wesentlich kürzer ist als die der hintereinander ausgeführten Schritte 1 bis 4, könnte man die Schritte 1 und 7 zusammen ausführen(Abb. 4.20).

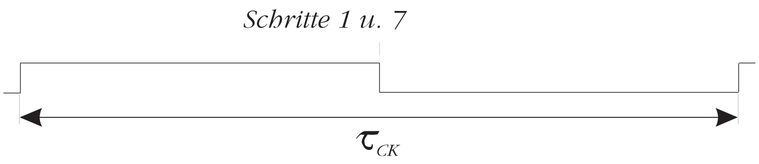


Abbildung 4.20: Ausführungszeit der Schritte 1 und 7

Man beachte dabei, daß die Daten bei der L/H Flanke von dem CK Signal gespeichert werden. Dies ist nötig, um die Zeit zur Änderung betreffender Register bei dem Wechsel vom Sortier– zu Datenaustauschvorgang durch die L/H Flanke von CK zu gewinnen (die restlichen Register werden während des Sortievorganges gesetzt).

Bezeichnet man die Verzögerungszeiten der Schritte 1, 3, 4, 5 und 6 mit τ_{Reg}, τ_{EA}, τ_{ADR}, τ_{DEC} und τ_{ANL}, so kann man die Zeit τ_{CK}, die man für den Austausch eines Datenelementes braucht, durch folgende Formel ausdrücken:

$$\tau_{CK} = \tau_{Reg} + \tau_{EA} + \tau_{ADR} + \tau_{DEC} + \tau_{ANL}$$

Berechnen wir nun die einzelnen Verzögerungszeiten[1]:

$$\begin{aligned}
\tau_{Reg} &= \tau_{idfr1a} \\
\tau_{EA} &= \tau_{CODE} + \tau_{LR} = 3 \cdot \tau_{imux2a} \\
\tau_{ADR} &= \tau_{imux2a} \\
\tau_{DEC} &= (2n+1) \cdot (\tau_{iand1a} + \tau_{iinv1a}) \\
\tau_{ANL} &= \tau_{imux1a}
\end{aligned}$$

[1] Hier sind τ_{idfr1a}, τ_{imux2a}, τ_{imux1a}, τ_{iand1a} und τ_{iinv1a} die Verzögerungszeiten vom Register, zwei Arten von Multiplexern, boolschen AND und Negation

Also gilt:

$$\begin{aligned}\tau_{CK} &= \tau_{idfr1a} + 3 \cdot \tau_{imux2a} + \tau_{imux2a} + (2n+1) \cdot (\tau_{iand1a} + \tau_{iinv1a}) + \tau_{imux1a} = \\ &= \tau_{idfr1a} + 4 \cdot \tau_{imux2a} + \tau_{imux1a} + (2n+1) \cdot (\tau_{AND} + \tau_{iinv}).\end{aligned}$$

Nehmen wir die maximalen Verzögerungszeiten einzelner Gatter, so erhält man:

$$\begin{aligned}\tau_{CK} &= 4,89 + 4 \cdot 3,02 + 3,02 + (2n+1) \cdot (3,06 + 1,32) = 4,89 + 5 \cdot 3,02 + 4,38 \cdot (2n+1) \\ &= 19,99 + 4,38 \cdot (2n+1) < 20 + 4,5 \cdot (2n+1).\end{aligned}$$

Die Gesamtzeit des Sortiervorganges $\tau_{DAT[n]}$ kann man dann folgendermaßen ausrechnen:

$$\tau_{DAT} = n \cdot 2^{2n+1} \cdot (20 + 4,5 \cdot (2n+1)).$$

- Sortieren

Um die Verzögerungszeit von $Sort_{2^k}$ ausrechnen zu können, muß man erst deren Tiefe $\mathcal{S}_{2^k}$ berechnen. Die ist aber $\mathcal{S}_{2^k} = \mathcal{S}_{2^{k-1}} + \mathcal{M}_{2^k}$, wo $\mathcal{M}_{2^k}$ die Tiefe von $Merge_{2^k}$ ist (siehe **Anhang A**).
Da $\mathcal{S}_1 = 1$ und $\mathcal{M}_{2^k} = k$ ist, gilt: $\mathcal{S}_k = \frac{(k+1) \cdot k}{2}$. In unserem Fall ist $k = 2n+1$. Deswegen gilt:

$$\mathcal{S}_{2^{2n+1}} = \frac{2(n+1)(2n+1)}{2} = 2n^2 + 3n + 1.$$

Also ist die Verzögerungszeit von $Sort[n]$ gleich:

$$\tau_{Sort[n]} = (2n^2 + 3n + 1) \cdot \tau_{Comp}$$

Hier ist τ_{Comp} die Verzögerungszeit von der Schaltung CMP (siehe Anhang A).

4.6.1 Die Synchronisation

Man kann leicht sehen, daß $\tau_{CK} < \tau_{Sort} < \tau_{DAT}$ gilt.
Also darf man den Sortierdurchgang nicht gleich nach der ersten $CK-$ Flanke abbrechen, andererseits wäre es eine Zeitverschwendung, das Sortieren τ_{DAT} lang laufen zu lassen. Eine alternative Lösung wäre es, als Sortierzeit $2 \cdot 2^{2n}$- mal τ_{CK} zu nehmen.

Kurz kann man das ganze folgendermaßen formulieren:
Das Sortierverfahren wird in acht Teilschritten durchgeführt. Dabei werden in den ungeraden Teilschritten, die $2^{2n+1} \cdot \tau_{CK}$ Zeit dauern, die Daten jedes Knotens lokal

sortiert. In den geraden Schritten (die $n2^{2n+1} \cdot \tau_{CK}$ Zeit dauern) werden die Daten zwischen den einzelnen Knoten ausgetauscht.

Aus dem oben gesagten folgt, daß der Zustandszähler entweder nach 2^{2n+1} Schritten (falls es sortiert wird, also $Sort$ ='H') oder nach $n \cdot 2^{2n+1}$ Schritten (bei dem Datenaustausch, $Sort$ ='L') getaktet sein soll. Außerdem sollen das Register A und die *Deblock*–Schaltung während des Datenaustausches nach 2^{2n} WR Signalen getaktet werden. Dabei wird der CK– Eingang der *Deblock*– Schaltung während des Sortierens blockiert und das Register A auf ihre Startposition (*1,0,...,0*) gesetzt.
Dies ist in Abb.4.3 gezeigt.

4.6.2 Einschätzung der gesamten Rechenzeit

Die gesamte Rechenzeit beträgt also:

$$\begin{aligned}\tau_n &= 4 \cdot \tau_{DAT[n]} + 4 \cdot \tau_{Sort[n]} = 4 \cdot 2^{2n+1} \cdot n \cdot \tau_{CK} + 4 \cdot 2^{2n+1} \cdot \tau_{CK} \\ &= 4 \cdot 2^{2n+1} \cdot \tau_{CK} \cdot (n+1) = \cdot 2^{2n+3}(n+1) \cdot (20 + 4,5 \cdot (2n+1)).\end{aligned}$$

Da bei den Ausrechnungen maximale Verzögerungszeiten einzelner Gatterelemente genommen werden, kann es keine Datenverlustgefahr bestehen.

Als ein konkretes Beispiel kann man $n = 7$ nehmen, es werden also insgesamt 2^{22} Zahlen sortiert.

$$\tau_{CK} = 20 + 4,5 \cdot 15 = 87,5\ ns.$$

(Einfaccheitshalber wird hier und weiter τ_{CK} ohne Parameter n angegeben.)

τ_{CK} kann man nach oben bis 100 "abrunden", ohne es zu den Datenverlußt zu kommen. Also nehmen wir an: $\tau_{CK} = 100\ ns$.

Dann ergibt sich die Berechnungzeit zu:

$$\begin{aligned}\tau_7 &= 4 \cdot 7 \cdot 2^{15} \cdot 100 + 4 \cdot 2^{15} \cdot 100 = 4 \cdot 2^{15} \cdot 100 \cdot 8 = \\ &= 2^{20} \cdot 100 = 104857600\ ns < 0,11\ sec.\end{aligned}$$

Also werden die 2^{22} Zahlen in weniger als 0,11 Sekunden sortiert.

Die oben durchgeführten Berechnungen gelten für die 1– Bit Zahlen. Für die größeren Zahlen sind die Berechnungen analog.

Kapitel 5

Berechnung der oberen Schranken für Zeit– und Oberflächenwachstum

In diesem Kapitel werden die Laufzeit– und Oberflächenwachstumfunktionen der entwickelten Schaltung angegeben und deren oberen Schranken eingeschätzt.
Im Gegenteil zu den anderen Schaltkreisen wird bei der Angabe der oberen Schranken auch der innere Speicher dazugezählt. Dies ist wegen der eigenartigen Konstruktion des inneren Speichers (externe und interne Eingänge etc., siehe Kap. 4) unvermeidbar. Zwar hat das keine Auswirkung auf die Angabe der oberen Schranken (wie es unten gezeigt wird), doch wird die Oberfläche der Schaltung dadurch exponentiell wachsen. Diesen Nachteil muß man aber in Kauf nehmen, um die Laufzeit der Schaltung in Grenzen zu halten.

5.1 Die Laufzeit

5.1.1 Laufzeit eines OEM– Sortierers

Um die Laufzeit eines Odd– Even– Mergesortierers einschätzen zu können, muß man erst dessen Tiefe $T(Sort_n)$ berechnen. Da für OEM– Sortierer die Gleichung $Sort_n = (Sort_{n-1} \ominus Sort_{n-1}) \oplus M_n$ gilt, kann man schreiben:

$$T(Sort_1) = T(M_1) = 1, \;\; T(Sort_n) = T(Sort_{n-1}) + T(M_n).$$

Für M_n gilt: $M_1 = 1$, $T(M_n) = T(M_{n-1}) + 1 = n$.

Also ist die Tiefe gleich: $T(Sort_n) = \sum\limits_{j=1}^{n} j = \frac{n\cdot(n+1)}{2}$.

Nimmt man statt n $2n+1$, so gilt: $T(Sort_{2n+1}) = \frac{(2n+1)(2n+2)}{2} = (2n+1)(n+1) = 2n^2+3n+1$.

Die Laufzeit dieser Schaltung stimmt mit deren Tiefe überein:
$L(Sort_{2n+1}) = T(Sort_{2n+1}) = 2n^2+3n+1$.

Für die oberen Schranken gilt: $O(2n^2+3n+1) = O(n^2)$.

Seien nun N Zahlen zu sortieren. Dann ist die Tiefe der entsprechenden $OEM-$ Schaltung $O(\log^2 N)$.

5.1.2 Laufzeit der Gesamtschaltung

Berechnen wir nun die Laufzeit der gesamtschaltung $CHIP_n$. Aufgrund der Parallelität des Gesamtsystems stimmt sie mit der Laufzeit der einzelnen Knoten überein.
Also gilt: $L(CHIP_n) = 4 \cdot n \cdot 2^{2n+1} + 4 \cdot L(Sort_{2n+1})$.

Wie man dem vorigen Abschnitt entnehmen kann, ist $L(Sort_{2n+1}) = 2n^2+3n+1$. Daraus folgt: $L(CHIP_n) = 4 \cdot (n \cdot 2^{2n+1} + 2n^2 + 3n + 1)$.

Für die oberere Schranke der Laufzeit gilt: $O(4 \cdot (n \cdot 2^{2n+1} + 2n^2 + 3n + 1)) = O(n \cdot 2^{2n})$.

Nimmt man an, man solle $N = 2^{3n+1}$ Elemente sortieren, so ist die obere Grenze der Laufzeit des Systems $O(\sqrt[3]{N^2} \cdot \log N)$.

5.2 Die Oberfläche

5.2.1 Oberfläche eines $OEM-$ Sortierers

Die Kosten eines $OEM-$ Sortierers betragen $C(Sort_n) = T(Sort_n) \cdot 2^n$. Aus den oben angegebenen Formeln folgt:
$C(Sort_{2n+1}) = 2^{2n+1} \cdot (2n^2+3n+1)$.

Die obere Schranke ist $O(n^2 \cdot 2^{2n})$.

Nimmt man an, daß $N = 2^{2n+1}$ Zahlen zu sortieren sind, beträgt dann die Oberflächenwachstumsrate $O(N \cdot \log^2 N)$.

5.2.2 Oberfläche der Gesamtschaltung

Die Kosten der Gesamtschaltung $C(CHIP_n)$ kann man mittels folgender Formel ausrechnen:

$$C(CHIP_n) = C(Cube_{n,j,n}) + C(Steuer_n).$$

Für $C(Cube_{n,j,n})$ und $C(Steuer_n)$ gilt:

$C(Steuer_n) = 19 \cdot n + 33,$
$C(Cube_{n,j,n}) = 2^n \cdot C(Knoten_n).$

Dabei läßt sich $C(Knoten_n)$ nach folgender Formel ausrechnen:
$C(Knoten_n) = C(Block_{n+1}) + C(RAM_n) + C(DAT_{2n+1}) + C(ADR_n) + C(Sort_n).$

Berechnen wir nun jede Formel einzeln:

$C(Block_{n+1}) = 2n + 2;$
$C(DAT_{2n+1}) = 2n + 1;$
$C(ADR_n) = 2n + 1;$
$C(RAM_n) = C(DEBLOCK_(2n + 1)) + 2^{2n+1} \cdot 5 = (2n + 1) \cdot 2^{2n+1} - 1 + 5 \cdot 2^{2n+1} = (2n + 6) \cdot 2^{2n+1} - 1.$
Hier ist $DEBLOCK_n$ die Schaltung eines Adressdekodierers.

Also gilt:

$$\begin{aligned} C(Cube_{n,j,n}) &= 2^n \cdot (6n + 4 + (2n + 6) \cdot 2^{2n+1} - 1 + 2^{2n+1} \cdot (2n^2 + 3n + 1)) \\ &= 2^n \cdot (2^{2n+1} \cdot (2n^2 + 5n + 7) + 6n + 3). \end{aligned}$$

Für die obere Schranke gilt: $O(C(CHIP_n)) = O(n^2 \cdot 2^{3n})$.

Daraus folgt, daß die obere Schranke für den Oberflächenwachstum der Schaltung $CHIP_n$ mit der von $Sort_{3n+1}$ übereinstimmt: $O(N \cdot \log^2 N)$.

Wie aus dem oben gesagten folgt, ist die entwickelte Schaltung nicht so schnell, wie z.B. die entsprechende OEM Schaltung, diesen Nachteil gleicht aber der geringere Platzbedarf aus. Da die Schaltung aber den OEM Sortierer beinhaltet, ist deren Wachstumsrate mindestens so groß wie die von OEM.

Aufgrund des exponentiellen Wachstums kann es passieren, daß die Schaltung ab einem gewissen n nicht mehr auf einem Kristall realisierbar ist. In solchen Fällen soll sie in mehrere Teile verteilt werden. Mit der Verteilung und damit verbundenen Problemen beschäftigen wir uns im nächsten Kapitel.

Kapitel 6

Verteilung der Schaltung in mehrere Teile

6.1 Zerlegung nach der Konstruktionsmethode

Wie man dem vorigen Kapitel entnehmen kann, nimmt die Oberfläche der Sortierschaltung nach dem Wachstum der Dimension exponentiell zu. Nach dem heutigen Stand der Technik kann man einen 16 Bit breiten Sortierer für $n = 5$ auf einem Kristall realisieren. Wenn man aber größere Sortierer zusammenstellen will, müssen mehrere kleinere Schaltungen zusammengestellt werden, sodaß das ganze System einen gewünschten Sortierer realisiert. Nach dem ersten Blick auf das

Definition 6.1 Als einen 'maximalen Teilwürfel' bezeichnen wir eine Schaltung $Cube_{n,j,m}$, für die gilt: $Cube_{n,j,m}$, nicht aber $Cube_{n,1-j,m+1}$ ist auf einem Kristall realisierbar ($j = \lceil \frac{m}{2} \rceil - \lfloor \frac{m}{2} \rfloor$).

Nach dem raschen Blick auf das Konstruktionsschema fällt einem als erstes die Idee der Verteilung nach der Konstruktionsmethode ein. D.h., es werden 'maximale Teilwürfeln' wie im Kap. 1 konstruiert, die größeren Schaltungen aber nach der weiteren Neben– und Übereinandersetzung zweier Teilwürfel, wie es bei der Konstruktion der Netzwerkstruktur der Fall ist. Ein Beispiel solcher Konstruktion ist in Abb. 6.1 gezeigt, wo ein siebendimensionale Würfel aus vier fünfdimensionalen Teilwürfeln zusammengestellt ist.

6.1.1 Berechnung der Pinzahl

Bei der oben dargestellten Konstruktionsmethode kann man einen falschen Eindruck bekommen, man könne durch dieses dauende Neben– und Übereinandersetzen beliebig große Schaltungen konstruieren. Das Problem bei der Konstruktion solcher Schaltungen ist, wie es später in diesem Kapitel gezeigt wird, daß es für gewisse n der 'maximale

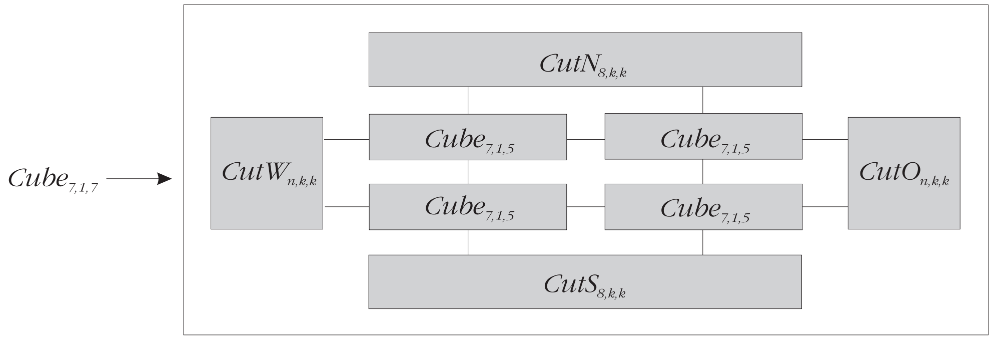


Abbildung 6.1: Die Würfelschaltung der siebten Dimension

Teilwürfel' zu viele Dimensionsleitungen auf O/W und N/S Seiten hat, was zu viele Pins auf einem Chip erfordert.

Seien $n, m \in N,\ j = \lceil \frac{m}{2} \rceil - \lfloor \frac{m}{2} \rfloor$.

Bezeichne ferner $W(Cube_{n,j,m})$ und $N(Cube_{n,j,m})$ die Anzahl der Dimensionseitungen auf der West– bzw. Nordseite der Schaltung $Cube_{n,j,k}$. Dann kann man die Gesamtzahl aller Leitungen auf den O/W und N/S Seiten von $Cube_{n,j,m}$ folgendermaßen ausdrücken:

$$L(Cube_{n,j,m}) = 2 \cdot (N(Cube_{n,j,m}) + W(Cube_{n,j,m})).$$

Da aber außer der Dimensionsleitungen in der eigentlichen Schaltung auch $7 \cdot n + 9 - 2 \cdot m$ Steuerleitungen ausgeführt werden, ist die Zahl aller auf die O/W und N/S Seiten ausgeführten Leitungen gleich:

$$\begin{aligned} P(Chip_{n,j,m}) &= L(Cube_{n,j,m}) + 7 \cdot n - 2 \cdot m + 9 \\ &= 2(N(Cube_{n,j,m}) + W(Cube_{n,j,m})) + 7 \cdot n - 2 \cdot m + 9. \end{aligned} \tag{6.1}$$

Folgender Satz drückt $W(Cube_{n,j,m})$ durch eine Formel aus:

Satz 6.1: Seien n, m und j wie oben. Dann kann man die Anzahl der Verknüpfungsleitungen auf der Westseite von $Cube_{n,j,m}$ $\ W_m = O(Cube_{n,j,m})$ folgendermaßen ausdrücken:

$$W_m = 2^{\lfloor \frac{m}{2} \rfloor} \cdot W_0 - 2^m + 2^{\lceil \frac{m}{2} \rceil}, \text{ wo } W_0 = 2^{\lceil \frac{n}{2} \rceil} - 1 \text{ ist.}$$

Um diesen Satz beweisen zu können, muß erst die Korrektheit folgender Lemmata gezeigt werden:

Lemma 6.1 : Sei $\{a_i\}_{i=1}^{\infty}$ eine Folge mit $a_1 = 1$, $a_{2i} = 2 \cdot a_{2i-1}$ und $a_{2i+1} = a_{2i} + 2^{2i}$. Dann kann man a_i für $\forall i \in N$ folgendermaßen ausdrücken:

$$a_i = 2^i - 2^{\lfloor \frac{i}{2} \rfloor}.$$

Beweis:

$$\begin{aligned} a_1 = & \quad 1 \\ a_2 = & \quad 2^2 - 2^1 = 2 = 2 \cdot a_1 \\ a_3 = & \quad 2^3 - 2^1 = 6 = a_2 + 2^2 \end{aligned}$$

Gelte die Behauptung für $\forall i \in \{1, ..., 2k-1\}$.

Induktionsschritt:

$$\begin{aligned} a_{2k} &= 2 \cdot a_{2k-1} = 2(2^{2k-1} - 2^{\lfloor \frac{2k-1}{2} \rfloor}) \\ &= 2^{2k} - 2 \cdot 2^{\lfloor \frac{2(k-1)}{2} \rfloor} \\ &= 2^{2k} - 2 \cdot 2^{k-1} \\ &= 2^{2k} - 2^k = 2^{2k} - 2^{\lfloor \frac{2k}{2} \rfloor} \end{aligned}$$

$$\begin{aligned} a_{2k+1} &= a_{2k} + 2^{(2k+1)-1} \\ &= 2^{2k} - 2^{\lfloor \frac{2k}{2} \rfloor} + 2^{2k} \\ &= 2^{2k+1} - 2^{\lfloor \frac{2k}{2} + \frac{1}{2} \rfloor} \\ &= 2^{2k+1} - 2^{\lfloor \frac{2k+1}{2} \rfloor}. \end{aligned}$$

Lemma 6.2 : Auf jeder Hierarchieebene m kann man W_m folgendermaßen ausdrücken:

$$W_m = 2^{\lfloor \frac{m}{2} \rfloor} \cdot W_0 - b_m.$$

(Hier ist b_m eine Zahlenfolge, die später definiert wird.)
Beweis :

$$
\begin{aligned}
W_0 &= W(Cube_{n,0,0}) = W(VW_n) = 2^{\lfloor \frac{2}{n} \rfloor} - 1 \\
W_1 &= W(Cut_{n,1} \ominus Cube_{n,0,0} \ominus Cube_{n,0,0} \ominus Cut_{n,1}) = W_0 - 1 \\
W_2 &= W(Cuben, 1, 1 \oplus Cuben, 1, 1) = 2 \cdot W_1 = 2 \cdot W_0 - 2 \\
W_3 &= W(Cut_{n,2} \ominus Cube_{n,0,2} \ominus Cube_{n,0,2} \ominus Cut_{n,2}) = 2 \cdot W_0 - 6
\end{aligned}
$$

Gelte die Behauptung für $\forall W_i$, $i \leq 2k - 1$.
Dann gilt:

$$
\begin{aligned}
W_{2k} &= W(Cube_{n,1,2k-1} \oplus Cube_{n,1,2k-1}) = 2 \cdot W_{2k-1} \\
&= 2 \cdot 2^{\lfloor \frac{2k-1}{2} \rfloor} \cdot W_0 - 2 \cdot b_{2k-1} = 2^{\lfloor \frac{2k+1}{2} \rfloor} \cdot W_0 - b_{2k}
\end{aligned}
$$

$$
\begin{aligned}
W_{2k+1} &= W(Cut_{n,2k+1} \ominus Cube_{n,0,2k} \ominus Cube_{n,0,2k} \ominus Cut_{n,2k+1}) \\
&= W_{2k} - 2^{2k} = 2^{\lfloor \frac{2k}{2} \rfloor} \cdot W_0 - (b_{2k} + 2^{2k}) = 2^{\lfloor \frac{2k}{2} \rfloor} \cdot W_0 - b_{2k+1}.
\end{aligned}
$$

Beweis des Satzes : Wie es leicht zu sehen ist, gilt:

$b_1 = 1, b_{2i} = 2 \cdot b_{2i-1}, b_{2i+1} = b_{2i} + 2^{2i}$.

Nach dem **Lemma 6.1** gilt: $b_i = 2^m - 2^{\lfloor \frac{m}{2} \rfloor}$.

Also gilt es:

$$W_m = 2^{\lfloor \frac{m}{2} \rfloor} \cdot W_0 - (2^m - 2^{\lfloor \frac{m}{2} \rfloor}) = 2^{\lfloor \frac{m}{2} \rfloor} \cdot W_0 - 2^m + 2^{\lfloor \frac{m}{2} \rfloor}.$$

Für $N_m = N(Cube_{n,j,m})$ gilt:

$$N_m = 2^{\lceil \frac{m}{2} \rceil} \cdot W_0 - 2^m + 2^{\lceil \frac{m}{2} \rceil}, \text{ wo } W_0 = 2^{\lfloor \frac{n}{2} \rfloor} - 1 \text{ ist.}$$

Dies kann man analog zu dem Satz 6.1 beweisen, nur muß man beachten, daß im Gegenteil zu W_n die "Verdoppelung" der Leitungen in jedem ungeraden Schritt stattfindet (deswegen auch $2^{\lceil \frac{m}{2} \rceil} \cdot N_0$ anstatt $2^{\lfloor \frac{m}{2} \rfloor} \cdot N_0$

Berechnen wir nun

$$\begin{aligned}
N_{n,m} + W_{n,m} &= 2^{\lceil \frac{m}{2} \rceil} \cdot N_{n,0} - 2^m + 2^{\lceil \frac{m}{2} \rceil} + 2^{\lfloor \frac{m}{2} \rfloor} \cdot W_{n,0} - 2^m + 2^{\lfloor \frac{m}{2} \rfloor} = \\
&= 2^{\lceil \frac{m}{2} \rceil}(N_{n,0} + 1) + 2^{\lfloor \frac{m}{2} \rfloor} \cdot (W_{n,0} + 1) - 2^{m+1} = \\
&= 2^{\lceil \frac{m}{2} \rceil}(2^{\lfloor \frac{m}{2} \rfloor} - 1 + 1) + 2^{\lceil \frac{m}{2} \rceil}(2^{\lfloor \frac{m}{2} \rfloor} - 1 + 1) - 2^{m+1} = \\
&= 2^{\lceil \frac{m}{2} \rceil} \cdot 2^{\lfloor \frac{n}{2} \rfloor} + 2^{\lfloor \frac{m}{2} \rfloor} \cdot 2^{\lceil \frac{n}{2} \rceil} - 2^{m+1} = \\
&= 2^{\lfloor \frac{n}{2} \rfloor + \lceil \frac{m}{2} \rceil} + 2^{\lceil \frac{n}{2} \rceil + \lfloor \frac{m}{2} \rfloor} - 2^{m+1}. \qquad (6.2)
\end{aligned}$$

Aus (6.1) und (6.2) ergibt sich folgende Formel für die Anzahl der Leitungen auf den Seiten von $Chip_{n,j,m}$:

$$\begin{aligned}
P(Chip_{n,j,m}) &= 2 \cdot (N_{n,m} + W_{n,m}) + 7 \cdot n - 2 \cdot m + 9 = \\
&= 2^{\lfloor \frac{n}{2} \rfloor + \lceil \frac{m}{2} \rceil + 1} + 2^{\lceil \frac{n}{2} \rceil + \lfloor \frac{m}{2} \rfloor + 1} - 2^{m+1} + 7 \cdot n - 2 \cdot m + 9.
\end{aligned}$$

6.2 Die Verteilungsmethode

Setzt man in die obere Formel die entsprechenden Werte ein, so kann man sehen, daß es höchstens ein achtdimensionale Würfel aus mehreren vieredimensionalen Würfeln zu konstruieren ist, ein neundimensionaler Würfel läßt sich dagegen nur aus den zweidimensionalen Teilwürfeln konstruieren, was 128 Teilbausteine benötigt.
Aus diesem Grund stellt sich die Frage, ob es auch andere Möglichkeiten gibt, den Würfel effektiver zu verteilen. Offensichtlich hängt die Verteilung von der Würfelkonstruktion ab. Wenn man das Konstruktionsschema aus dem Kap. 1 als Basis für das Schaltungnetz nimmt, gibt es also keine alternativen Teilungsmöglichkeiten.
Das eigentliche Problem bei dieser Verteilungsmethode ist der exponentielle Wachstum der Ausgänge in den Teilwürfeln. Dieses Problem wird aber durch das Durchschneiden des Basisknotens in zwei Teile gelöst.
Schneidet man den Knoten wie in Abb. 6.3, so bekommt jeder Knoten nur einen Ein– und einen Ausgang. Die Würfelstruktur selbst wird in einem zweiten Baustein wie in Kap. 1 realisiert, der Unterschied ist nur, daß als Basis zur Konstruktion der obere Teil der aufgespaltenen Schaltung genommen wird und in jedem Konstruktionsschritt k 2^{k+1} zusätzliche Ausgänge ausgeführt werden (siehe Abb. 6.4).
Diese Ausgänge dienen dazu, um die nach in Abb. 6.5 gezeigtem Schema zusammengefügten Bausteine anzuschließen. Damit wird es erreicht, daß die ganze Würfelstruktur mit ihren exponentiell wachsenden Verbindungsleitungen in dem Chip "verborgen"

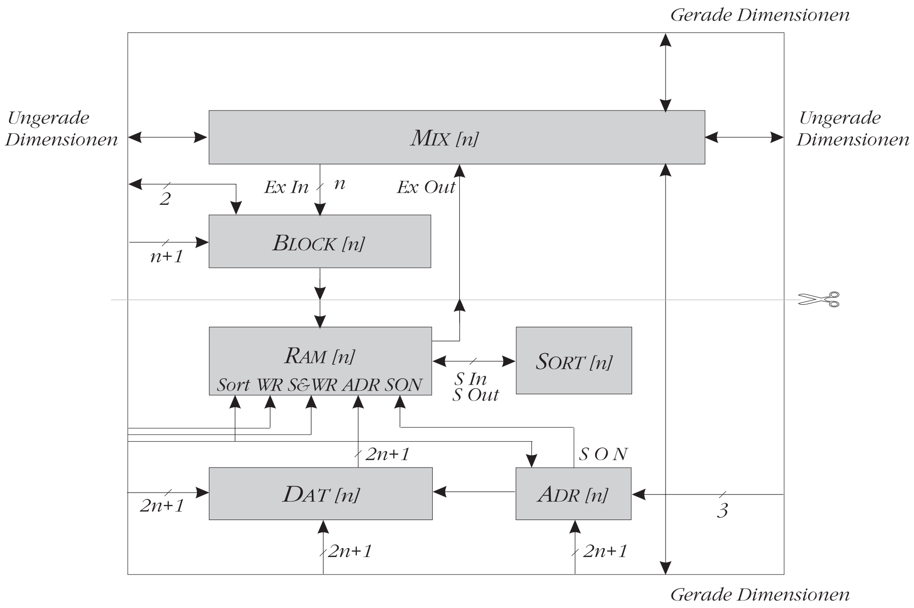


Abbildung 6.2: Schnittlinie des Basisknoten

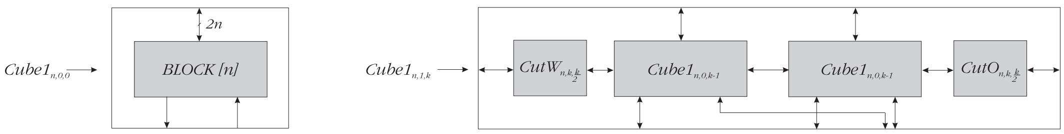


Abbildung 6.3: Konstruktion des Verbindungbausteines in einem ungeraden Schritt

bleibt. Da dieses Teil nur aus $n \cdot p \cdot 2^k$ Elementen und den Verdrahtungsleitungen besteht (hier ist p die Breite der Zahlen in Bits), kann man es für ziemlich große n auf einem Kristall realisieren.
Die Anzahl der Leitungen auf den Außenseiten ist also: $P(Wire_{n,0}) = 2^{n+1} \cdot p$ (hier ist p wiederum die Breite der zu sortierenden Zahl).
Sei nun $Cube_{n,j,m}$ der 'maximale Teilwürfel'. Dann ist die Schaltung *Teil[n,m]*, gezeigt in Abb. 6.5, auch auf einem Kristall realisierbar. Also kann man die Schaltung $Chip_n$ wie in Abb. 6.6 realisieren.
Diese Methode hat den Vorteil, daß es auf die Ränder der Teilchips nicht $2^{n+1} \cdot p$, sondern nur $2^{m+1} \cdot p$ Ausgänge ausgeführt werden.

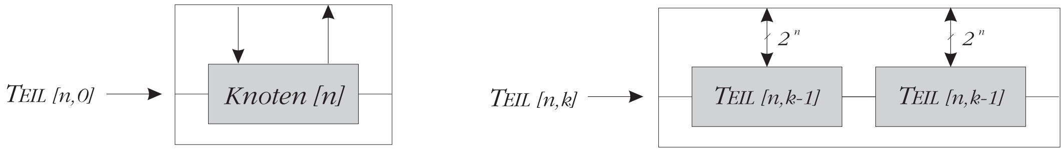


Abbildung 6.4: Zusammenfügen der Basisknoten

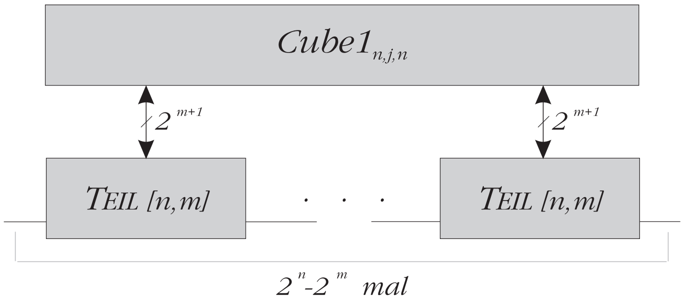


Abbildung 6.5: Konstruktion eines großen Würfels

Anhang A

Korrektheitsbeweis Des OEM– Sortierers

Die Sortierschaltung *Sort[n]* hat folgende hiererchische Struktur:

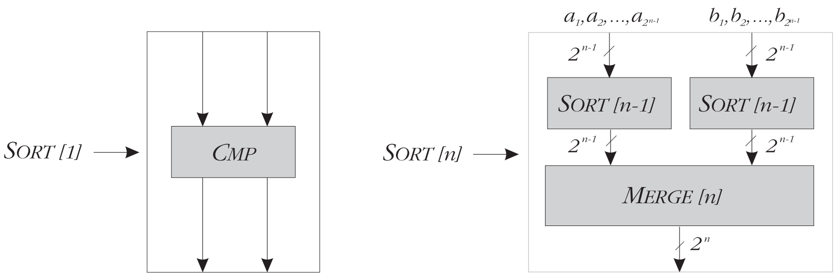


Abbildung A.1: Die Schaltung *Sort[n]*

Die wichtigste Komponente dieser Schaltung ist *CMP* (die auf dieser Hierarchieebene nicht zu sehen ist), die zwei Zahlen sortiert. Dabei stimmt die mit der Schaltung *Sort[1]* überein. Die expandierte Schaltung *Sort[n]* besteht nur aus den *CMP*– Elementen, deren Ein– bzw. Ausgänge in bestimmter Reihenfolge verknüpft sind.
In Abb. A1 ist *Merge[n]* eine Schaltung, die die am Eingang anliegende Zahlenfolge korrekt sortiert, falls die Folgen $\{a_i\}_1^{2^{n-1}}$ und $\{b_i\}_1^{2^{n-1}}$ korrekt vorsortiert sind (siehe Abb. A2).
Für *Sort[1]* gilt die Korrektheit offensichtlich. Gelte die nun auch für $\forall i \in \{1, ..., n-1\}$. Offensichtlich genügt es für den Korrektheitsbeweis von *Sort[n]*, die Korrektheit von *Merge[n]* zu zeigen.

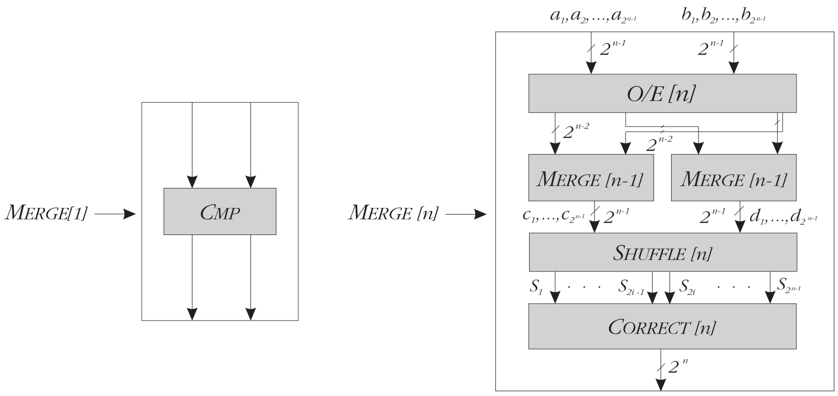


Abbildung A.2: Die *Merge[n]*– Schaltung

A.1 Korrektheitsbeweis der *Merge[n]*– Schaltung

In Abb. A2 ist die Hierarchische Struktur von *Merge[n]* gezeigt.
Hier ist die Schaltung $O/E[n]$ wie in Abb. A.3 definiert.

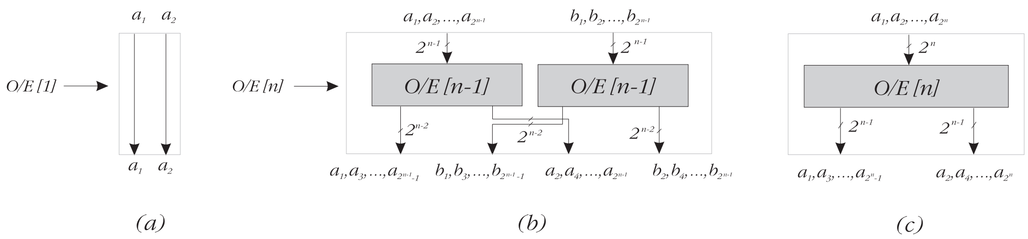


Abbildung A.3: Das Prinzip von $O/E[n]$

Gibt man als Eingabe in diese Schaltung die Zahlenfolge $\{a_i\}_{i=1}^{2^n}$ ein, so gibt die Schaltung als Ausgabe links die Elemente mit dem ungeraden, rechts — die mit dem geraden Index aus (Abb. A3c).

Die Schaltung *Shuffle[n]* dagegen ist eine Umkehrung der *O/E[n]*– Schaltung (Abb. A.4).

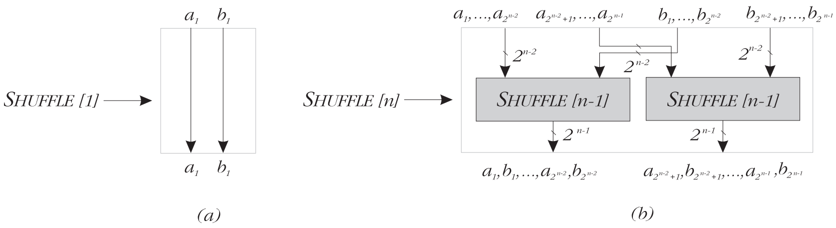


Abbildung A.4: Die Schaltung *Shuffle[n]*

Letztendlich ist *Correct[n]* eine Schaltung, die aus 2^n nebeneinandergesetzten *CMP*–Schaltungen besteht und je zwei benachbarte Elemente paarweise sortiert (abb.A5).

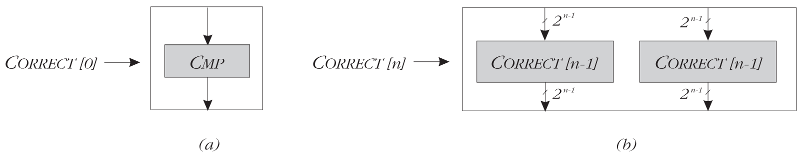


Abbildung A.5: Die *Correct[n]*– Schaltung

Die Korrektheit dieser Schaltungen läßt sich leicht durch die strukturelle Induktion beweisen, was dem Leser überlassen ist.

Der Beweis der Korrektheit von *Merge[n]* wird mit Hilfe der strukturellen Induktion durchgeführt. Für $n = 1$ funktioniert die Schaltung korrekt, da die nur aus einem *CMP*– Baustein besteht.

Induktionsannahme : Gelte nun die Behauptung für alle *Merge[i]*, $i \in \{1, 2, ..., n-1\}$.

Induktionsschritt : $n \longrightarrow n + 1$.

Bemerkung: Nach der Induktionsannahme gilt: $c_j \geq c_{j+i}$, $d_j \geq d_{j+i}$, $i + j \in \{1, ..., 2^n\}$.

Die Schaltung *Merge[n]* hat folgende wichtige Eigenschaften:

- falls $S_{2j} > S_{2j-1}$ ist, wird dieser Fehler in letztem Teil der Schaltung, nämlich in *Correct[n]* berichtigt (siehe das Konstruktionsschema oben);

- Sei $S_{2j+1} \leq S_{2j}$. Seien ferner $S_{2j+1} = c_k$, $S_{2j} = d_l$ $(k, l \in N)$.
  Betrachten wir nun S_{2j-m}, $m \in N$. Es gibt zwei Möglichkeiten:

  1. $s_{2j-m} = c_{k-p} \leq c_{k+1} = s_{2j+1}$;
  2. $s_{2j-m} = d_{l-p} \leq d_{l+1} = s_{2j+1}$.

  In beiden Fällen gilt also: $S_{2j+1} \leq S_{2j}$.

Aus dem oben gesagten folgt, daß man für den Korrektheitsbeweis von *Merge[n]* die Ungleichung $s_{2j+1} \leq s_{2j}$ zeigen soll.

Ohne Beschränkung der Allgemeinigkeit kann man annehmen: $j = 1$, $s_2 = d_1$, $s_3 = c_2$ (nachher könnte man den Beweis für beliebige $j \in N$ erweitern).

Da es nach der Induktionsannahme jede Teilschaltung korrekt funktioniert, gilt:

$$c_1 \in \{a_1, b_2\}; d_1 \in \{a_2, b_1\}; c_2 \in \{a_1, a_3, b_2, b_4\}.$$

Deswegen können nur folgende Fälle auftreten:

- $d_1 = b_1$

  1. $c_1 = a_1, c_2 = a_3, \Longrightarrow d_1 = b_1 \geq a_2 \geq a_3 = c_2$, also $\underline{d_1 \geq c_2}$
  2. $c_1 = a_1, c_2 = b_2 \Longrightarrow d_1 = b_1 \geq b_2 = c_2$, also $\underline{d_1 \geq c_2}$
  3. $c_1 = b_2, c_2 = a_1 \Longrightarrow d_1 = b_1 \geq b_2 \geq a_1 = c_2$, also $\underline{d_1 \geq c_2}$
  4. $c_1 = b_2, c_2 = b_4 \Longrightarrow d_1 = b_1 \geq b_4 = c_2$, also $\underline{d_1 \geq c_2}$

  Bemerkung : In beideb ersten Fällen ist $c_2 = b_4$, in den letzten Beiden — $c_2 = a_3$ ausgeschlossen.

- $d_1 = a_2$

  1. $c_1 = a_1, c_2 = a_3 \Longrightarrow d_1 = a_2 \geq a_3 = c_2$, also $\underline{d_1 \geq c_2}$
  2. $c_1 = a_1, c_2 = b_2 \Longrightarrow d_1 = a_2 \geq b_1 \geq b_2 = c_2$, also $d_1 \geq c_2$
  3. $c_1 = b_2, c_2 = a_1 \Longrightarrow d_1 = a_2 \geq d_{i+1} = b_1 \geq b_2 \geq a_1$,
     also $a_2 \geq a_1$ und, nach der I.A., $a_2 = a_1 \quad (i > 0)$
  4. $c_1 = b_2, c_2 = b_4 \Longrightarrow d_1 = a_2 \geq b_1 \geq b_4 = c_2$, also $d_1 \geq c_2$

Damit ist die Korrektheit der Schaltung *Merge[n]* komplett bewiesen.

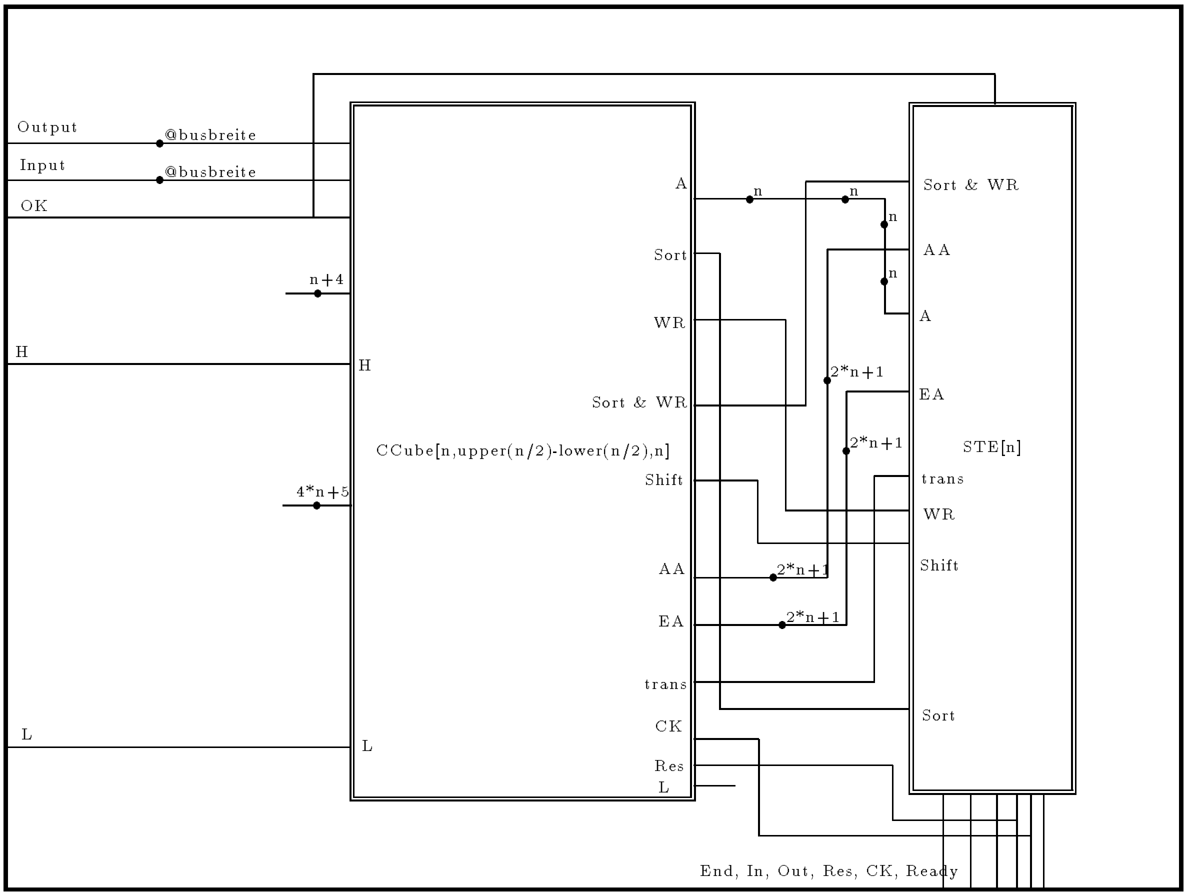
Anhang B

Einige Schaltungen

In diesem Anhang werden die einzelnen Teile der gesamten Sortierschaltung so dargestellt, wie die in *CADIC* [Bur94] entwickelt wurden.

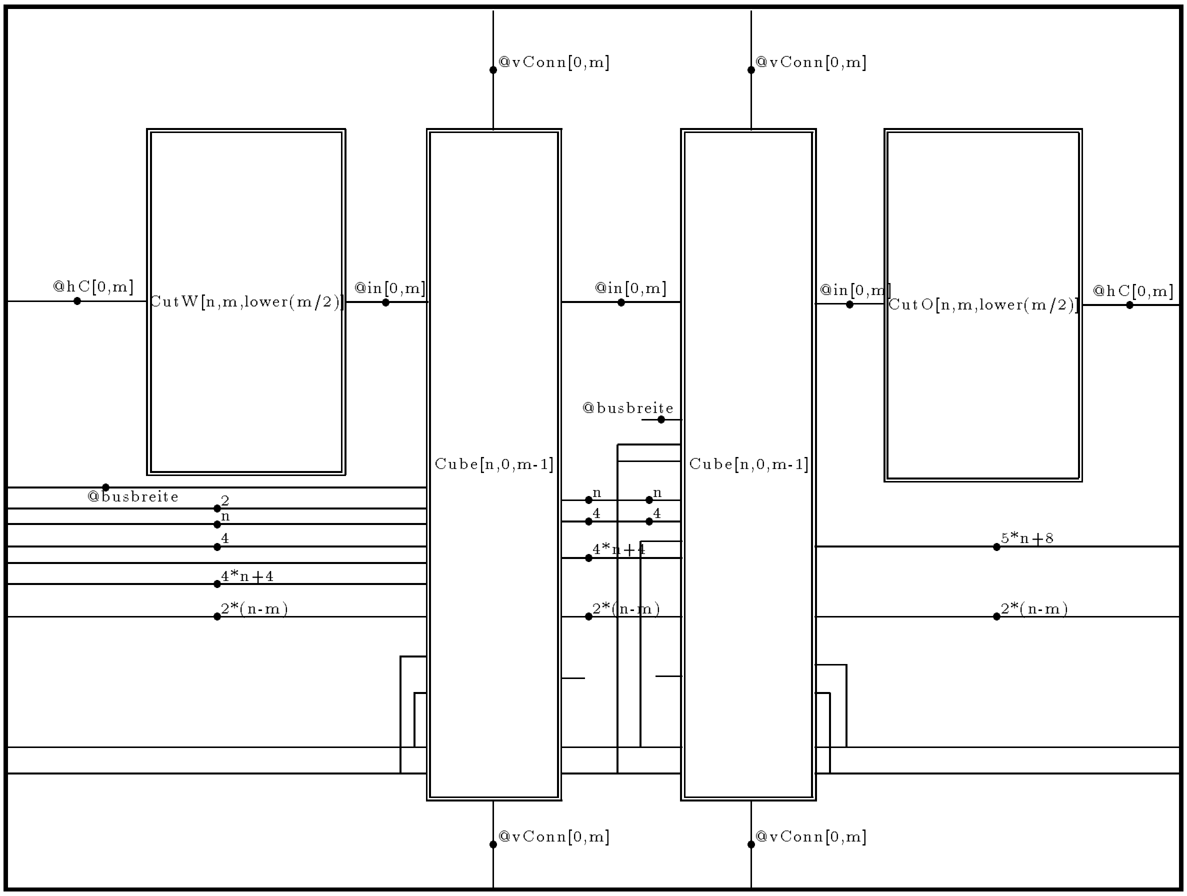
In einigen Fällen stimmt die Darstellung der Schaltungen in diesem Abschnitt nicht mit denen im Text überein, was z.B. in Abb. A1 für die Chipschaltung auf der obersten Hierarchieebene der Fall ist.
Diese Mehrdeutigkeit ist durch die relativ komplizierte Leitungsnetze verursacht worden, die durch die Verbindung der Anschlüsse mehrerer Teilschaltungen entstehen und Für allgemeine Funktionsbeschreibung des Systems unwichtig sind.

Als Grundelemente wurden die CMOS Zellen genommen, deren Liste in Anhang C folgt.



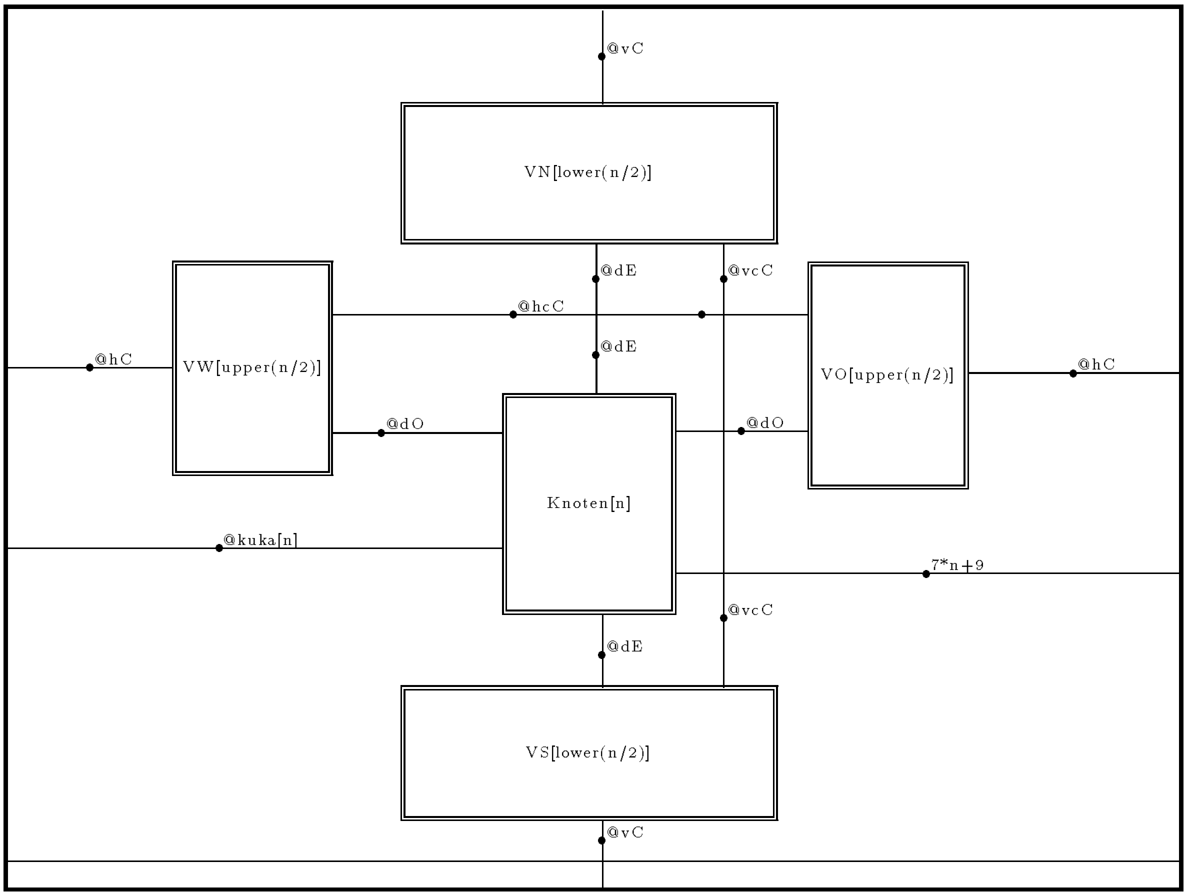
chip

Schaltung $CHIP[n]$ — Das Gesamtsystem auf oberster Chierarchieebene.
Die Leitungen H und L müssen an die $'H'$ und $'L'$ Sugnale angeknüpft werden. Die restlichen Signale sind im Kap. 4 beschrieben.

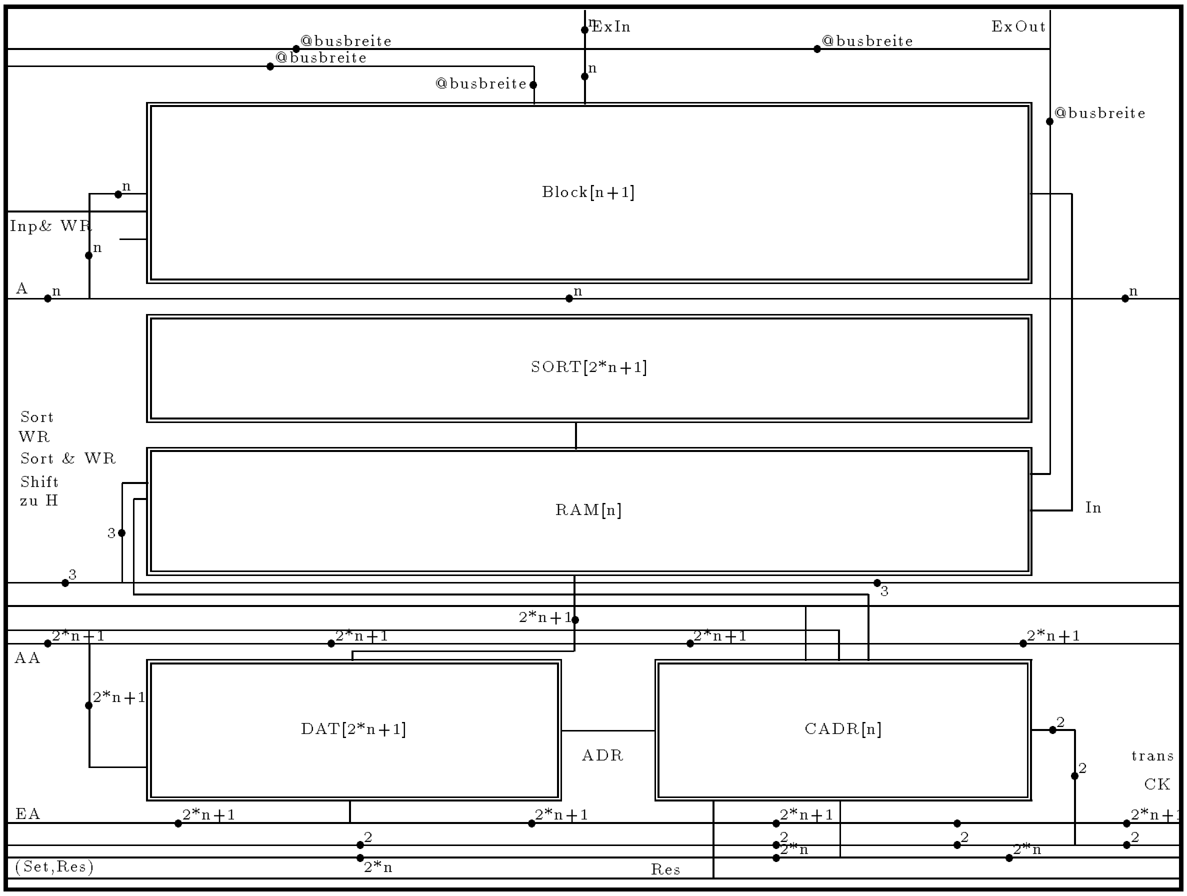


wurfel

Die Würfelkonstruktion im ungeraden Schritt m



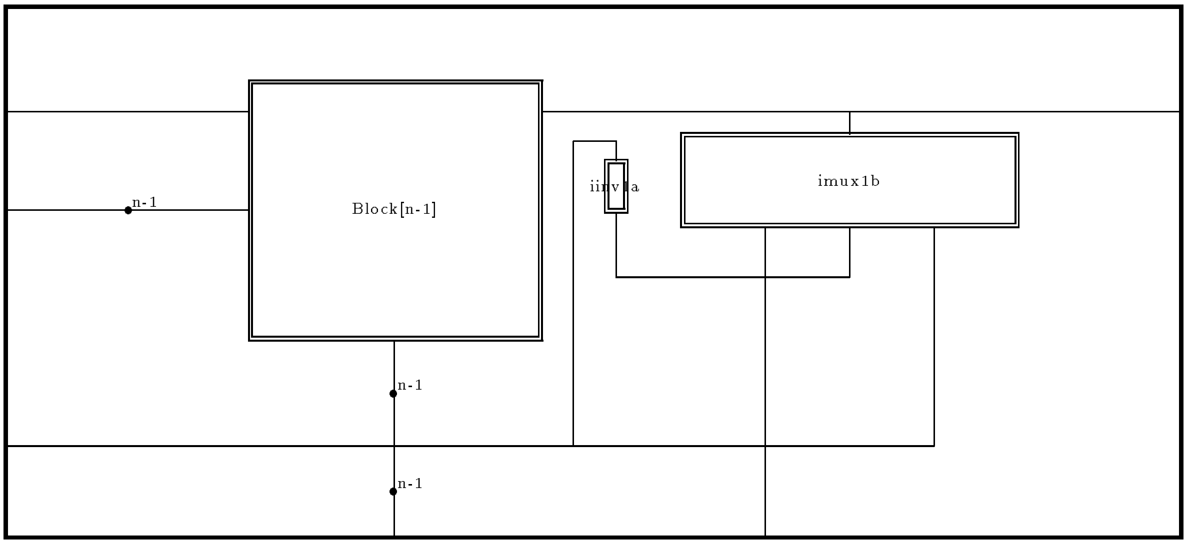
Vorverarbeitung der Dimensionsleitungen

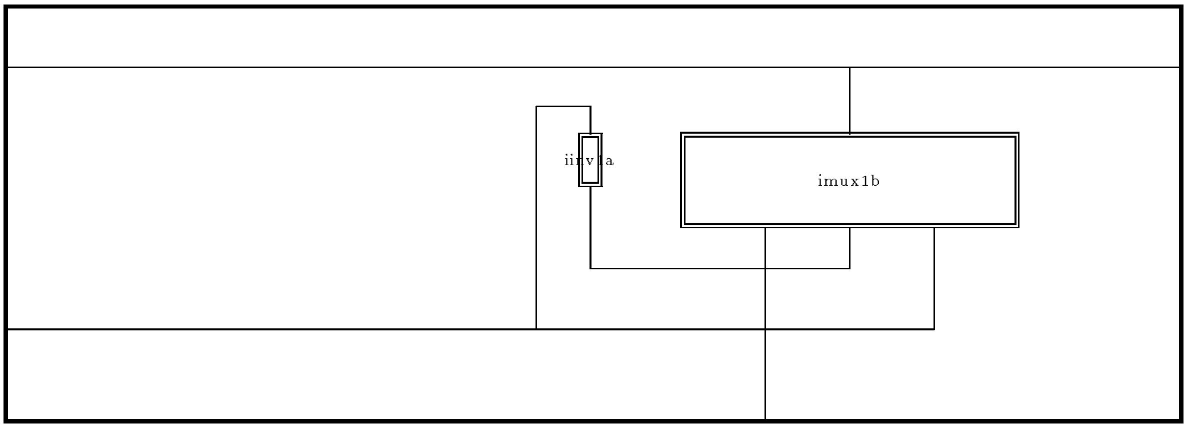


knoten

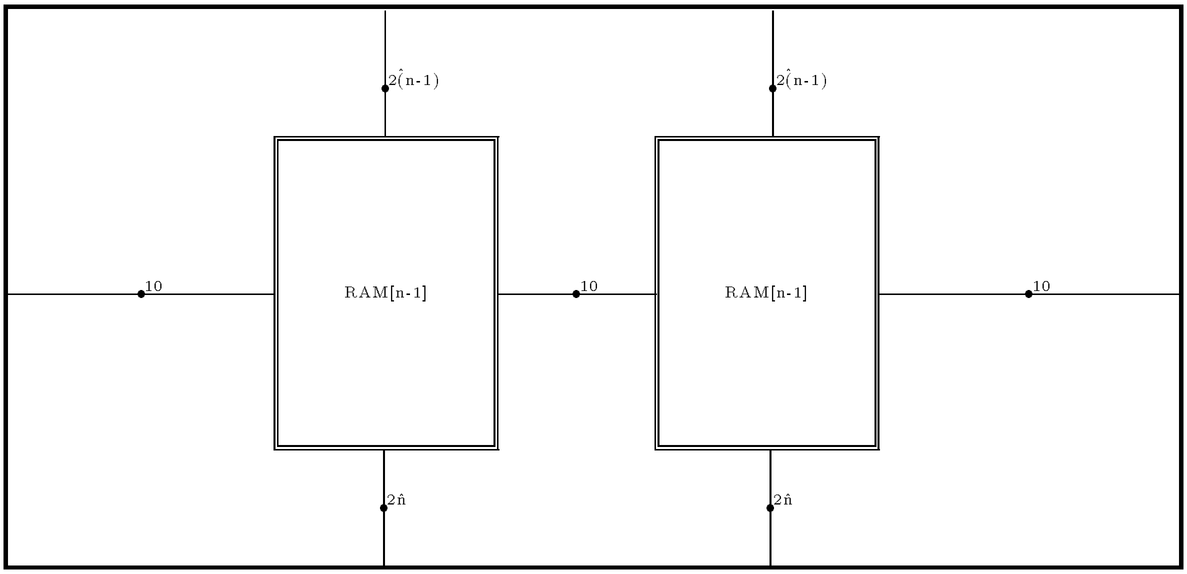
Die Knotenschaltung auf oberster Hierarchieebene

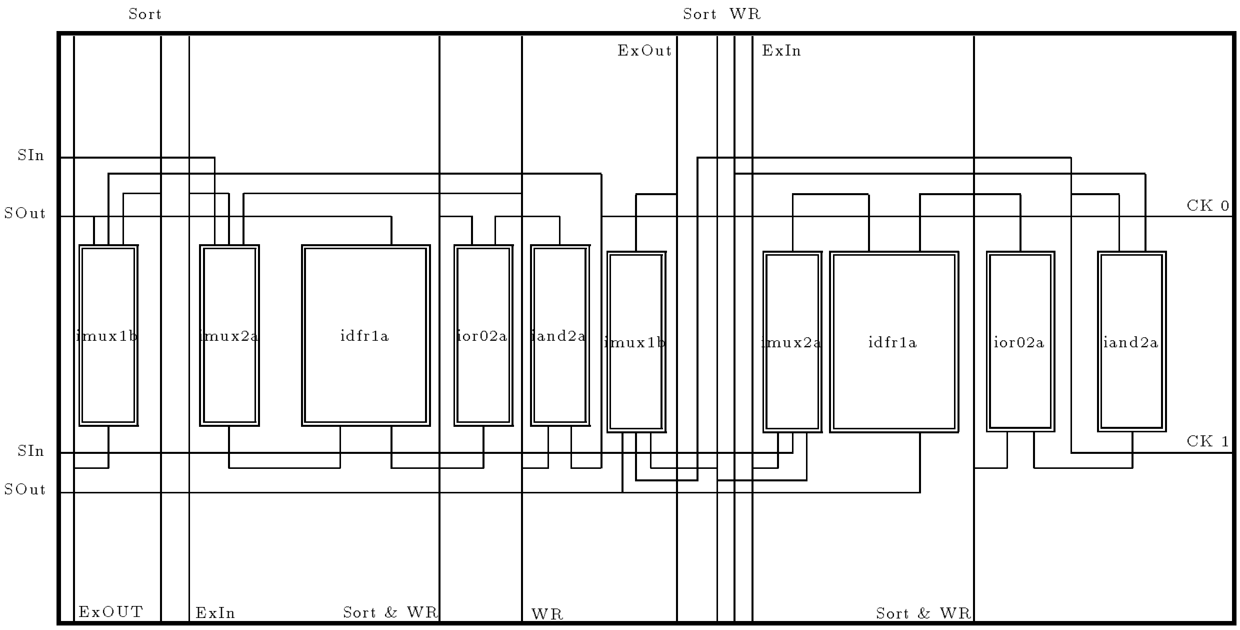
Die Schaltung $CADR[n]$ (Count Address) ias ein Schiftregister, wo die binäre Zellenadresse gespeichert ist. Außerdem setzt sie auch die Variable S_O_N (siehe Kap. 4).



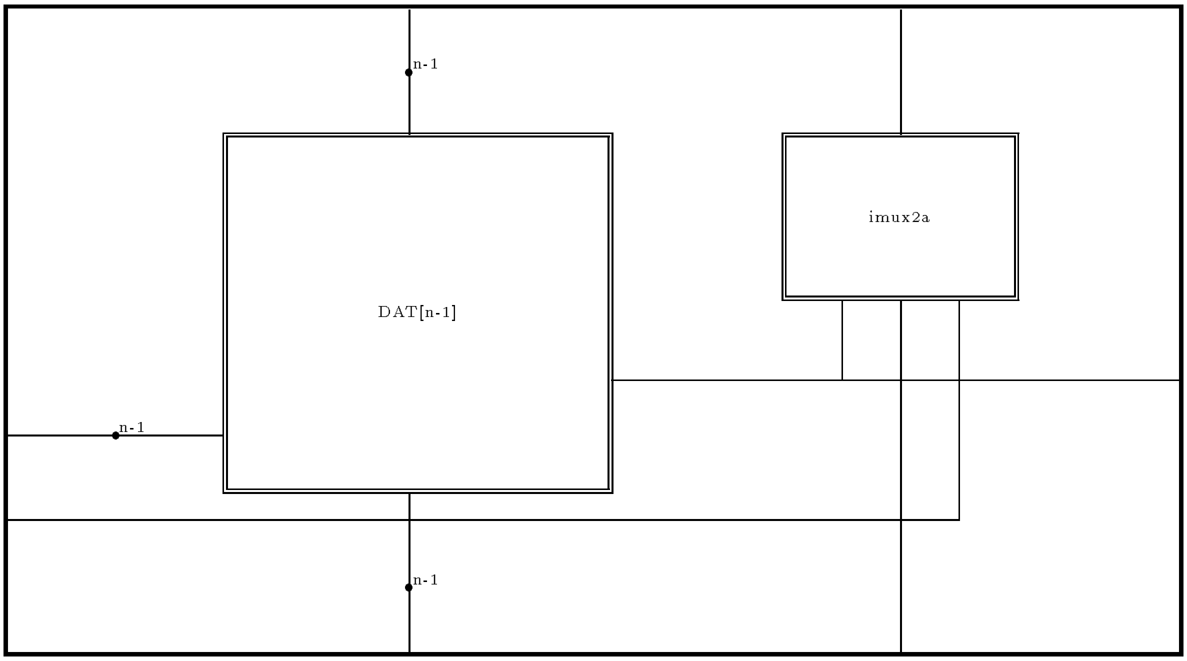


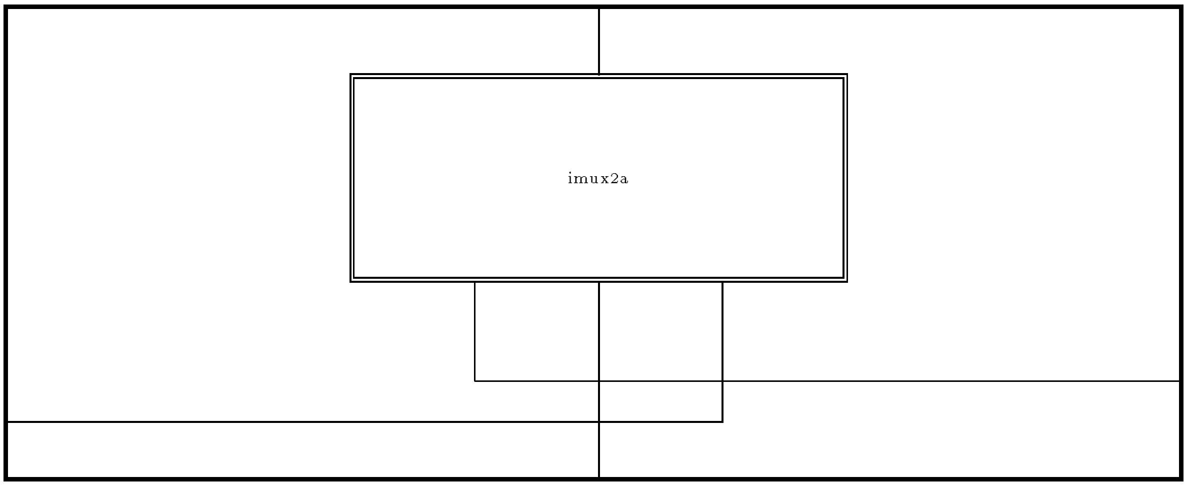
Oben: Hierarchische Darstellung der Schaltung $BLOCK[n]$;
Unten: Abschluss der Hierarchie von $BLOCK[n]$ — $BLOCK[1]$



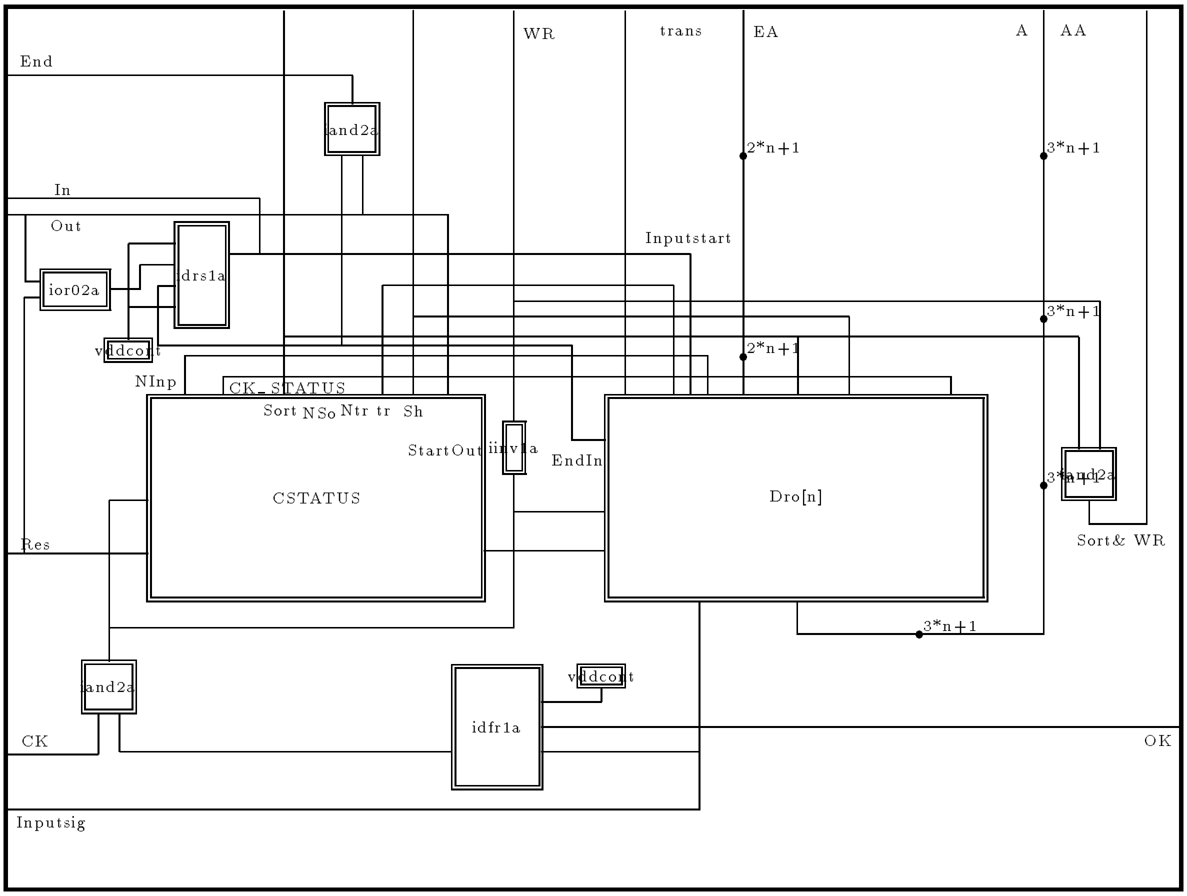


Oben: Hierarchische Honstruktion des Speichers;
Unten: Abschluß der Iteration (zwei Zellen in einem)





Oben: die Schaltung $DAT[n]$ (oben) zur Ausrechnung der aktuellen Zellenadresse mittels EA, AA und $\overline{a}_j$;
Unten: Abschluß der Hierarchie der Schaltung $DAT[n]$ — $DAT[1]$.



steuer

Die Steuerleitung auf der obersten Hierarchieebene

Die Schaltung $CSTATUS$ (Count Status) bestimmt die Zustandsvariablen (siehe Kap. 4);

$DRO[n]$ dagegen berechnet EA, A und AA.

Anhang C

Beschriebene Grundzellen

Zellen	Beschreibung
and2a	Boolsches AND
idfr1a	Flipfiop
iinv1a	Bolsches NOT
imux1b	Multiplexer
imux2a	Multiplexer mit zwei Eingängen
ior2a	Boolsches AND
vddcont	Das 'L' Signal

LITERATURVERZEICHNIS

[Bur94] *Eine graphische Arbeitsumgebung für den parametrisierten Entwurf integrierter Schaltungen*
PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Postfach 15 11 50, 66041 Saarbrücken, 1994

[Knu73] D. E. Knuth. Sorting and Searching, The Art of Computer Programming. Addison — Wesley, 1973.

[KMO89] R. Kolla, P. Molitor, H.– G. Osthoff Einführung in den *VLSI*– Entwurf. Leitfäden und Monographien der Informatik. B. G. Teubner Verlag, Stuttgart, 1989

[Lei85] T. Leighton. Tight Bounds on the Complexity of Parallel Sorting. *IEEE* Transactions on Computers, Vol. C34(4):344-354, April 1985