A Parametrized Sorting System for a Large Set of k- bit Elements

Alexander Gamkrelidze and Thomas Burch

Technical Report A 04/1998

Department of Computer Science, University of Saarland
66041 Saarbrücken, Germany
e- mail: sandro@cs.uni-sb.de burch@cs.uni-sb.de

A Parametrized Sorting System for a Large Set of k- bit Elements*

Alexander Gamkrelidze and Thomas Burch
Department of Computer Science, University of Saarland
66041 Saarbrücken, Germany
e- mail: sandro@cs.uni-sb.de burch@cs.uni-sb.de

Abstract

In this paper, we describe a parametrized sorting system for a large set of k- bit elements. The structure of the system is independent from the problem size (the number of elements to be sorted) and the type of the sorting set (for example, a set of k- bit numbers, an alphabetical list of k- bit words etc.), as well as from the ordering relation defined on the set of the elements (such as ascending or descending order of k- bit numbers, or a specific order of alphabetical words).
The general structure of the underlying parallel network is based on the n- dimensional hypercube. The node circuit construction defines the type of the sorting elements, thus defining the semantics of the system. The structure of the circuit implements the Columnsort algorithm introduced by Leighton in [Lei85]. By changing only one subcircuit of the size $O(k)$ in the node, we can define different ordering relations of the sorted elements. The system is based on specific VLSI chips that were developed in [Gam96] with the CAD system Cadic [Bur95], that has been developed in the project B1 "VLSI design systems and parallelity" under guidance of Prof. G. Hotz.
The result is a fast system that sorts the sets of up to 2^{28} 64- bit numbers. The maximal sorting time is less than 43,6 seconds that is better than some of the fastest software realizations implemented at 32- processor Paragon ([Hard96]), Cray Y – MP ([ZagBlel91]) and MasPar MP – 1 ([BrockWan97]).

Key words: sorting, n- dimensional hypercube, bi-categorial calculus, logic topological net, graphical user interface

I. Introduction

The classical calculus for dealing with logical circuits, the Boolean Algebra, was sufficient as long as the cost of wires were negligible compared with the cost of gates. Since this is no longer the case in integrated circuit design, new calculus has been developed, the bi-categorial calculus [Mol 86], where the design is represented by its logical function as well as by some information about the geometrical arrangement of its components. The first step to extend the Boolean Algebra to the bi-categorial calculus was given by the introduction of x-category by G. Hotz in 1965 [Hot 65] and is described in [HotRe 96].
Consider the circuits layed out into a rectangular R. In order to suppress geometrical and physical details of manufacturing processes and thus to become independent of technology, we forget the width and the layer of wires. In doing so, wires become simple lines which may branch and cross one other. Furthermore we suppose that the circuit is constructed by cells which compute digital values. Assuming that these cells are physically correctly designed, we suppress their internal structure and size and maintain only the order of external connectors on their boundaries. If we consider crossings and branchings of wires also as cells which perform crossings and branchings of signals, this abstraction results in an arrangement of cells in the plane whose interconnections consist of crossing-free non-overlapping lines (Fig. 1).
Now we define for each cell a northern, southern, eastern and western side, on which connectors are placed (note that no connector should belong to a corner). We denote for a cell A the number of connectors onto the northern (southern, eastern and western) side by $N(A)$ ($S(A)$,

*Supported by the DFG, SFB 124 'VLSI design methods and parallelism'

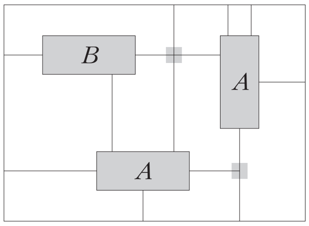


Fig. 1.

E(A), W(A)). To suppress precise geometrical relations of this abstract layout, we consider two such layouts to be equivalent iff they can be transformed into each other by a sequence of deformations that maintain the planar topological structure of the layout. We call a set of nets that can be transformed into each other by a sequence of such deformations, a **logic topological net**, the elements of which are called **topographical representatives** of logic topological net.

The advantage of a logic topological net is that it gives a precise characterization of an integrated circuit, which is sufficiently abstract to suppress geometrical and physical details, and which is sufficiently concrete to control the arrangement of cells and the global routing of wires. A detailed and precise theoretical background of this calculus is given in [Mol 86]. If we relate to each cell its behavior by a boolean function or a more general model, we also get a precise mathematical characterization of the behavior of logical nets ([Kol 87], [Mol 86]).

Having the components, logic topological nets, we can now define the operations between them, namely the compositions of nets which are defined by abutment of topographical representatives. There are two kinds of compositions, namely the horizontal composition $\ominus$ ("left ftom") and the vertical composition Φ ("above"). The composition $N_1 \Phi N_2$ is defined for two nets N_1, N_2 iff there are two topographical representatives of N_1, N_2 so that the southern side of N_1 matches the northern side of N_2. his operation can be carryed out iff $S(N_1)=N(N_2)$.

Consider the construction of a full adder as shown in Fig. 2.

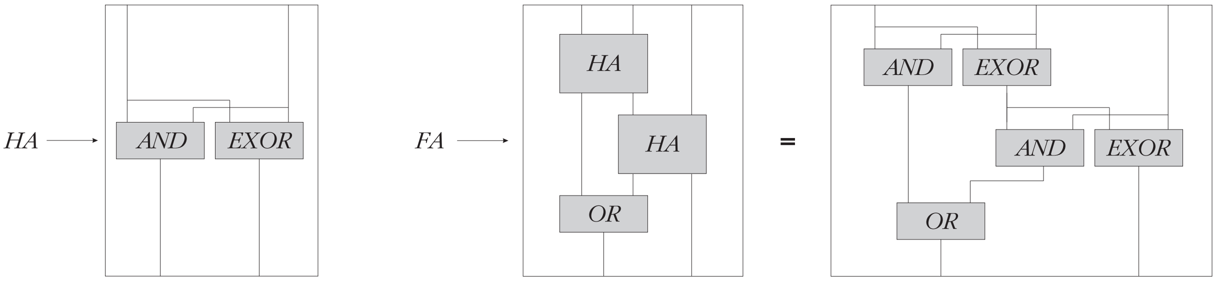


Fig. 2.

In the bi-categorial calculus, it could be described as follows:

$$HA = (\vdash \ominus (- \Phi \ulcorner) \ominus (\neg \Phi +) \ominus \dashv) \Phi (AND \ominus EXOR),\; FA = (HA \ominus |) \Phi (| \ominus HA) \Phi (OR \ominus |).$$

In the following example, we consider the construction of the circuit shown in Fig. 3 (a).

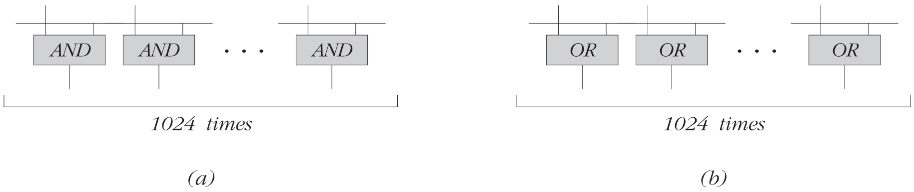


Fig. 3.

In CADIC, the VLSI design system that is based on the bi-categorial calculus, it could be realized as shown in Fig. 4.

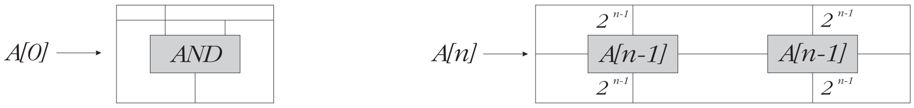


Fig. 4.

Using the bi-categorial calculus, these circuits can be described as follows:

$$A[1] = (+ \ominus \top) \obar AND, \quad A[n] = A[n-1] \ominus A[n-1].$$

The above example shows how compact the hierarchical representations of parametrized circuits could be compared to the traditional methods. To build a circuit of 1024 elements with CADIC, one has to generate the circuit *A[10]*.

Now consider a circuit shown in Fig. 3 (b). This is the same circuit as in Fig. 3 (a), with ANDs changed with ORs. In CADIC, you can build it by substituting only one element: AND with OR in *A[0]*. That means, to change the semantics of the parametrized circuit in hierarchical representation, one has to modify only one element in the design. In non- hieerarchical design, up to 1024 modifications would be necessary.
This advantage pays out in complex systems, such as a sorting system of a large set of elements described in this work. Having a basic structure of an underlying graph (such as an n- dimensional hypercube, $n \times m$ dimensional meshes, a butterfly network etc.), you can implement different parallel divide-and-conquer algorithms constructing the specific node circuits.
As an example of the application of our system, we have constructed a sorting system for a large set of k- bit elements. By changing only one subcircuit of the size $O(k)$ in the design (and by the automatic generation of the whole system with Cadic), we can specify different sorting systems that suits for different purposes, such as sorting a large set of k- bit numbers in ascending or descending order or sorting the alphabetical lists of k- bit entrys in specific order.

As a basic structure in our system, we have chosen an n- dimensional hypercube, that is emerging as a popular network for parallel machines. For example, it is used in the Intel

iPSC, NCube of NCube Corporation and the Connection Machine. One of the key features of a hypercube is a rich interconnection structure which permits many other important network topologies. Furthermore, the hypercube can be divided into subcubes of which the implementation of recursive divide and conquer algorithms is supported.
However, this structure has its disadvantages as well in terms of poor scalability and exponential growth of the circuit. It rises a problem of circuit partitioning — the development of partial circuits on several chips and assembling them as a whole system on a specific board.

The system can be applyed to sort up to 2^{28} 64- bit numbers. The time needed to sort 2^{28} 64- bit numbers is less than 43,6 sec. that is faster than the software realizations implemented at the 32- processor Paragon ([Hard96]), Cray Y – MP ([ZagBlel91]) and MasPar MP – 1 ([BrockWan97]) supercomputers.

This paper is organized in the following manner. Section 2 describes the general topological structure of an n- dimensional hypercube, independent from further implementations of the network semantics. In section 3, the Columnsort algorithm will be introduced; The problems of the efficient implementation of Columnsort at the n- dimensional hypercube are discussed in section 4. Section 5 gives the upper bounds for the growth of time and area complexity of the whole system. Based on the results of section 5, we can realize that for a large number of k- bit elements, the whole system can not be implemented as one chip. That raises the problem of circuit partitioning discussed in section 6, where we give a method of building a sorting system of a large number of k- bit keys. In section 7, we give the runtimes of our system on different problem sizes and compare it with other software realizations of sorting algorithms on different parallel supercomputers. Some of the most important parts of the circuit are introduced in section 8, followed by the layouts of the 3- dimensional hypercube system and its node.

II. Description of the Network

In this section, we give a parametrized description of the hypercube, irrespective of the construction of the vertexes: we describe the topology of the network based on the n- dimensional hypercube, the semantic of which could be specified by the processors inserted into the vertexes.
The n- dimensional hypercube is represented as a graph $G_n = (V_n, E_n)$ where V_n contains 2^n elements. A unique address is corresponded to every node $v \in V_n$:

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

Additionally, every vertex has degree n and every edge between two vertexes corresponds to a dimension d $(0 \leq d < n)$.
Two vertexes $v, v' \in V_n$ with $addr(v) = (a_{n-1}, ..., a_0)$ and $addr(v') = (a'_{n-1}, ..., a'_0)$ are connected via an edge iff

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

We assume that, for the moment, all the connections in the hypercube are of undefined type. The type will be defined later, depending on specific vertex circuit. For example, the edge of type $@t$ could be a k- bus, a single- or bidirectional connector etc. For simplicity we assume that all the edges are of the same type $@t$.
In order to get a compact topological description of the hypercube with short connections between the vertexes, we construct any n- dimensional hypercube by setting two $n-1$ dimensional subcubes aside or above one another and connecting the corresponding vertexes via $n-1$ -dimensional edges: if n is odd, set the subcubes besides, else over one another. An example of the construction of a 3- dimensional hypercube is shown in Fig. 5.

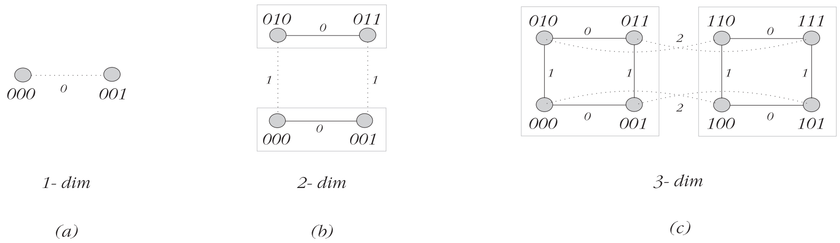


Fig. 5. Construction of a 3- dimensional hypercube

Applying this method, we can recursively describe any n- dimensional hypercube. The recursion terminates on the cube of dimension 0, the vertex of the hypercube (Fig. 6).

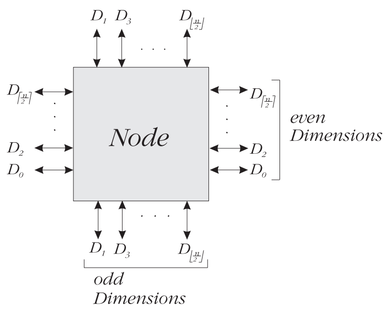


Fig. 6. The vertex of the hypercube

In order to work for any n, the interface of the vertex must intend all the connections for the higher dimensions. To beware the symmetry of the construction, the dimensions are attached by turns on the N/S and E/W sides respectively. Moreover, same dimensions are passed through N/S and E/W sides of the circuit.

Consider another example of the construction of a 3- dimensional hypercube:
Example 1.1: Construction of a 3- dimensional hypercube

- The 1- dimensional subcube is constructed by setting two 0- dimensional subcubes (vertexes) one besides the other and connecting the 0- dimensional edges, cuted (projected) on the eastside (resp. westside) of the nodes.

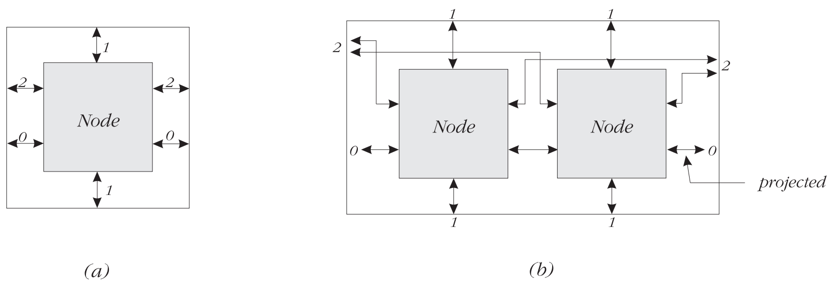


Fig. 7. The vertex *(a)* and the 1– dimensional subcube *(b)*

  In Fig. 7 (b), the edges of dim. 2 are passing the vertexes and are connected to the E/W sides of the subcube of dimension 1, providing the connections in higher steps of the construction.

- To construct a 2- dimensional hypercube, two 1- dimensional subcubes are set one above the other, connected via dimension 1 (Fig. 8).

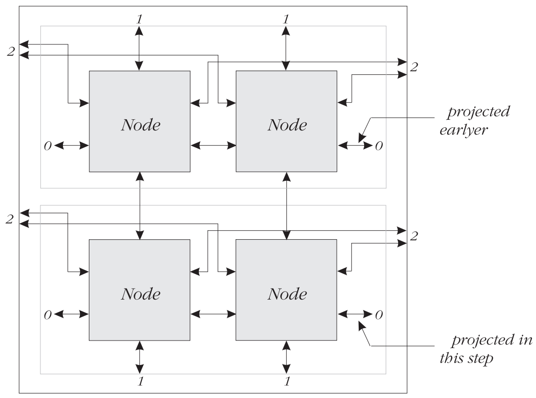


Fig. 8. Construction of a 2– dimensional hypercube

  The edges of dimension 1 on the northside of the upper and the southside of the lower subcubes are cuted (note that the edges of dimension 0 are cuted earlier and do not appear at this level).

- In the last step of the iteration (Fig. 9), there is no direct connection between the vertexes via dimension 2. But still it is possible to connect the 2- dimensional subcubes directly, connecting their west- and eastsides, because of the correct allocation of the edges of dimension 2 in previous steps.

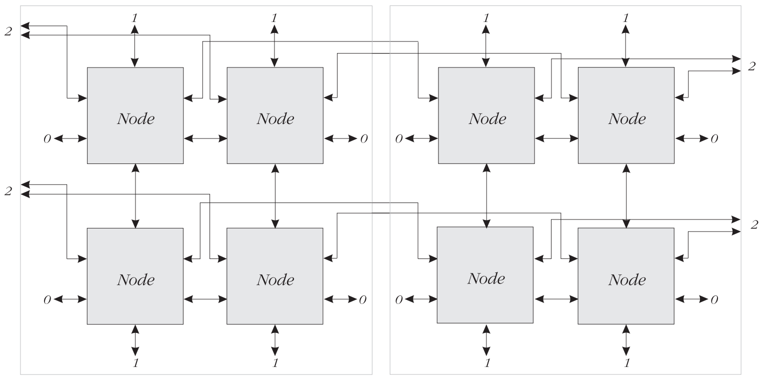
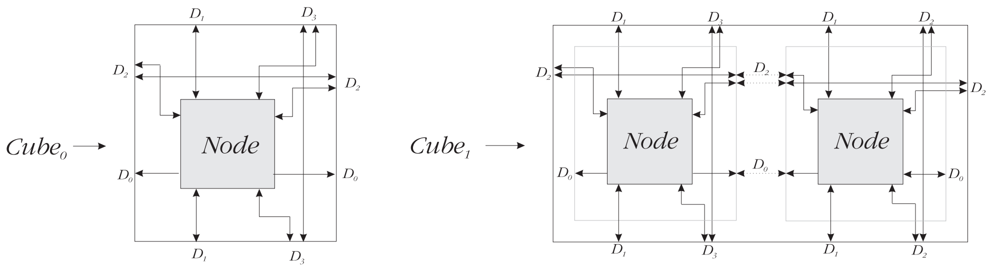


Fig. 9. Construction of a 3– dimensional hypercube

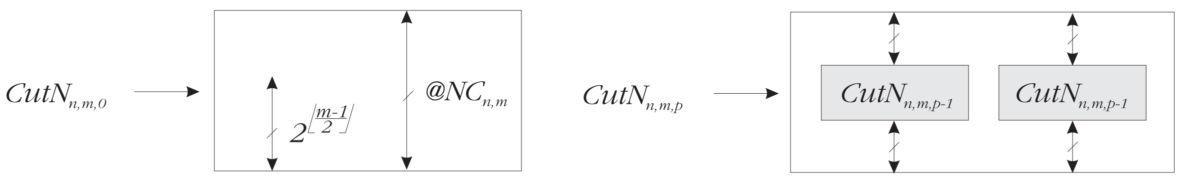
As shown in previous example, in every step of the construction, some edges must bypass the nodes and be connected to the N/S respectively E/W sides of the subcube. Until now, this was done manually, that means, in every step, each edge of this kind is passed to specific node.



(a) (b)

Fig. 10. Preparation of the edges *(a)* and the 1– dimensional subcube *(b)*

If we define the vertex as in Fig. 10a, the process of the construction will be simplified in terms of connecting all the edges at the E/W resp. N/S sides of the subcubes: the edges with higher dimensions will be bypassed automatically (Fig. 10b). Fig. 12 shows a 4- dimensional hypercube constructed with the vertexes defined as in Fig. 10a.
As shown in the examples, some edges must be cuted in the respective steps of the iteration. This is provided by the circuits $CutE$, $CutW$, $CutN$ and $CutS$, defined in Fig. 11.



(a) (b)

Fig. 11. The circuit $Cut_{n,m,p}$

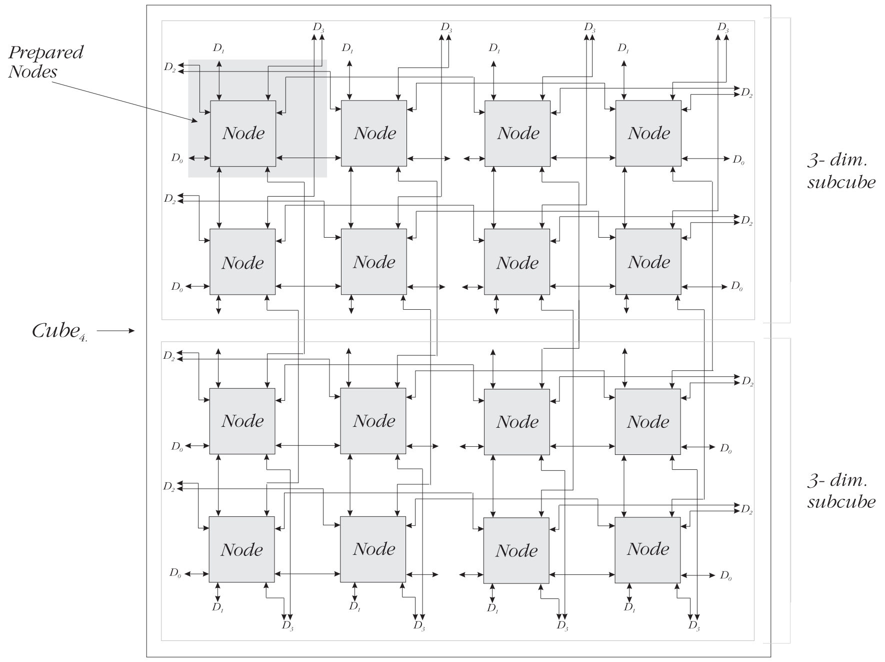


Fig. 12. the four- dimensional hypercube

$CutS$, $CutE$ and $CutW$ are projecting the respective edges on the southern, eastern and western sides of the subcube and are constructed similarly to $CutN$: $CutW$ is the 90^o rotated circuit $CutN$, $CutS$ — vertically mirrored $CutN$, $CutE$ — horizontally mirrored $CutW$. The preparation of the edges of the vertex could be described as in Fig. 13

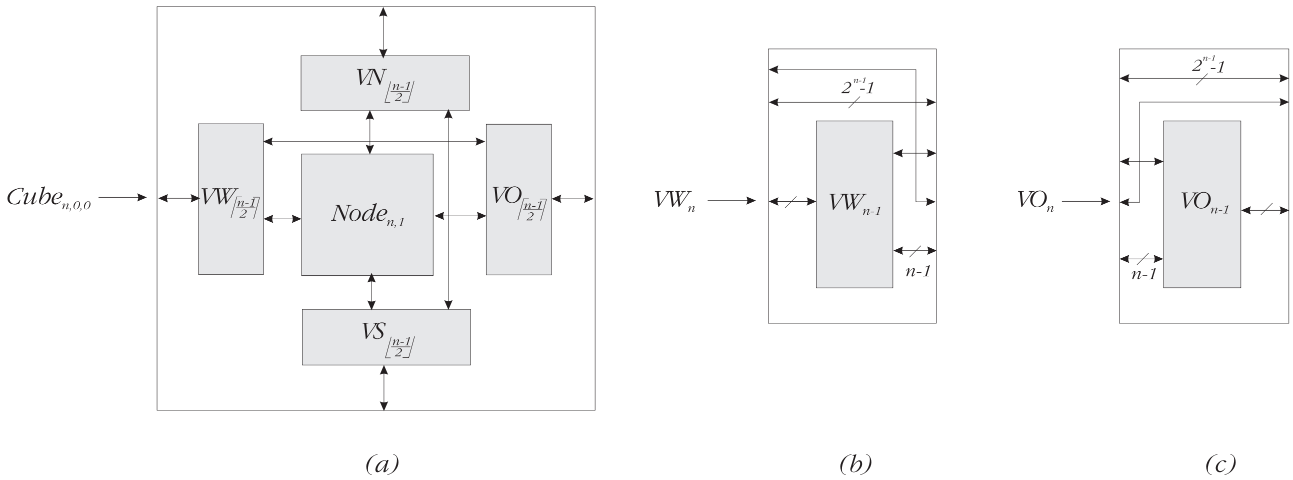


Fig. 13. The edge- preparation circuits

The circuit VN (respectively VS) is the circuit VE (respectively VW), rotated 90^o clockwise.

Using the bi-categorial notation from [KMO 89] (where $\ominus$ means 'set besides', $\oplus$ 'set above'), we can write:

$$Cube_{n,j,k} = \begin{cases} CutW_{n,k,\lfloor\frac{k}{2}\rfloor} \ominus Cube_{n,0,k-1} \ominus Cube_{n,0,k-1} \ominus CutE_{n,k,\lfloor\frac{k}{2}\rfloor}, & \text{for odd } k; \\ CutN_{n,k,\lfloor\frac{k}{2}\rfloor} \oplus Cube_{n,1,k-1} \oplus Cube_{n,1,k-1} \oplus CutS_{n,k,\lfloor\frac{k}{2}\rfloor}, & \text{for even } k. \end{cases}$$

(here and further in the notations of $Cube_{n,j,k}$, $j = \lceil\frac{k}{2}\rceil - \lfloor\frac{k}{2}\rfloor$).

Fig. 14 shows the construction of the subcube in the odd step m.

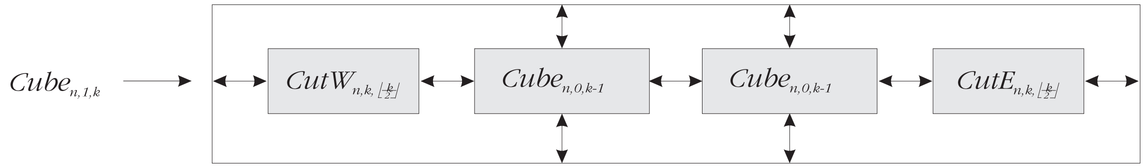


Fig. 14. construction of the subcube in the odd step k

III. Columnsort

In this section, we describe a sorting algorithm *Columnsort*, introduced by Leighton in [Lei 85].

For simplicity, we describe the algorithm as a series of elementary matrix operations.

Let Q be an $r \times s$ matrix of N numbers to be sorted where $r \cdot s = N$, $s|r$, and $r \geq 2(s-1)^2$. Initially, each entry of the matrix is one of the numbers to be sorted. After completion of the algorithm, the $i,\ j$ entry $(0 \leq i \leq r,\ 0 \leq j \leq s)$ of Q will contain the p- th sorted number $(0 \leq p < N)$ where $p = i + j \cdot r$.

As an example, we give a 3×9 matrix before and after sorting:

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

Columnsort works in eight steps. In steps 1, 3, 5 and 7 it sorts the numbers within each column of the matrix. In steps 2, 4, 6 and 8, the entries of the matrix are permuted.

<u>Steps 2 & 4</u> The permutation of the matrix in step 2 corresponds to a 'transpose' of the matrix. The entries are pieced up column by column and then deposited row by row (always going from top to bottom in a column and from left to right in a row). The permutation in step 4 is the inverse of that in step 2.

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

Note that these steps do not match to real transposition if they are not applyed to a square matrix, but, for simplicity, we still call them 'transpose' and 'retranspose'.

<u>Steps 6 & 8</u> The permutation in step 6 corresponds to an $\lfloor \frac{n}{2} \rfloor$- shift of the entries.

In this step, the matrix $r \times s$ is transformed into the matrix $r \times (s+1)$.

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

The permutation in step 8 is the reverse of that in step 6. We call these steps 'shift' and 'reshift'. For short, we call all these steps 'permutation' steps.
The following example shows a step- by- step application of *Columnsort* to a 9×3 matrix:

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

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

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

IV. Implementation of Columnsort

To implement *Columnsort* on the n- dimensional hypercube, we proceed as follows:

- Every vertex of the n- dimensional hypercube corresponds to a column of the matrix Q. That means, the number of columns in Q must be $s = 2^n$. The elements of the columns are stored in the memory of size r of corresponding vertexes;

- Hence $r|s$ and $s \geq 2(r-1)^2$ must hold, we assume $r = 2^{2n+1}$.

That means, we have a $2^{2n+1} \times 2^n$ matrix, where 2^{3n+1} elements will be sorted.

In this section, we represent the method of the implementation of each 'sorting' and 'permutation' steps of *Columnsort*.

The 'sorting' steps are implemented with *OddEvenMergeSort* circuit [Knu 76], contained in each vertex. Since every vertex sorts its data independent from his neighbours, no problems of synchronization arise.

The problems arise while implementing 'permutation' steps. As an example, transform a 4×4 matrix (Fig. 15):

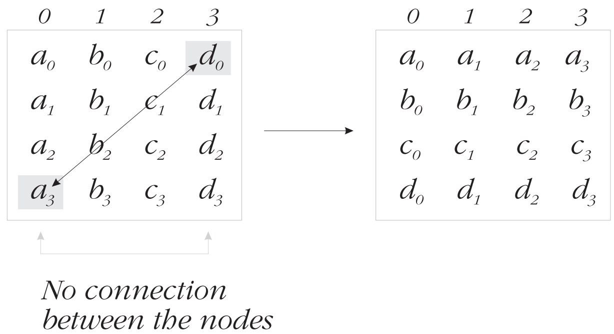


Fig. 15. Connection problems while transforming a matrix

The columns 0 and 3 must share data with one another, but there is no direct connection between them. The solution is to send data along the shortest path to the neighbours.

We apply the following scheme of transformation:
- Transform the matrix in n steps;
- Share data between the $p-1$ -th neighbour via the edge of dimension $p-1$.

In step $p \in \{1, ..., n\}$, the data will be shared between the vertexes with the addresses $(a_0, ..., a_{n-p-2}, 0, a_{n-p}, ..., a_{n-1})$ and $(a_0, ..., a_{n-p-2}, 1, a_{n-p}, ..., a_{n-1})$ via dimension $n-p$ as follows:

The data in memory of each vertex is divided into 2^{p-1} blocks. The vertexes exchange the halfs of each block with one another: the vertex with lower address changes the lower half of the block with the upper half of the corresponded block of the vertex with higher address.

As an example, we transform a 4×4 matrix with this method (Fig. 16):

Step 1: $p = 1$; divide each column in $2^{1-1} = 1$ blocks;
Exchange data via dimension 1 (node 0 with node 2, node 1 with node 3).
Vertexes with lower addresses (0 and 1) change lower halfs with upper halfs of the vertexes with higher addresses (2 and 3).

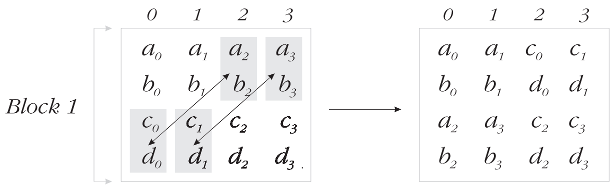


Fig. 16. Step 1 of transposition

Step 2: $p = 2$; divide each column in $2^{2-1} = 2$ blocks;
exchange data as described (Fig. 17)

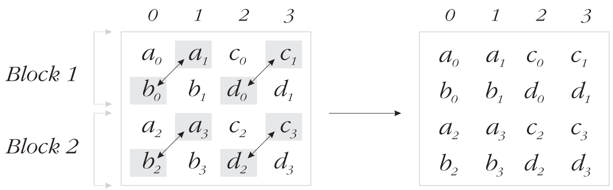


Fig. 17. Step 2 of transposition

The above method transposes a $n \times n$ matrix. To implement it on the $r \times s$ matrix, we apply it step by step on the $s \times s$ partial matrixes.

As an example, consider a 8×4 matrix to be transposed. After applying the above method on the upper and lower partial matrixes, we get

$$\left(\begin{array}{|cccc|}
\hline
a_1 & b_1 & c_1 & d_1 \\
a_2 & b_2 & c_2 & d_2 \\
a_3 & b_3 & c_3 & d_3 \\
a_4 & b_4 & c_4 & d_4 \\
\hline
a_5 & b_5 & c_5 & d_5 \\
a_6 & b_6 & c_6 & d_6 \\
a_7 & b_7 & c_7 & d_7 \\
a_8 & b_8 & c_8 & d_8 \\
\hline
\end{array}\right)
\longrightarrow
\left(\begin{array}{|cccc|}
\hline
a_1 & a_2 & a_3 & a_4 \\
b_1 & b_2 & b_3 & b_4 \\
c_1 & c_2 & c_3 & c_4 \\
d_1 & d_2 & d_3 & d_4 \\
\hline
a_5 & a_6 & a_7 & a_8 \\
b_5 & b_6 & b_7 & b_8 \\
c_5 & c_6 & c_7 & c_8 \\
d_5 & d_6 & d_7 & d_8 \\
\hline
\end{array}\right)$$

This matrix does not correspond the transposed matrix described in section 2, but as one can see, each column of it is a permutation of the corresponded column of *Columnsort*- matrix in step 2. Hence, after applying the sorting algorithm in step 3, we get exactly the same matrix as in *Columnsort*.

To implement the 'retransposition', we apply the above method of $n \times n$ matrix transposition on the whole matrix (Fig. 18).

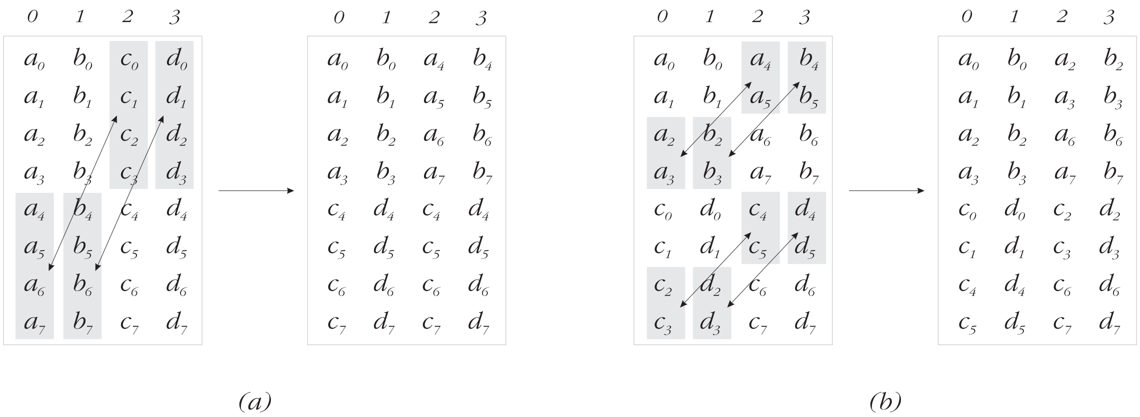


Fig. 18. retrans

Note that, similar to 'transposition', the matrix does not correspond that of *Columnsort*, but after applying 'Sorting' in step 5, we get the same result.

While 'shifting', the method of Fig. 19 will be applied:

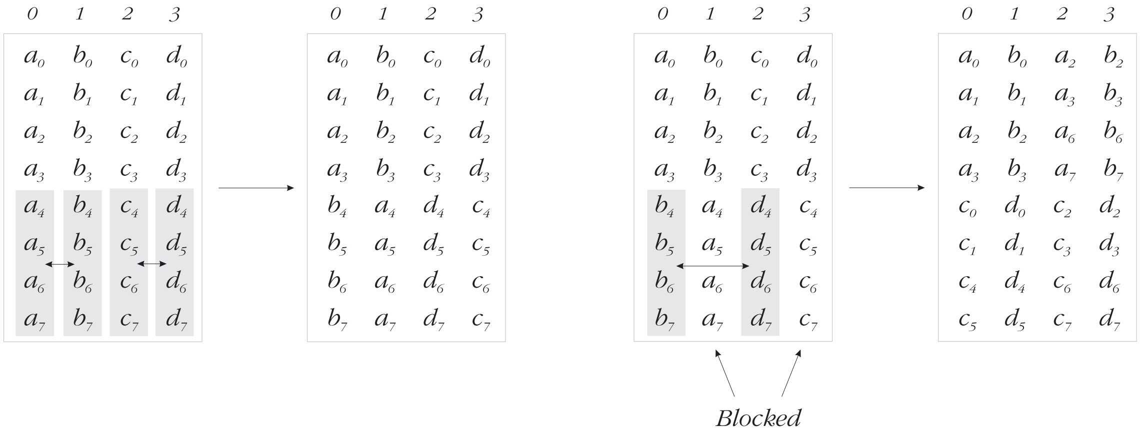


Fig. 19. shift

The nodes share data in ascending order, that means, in step k, they share data with the $k-1$ -th neighbour via edge $k-1$. Note that, in step k, only the nodes with the addresses $(a_{n-1}, ..., a_k, 0, ..., 0)$ share data with their neighbours, the rest of them is blocked (on the exception of step 1, where all the nodes share data with their corresponding neighbours).

The principle of the 'reshift'- step (Fig. 20) is similar to 'shift', with some exceptions:

- The nodes share the upper halfs of their data with their neighbours (on exception of node 0, that shares the lower half of its data with the upper half of its neighbours);
- Only the nodes with the addresses $(a_{n-1}, ..., a_k, 1, ..., 1)$ share data, the rest is blocked.

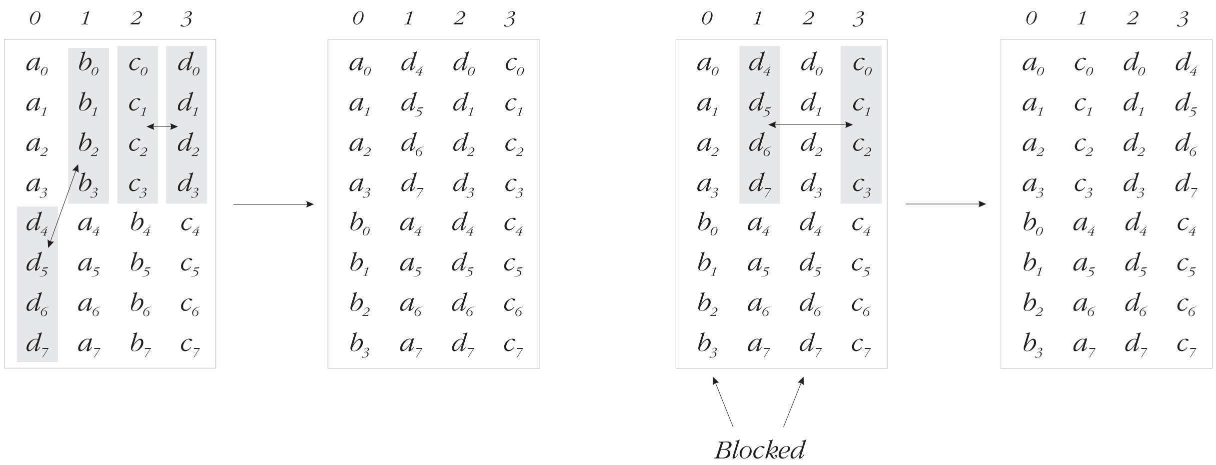


Fig. 20.

There is one important thing to be noted: normally, the elements in each node must be sorted. In node 0 of the 'reshifted' matrix , however, the upper and lower parts are exchanged. This is considered in the data output algorithm described later in this work, so that the user gets the sorted elements in ascending order.

Input and Output

The Input / Output algorithm is realized as follows:
For a 1- dimensional cube:
1. Write (read) data to (from) the node 0;
2. Exchange data between the nodes 0 and 1;
3. Execute step 1

This process could be generalized for an n -dimensional hypercube:
1. Write (read) data to (from) the lower $n-1$ dimensional subcube;
2. Exchange data between the $n-1$ dimensional subcubes;
3. Execute step 1

Note that in step 1, no changes must be undertaken in the higher $n-1$ dimensional subcube.

V. Area and Time Complexity

In this section, we give the upper bounds of the growth functions for time (the depth) and area (number of cells) of the sorting circuit discussed in previous sections.
Because of the specific construction, the vertexes of the hypercube contain the memory cells, 2^{2n+1} each. It has the negative effect in terms of the area expansion, but the alternative construction would require at least $2^n \cdot k$ pins (the connection of each vertex to memory) or the system should be realized sequentially (we assume that each number to be sorted consists of k bits).

A. Time Complexity of the Circuit

First we give the time function of the Odd-Even-Merge-Sort, shown in Fig. 21:

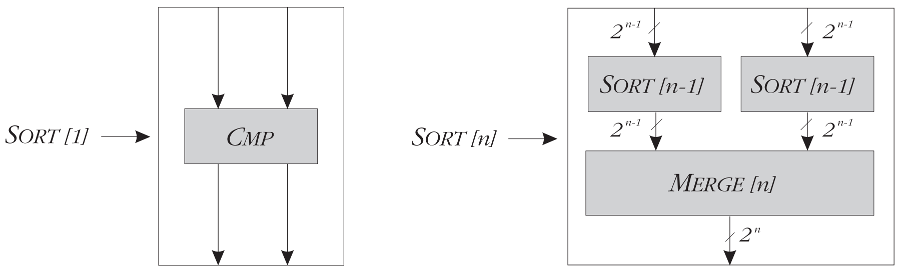


Fig. 21.

The circuit $Merge[n]$ (Fig. 22) sorts two pre- sorted sequences of elements. Its components are explained in Fig. 23.

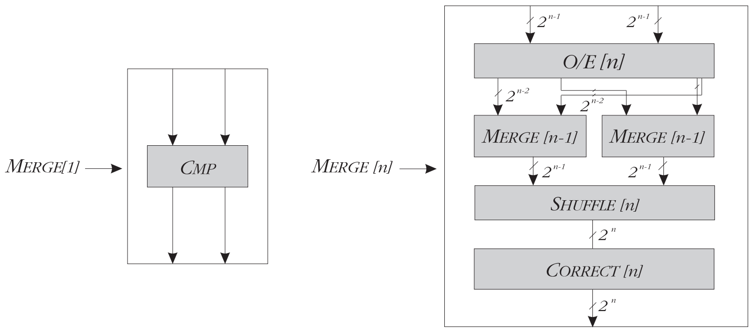


Fig. 22.

The circuit $Shuffle[n]$ is the reverse of $O/E[n]$, CMP sorts two elements of its input.

As one can easily see, the depth (i. e. time) of $Sort[n]$ could be calculated with the following formula:

$$T(Sort_n) = \frac{n \cdot (n+1)}{2}.$$

For $Sort_{2n+1}$, it is

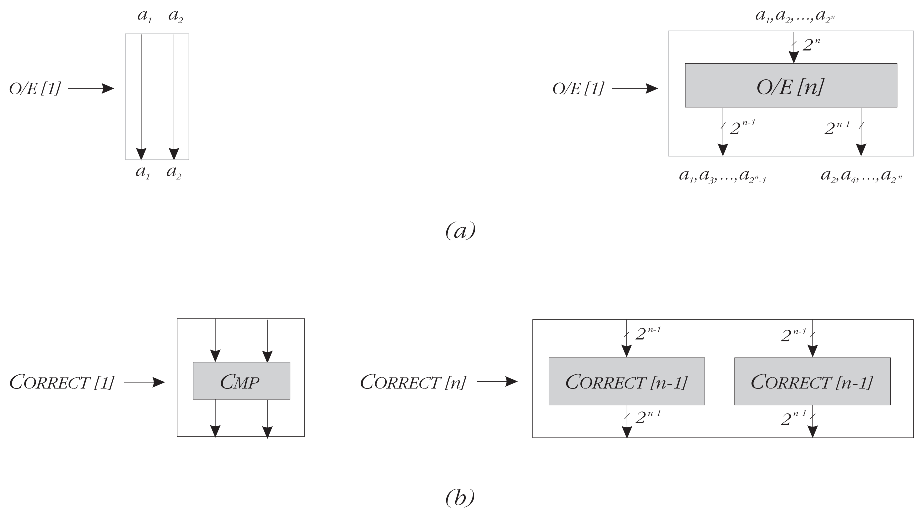


Fig. 23.

$$T(Sort_{2n+1}) = \frac{(2n+1)(2n+2)}{2} = (2n+1)(n+1) = 2n^2 + 3n + 1.$$

The upper bound of it is $O(2n^2 + 3n + 1) = O(n^2)$.
For N elements to be sorted, the upper bound would be $O(\log^2 N)$.

Let $Cube_n$ be our sorting circuit, realized as the n- dimensional hypercube. Than the time function would be:

$$T(Chip_n) = 4 \cdot n \cdot 2^{2n+1} + 4 \cdot T(Sort_{2n+1})$$

Its upper bound is $O(4 \cdot (n \cdot 2^{2n+1} + 2n^2 + 3n + 1)) = O(n \cdot 2^{2n})$.
That means, the circuit sorts $N = 2^{3n+1}$ elements in time $O(\sqrt[3]{N^2} \cdot \log N)$.

B. The Area Complexity

As described in [Gam 96], the circuit $Chip_n$ consists of two parts — the n- dimensional hypercube and the logic unit. Hence, the area function $C(Chip_n)$ could be represented as

$$C(Chip_n) = C(Cube_{n,j,n}) + C(LU_n),$$

where LU_n is the logic unit of the system.

It is also shown in [Gam 96], that

$$C(ST_n) = 19 \cdot n + 33 \text{ and } C(Cube_{n,j,n}) = 2^n \cdot (2^{2n+1} \cdot (2n^2 + 5n + 7) + 6n + 3).$$

It follows:

$$O(C(Chip_n)) = O(n^2 \cdot 2^{3n}).$$

That means, the upper bound of the area of the network that sorts $N = 2^{3n+1}$ elements is

$$O(N \cdot \log^2 N).$$

VI. Circuit Partitioning

Because of the exponential growth of the network, the circuit could be unrealizable for sufficiently large n.

<u>*Definition 5.1:*</u> Let $Cube_{n,j,n}$ be a hypercube to be constructed. A subcube $Cube_{n,j,m}$ is called a maximal subcube of n- dimensional hypercube, if following holds:

$$\begin{array}{ll} Cube_{n,j,m} & \text{is realizable as one chip;} \\ Cube_{n,1-j,m+1} & \text{is not realizable as a chip.} \end{array}$$

One method of the construction of large hypercubes could be a construction of their maximal subcubes and assembling them to a hypercube. Fig. 24 shows a construction of a 7- dimensional hypercube with four 5- dimensional subcubes.

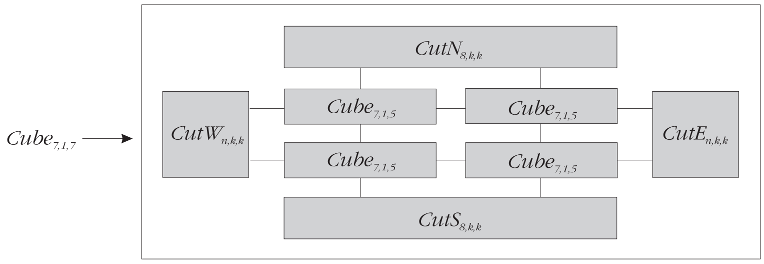


Fig. 24.

The closer look at the system shows the impossibility of this method. As an example, we construct a 7- dimensional hypercube from four 5- dimensional subcubes. First we count the number of connections on the E/W and N/S sides of these subcubes.
As shown in [Gam 96], the number of pins of $Cube_{n,j,m}$ could be calculated as follows:

$$P(Cube_{n,j,m}) = 2^{\lfloor \frac{n}{2} \rfloor + \lceil \frac{m}{2} \rceil + 1} + 2^{\lceil \frac{n}{2} \rceil + \lfloor \frac{m}{2} \rfloor + 1} - 2^{m+1} + 7 \cdot n - 2 \cdot m + 9.$$

In our case it is:

$$P(Cube_{7,1,5}) = 2^7 + 2^7 - 2^6 + 49 - 10 + 9 = 240.$$

In a 1- bit model, the number of pins (corresponding to the number of connections) of the 5- dimensional subcube chip is at least 240. That means, $Cube_{7,1,5}$ is not realizable because of the large number of its pins.
Hence the only problem is the large number of pins, we can solve it by reducing the number of connections in $Cube_{n,0,0}$.

To avoid the above problems, we proceed as follows:
We construct a maximal hypercube (for a 64- bit sorting system, it is a 4- dimensional hypercube).
A 4- dimensional hypercube sorts 2^{13} elements, that means, it could be used as a node of a 6- dimensional hypercube. Figure 25 shows a possible scheme for a construction of a 6- dimensional hypercube system. Each chip is observed as a node of a 6- dimensional hypercube, sorting its data independent from one another and sharing it with its neighbors according to Columnsort. The data paths connecting the nodes of the cube with one another or with the

outside world are set with additional circuits which we denote as C. It is not realized as one chip because of a large number of pins, but, for simplicity, we still represent it as a unit.
In other words, we develop the same system as in previous chapters that do not fit in one chip and is placed on a specific board.

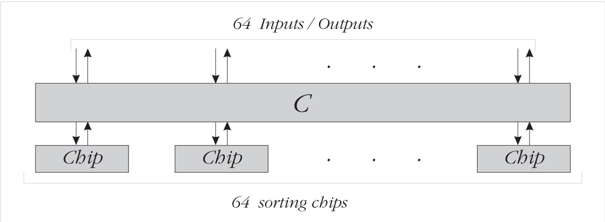


Fig. 25.

The system can be even enlarged, if we use a 6- dimensional hypercube as a vertex of a hypercube with a dimension up to 9 (up to 2^{28} 64- bit elements could be sorted). We describe the method, constructing an n- dimensional hypercube (see Fig. 26).

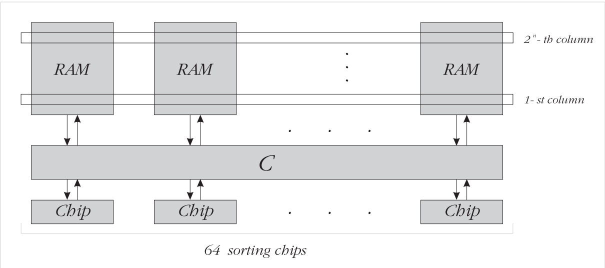


Fig. 26.

For the sake of parallelization, we use at least 64 RAM- blocks (because of the same number of sorting chips), 2^{28} 64- bit words in total. These RAMs build the columns of the Columnsort matrix as shown in Fig. 27.

The general algorithm is implemented as follows:

```
for (i = 0, 4, +i)
    {
     for (j = 0, 2^n, ++j)
         {
         Read j- th column into the system;
         Sort the red- in data;
         Write sorted data;
         }
     /* all 2^n columns of the matrix are sorted */
     Exchange data between the columns according to Columnsort;
     }
```

In other words, we read the data of each RAM column, sort it and store it back into RAM. Then we exchange data between the nodes according to Columnsort (note that some steps in this scheme,such as data exchange and read/write could be parallelized).

VII. Performance Results

In this section, we give the sorting times of the whole system and compare it with other software realizations on several parallel supercomputers.
Our calculations are based on the implementation algorithm from previous section.
The system could be even accelerated by the parallelization of the data read/write and exchange steps:

- write the sorted data in two columns that must exchange data with one another;
- while exchanging data between these two columns, write data to another pare of columns.

The parallelization requires exact analysis based on the timing diagrams of the selected RAM modules, so we do not discuss it in depth in this paper and present the sorting times of the unparallelized system.

The total sorting time of a non- parallelized $n-$ dimensional system can be expressed wit the following formula:

$$T_n < 4 \cdot 2^n \cdot (T_R + T_{sort} + T_W) + 4 \cdot 2^{n-1} \cdot T_{ex},$$

where $T_R = T_R = 2^{13}$ are times needed to read/write data from/to sorting chips, T_{sort} is the time needed to sort data in a 6- dimensional system and T_{ex} is the time needed to exchange data between the nodes of an n dimensional hypercube (the system to be developed).

It is easy to show that

$$T_{sort} = 4 \cdot T_s + 4 \cdot 6 \cdot 2^{12} = 4 \cdot 3 \cdot 2^{11} + 4 \cdot 6 \cdot 2^{12} = 15 \cdot 2^{13},$$

where $T_s = 3 \cdot 2^{11}$ is the sorting time of one sorting chip.

For T_{ex} we have $T_{ex} = n \cdot 4 \cdot 2^{12}$ (2^{12} elements exchanged in n steps).

Applying these formulae, we get:

$$T_n < 4 \cdot 2^n \cdot (15 \cdot 2^{13} + 2^{14}) + 4 \cdot 2^{n-1} \cdot n \cdot 4 \cdot 2^{12} = 2^{n+15} \cdot (n + 17)$$

So, we have estimated the sorting time of the whole system:

$$\boxed{T_n < 2^{n+15} \cdot (n + 17)}$$

The Following table shows the number of elements and sorting times of various sorting systems (for time calculations, we take the clock delation of 100ns).

n	elements	time / sec
1	2^{20}=1.048.576	0,1179648
2	2^{21}=2.097.152	0,2490368
3	2^{22}=4.194.304	0,5242880
4	2^{23}=8.338.608	1,1010048
5	2^{24}=16.777.216	2,3068672
6	2^{25}=33.554.432	4,8234496
7	2^{26}=67.108.864	10,0663296
8	2^{27}=134.217.728	20,9715200
9	2^{28}=268.435.456	43,6207616

In the following diagram, we compare the sorting time of our system with that of Bitonicsort and Samplesort algorithm realizations on the 1024 processor Partysec GCel [Dea] (we have chosen the fastest implementation from the software realizations on the 32- processor Paragon [Hard96], Cray Y – MP [ZagBlel91], MasPar MP – 1 [BrockWan97] and 1024 processor Partysec GCel [Dea]).

Because of the assumption of Columnsort (section III), to sort $2^n - l$ elements $(1 \leq l < 2^n - 1)$, we build a sorting system of the size 2^n, that explains the discrete time graph in the diagram.

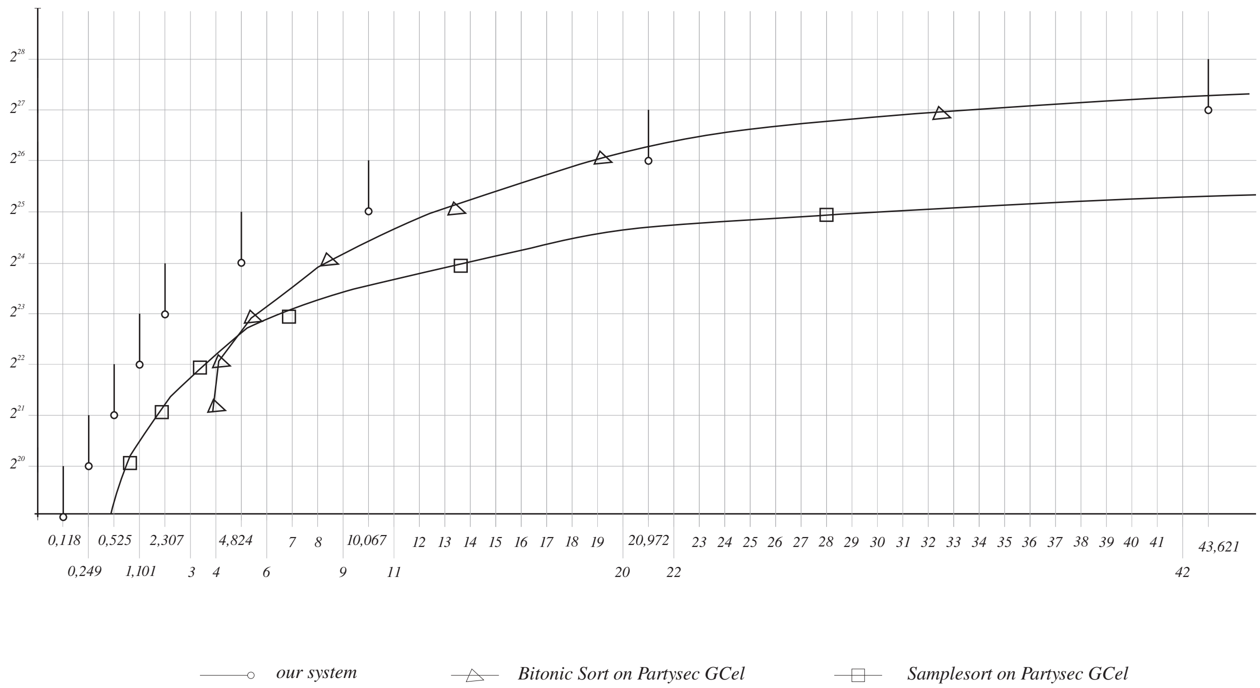


Fig. 27.

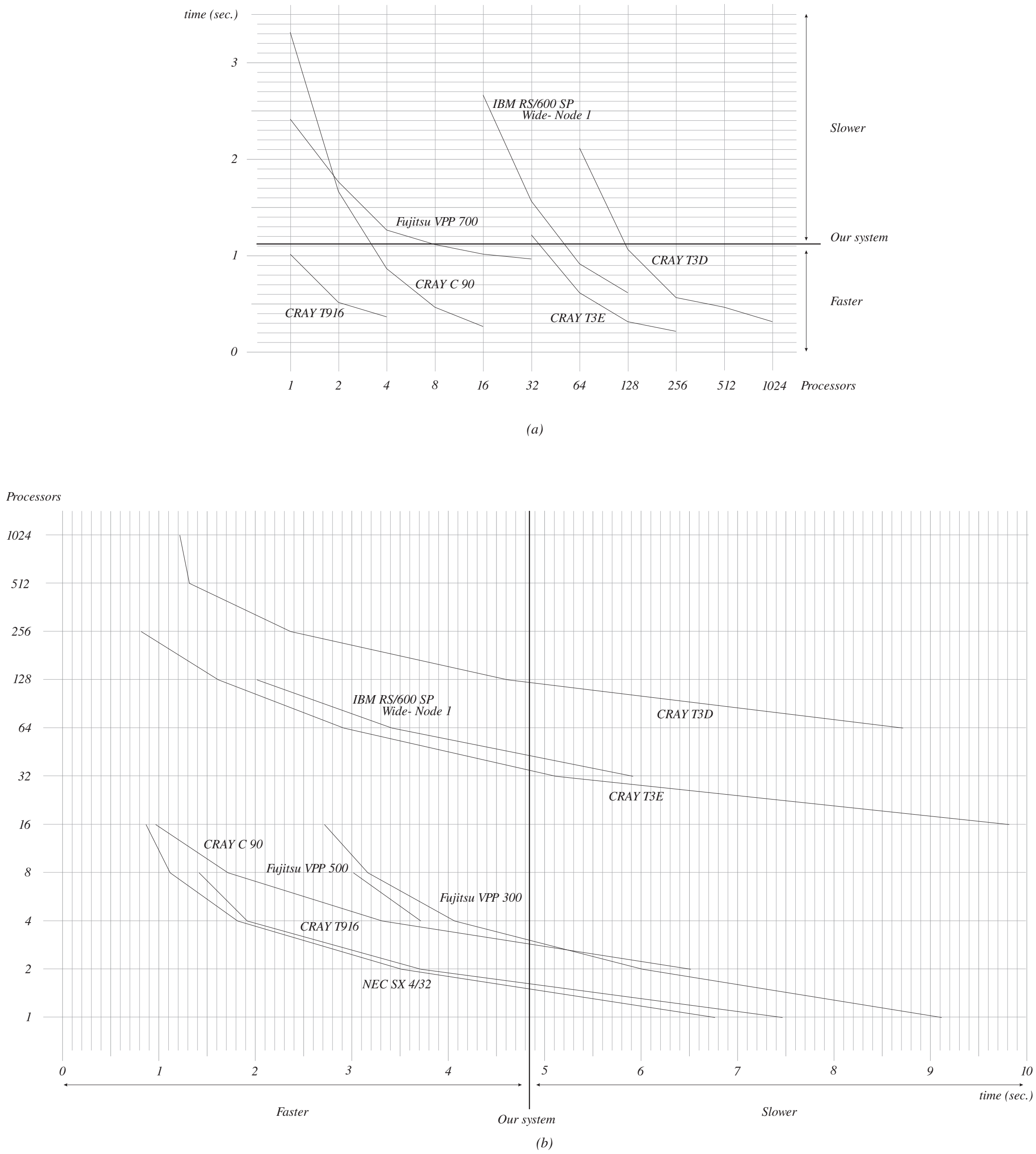


Fig. 28.

According to the benchmark results from the NASA Ames Research Center [BBDS 94], we can compare our system with the realizations of the Integer Sort algorithm on different supercomputers of the problem sizes 2^{23} (Fig. 28(a)) and 2^{25} (Fig. 28(b)).

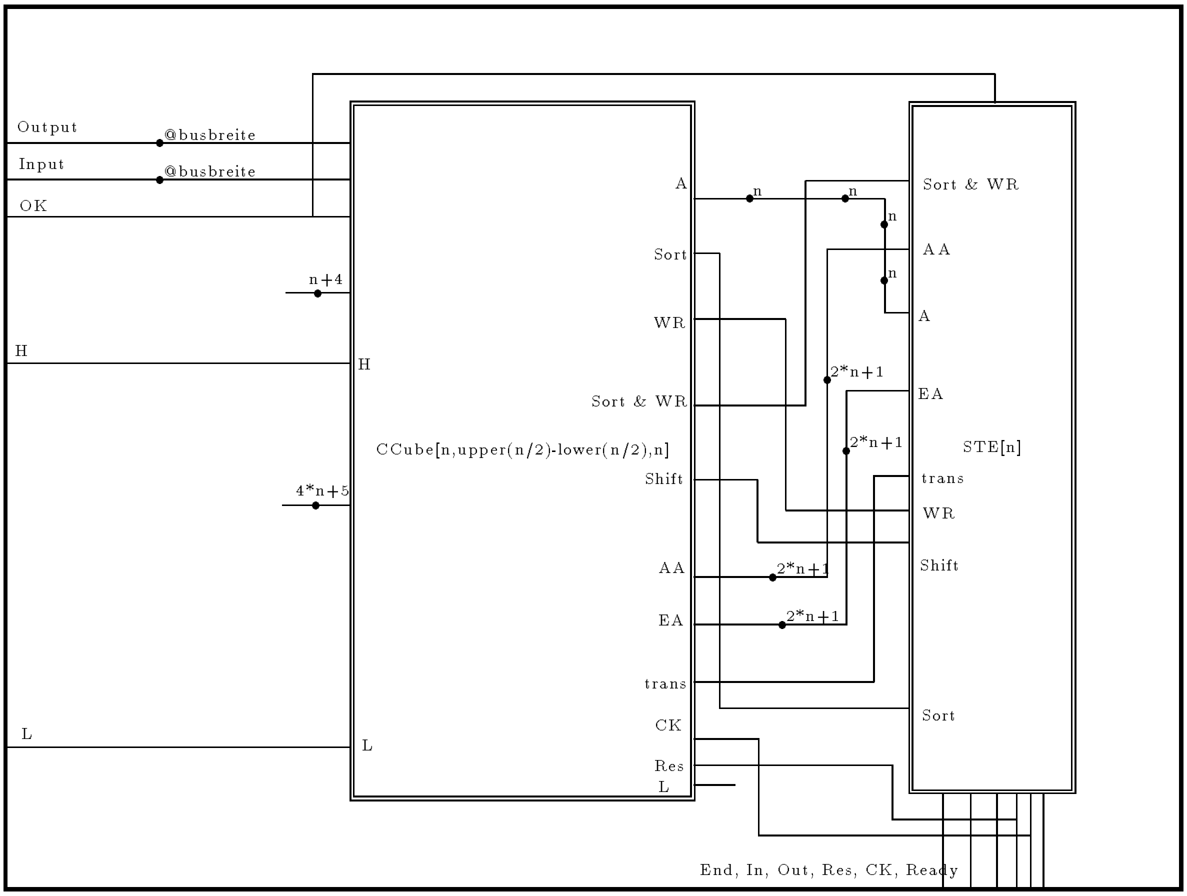
VIII. Important Circuits

In this section, we describe some of the most important circuits of the sorting chip as developed in CADIC [Bur 94].

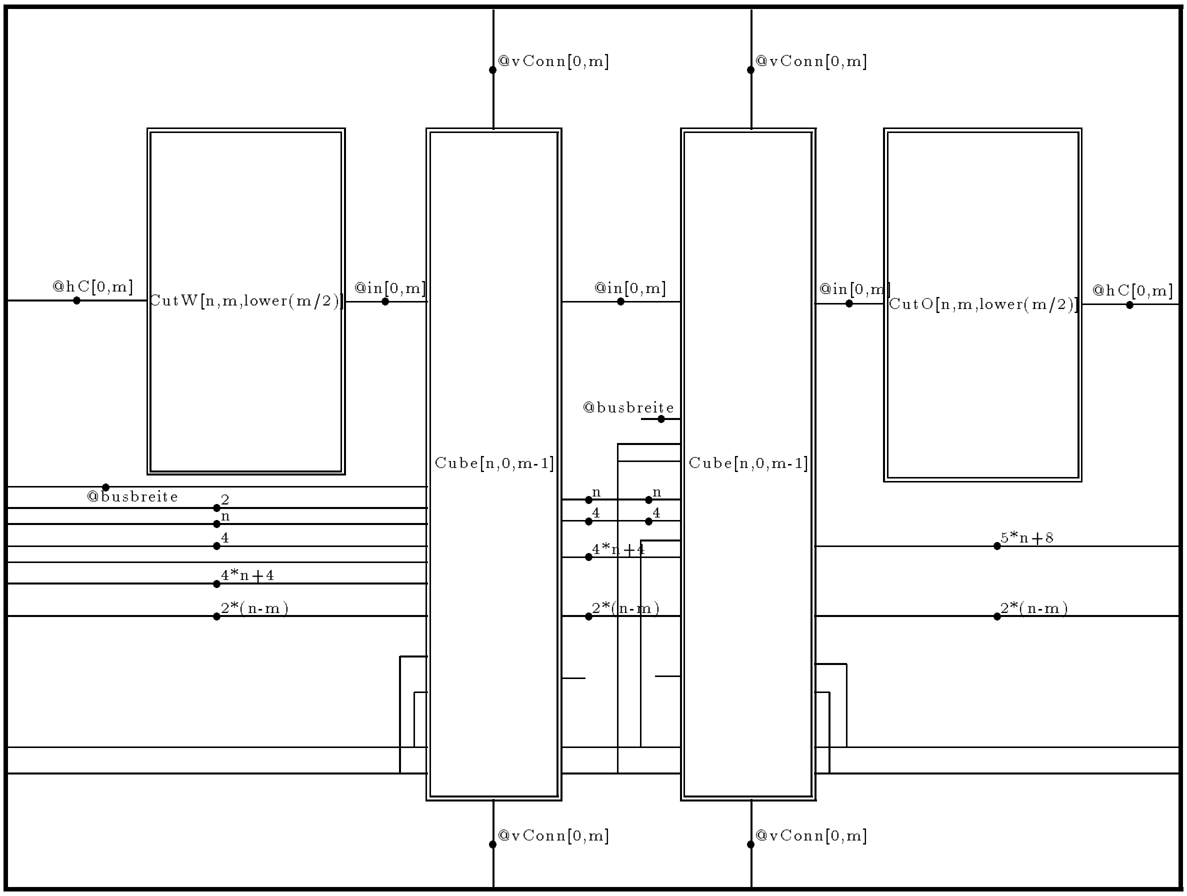
In several cases, the diagrams do not match the circuits analyzed earlier because of the complicated wiring system.

As basic elements, we use the CMOS cells described in the following table:

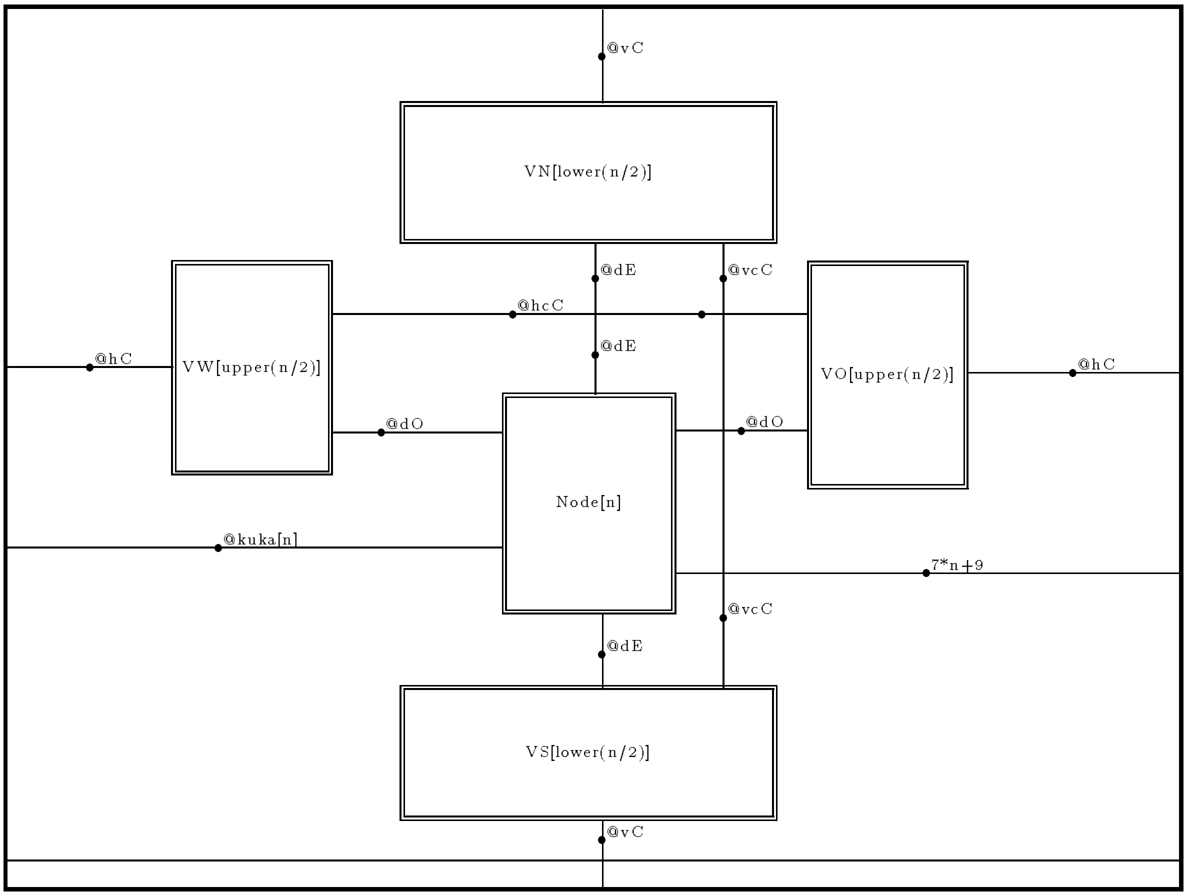
Cells	Description
and2a	Boolean AND
idfr1a	Flipfiop
iinv1a	Boolean NOT
imux1b	Multiplexer
imux2a	Multiplexer with two inputs
ior2a	Boolean OR
vddcont	The 'Lo' Sygnal



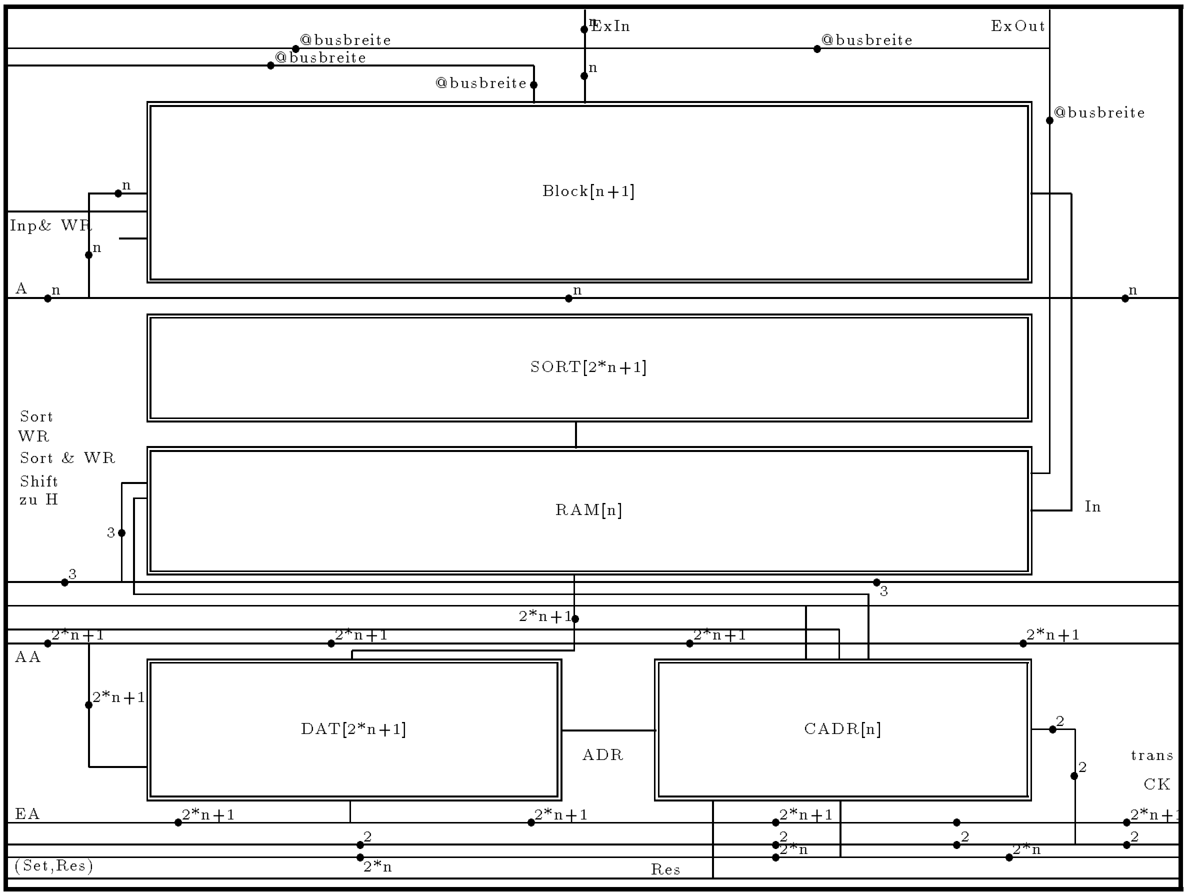
The circuit $CHIP[n]$ — The system at its highest hierarchy level with the control unit $STE[n]$



Construction of the odd subcube

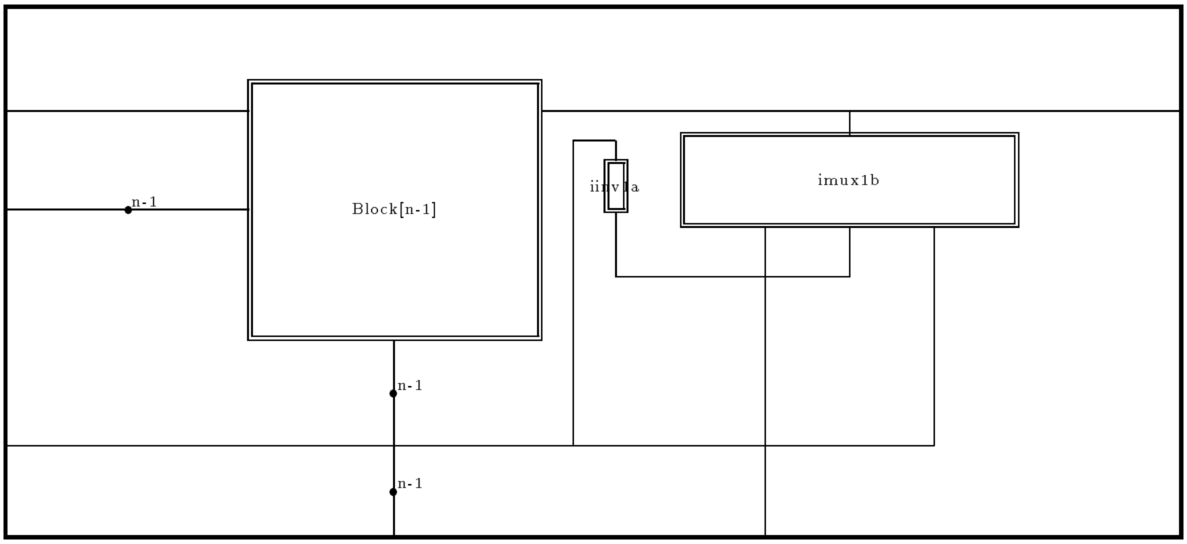


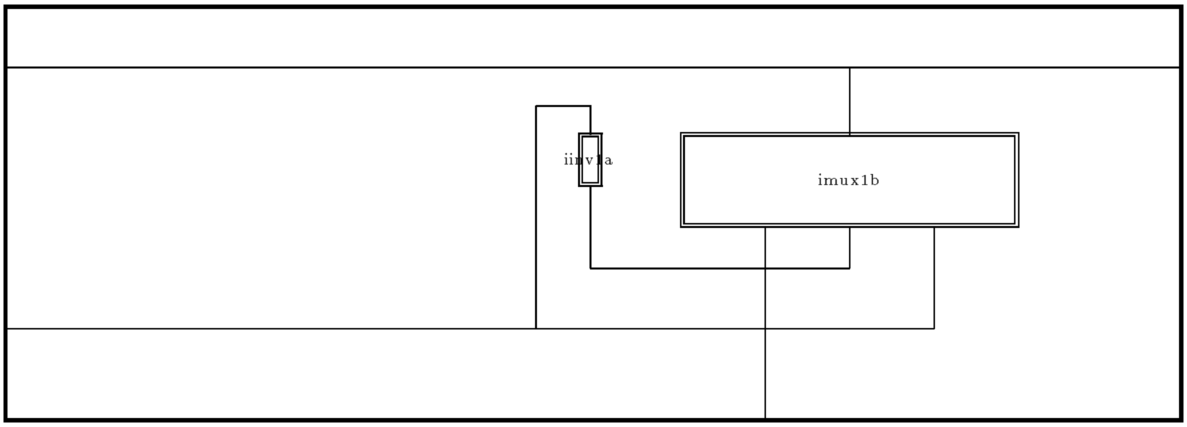
Preparation of the edges



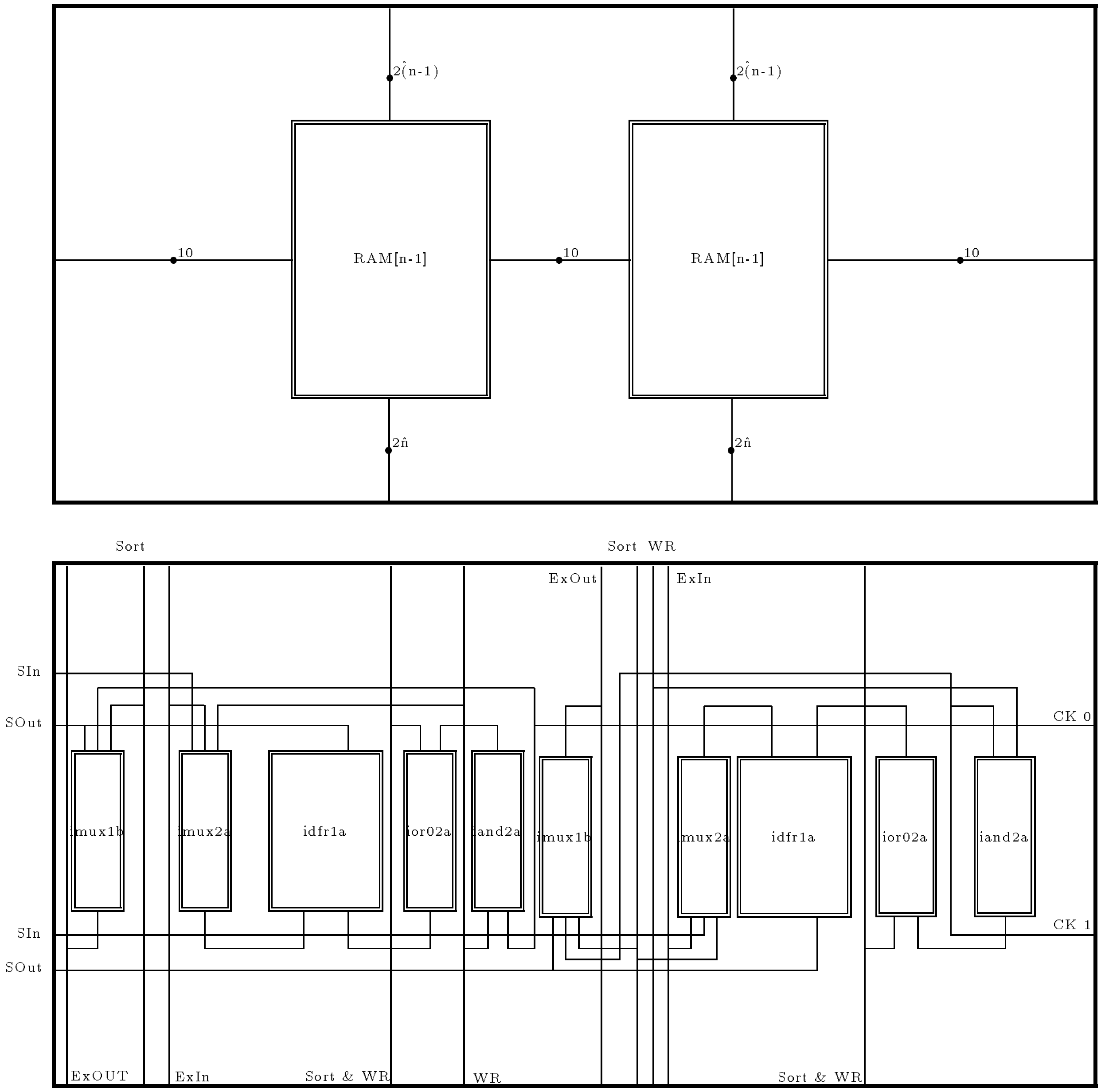
The node of the hypercube at the highest hierarchy level

The circuit $CADR[n]$ (Count Address) contains the binary address of the node. It determines, whether the data of the node must be shifted.

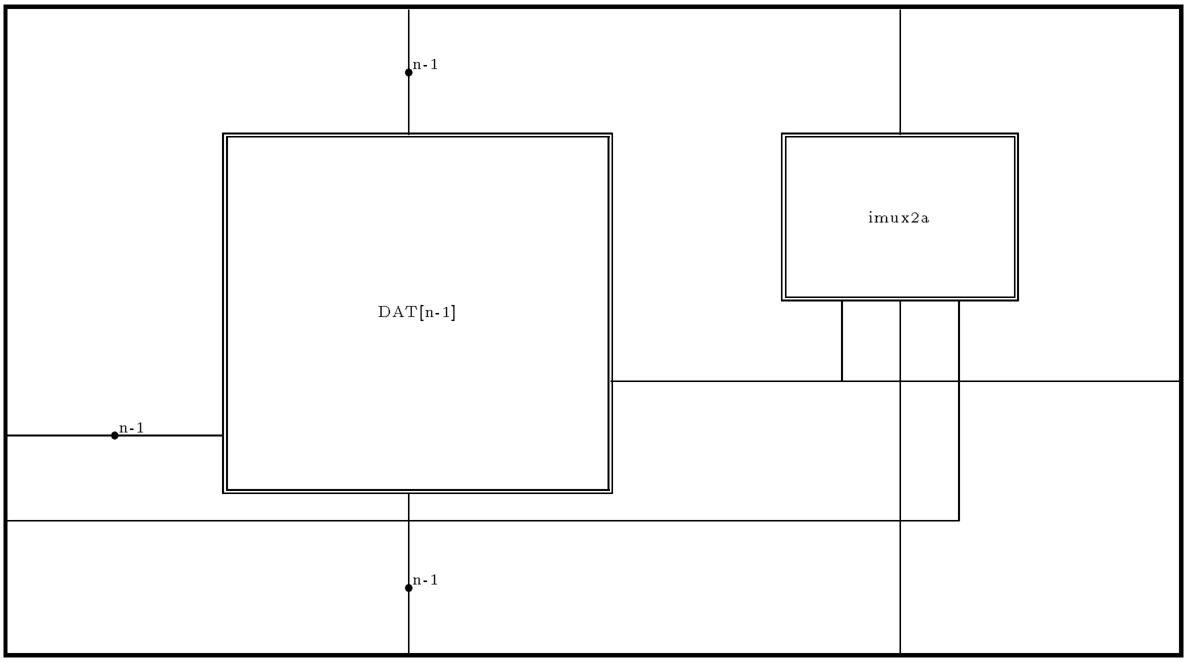


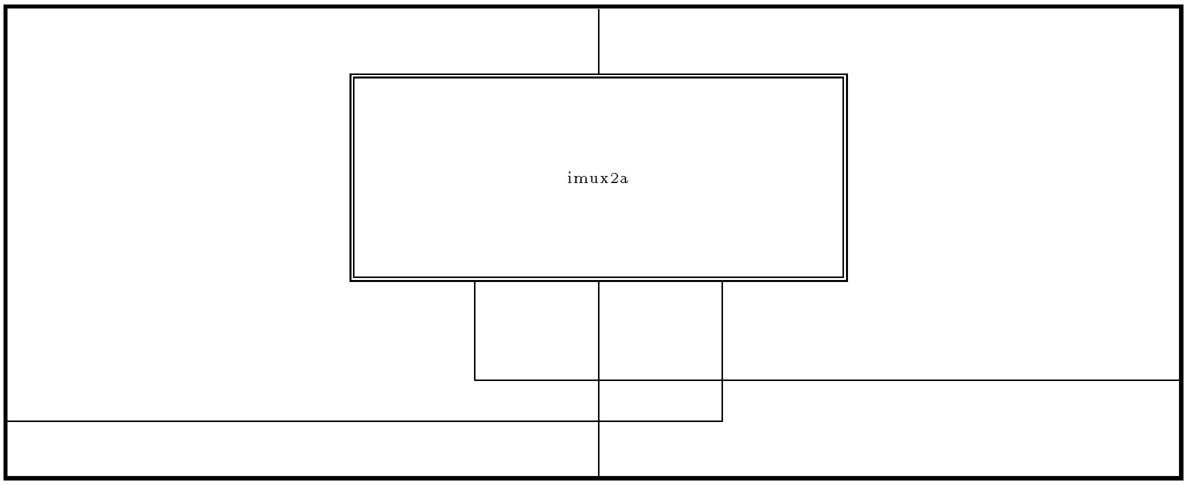


Upper diagram: Hierarchical representation of the circuit $BLOCK[n]$ that determines the dimension along which the data must be exchanged;
Lower diagram: Termination of the hierarchy of $BLOCK[n]$ — $BLOCK[1]$

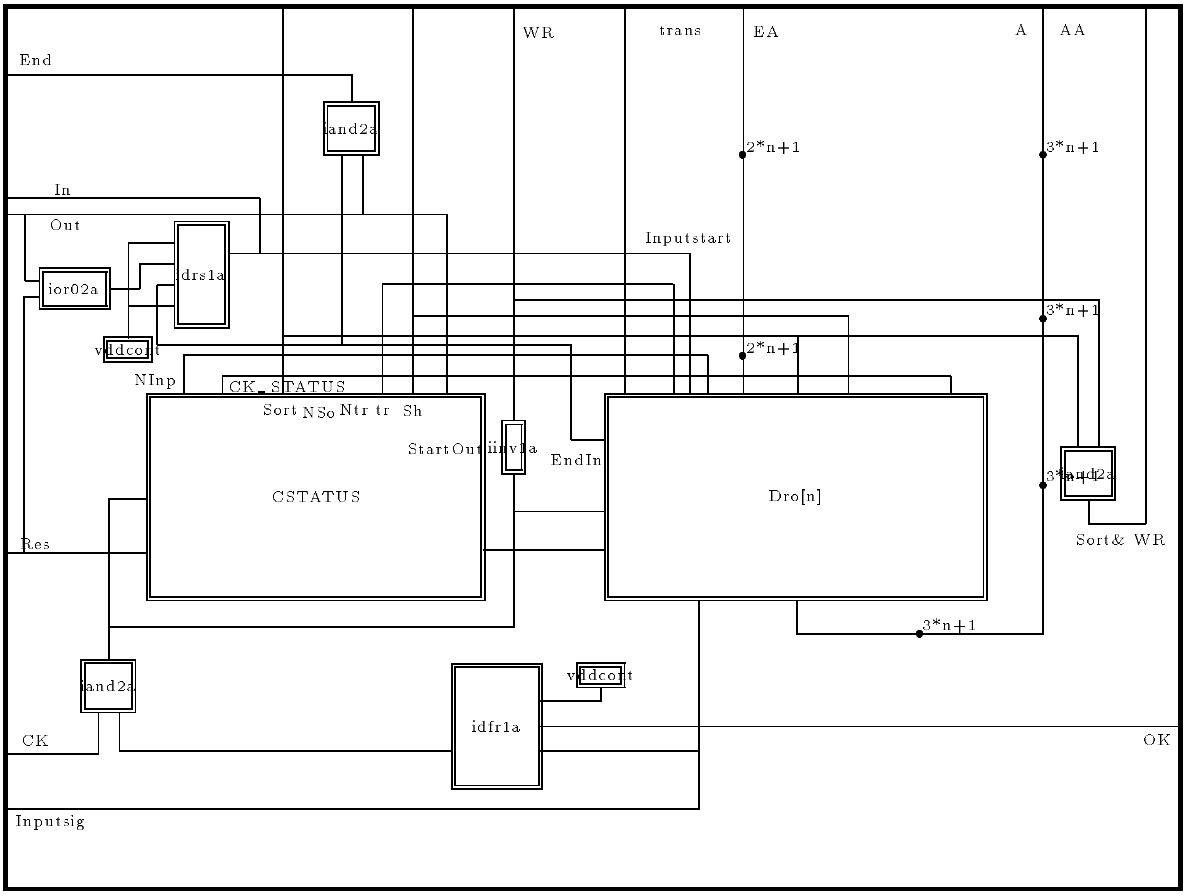


Upper diagram: Hierarchical construction of the memory cells;
Lower diagram: Termination of the hierarchy (two memory cells)





Upper diagram: The circuit $DAT[n]$ used to calculate the actual addresses of the data to be shared using EA, AA and $\overline{a}_j$;
Lower diagram: Termination of the Hierarchy of the circuit $DAT[n]$ — $DAT[1]$.



The control unit at the highest hierarchy level

The circuit $CSTATUS$ (Count Status) determines the state variables of the system;

$DRO[n]$ determines the variables EA, A and AA used to calculate the data addresses to be exchanged.

References

[BBDS] D. H. Bailey, E.Barszcz, L. Dagum and H. D. Simon
NAS Parallel Benchmark Results 3-94
RNR Technical Report RNR-94-006
NASA Ames Research Center, March 1994

[BrockWan97] K. Brockmann, R. Wanka
Efficient Oblivious Parallel Sorting on the MasPar MP – 1
In Proc. 30th Hawaii International Conference on System Sciense (HICSS), IEEE, Jan. 1997

[Bur94] Th. Burch
Eine graphische Arbeitsumgebung für den parametrisierten Entwurf integrierter Schaltungen
PhD thesis, department of Computer Science, University of Saarland, 1994

[Dea] Ralf Dickmann et al.
Sortieren großer Datenmengen auf einem massiv parallelen System

[Gam96] A. Gamkrelidze
Entwurf eines booleschen Sortiernetzes mit der Struktur eines $n-$ dimensionalen Würfels
Masters thesis, University of Saarland, 1996

[Hard96] J. C. Hardwick
An Efficient Implementation of Nested Data Parallelism for Irregular Divide- and- Conquer Algorithms
In Proceedings of First International Workshop on High- Level Programming Models and Supportive Environments, pp. 105 – 114, Apr. 1996

[Hot 65] G.Hotz
Eine Algebraisierung des Syntheseproblems für Schaltkreise
EIK 1, 1965, pp. 185 – 205, 209 – 231

[HotRe 96] G. Hotz and A. Reichert
Hierarchischer Entwurf komplexer Systeme
In: I. Wegener (editor) Highlights aus der Informatik, Springer Verlag, 1996

[Kol 87] R. Kolla
Spezifikation und Expansion Logisch Topologischer Netze
PhD thesis, Saarbrücken, 1986/87

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

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

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

[Mol 86] P. Molitor
Über die Bikategorie der Logisch- Topologischen Netze und ihre Semantik
PhD thesis, Saarbrücken, 1986

[ZagBlel91] M. Zagha, G. Blelloch
Radix Sort for Vector Multiprocessors
Supercomputing' 91, November 1991