Routing Partial Permutations in General Interconnection Networks based on Radix Sorting

Tripti Jain and Klaus Schneider
University of Kaiserslautern, Germany
http://es.cs.uni-kl.de/

Abstract. In general, sorting networks can be used as interconnection networks in that the input messages are simply sorted according to their target addresses. If the target addresses form a permutation of *all* addresses, this is obviously correct since then the sorting algorithm routes each message to its target address. However, if not all inputs need a connection to one of the outputs, then some output addresses do not appear as target addresses, and thus, partial permutations have to be implemented. In this case, sorting networks work no longer correctly as interconnection networks since all messages with target addresses larger than the smallest missing target address will be routed to the wrong outputs. For merge-based sorting networks, there is a well-known general solution called the Batcher-Banyan network. However, for the larger class of radix-based sorting networks this does not work, and there is only one solution known for a particular permutation network [28]. In this paper, we present three general constructions to convert any binary sorter into a ternary split module which is the key to construct a radix-based interconnection network that can cope with partial permutations. We compare the sizes and depths of the circuits obtained by our constructions for six known binary sorters and show this way that the obtained circuits are of practical interest.

1. Introduction

Non-blocking unicast interconnection networks allow every input component $x_0, \ldots, x_{n-1}$ to be connected with any output component $y_0, \ldots, y_{n-1}$ provided that none of the outputs y_j is the target of more than one input x_i. Hence, such networks can implement *all* $n!$ *permutations* of the addresses $\{0, \ldots, n-1\}$ as routes through a switching network that is typically built by 2×2 crossbar switches. In practice, however, not all input components have to be always connected to an output component. For this reason, even *all* $\sum_{i=0}^{n} i!\binom{n}{i}^2$ *partial permutations* have to be implemented by non-blocking unicast interconnection networks.

Even for total permutations, the efficient implementation of such networks turned out to be a difficult challenge for many decades: The simplest non-blocking network is the *crossbar* that can be implemented as circuit of size $O(n^2)$ and depth $O(\log(n))$ for connecting n components. While the depth of the crossbar is optimal, its size grows with $O(n^2)$ and becomes quickly prohibitive for large n. The challenge is therefore to develop non-blocking interconnection networks with a size of less than $O(n^2)$ and with a poly-logarithmic depth $O(\log(n)^c)$ (for some small constant $c \in \mathbb{N}$).

To reduce the size of the crossbar network, Clos [7] constructed a three-stage network using q $r \times p$ crossbars in the first stage, p $q \times q$ crossbars in the second stage, and q $p \times r$ crossbars in

the third stage with $n = q \cdot r$. He proved that his network is non-blocking without rearranging existing connections iff $p \geq 2r - 1$ and still non-blocking for $p \geq r$ when existing connections are rearranged. Based on these observations, Beneš [3, 35] constructed special Clos networks built by $2\log(n) - 1$ stages of 2×2 crossbars only. Many other networks based on 2×2 crossbars were then proposed, e.g., the Ω-network [21], the butterfly (Banyan) networks [11], fat trees [25], flattened butterfly [18] to name just a few (see textbooks like [8, 33] for further examples). For all of these networks, however, it turned out that they are either blocking or that it is very difficult to determine the configuration of their 2×2 crossbars to establish a desired permutation. In particular, the routing problem for the Beneš network has been considered in many research papers [23, 29] and the known parallel algorithms to compute configurations have a depth larger than the network itself.

Sorting networks [2] are therefore an attractive alternative for the design of non-blocking interconnection networks [10]: To that end, the inputs $x_0, \ldots, x_{n-1}$ are simply sorted according to their target addresses to implement the desired permutation. The 2×2 crossbars become then compare-and-swap switches that determine their configuration by simply comparing the incoming target addresses. The AKS network [1] proves that there exist sorting networks with depth $O(\log(n))$ and size $O(n \log(n))$, so that the use of sorting networks can significantly improve the costs of crossbars[1]. However, the constants hidden behind the O-calculus turned out to be prohibitively large so that the AKS network is unfortunately impractical [24, 31, 34]. However, well-known sorting networks like Batcher's bitonic and odd-even merge networks [2] and related variants [9, 20, 30] with a depth $O(\log(n)^2)$ and size $O(n \log(n)^2)$ are still competitive[2].

However, the *implementation of partial permutations by sorting networks* is not straightforward and depends on the used sorting algorithms as we will outline in the next section. In particular, there is a general solution for the class of merge-based networks, while for radix-based sorting networks that were often considered for the design of interconnection networks [4, 5, 16, 17, 19, 22, 28], only a special solution given by Narasimha [28] was known so far. Unfortunately, his network has a bad depth of $O(n)$ and is therefore not efficient enough for many applications.

In this paper, we present three general constructions that can transform binary sorters to ternary Split modules. The latter are the key modules that directly lead to interconnection networks for partial permutations based on radix-based sorting. This way, many efficient interconnection networks [4, 5, 16, 17, 19, 22, 28] that were designed for total permutations can be transformed so that the resulting networks can also work with partial permutations. While our constructions roughly double the size of the circuits, they nearly maintain their depths, and thus do not influence the latency of the networks. In particular, we still obtain networks of size $O(n \log(n)^3)$ and depth $O(\log(n)^2)$ (see [16]). By experimental results, we also show that the sizes of the obtained networks are still in a practical range and are competitive to other known solutions.

The outline of the paper is as follows: In the next section, we report about related work on the use of sorting networks as interconnection networks. In particular, we discuss problems and known solutions for establishing partial permutations with sorting networks. Section 3 contains the core of the paper where we show how to transform binary sorters to ternary Split modules to implement radix-sorting interconnection networks for partial permutations. Finally, Section 4 shows by experimental results that the obtained networks still have a competitive size and depth.

[1] These complexities refer to the compare-and-swap modules that have to compare addresses $0, \ldots, n-1$ having $\log(n)$ bits. Gate-level implementations of these modules have size $O(\log(n))$ and depth $O(\log(\log(n)))$, so that gate-level implementations of the AKS network will have circuit depth $O(\log(n)\log(\log(n)))$ and size $O(n \log(n)^2)$.

[2] Their gate-level implementations have a depth $O(\log(n)^2 \log(\log(n)))$ and size $O(n \log(n)^3)$.

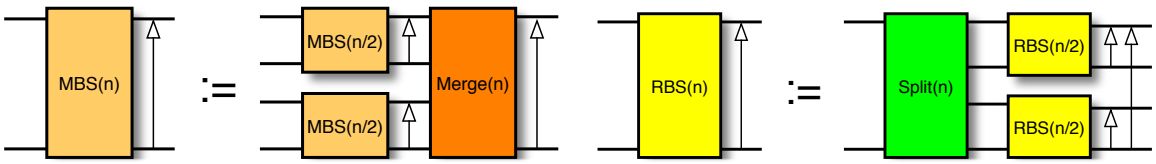


Figure 1: Merge-based sorting (MBS) versus radix-based sorting (RBS): MBS merges already sorted sequences with Merge modules while RBS partitions input sequences by Split modules into two halves that are independently sorted.

2. Sorting Networks as Interconnection Networks

2.1. Routing Total Permutations by Sorting Networks

There are two important classes of sorting networks, namely the merge-based (MBS) and the radix-based (RBS) sorting networks which are recursively defined as shown in Figure 1. In the *merge-based approach*, a sorting network $\mathsf{MBS}(n)$ for n inputs is recursively constructed by splitting the given sequence into two halves, recursively sorting these by two sorting networks $\mathsf{MBS}(\frac{n}{2})$ of half the size, and then merging the two sorted halves by a merge module $\mathsf{Merge}(n)$. Well-known sorting networks following this paradigm are Batcher's bitonic and odd-even sorting networks [2] and related ones [9, 20, 30].

In *radix-based sorting networks*, the given inputs are partitioned into two halves by a $\mathsf{Split}(n)$ module, e.g., by sorting them according to the most significant bit of their target address. Thus, after the $\mathsf{Split}(n)$ module, the given inputs have already been routed to the right halves, so that the remaining problems can be solved recursively in the same way (ignoring now the most significant bits of the target addresses). The implementation of radix-based sorting networks is completely determined by the implementation of the Split modules.

There are many ways to implement a Split module for total permutations, e.g., by means of binary sorters [4, 5, 16, 17, 19, 22, 28] or concentrators [6, 13, 26, 32]: A (n, m)-concentrator is a circuit with n inputs and $m \leq n$ outputs that can route any given number $k \leq m$ of valid inputs to k of its m outputs. Split modules for total permutations can be obtained by two $(n, \frac{n}{2})$-concentrators: One that routes the $\frac{n}{2}$ inputs with a most significant bit 1 from the n inputs to the upper half of outputs, and another one routing the other $\frac{n}{2}$ inputs with a most significant bit 0 to the lower half of outputs.

2.2. Routing Partial Permutations by Sorting Networks

Independent on the choice of a particular sorting algorithm, sorting networks at first only implement *total permutations* in that they can sort the n inputs by their target addresses which are numbers $0, \ldots, n-1$. If some inputs do not need a connection to an output, their target addresses are invalid, denoted as $\perp$ in the following. Note that there is no ordering of $\{\perp, 0, \ldots, n-1\}$ that would still solve the routing problem by a simple sorting approach, since many values $\perp$ may now occur and they may have to be routed to different places in the final output sequence.

For merge-based sorting networks, there is a well-known solution known as the Batcher-Banyan network [12, 27]. The main idea is thereby to first treat $\perp$ as a number larger than all target addresses so that after using a normal sorting network this way, one obtains a preliminary output sequence $y_0, \ldots, y_{k-1}, y_k, \ldots, y_{n-1}$ where the k valid inputs were sorted as the prefix $y_0, \ldots, y_{k-1}$ while the invalid ones are placed in the suffix $y_k, \ldots, y_{n-1}$. A final Banyan permutation network can then

be used to move the valid inputs $y_0, \ldots, y_{k-1}$ to the right places. To that end, one can simply use a bit-controlled network like the Ω-network [21] where invalid target addresses $\perp$ are ignored, so that the valid ones are routed to their final destination. It can be shown [27] that the Ω-network [21] while being blocking in general will never block in this setting.

The same approach does however not work for the radix-based networks: If we treat $\perp$ as a number larger than all target addresses, it may happen that valid inputs with a most significant bit 1 will be erroneously routed by Split modules to the lower sub-network, where they are mixed up with other valid inputs having a most significant bit 0. Hence, the resulting preliminary output sequence will not consist of a sorted prefix of valid inputs as in the case of merge-based networks.

Hence, the Batcher-Banyan construction does not work for RBS networks. Recall that the task of Split modules was to route the inputs already in the right halves. Inputs with invalid target addresses can be routed to any half, but inputs with valid target address must be routed to the lower and upper sub-networks in case the most significant bit of the target address is 0 and 1, respectively.

Instead of using binary sorters as in case of total permutations, one could therefore use *ternary sorters* as Split modules using the ordering $0 \preceq \perp \preceq 1$. This way, the output sequence of a Split module will still route the inputs with valid target addresses to the right halves, while invalid inputs may be routed to any half (note that still at most $\frac{n}{2}$ inputs can have most significant bits 0/1). However, while many constructions for binary sorters have been proposed [4, 5, 16, 17, 19, 22, 28], none are known for the ternary case. We therefore show in the next section how to construct ternary sorters from any binary sorter with almost the same circuit depth, but doubling the circuit size. This way, we can transform any RBS network that has been constructed for total permutations into a more powerful one that can work with partial permutations as well. Additionally, we consider two optimizations of the ternary sorters for implementing RBS networks.

Narasimha addressed the problem to route partial permutations in his RBS network in [28]. In Section III of [28], he explains without giving a proof that his network can also work with partial permutations if an additional Split module is added on the left side of his RBS network. While this is true for his network, and also for some others (we prove a characterization which networks can be transformed this way in an upcoming publication), it is definitely not true for general RBS networks (like three of the six we consider in the experimental results).

3. Split Modules for Routing Partial Permutations in RBS Networks

In this section, we present three constructions based on binary sorters to implement Split modules for routing partial permutations in RBS networks. For all implementations, we assume that any input x_i is a bitvector in the format given on the left-hand side below:

$x_{i,0} \ldots x_{i,q-1}$	$x_{i,q}$	$x_{i,q+1} \ldots x_{i,q+\log(n)}$

$x_{i,q}$	$x_{i,q+\log(n)}$	msb
0	$*$	$\perp$
1	0	0
1	1	1

The leftmost q bits are the *message bits* that should be sent to an output, bit q is the *valid bit* that indicates whether this input shall be connected to some output, and the remaining bits are the *bits of the target address* where the most significant bit (msb) is the rightmost one (that is consumed by each Split module in the RBS network). Considering $x_{i,q}$ and $x_{i,q+\log(n)}$ only, we interpret inputs x_i as ternary values $\{0, \perp, 1\}$ as shown on the right hand side above.

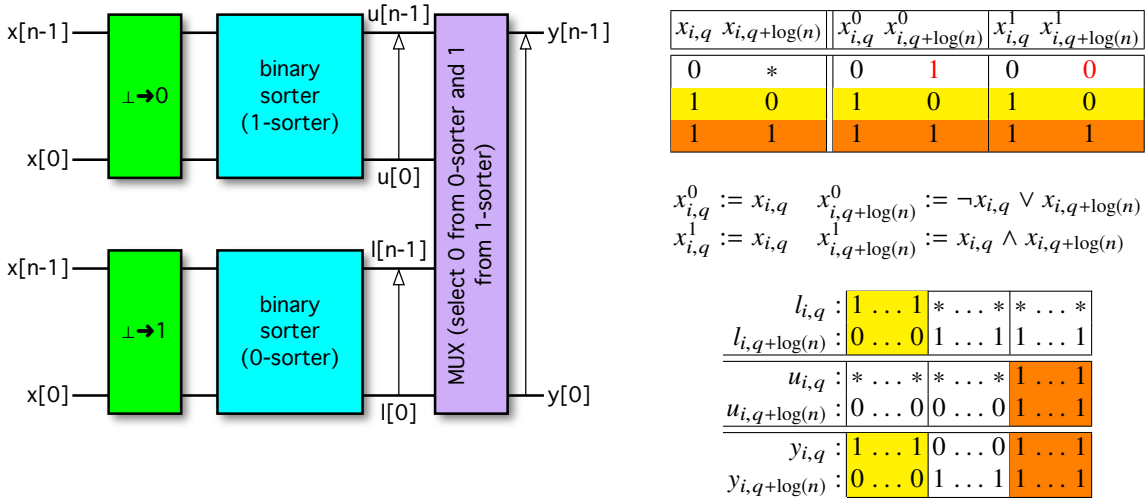


Figure 2: Construction of a Ternary Sorter by two Binary Sorters.

3.1. Constructing Split Modules by Ternary Sorters

The left-hand side of Figure 2 shows how a ternary sorter can be constructed by two binary sorters that we call the 0-sorter and the 1-sorter, respectively. Both binary sorters obtain the n inputs $x_0, \ldots, x_{n-1}$ after a preprocessing step that modifies the msbs $x_{i,q+\log(n)}$ of the invalid target addresses as shown on the upper right part of Figure 2 as $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$ for the 0- and 1-sorter, respectively. Note that after the preprocessing step, only the valid inputs have msbs 0 and 1 for the 0- and 1-sorter, respectively.

After this, the 0-sorter and the 1-sorter sort their input sequences to output sequences $l_0, \ldots, l_{n-1}$ and $u_0, \ldots, u_{n-1}$, respectively, by only considering the modified msbs $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$. Hence, the 0-sorter uses the ordering $0 \prec \{\perp, 1\}$ while the 1-sorter uses ordering $\{0, \perp\} \prec 1$ (regarding the original inputs).

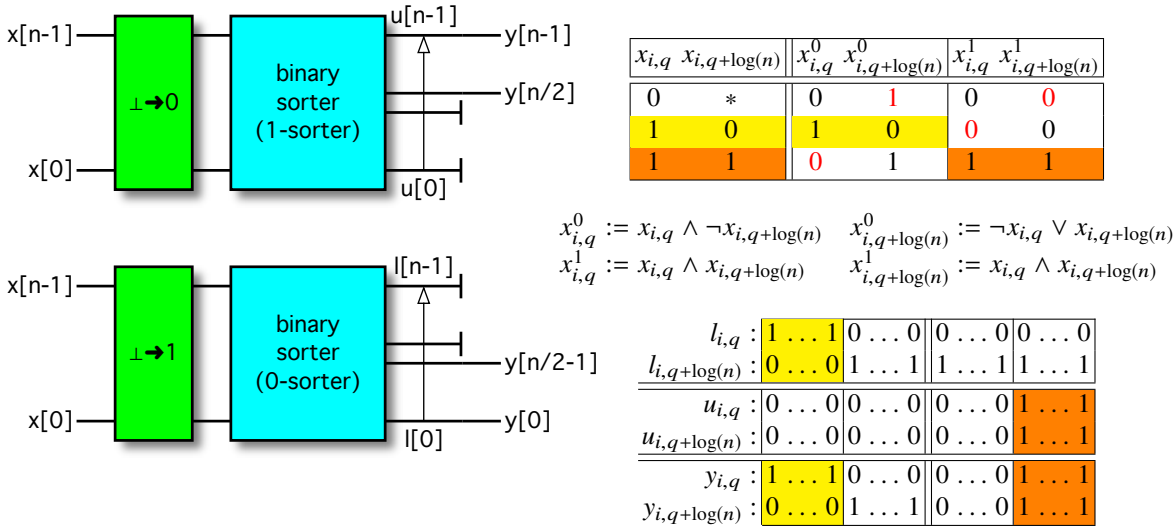
The lower right part of Figure 2 shows how the 0- and 1-sorter's output sequences look like in general: The 0-sorter's output sequence starts with values $(l_{i,q}, l_{i,q+\log(n)}) = (1, 0)$, i.e., 0, followed by values $(l_{i,q}, l_{i,q+\log(n)}) = (*, 1)$, i.e., $\perp$ or 1, while the 1-sorters output sequence starts with values $(u_{i,q}, u_{i,q+\log(n)}) = (*, 0)$, i.e., 0 or $\perp$, followed by values $(u_{i,q}, u_{i,q+\log(n)}) = (1, 1)$, i.e., 1.

The final stage of multiplexers will then determine output y_i by selecting one of the corresponding values l_i or u_i as follows where l'_i is obtained from l_i by setting its valid bit to 0:

$$y_i := \begin{cases} u_i & : \text{if } u_{i,q} \wedge u_{i,q+\log(n)} \\ l_i & : \text{if } l_{i,q} \wedge \neg l_{i,q+\log(n)} \\ l'_i & : \text{otherwise} \end{cases}$$

Note that the number of valid inputs can be at most n, hence, we never have both $u_{i,q} \wedge u_{i,q+\log(n)}$ and $l_{i,q} \wedge \neg l_{i,q+\log(n)}$. Note further that we have to set $l_{i,q} := 0$ in case l_i is chosen for y_i, but $l_{i,q} \wedge \neg l_{i,q+\log(n)}$ does not hold (this way, we avoid that an input with $(x_{i,q}, x_{i,q+\log(n)}) = (1, 1)$ will be taken from the 0-sorter that has already been copied from the 1-sorter).

It can be easily verified that the circuit shown in Figure 2 implements a ternary sorter, i.e., any input sequence $x_0, \ldots, x_{n-1}$ of values $\{0, \perp, 1\}$ is correctly sorted using the total order $0 \prec \perp \prec 1$.



$x_{i,q}$	$x_{i,q+\log(n)}$	$x^0_{i,q}$	$x^0_{i,q+\log(n)}$	$x^1_{i,q}$	$x^1_{i,q+\log(n)}$
0	*	0	1	0	0
1	0	1	0	0	0
1	1	0	1	1	1

$$x^0_{i,q} := x_{i,q} \wedge \neg x_{i,q+\log(n)} \qquad x^0_{i,q+\log(n)} := \neg x_{i,q} \vee x_{i,q+\log(n)}$$
$$x^1_{i,q} := x_{i,q} \wedge x_{i,q+\log(n)} \qquad x^1_{i,q+\log(n)} := x_{i,q} \wedge x_{i,q+\log(n)}$$

$l_{i,q}$:	1 . . . 1	0 . . . 0	0 . . . 0	0 . . . 0
$l_{i,q+\log(n)}$:	0 . . . 0	1 . . . 1	1 . . . 1	1 . . . 1
$u_{i,q}$:	0 . . . 0	0 . . . 0	0 . . . 0	1 . . . 1
$u_{i,q+\log(n)}$:	0 . . . 0	0 . . . 0	0 . . . 0	1 . . . 1
$y_{i,q}$:	1 . . . 1	0 . . . 0	0 . . . 0	1 . . . 1
$y_{i,q+\log(n)}$:	0 . . . 0	1 . . . 1	0 . . . 0	1 . . . 1

Figure 3: Construction of a Ternary Splitter by Binary Sorters.

3.2. Constructing Split Modules by Ternary Concentrators

We have already mentioned that the Split modules do not have to be ternary sorters to partition the inputs according to their msbs. Instead, it is sufficient to route all inputs x_i with $(x_{i,q}, x_{i,q+\log(n)}) = (1,1)$ to the upper half and all inputs x_i with $(x_{i,q}, x_{i,q+\log(n)}) = (1,0)$ to the lower half, while the invalid inputs x_i with $x_{i,q} = 0$ may be routed to any half among the other values routed there.

For this reason, we can also consider the slightly simplified construction given in Figure 3. Compared to Figure 2, we modify the msbs $x_{i,q+\log(n)}$ of the target addresses in the same way, but additionally invalidate all 1s and 0s in the 0- and 1-sorter, respectively, as shown in the upper right part of Figure 3. Hence, the 0-sorter will only have inputs $(x^0_{i,q}, x^0_{i,q+\log(n)}) \in \{(0,1),(1,0)\}$, i.e., $\{\perp, 0\}$, and the 1-sorter will only have inputs $(x^1_{i,q}, x^1_{i,q+\log(n)}) \in \{(0,0),(1,1)\}$, i.e., $\{\perp, 1\}$.

Again, the 0- and 1-sorter only consider the modified msbs $x^0_{i,q+\log(n)}$ and $x^1_{i,q+\log(n)}$, respectively, to generate their output sequences $l_0, \ldots, l_{n-1}$ and $u_0, \ldots, u_{n-1}$, respectively.

Assuming now that at most $\frac{n}{2}$ inputs x_i satisfy $(x_{i,q}, x_{i,q+\log(n)}) = (1,0)$ and also at most $\frac{n}{2}$ inputs x_i satisfy $(x_{i,q}, x_{i,q+\log(n)}) = (1,1)$, we can simply determine y_i as follows (see lower right part of Figure 3):

$$y_i := \begin{cases} u_i & : \text{if } i \in \{\frac{n}{2}, \ldots, n-1\} \\ l_i & : \text{if } i \in \{0, \ldots, \frac{n}{2}-1\} \end{cases}$$

As long as at most $\frac{n}{2}$ inputs x_i are 0 and 1, the output sequence will even be a sorted ternary sequence. However, if more than $\frac{n}{2}$ inputs x_i should be 0 or more than $\frac{n}{2}$ inputs x_i should be 1, the circuit will omit some of the inputs and will therefore no longer be correct. We therefore do not consider the circuit of Figure 3 as a ternary sorter, but each part of it is a $(n, \frac{n}{2})$ concentrator that concentrates on the 0 and 1 values, respectively.

While not yielding a ternary sorter for general ternary sequences, Figure 3 still sorts all ternary input sequences that will appear in RBS networks for partial permutations. However, it does not allow some further optimizations as the one shown in the next section.

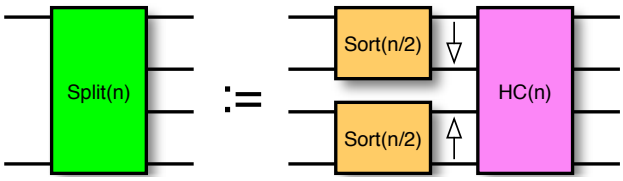


Figure 4: Construction of a Ternary Splitter by Ternary Sorters and a Half Cleaner.

3.3. Constructing Split Modules by Ternary Sorters and Half Cleaners

In previous work [14, 15], we have shown how (ternary) Split modules with n inputs/outputs can be constructed as shown in Figure 4 using two (ternary) sorters with $\frac{n}{2}$ inputs/outputs and a half cleaner circuit. Half cleaners were introduced by Batcher in [2] for the construction of his bitonic sorting networks. We observed that half cleaners can also be used to implement binary [15] and ternary [14] Split modules as shown in Figure 4. Due to lack of space, we cannot list details of the definition of half cleaners, and just mention here that these circuits have size $O(n)$ and depth $O(1)$, so that the depth is mainly determined by the used sorting networks (see [14, 15] for further details).

As outlined in [14], it is required to use sorting networks for the construction of Figure 4. In particular, the construction shown in the previous section, i.e., in Figure 3 cannot be used. Hence, even though our initial construction of Figure 2 cannot compete with the one in Figure 3, it allows the optimization shown in Figure 4. As our experimental results show, this implementation often turns out as the best one of the three versions discussed in this paper.

4. Experimental Results

The depth of the circuits is mainly determined by the depth of the binary sorters which is $O(\log(n))$ or $O(\log(n)^2)$ for known implementations. Note that modifying the msbs and the selection of the outputs as either l_i or u_i does only require circuits of depth $O(1)$. Also the size of the circuits is mainly dominated by the size of the binary sorters. While the depth does only increase by some constant, the size obviously is twice the size of the binary sorters plus some $O(n)$ gates for the mapping and possible multiplexer stages.

To consider concrete circuits, we have implemented the constructions described in the previous section for six binary sorters that we abbreviate by the acronyms of the authors of the paper where these binary sorters were published: Batc68 [2] (the bitonic sorter reduced to one bit), ChOr94 [5], ChCh96 [4], JaSJ17 [16], KoOr90 [19], and Nara94 [28].

The tables shown in Figure 5, Figure 6, and Figure 7 show the experimental results that we obtained for the constructions given in Figure 2, Figure 3, and Figure 4, respectively. In these tables, we list the depths and sizes of the generated circuits for n inputs/outputs, and the numbers of NOT, AND, OR, XOR gates, half adders (HA), full adders (FA), 2:1 multiplexers (MX) and 2×2 crossbar switches (SW). The tables show only the size of the Split modules, and not of the corresponding RBS networks. The size always improves from Figure 5 via Figure 6 to Figure 7, but the depths are sometimes best for Figure 6 and sometimes for Figure 7.

5. Conclusions

In this paper, we presented three transformations that convert binary sorters to ternary Split modules. Using the latter, interconnection networks based on radix-based sorting can be implemented that can correctly route also partial permutations. Our transformations yield Split modules with the same asymptotic complexities as the binary sorters in terms of circuit size and depth, and even only add a constant to the circuit depth, but roughly double the size of the circuits. Nevertheless, the sizes are still competitive since very good implementations of binary sorters have been developed in many previous research papers that can now be used also for the implementation of interconnection networks.

References

[1] Ajtai, M., J. Komlos, and E. Szemeredi: *An $O(n \log(n))$ sorting network*. In *Symposium on Theory of Computing (STOC)*, pages 1–9. ACM, 1983.

[2] Batcher, K.E.: *Sorting networks and their applications*. In *AFIPS Spring Joint Computer Conference*, volume 32, pages 307–314, 1968.

[3] Beneš, V.E.: *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, 1965.

[4] Cheng, W. J. and W. T. Chen: *A new self-routing permutation network*. IEEE Transactions on Computers, 45(5):630–636, May 1996.

[5] Chien, M.V. and A.Y. Oruç: *High performance concentrators and superconcentrators using multiplexing schemes*. IEEE Transactions on Communications, 42(11):3045–3050, November 1994.

[6] Chung, F.R.K.: *On concentrators, superconcentrators, generalizers, and nonblocking networks*. The Bell Systems Technical Journal, 58(8):1765–1777, October 1978.

[7] Clos, C.: *A study of non-blocking switching networks*. Bell System Technical Journal, 32(2):406–424, March 1953.

[8] Dally, W.J. and B. Towles: *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[9] Dowd, M., Y. Perl, M. Saks, and L. Rudolph: *The balanced sorting network*. In *Symposium on Principles of Distributed Computing (PODC)*, pages 161–172, Montreal, Quebec, Canada, 1983. ACM.

[10] Galil, Z. and W.J. Paul: *An efficient general-purpose parallel computer*. Journal of the ACM (JACM), 30(2):360–387, 1983.

[11] Goke, L.R. and G. Jack Lipovski: *Banyan networks for partitioning multiprocessor systems*. In *25 Years of the International Symposia on Computer Architecture (ISCA)*, pages 117–124, Barcelona, Spain, 1998. ACM.

[12] Huang, A. and S. Knauer: *Starlite: A wideband digital switch*. In *Global Telecommunications Conference (GLOBECOM)*, pages 121–125, 1984.

[13] Jain, T. and K. Schneider: *Verifying the concentration property of permutation networks by BDDs*. In Leonard, E. and K. Schneider (editors): *Formal Methods and Models for Codesign (MEMOCODE)*, pages 43–53, Kanpur, India, 2016. IEEE Computer Society.

[14] Jain, T. and K. Schneider: *The half cleaner lemma: Constructing efficient interconnection networks from sorting networks*. Parallel Processing Letters, 28(1), March 2018.

[15] Jain, T., K. Schneider, and A. Jain: *Deriving concentrators from binary sorters using half cleaners*. In *Reconfigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2017. IEEE Computer Society.

[16] Jain, T., K. Schneider, and A. Jain: *An efficient self-routing and non-blocking interconnection network on chip*. In *Network on Chip Architectures (NoCArc)*, pages 4:1–4:6, Boston, MA, USA, 2017. ACM.

[17] Jan, C.Y. and A.Y. Oruç: *Fast self-routing permutation switching on an asymptotically minimum cost network*. IEEE Transactions on Computers, 42(12):1469–1479, December 1993.

[18] Kim, J., W.J. Dally, and D. Abts: *Flattened butterfly: a cost-efficient topology for high-radix networks*. In Tullsen, D.M. and B. Calder (editors): *International Symposium on Computer Architecture (ISCA)*, pages 126–137, San Diego, California, USA, 2007. ACM.

[19] Koppelman, D.M. and A.Y. Oruç: *A self-routing permutation network*. Journal of Parallel and Distributed Computing, 10(2):140–151, 1990.

[20] Kutyłowski, M., K. Loryś, B. Oesterdiekhoff, and R. Wanka: *Periodification scheme: constructing sorting networks with constant period*. Journal of the ACM (JACM), 47(5):944–967, September 2000.

[21] Lawrie, D.H.: *Access and alignment of data in an array processor*. IEEE Transactions on Computers (T-C), 24:1145–1155, December 1975.

[22] Lee, C. Y. and A.Y. Oruç: *Design of efficient and easily routable generalized connectors*. IEEE Transactions on Communications, 43(2-4):646–650, 1995.

[23] Lee, C. Y. and A.Y. Oruç: *A fast parallel algorithm for routing unicast assignments in Beneš networks*. IEEE Transactions on Parallel and Distributed Systems, 6(3):329–334, March 1995.

[24] Leighton, F.T.: *Tight bounds on the complexity of parallel sorting*. IEEE Transactions on Computers (T-C), 34(4):344–354, April 1985.

[25] Leiserson, C.E.: *Fat-trees: Universal networks for hardware efficient supercomputing*. IEEE Transactions on Computers (T-C), 34(10):892–901, October 1985.

[26] Masson, G.M., G.C. Gingher, and S. Nakamura: *A sampler of circuit switching networks*. IEEE Computer, 12(6):32–48, 1979.

[27] Narasimha, M.J.: *The Batcher-Banyan self-routing network: universality and simplification*. IEEE Transactions on Communications, 36(10):1175–1178, October 1988.

[28] Narasimha, M.J.: *A recursive concentrator structure with applications to self-routing switching networks*. IEEE Transactions on Communications, 42(2-4):896–898, February/March/April 1994.

[29] Nassimi, D. and S. Sahni: *Parallel algorithms to set up the Beneš permutation network*. IEEE Transactions on Computers (T-C), 31(2):148–154, February 1982.

[30] Parberry, I.: *The pairwise sorting network*. Parallel Processing Letters (PPL), 2(2-3):205–211, 1992.

[31] Paterson, M.S.: *Improved sorting networks with $O(log(N))$ depth*. Algorithmica, 5(4):75–92, 1990.

[32] Pinsker, M.S.: *On the complexity of a concentrator*. In *International Teletraffic Conference (ITC)*, pages 318:1–318:4, Stockholm, Sweden, 1973.

[33] Schwederski, T. and M. Jurczyk: *Verbindungsnetze – Strukturen und Eigenschaften*. Springer, 1996.

[34] Seiferas, J.: *Sorting networks of logarithmic depth, further simplified*. Algorithmica, 53(3):374–384, March 2009.

[35] Waksman, A.: *A permutation network*. Journal of the ACM (JACM), 15(1):159–163, January 1969.

CNC-TRP-HC0-Batc68

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	8	96	6	8	0	2	0	0	4	8
4	14	576	20	24	0	4	0	0	12	60
8	23	2576	64	72	0	8	0	0	32	288
16	35	9696	192	208	0	16	0	0	80	1120
32	50	32640	544	576	0	32	0	0	192	3840
64	68	101632	1472	1536	0	64	0	0	448	12096
128	89	298752	3840	3968	0	128	0	0	1024	35840
256	113	840192	9728	9984	0	256	0	0	2304	101376
512	140	2281472	24064	24576	0	512	0	0	5120	276480
1024	170	6021120	58368	59392	0	1024	0	0	11264	732160

CNC-TRP-HC0-ChOr94

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	6	76	4	6	0	2	0	0	16	0
4	10	710	8	14	2	6	0	0	170	0
8	17	3416	16	32	8	16	0	0	836	0
16	27	12442	32	70	22	38	0	0	3070	0
32	40	39196	64	148	52	84	0	0	9712	0
64	56	112990	128	306	114	178	0	0	28066	0
128	75	306848	256	624	240	368	0	0	76340	0
256	97	798434	512	1262	494	750	0	0	198854	0
512	122	2011940	1024	2540	1004	1516	0	0	501464	0
1024	150	4944742	2048	5098	2026	3050	0	0	1233130	0

CNC-TRP-HC0-ChCh96

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	10	124	10	6	2	2	0	8	4	6
4	14	478	20	12	8	4	0	26	12	32
8	17	1546	42	24	24	8	0	72	32	120
16	20	4522	92	48	64	16	0	182	80	384
32	23	12402	206	96	160	32	0	436	192	1120
64	26	32522	464	192	384	64	0	1010	448	3072
128	29	82498	1042	384	896	128	0	2288	1024	8064
256	32	203962	2324	768	2048	256	0	5102	2304	20480
512	35	494002	5142	1536	4608	512	0	11244	5120	50688
1024	38	1176490	11288	3072	10240	1024	0	24554	11264	122880

CNC-TRP-HC0-JaSI17

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	6	76	4	6	0	2	0	0	4	6
4	9	364	8	12	4	4	0	0	12	36
8	13	1286	16	24	22	8	0	0	32	136
16	17	3952	32	48	80	16	0	0	80	432
32	23	11186	64	96	242	32	0	0	192	1248
64	31	29972	128	192	660	64	0	0	448	3392
128	39	77206	256	384	1686	128	0	0	1024	8832
256	49	193048	512	768	4120	256	0	0	2304	22272
512	61	471578	1024	1536	9754	512	0	0	5120	54784
1024	75	1130524	2048	3072	22556	1024	0	0	11264	132096

CNC-TRP-HC0-KoOr90

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	11	106	10	6	2	2	2	2	6	6
4	14	528	24	12	2	4	8	14	20	40
8	19	1898	56	24	2	8	22	52	56	160
16	24	5858	128	48	2	16	52	152	144	528
32	30	16568	288	96	2	32	114	394	352	1568
64	35	44292	640	192	2	64	240	954	832	4352
128	41	113830	1408	384	2	128	494	2216	1920	11520
256	46	284094	3072	768	2	256	1004	5012	4352	29440
512	52	693068	6656	1536	2	512	2026	11134	9728	73216
1024	57	1660112	14336	3072	2	1024	4072	24422	21504	178176

CNC-TRP-HC0-Nara94

n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	6	76	4	6	0	2	0	0	4	6
4	9	364	8	12	4	4	0	0	12	36
8	12	1288	16	24	24	8	0	0	32	136
16	18	3960	32	48	88	16	0	0	80	432
32	27	11208	64	96	264	32	0	0	192	1248
64	44	30024	128	192	712	64	0	0	448	3392
128	77	77320	256	384	1800	128	0	0	1024	8832
256	142	193288	512	768	4360	256	0	0	2304	22272
512	271	472072	1024	1536	10248	512	0	0	5120	54784
1024	528	1131528	2048	3072	23560	1024	0	0	11264	132096

Figure 5: Circuit Size and Depth for Split Modules with n Inputs/Outputs as Discussed in Section 3.1 using Different Binary Sorters

CNC-TRC-HC0-Batc68										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	2	5	75	5	4	0	2	0	0	8
4	4	11	518	18	16	0	4	0	0	60
8	8	20	2428	60	56	0	8	0	0	288
16	16	32	9336	184	176	0	16	0	0	1120
32	32	47	31792	528	512	0	32	0	0	3840
64	64	65	99680	1440	1408	0	64	0	0	12096
128	128	86	294336	3776	3712	0	128	0	0	35840
256	256	110	830336	9600	9472	0	256	0	0	101376
512	512	137	2259712	23808	23552	0	512	0	0	276480
1024	1024	167	5973504	57856	57344	0	1024	0	0	732160
CNC-TRC-HC0-ChOp94										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	56	4	2	0	2	0	0	12	0
4	8	654	8	6	2	6	0	0	158	0
8	15	3272	16	16	8	16	0	0	804	0
16	25	12090	32	38	22	38	0	0	2990	0
32	38	38364	64	84	52	84	0	0	9520	0
64	54	111070	128	178	114	178	0	0	27618	0
128	73	302496	256	368	240	368	0	0	75316	0
256	95	788706	512	750	494	750	0	0	196550	0
512	120	1990436	1024	1516	1004	1516	0	0	496344	0
1024	148	4897638	2048	3050	2026	3050	0	0	1221866	0
CNC-TRC-HC0-ChCh96										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	7	104	10	2	2	2	0	8	0	6
4	11	422	20	4	8	4	0	26	0	32
8	14	1402	42	8	24	8	0	72	0	120
16	17	4170	92	16	64	16	0	182	0	384
32	20	11570	206	32	160	32	0	436	0	1120
64	23	30602	464	64	384	64	0	1010	0	3072
128	26	78146	1042	128	896	128	0	2288	0	8064
256	29	194234	2324	256	2048	256	0	5102	0	20480
512	32	472498	5142	512	4608	512	0	11244	0	50688
1024	35	1129386	11288	1024	10240	1024	0	24554	0	122880
CNC-TRC-HC0-JaSI17										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	56	4	2	0	2	0	0	0	6
4	6	308	8	4	4	4	0	0	0	36
8	10	1142	16	8	22	8	0	0	0	136
16	14	3600	32	16	80	16	0	0	0	432
32	20	10354	64	32	242	32	0	0	0	1248
64	28	28052	128	64	660	64	0	0	0	3392
128	36	72854	256	128	1686	128	0	0	0	8832
256	46	183320	512	256	4120	256	0	0	0	22272
512	58	450074	1024	512	9754	512	0	0	0	54784
1024	72	1083420	2048	1024	22556	1024	0	0	0	132096
CNC-TRC-HC0-KoOp90										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	8	84	8	2	2	2	2	2	2	6
4	11	468	20	4	2	4	8	14	8	40
8	16	1746	48	8	2	8	22	52	24	160
16	21	5490	112	16	2	16	52	152	64	528
32	27	15704	256	32	2	32	114	394	160	1568
64	32	42308	576	64	2	64	240	954	384	4352
128	38	109350	1280	128	2	128	494	2216	896	11520
256	43	274110	2816	256	2	256	1004	5012	2048	29440
512	49	671052	6144	512	2	512	2026	11134	4608	73216
1024	54	1611984	13312	1024	2	1024	4072	24422	10240	178176
CNC-TRC-HC0-Nara94										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	56	4	2	0	2	0	0	0	6
4	6	308	8	4	4	4	0	0	0	36
8	9	1144	16	8	24	8	0	0	0	136
16	15	3608	32	16	88	16	0	0	0	432
32	24	10376	64	32	264	32	0	0	0	1248
64	41	28104	128	64	712	64	0	0	0	3392
128	74	72968	256	128	1800	128	0	0	0	8832
256	139	183560	512	256	4360	256	0	0	0	22272
512	268	450568	1024	512	10248	512	0	0	0	54784
1024	525	1084424	2048	1024	23560	1024	0	0	0	132096

Figure 6: Circuit Size and Depth for Split Modules with n Inputs/Outputs as Discussed in Section 3.2 using Different Binary Sorters

CNC-TRP-HC1-Batc68										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	12	328	14	20	0	6	0	0	16	28
8	18	1584	44	56	0	12	0	0	40	164
16	27	6464	136	160	0	24	0	0	96	720
32	39	23168	400	448	0	48	0	0	224	2672
64	54	75648	1120	1216	0	96	0	0	512	8896
128	72	230656	3008	3200	0	192	0	0	1152	27456
256	93	667648	7808	8192	0	384	0	0	2560	80128
512	117	1855488	19712	20480	0	768	0	0	5632	224000
1024	144	4990976	48640	50176	0	1536	0	0	12288	605184
CNC-TRP-HC1-ChOp94										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	10	288	10	16	0	6	0	0	48	8
8	14	1932	20	36	4	16	0	0	424	20
16	21	8480	40	80	16	40	0	0	1980	48
32	31	29572	80	172	44	92	0	0	7072	112
64	44	90696	160	360	104	200	0	0	21956	256
128	60	256588	320	740	228	420	0	0	62568	576
256	79	687056	640	1504	480	864	0	0	168332	1280
512	101	1768020	1280	3036	988	1756	0	0	434608	2816
1024	126	4415192	2560	6104	2008	3544	0	0	1087956	6144
CNC-TRP-HC1-ChCh96										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	14	384	22	16	4	6	0	16	16	24
8	18	1324	44	32	16	12	0	52	40	100
16	21	4020	92	64	48	24	0	144	96	336
32	24	11284	200	128	128	48	0	364	224	1008
64	27	30052	444	256	320	96	0	872	512	2816
128	30	77076	992	512	768	192	0	2020	1152	7488
256	33	192132	2212	1024	1792	384	0	4576	2560	19200
512	36	468340	4904	2048	4096	768	0	10204	5632	47872
1024	39	1121124	10796	4096	9216	1536	0	22488	12288	116736
CNC-TRP-HC1-JaSI17										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	10	288	10	16	0	6	0	0	16	24
8	13	1096	20	32	8	12	0	0	40	108
16	17	3500	40	64	44	24	0	0	96	368
32	21	10144	80	128	160	48	0	0	224	1104
64	27	27620	160	256	484	96	0	0	512	3072
128	35	71976	320	512	1320	192	0	0	1152	8128
256	43	181548	640	1024	3372	384	0	0	2560	20736
512	53	446512	1280	2048	8240	768	0	0	5632	51456
1024	65	1076276	2560	4096	19508	1536	0	0	12288	124928
CNC-TRP-HC1-KoOp90										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	15	348	22	16	4	6	4	4	20	24
8	18	1424	52	32	4	12	16	28	56	116
16	23	4724	120	64	4	24	44	104	144	416
32	28	13956	272	128	4	48	104	304	352	1296
64	34	38384	608	256	4	96	228	788	832	3712
128	39	100616	1344	512	4	192	480	1908	1920	10048
256	45	254796	2944	1024	4	384	988	4432	4352	26112
512	50	628604	6400	2048	4	768	2008	10024	9728	65792
1024	56	1519256	13824	4096	4	1536	4052	22268	21504	161792
CNC-TRP-HC1-Nara94										
n	depth	size	#NOT	#AND	#XOR	#OR	#HA	#FA	#MX	#SW
2	4	28	1	2	0	1	0	0	0	3
4	10	288	10	16	0	6	0	0	16	24
8	13	1096	20	32	8	12	0	0	40	108
16	16	3504	40	64	48	24	0	0	96	368
32	22	10160	80	128	176	48	0	0	224	1104
64	31	27664	160	256	528	96	0	0	512	3072
128	48	72080	320	512	1424	192	0	0	1152	8128
256	81	181776	640	1024	3600	384	0	0	2560	20736
512	146	446992	1280	2048	8720	768	0	0	5632	51456
1024	275	1077264	2560	4096	20496	1536	0	0	12288	124928

Figure 7: Circuit Size and Depth for Split Modules with n Inputs/Outputs as Discussed in Section 3.3 using Different Binary Sorters