Arithmetic Design for Permutation Groups

Tamás Horváth

Secunet AG, Im Teelbruch 116, 45219 Essen, Germany, email: horvath@secunet.de

Abstract. This paper investigates the hardware implementation of arithmetical operations (multiplication and inversion) in *symmetric* and *alternating groups*, as well as in *binary permutation groups* (permutation groups of order 2^r). Various fast and space-efficient hardware architectures will be presented. High speed is achieved by employing switching networks, which effect multiplication in one clock cycle (full parallelism). Space-efficiency is achieved by choosing, on one hand, proper network architectures and, on the other hand, the proper representation of the group elements. We introduce a non-redundant representation of the elements of binary groups, the so-called *compact representation*, which allows low-cost realization of arithmetic for binary groups of large degrees such as 128 or even 256. We present highly optimized multiplier architectures operating directly on the compact form of permutations. Finally, we give complexity and performance estimations for the presented architectures.

Keywords: permutation multiplier, switching network, destination-tag routing, sorting network, separation network, binary group, compact representation, secret-key cryptosystem, PGM.

1 Introduction

Several cryptosystems, such as RSA, elliptic curve systems, IDEA or SAFER, utilize operations in algebraic domains like polynomial rings or Galois-fields. Efficient implementations of the basic arithmetical operations in those domains have been extensively studied but not much attention has been spent to simpler constructs like permutation groups. Our research on permutation group arithmetic has been motivated by the implementation of a secret-key cryptosystem called PGM (Permutation Group Mapping) [11,12], which utilizes some generator sets, called *group bases*, for encryption.

Briefly, a *basis* for a permutation group G is an ordered collection $\beta = (B_0, B_1, \ldots, B_{w-1})$ of ordered subsets (so-called *blocks*) $B_i = (b_{i,0}, \ldots, b_{i,r_i-1})$ of G, such that each element $g \in G$ has a unique representation in form of a product $g = b_{0,x_0} \cdot b_{1,x_1} \cdots b_{w-1,x_{w-1}}$, where $b_{i,x_i} \in B_i$ and $0 \le x_i \le r_i - 1$. Thus, β defines a bijective mapping $\hat{\beta} : G \to X$ which assigns to each element $g \in G$ a unique vector $x = (x_0, \ldots, x_{w-1}) \in X$, where $X = \mathbb{Z}_{r_0} \times \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_{w-1}}$. Clearly, $|G| = |X| = r_0 r_1 \cdots r_{w-1}$. $\hat{\beta}^{-1}$ can be effected by means of permutation multiplications, whereas $\hat{\beta}$ involves finding the proper factors in β, and is hence called *factorization.* There is a huge amount of so-called *transversal* bases, for which factorization can be effected very efficiently by means of permutation multiplications and inversions. A pair of such, randomly chosen bases β_1 and β_2 for some group G (called the *carrier group*) form the key for PGM. The encryption of a cleartext message m is $c = \hat{\beta}_2(\hat{\beta}_1^{-1}(m))$. Decryption is performed in the same manner by exchanging the roles of β_1 and β_2, i.e. $m = \hat{\beta}_1(\hat{\beta}_2^{-1}(c))$. To accomodate the cryptosystem to some binary cleartext and ciphertext space $\mathcal{M} = \mathcal{C} = \mathbb{Z}_{2^k}$, an additional, fixed mapping $\lambda : \mathcal{M} \to X$ has to be effected prior to and λ^{-1} after the actual encryption.

It is very natural to represent permutations in a computer in the so-called *Cartesian* form. Section 2 introduces the basic principle of multiplying two Cartesian permutations in a switching network. Section 3 presents different, mostly novel multiplier architectures operating in the symmetric group. Unfortunately, a symmetric carrier group S_n has a serious drawback. It is namely that any basis for S_n $(n > 2)$ has several blocks with length $r_i \neq 2^{k_i}$. It follows that λ, which can be seen as a conversion from a binary to a *mixed* radix $r = (r_0, r_1, \ldots, r_{w-1})$, is computationally rather intensive [14].

As oppesed to a symmetric group, any basis for a *binary group* (a permutation group of order 2^r) has block lengths $r_i = 2^{k_i}$, and thus the mapping λ for such a carrier group is trivial. Since a binary group of degree n is only a small subgroup of the symmetric group S_n, the use of some large degree ($n = 128 \ldots 256$) is indicated, which makes the use of use of the multipliers, proper proper for symmetric groups, infeasible. The problem lies in the fact that the Cartesian representation of binary group elements contains a large amount of redundancy. In Sec. 4 we introduce a novel, non-redundant representation, the so-called *compact representation*, and present various multiplier architectures operating directly on the compact form of permutations. Finally, Sec. 5 gives complexity and performance estimations for the presented architectures. It turns out that a PGM system with a binary carrier group is indeed much more efficient than one based on a symmetric group of similar order.

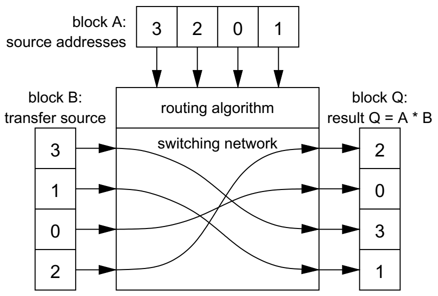
2 Multiplication in Permutation Networks

To briefly recall, a permutation p of degree n is a bijection $p : L \to L$, where L is a set of n arbitrary symbols or *points.* In the arithmetic we propose, elements of the symmetric group $\mathcal{S}_n$ are represented in the so-called *Cartesian* form. In this representation, $L = \{0, 1, \ldots, n-1\}$ and a permutation $p \in \mathcal{S}_n$ is a vector $p = (p(0), p(1), ..., p(n-1))$ of the n function values. Suppose now, elements of vector p are physically stored in their natural order in a block P of registers, i.e. $P[i] = p(i)$ for $0 \leq i \leq n-1$, where $P[i]$ denotes the content of the i^{th} register.

By definition, the product of two permutations a and b is permutation $q = a \cdot b : q(i) = b(a(i))$, for $0 \leq i \leq n-1$. Using the Cartesian representation, the product q can be computed by means of n memory transfer operations: $Q[i] := B[A[i]]$ for $0 \leq i \leq n-1$, where A, B and Q are the memory blocks storing a, b and q, respectively. For simplicity of the notation, we are not going to distinguish register blocks from their content, but simply write $Q = A \cdot B$ to denote the product.

By definition, the inverse of a permutation a is permutation $q = a^{-1}$, for which $a \cdot a^{-1} = a^{-1} \cdot a = \iota$, where ι denotes the *identity* permutation (i.e. $\iota(i) = i$). The inverse a^{-1} can be obtained in block Q by applying n memory transfers $Q[A[i]] := i$. We denote the inverse simply as $Q = A^{-1}$.

The memory transfers can be carried out either sequentially or in parallel. The former is the typical software implementation. The parallel implementation exploits the fact that the n memory transfers are completely independent and can thus be carried out simultaneously in *switching networks*, as follows. For multiplication, the $A[i]^{\text{th}}$ register of source block B is connected to the i^{th} register of the destination block Q, i.e. A is interpreted during *routing* as a vector of source addresses. After setting up the network, the content of B is copied to Q via the established connections, forming the product $A \cdot B$ in Q. Fig. 1a illustrates this principle on a small example.



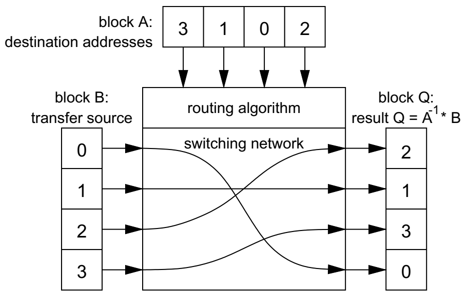


Fig. 1. a, Multiplication and **b,** inversion in a switching network

When interpreting A as a vector of destination addresses, the reverse connections are established. By copying then the content of B to Q, the product $A^{-1} \cdot B$ is obtained in Q. By substituting $B = \iota$, the network delivers the inverse of A, as shown in Fig. 1b.

3 Arithmetic in Symmetric Groups

According to the computing principles introduced above, a multiplier network for $\mathcal{S}_n$ should be an n-input, n-output network (briefly (n, n) network). For the sake of full parallelism, the network must be able to connect the n inputs to the n outputs simultaneously, that is without collision (or *blocking*) at any of the links. Since the network has to be completely re-routed after each multiplication, *rearrangeable networks* are favourable compared to the more complex *non-blocking* networks [3]. Moreover, since the routing operand may come from the entire symmetric group $\mathcal{S}_n$, the network must be able to realize all possible n-to-n connections. Exactly these characteristics describe a specific class of switching networks, called *permutation networks*.

3.1 Crossbar Networks

The *crossbar network* is the most fundamental *single stage* permutation network [3]. As depicted in Fig. 2, it consists of $n \times n$ switches in a matrix form, which pass the signals from input port C to output port Q. In the following, we propose three different schemes for fast routing of the network.

In the first routing scheme the pure crossbar network is equipped with a routing port A and with corresponding horizontal lines, each controlling an n-to-1 multiplexer (MUX), as shown in Fig. 2a. According to the control signal, each MUX selects one of the n signals of port C, and forwards it to port Q. Note in this mechanism that the data items entered at port A are used as source addresses. Hence, when entering Cartesian permutations at A and C, the network computes the product $Q = A \cdot C$.

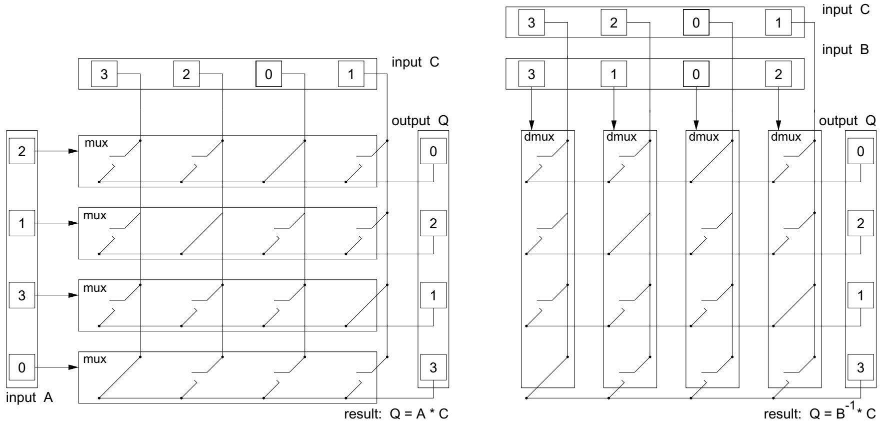


Fig. 2. a, MUX-type multiplier **b,** DMUX-type multiplier

The second routing scheme in Fig. 2b adds routing port B and corresponding vertical lines to the pure crossbar network. Each of these lines controls a 1-to-n demultiplexer (DMUX) which transmits the input signal through one of the n output lines towards Q, while disconnecting from all other output lines. Note that data items of A are interpreted in this mechanism as destination addresses. Accordingly, when entering Cartesian permutations at B and C, the network computes the product $Q = B^{-1} \cdot C$.

A combination of the above two routing schemes yields the third one of Fig. 3. Both routing ports A and B are included here, and are connected to horizontal and, respectively, vertical addressing lines. In addition, each switching cell is equipped with an **equivalence** comparator, which compares the addresses received from the neighboring addressing lines. If the addresses are equal, the comparator closes the switch, otherwise opens it. By entering Cartesian permutations at ports A, B and C respectively, the result obtained at port Q is the product $Q = A \cdot B^{-1} \cdot C$. A more detailed description of this architecture and of a bit-parallel realization has been published in [13].

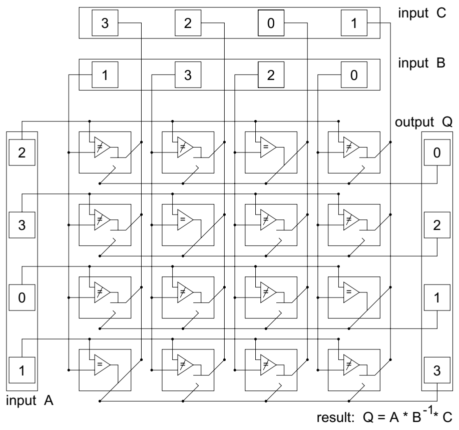


Fig. 3. A 3-operand crossbar multiplier

3.2 Sorting Networks

The *Beneš-network* [3,5] is known to be the most efficient rearrangeable multistage network topology, based on elementary (2,2) switching cells. However, its routing algorithm, the so-called *looping algorithm* [5], is intrinsically sequential and can only be effected in a *centralized control* unit. Accordingly, the mechanism is rather slow and thus not suitable for a multiplier network. On the other hand, there exists a large class of multistage networks, the so-called *digit-controlled* (or *delta*) *networks*, which possess a very convenient routing algorithm, the so-called *destination-tag routing* (or *self-routing*) [5,7,8,10]. This *distributed control* mechanism is very fast, since the individual cells decide independently and simultaneously. Unfortunately, delta networks are blocking ones.

A *sorting network* is an (n, n) multistage network effecting some deterministic sorting algorithm [2,4,6,9]. The network is built from elementary (2,2) *compare-exchange modules*. Each module compares the two incoming numbers and routes them according to their magnitudes. No matter in which order input numbers are entered at the input, the network applies the proper permutation to them and delivers the sorted sequence. Hence, any sorting network can be regarded as a rearrangeable permutation network.

Though all known sorting networks are more complex than the Beneš network, they offer a way for destination-tag routing, as follows. If entering elements $a(i)$ of a Cartesian permutation a at input $A[i]$ $(0 \leq i \leq n-1)$, the network forwards each $a(i)$ to output $Q[a(i)]$. Put another way, vector a carries routing information and designates n parallel paths from $A[i]$ to $Q[a(i)]$. When now attaching the elements of a permutation b to a, i.e. entering packets of the form $(a(i), b(i))$ at input line i, destination tag $a(i)$ will route $b(i)$ through the network towards $Q[a(i)]$, i.e. eventually $Q[a(i)] = b[i]$ is obtained. It is seen that the Cartesian permutation q obtained at Q is $q(a(i)) = b(i)$, or equivalently, $a \cdot q = b$, and thus $q = a^{-1} \cdot b$, which fact amounts to the general multiplication principle of Sec. 2. Figure 4 illustrates the method on a small example.

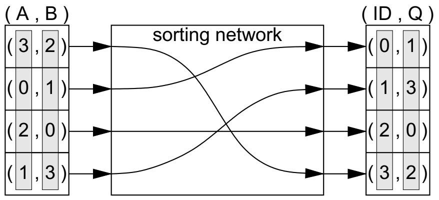


Fig. 4. Multiplication in a sorting network

In Fig. 5 we introduce two classical sorting networks. The *odd-even transposition sorter* is the parallel implementation of the insertion sort and, at the same time, of the selection sort algorithms. The n input numbers are sorted in n stages, comprising $n(n-1)/2$ modules. Accordingly we say that the network has depth n and complexity $n(n-1)/2$. The arrows in the symbols of the compare-exchange modules indicate the direction which the larger numbers are forwarded to. Note that this network has a completely "straight" wiring topology, which is advantageous in view of wiring area. The *bitonic sorter* as well as *Batcher's odd-even sorter* [9] are known to be the most efficient regular topologies, having $O(\log^2 n)$ stages in a recursive structure. Note that many lines cross between certain stages, which is in direct correspondence with the wiring area.

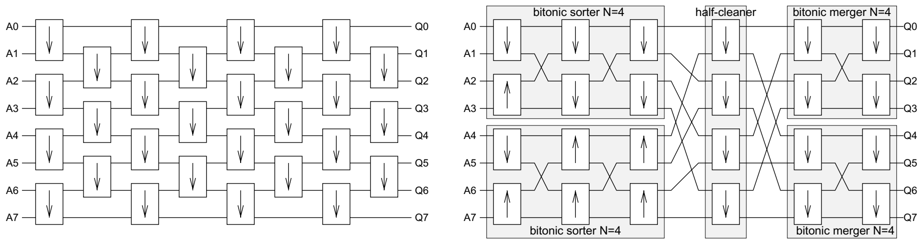


Fig. 5. a, The odd-even transposition and **b,** the bitonic sorters

In a straightforward realization of the compare-exchange modules, comparison is carried out first, and the result is then used to set the switches. In this method, no data can be transferred until the comparison is completed. Considerable acceleration can be achieved by recognizing that comparison can be performed sequentially, scanning from the MSBs towards the LSBs of the input numbers X and Y, according to the following algorithm:

1. As long as $X_i = Y_i$ while scanning bits in decreasing order of i, it does not matter how the switch is set, and thus X_i and Y_i can be passed to the next stage;
2. as soon as difference is noticed at bit j, i.e. $X_j \neq Y_j$, all switches for bits $i \leq j$ can be set to the same state, which is determined by the relation of X_j and Y_j.

In the improved scheme, corresponding bits X_i and Y_i are transmitted to the next stage immediately after their comparison, that is before the comparison of lower bits is completed. The higher order bits reach the next stage therefore earlier than the lower order ones, where their comparison starts immediately. In this way, each comparator stage delays the destination tag effectively only by the time of a single bit-comparison. For implementation details we refer to [14].

3.3 Separation Networks

Sorting networks are able to sort arbitrary number sequences. Note however that Cartesian permutations are special sequences, such that each number of the range $0 \ldots n-1$ occurs exactly once. This kind of sequence we call a *permutation sequence*. In the following, we introduce a class of novel permutation network architectures, which exploit this property to reduce hardware complexity. The new networks employ the *radix sorting* algorithm [1] for routing: destination addresses are represented as binary strings, starting with the MSB as first letter. Sorting proceeds as follows: first the strings starting with a '1' as first letter are separated from those starting with a '0'. As second step, both of the resulting subsequences are further be split up so that strings having '1' as second letter get separated from those having '0' at the same position. The 'divide-and-conquer' principle is followed in this way till the last step, where strings with trailing '1' are separated from those with trailing '0'.

Since a permutation sequence contains a predetermined set of strings, the number of strings with '1' and respectively '0' at any particular position is constant, irrespective of the actual sequence. Due to this fact, the length of the separated subsequences is known and constant for all separation steps. In the specific case of $n = 2^m$, all separated subsequences are *balanced*, i.e. contain exactly as many 1's as 0's at any particular position. This property is the basis for the design of *separator networks*. Each separation step is effected in a dedicated *separator stage*. The first separator stage splits the input sequence in two halves of length $n/2$ (without actually achieving perfect ordering), the next stage produces subsequences of length $n/4$, and so on. Networks of degree $n \neq 2^m$ can be constructed by omitting parts of a network of degree 2^m, where $n < 2^m$.

The strength of the technique lies in the fact that any particular stage can achieve the separation by looking at corresponding single bits of the destination tags. The method can thus be considered as the generalization of the bit-controlled self-routing algorithm for permutation networks. Interestingly, comparing corresponding bits X and Y of two destination tags and routing them towards the proper output H ("higher" value) and respectively L ("lower" value) requires no logic at all. To see this, consider the truth-table of the "binary" compare-exchange module:

X	Y	switch state required	switch state chosen	H	L
0	0	don't care	across	0	0
0	1	across	across	1	0
1	0	straight	straight	1	0
1	1	don't care	straight	1	1

By choosing the switch state for *don't care*'s as shown above, the switches can be controlled directly by input bit X, whereas H and L can be formed by a single OR- and respectively AND-gate. See [14] for implementation details.

In the following, we present a couple of novel separator network architectures. The first scheme is related to the bitonic sorter (Fig. 5b). The two bitonic sorters of length $n/2$ and a *half-cleaner stage* of this network form a so-called *selection network* [9], which separates the $n/2$ largest from the $n/2$ smallest elements. If entering a sequence of $n/2$ 1's and $n/2$ 0's, the selection network separates the 1's from the 0's. Clearly, this is also achieved when the magnitude comparator modules are replaced with "binary" comparator modules. Such *bitonic separator* stages can be used to build a *bitonic separator network*, as illustrated in Fig. 6 for $n = 8$. The network has depth of order $O(\log^3 n)$.

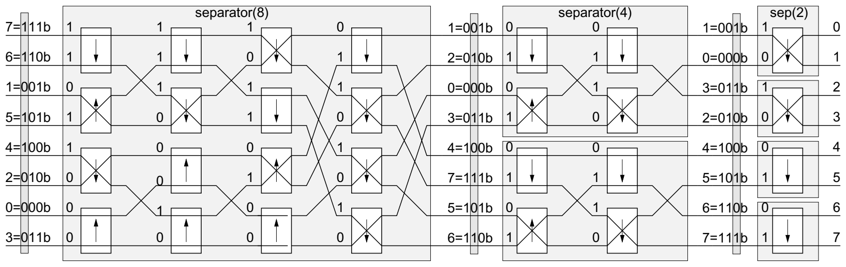


Fig. 6. A separator network based on bitonic separators

Note that the (n/2,n/2)-sorters in front of the half-cleaner can actually be replaced by any kind of sorting network, for instance, by odd-even transposition sorters. We call the network obtained in this way the *linear odd-even separator network*. As the name suggests, it has depth of order $O(n)$.

Separator stages can rely on other principles, too. The sorter depicted in Fig. 7 employs a novel separator type of depth $O(n)$, which we call a *diamond separator*. The underlying sorting principle is similar to that of the odd-even transposition sorter. The advantage of this architecture is the completely straight wiring pattern. Its drawbacks are that the network is rather deep and hard to lay-out in a rectangular form.

The *rotation separator* offers lower depth, rectangular layout and still a "nearly" straight wiring topology. The separation principle can be followed in Fig. 8. Links running across the network are considered to be of two types: *0-lines*, which are expected to deliver 0's at the output, and *1-lines*, that should deliver 1's. A '1' on a 0-line (and a '0' on a 1-line, respectively) is considered as a *1-error* (a *0-error*, respectively). Due to the balance in a permutation sequence of length $n = 2^m$, 1-errors are present in the same number as 0-errors at any particular stage. Each compare-exchange module receives input from a 0-line and a 1-line, and outputs to a 0-line and a 1-line. When a 1-error and a 0-error are received, they "neutralize" each other, i.e. both errors disappear.

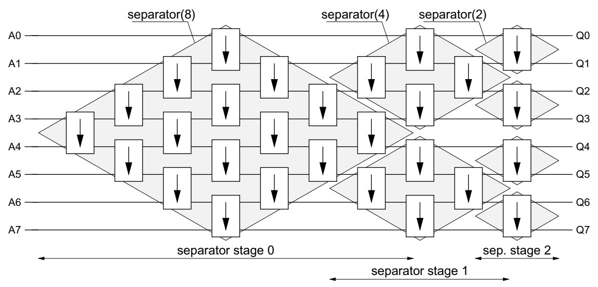


Fig. 7. The "diamond" separator network with 8 inputs

The topology of the network implements the following strategy for eliminating all errors in the input sequence: 0-lines (carrying potentially 1-errors) are iteratively "rotated around" and combined pairwise with 1-lines (carrying potentially 0-errors). In order for all 0-lines to be combined with all 1-lines, $n/2$ rotation steps are needed, and hence the separator stage has depth $n/2$. The total depth of the entire network is $n-1$.

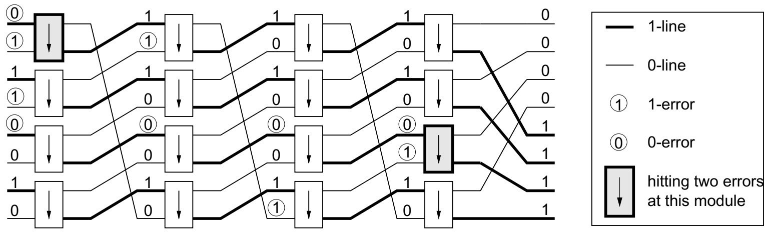


Fig. 8. A "rotation" separator stage with 8 inputs

4 Arithmetic in Binary Groups

As mentioned, any binary group of degree n is a subgroup of S_n. Unfortunately, even a so-called Sylow-2 subgroup $\mathcal{H}_s$ of $\mathcal{S}_n$, which is of maximal order, is rather small; it has order $|\mathcal{H}_s| = 2^{n-1}$ if $n = 2^s$. Hence, if a certain group size is required, the usage of a binary group of some large degree is indicated. For instance, if a group order of at least 2^{127} is required, not unusual in cryptographic applications, either a symmetric group of degree $n = 34$ or a Sylow-2 subgroup of degree $n = 128$ may be chosen. Unfortunately, the storage of Cartesian permutations (7∗127=896 bits in the above example) as well as the multipliers based on permutation networks are very extensive for binary groups of such large degrees. Note however that the Cartesian form is very redundant for representing binary group elements, and that the multiplier networks would be used rather inefficiently too, because most of the possible permutation patterns, namely thus in $\mathcal{S}_n$ but not in $\mathcal{H}_s$, would never be configured.

A study of the *indirect binary cube* (IBC) network for $n = 2^s$ has shown that though the set of permutations realized by the network is not a group, it embeds a Sylow-2 subgroup $\mathcal{H}_s$ of $\mathcal{S}_n$. Similar results can be obtained for the "inverse" of the IBC network, the so-called *generalized cube* (also called *butterfly* or *SW-Banyan*) network, as well as for other (n, n) delta networks, such as the *omega*, the *baseline*, the *modified data manipulator* MDM and their respective "inverses", the *reverse omega* (also called *flip*), the *reverse baseline* and the *inverse* MDM networks [5,10]. The different delta networks realize various instances of $\mathcal{H}_s$, while it is known that all Sylow-2 subgroups $\mathcal{H}_s$ of $\mathcal{S}_n$ are isomorphic.

A delta network of degree $n = 2^s$ comprises $s * n/2$ switches in s stages, and is thus considerably more efficient for $\mathcal{H}_s$ then any of the permutation networks. The construction of a multiplier we illustrate on the IBC network of degree $n = 8$, depicted on the left of Fig. 9. The network contains 12 binary switches and since it is a banyan network (i.e. there is one unique path from each input to each output), all of the 2^{12} different configurations realize different permutations. This permutation set of size 2^{12} is not a group, but it contains $\mathcal{H}_3$, which is of order 2^7. Clearly, some configurations will never be used if working in $\mathcal{H}_3$.

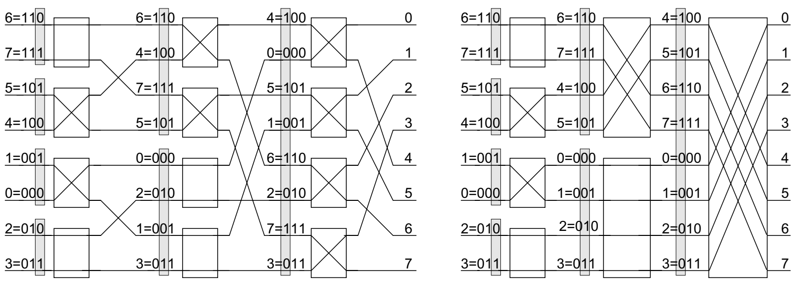


Fig. 9. The indirect binary cube network for $n = 8$ in two presentations

As illustrated in the figure, the IBC network can be configured by the bit-controlled self-routing algorithm, where switches in the first stage are controlled by the LSBs, while subsequent stages by succeeding bits of the destination tags. If first routing with a Cartesian permutation $a \in \mathcal{H}_3$ and then transferring another permutation $b \in \mathcal{H}_3$, the network delivers, according to the general multiplication principle of Sec. 2, the product $q = a^{-1} \cdot b$.

It turns out that if working in $\mathcal{H}_3$, switches of certain switch groups are always set to a common state, and can thus be unified in one *switching module*, as depicted on the right side of the figure. The unified switches can be controlled by one common signal, which sets either a "straight" or a "swapping" connection pattern. We call an IBC network with unified switches an UIBC network. The 2^{n-1}=2^7 different connections patterns of the UIBC network realize exactly the elements of (a specific instance of) $\mathcal{H}_3$.

The control bits can be extracted from the Cartesian permutation a by simply selecting certain bits of a. Actually, the n−1=7 control bits can be seen as a special representation of the group elements of $\mathcal{H}_3$, which we call the *compact representation.* From the fact $|\mathcal{H}_s| = 2^{n-1}$ it is seen that the compact representation is non-redundant and hence optimal. Expanding the compact form to the Cartesian form is similarly simple, it can achieved by reproducing (copying) certain bits of the compact permutation. Note that the ease of the conversions is not a general feature but specific to the instance of H_s induced by the use of the UIBC network.

A great advantage of the compact representation is that it allows space-efficient storage of elements of $\mathcal{H}_s$. Note furthermore that since the Cartesian form of $b \in \mathcal{H}_s$ is redundant, more bits than actually necessary are transferred by the UIBC network while multiplying. By removing links and switching components from the network which convey redundant bits of b, the complexity of the scheme can be significantly reduced. The optimized scheme transmits merely the compact form of b. The resulting multiplier network, called MULAIB, is shown in Fig. 10 for $n = 8$.

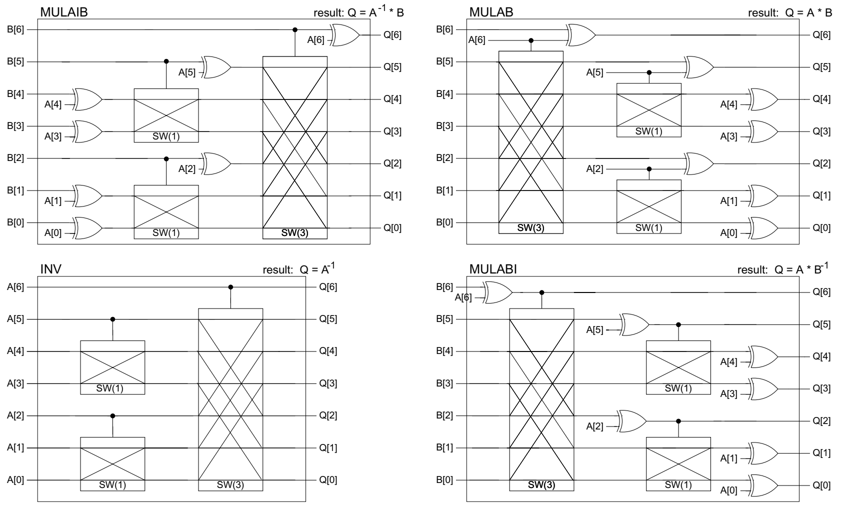


Fig. 10. Different multipliers and inverters working on compact permutations

Figure 10 illustrates further multiplier and inverter architectures deduced from the IBC network and respectively, from its inverse, the generalized cube network. All architectures work directly on the compact form of operands.

Above we followed an illustrative approach to introduce the arithmetic for binary groups. An accurate description of the construction of group $\mathcal{H}_s$ underlying the arithmetic, a formal definition of the compact representation, proofs of the multiplication algorithms, further multiplier and inverter schemes as well as a generalization of the theory to a large class of binary groups of arbitrary degree n have been omitted here in lack of space, but can be found in [14].

5 Conclusions

In the following, we give complexity and performance estimations for the presented multiplier architectures. The examined multipliers operate in the symmetric group S_{32} and in the binary group H_7 (degree $n = 128$), which have comparable orders: $|\mathcal{S}_{32}| \approx 2^{117}$ and respectively $|\mathcal{H}_7| = 2^{127}$. Estimations of complexity and delays have been made for the $0.7\,\mu$m ES2 standard-cell CMOS technology of European Silicon Structures. The complexity of the typically extensive wiring of switching networks, indicated also by the measure *wiring width*, has been taken into account. The throughput of the networks has been calculated for a purely combinational, full-parallel, non-pipelined implementation. The estimation methodology as well as other implementation styles are detailed in [14]. Table 1 below summarizes the results.

Table 1. A comparison of multipliers for S_{32} and $\mathcal{H}_7$

	Topology		Complexity			Performance		
Multiplier Design	depth	# sw. modules	gate count	wiring width	area mm^2	gate delay	perf. MMPS	perf./ area
MUX-type crossbar	1	1024	8.68K	640	11.8	9	222	18.8
DMUX-type crossbar	1	1024	12.7K	640	15.8	12	167	10.5
3-operand crossbar	1	1024	25.0K	640	27.6	12	167	6.04
Linear oev. sorter	32	496	21.5K	0	16.3	72	27.8	1.70
Bitonic sorter	15	240	10.4K	820	14.4	38	52.6	3.65
Linear oev. sep.net	35	560	8.47K	526	10.2	71	28.2	2.76
Bitonic sep.net	25	400	5.95K	1030	10.7	51	39.2	3.67
Diamond sep.net	46	496	7.54K	0	5.73	93	21.5	3.75
Rotation sep.net	31	496	7.54K	526	9.26	63	31.7	3.43
MULAIB	7	756	1.61K	240	3.17	15	133	42.0
MULAB	7	756	1.61K	240	3.17	19	105	33.1
MULABI	7	756	1.61K	240	3.17	44	45.5	14.4
INV	7	756	1.33K	240	2.82	13	154	54.6

Among the multipliers for $\mathcal{S}_{32}$, the crossbar architectures are very fast and cost-effective too. The bitonic separator network and the rotation separator network perform quite similarly, and are slightly smaller and slower than the well-known bitonic sorter. All multipliers for $\mathcal{H}_7$ have extremely low gate-complexity, whereas about 60 % of the total area is spent for global wiring in all designs. The reason that MULAB performs significantly worse than MULAIB is that control signals, that are to be distributed at a particular stage, are produced by the preceeding stage. Therefore, the delay of signal distribution adds to the total delay at each stage, a rather undesirable phenomenon.

To summarize, the multipliers for the binary groups outperform those for the symmetric group and because of their $O(n\log n)$ complexity, the gain becomes even more striking for larger groups. We stress here again that the very fundamental invention which allows both space-efficient storage and efficient computation in binary groups is that of the *compact representation.*

References

1. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullmann:
The Design and Analysis of Computer Algorithms
Addison-Wesley, 1974.
2. Selim G. Akl: Parallel Sorting Algorithms
Academic Press, 1985.
3. V. E. Beneš, Mathematical Theory of Connecting Networks and Telephone Traffic, Academic Press, 1965
4. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest:
Introduction to Algorithms, MIT Press, 1990.
5. Tse-yun Feng, A Survey of Interconnection Networks, *IEEE Computer* , December 1981, pp 12–27.
6. Alan Gibbons, Wojciech Rytter: Efficient Parallel Algorithms
Cambridge University Press, 1988.
7. A.J. van de Goor: Computer Architecture and Design
Addison-Wesley, 1989.
8. J.P. Hayes: Computer Architecture and Organization
McGraw-Hill, 1988.
9. D. Knuth: The Art of Computer Programming
Volume III: Sorting and Searching, Addison-Wesley, 1973.
10. C. P. Kruskal, M. Snir: "A Unified Theory of interconnection Network Structure", Theoretical Computer Science, Volume 48, 1986.
11. S. S. Magliveras, A cryptosystem from logarithmic signatures of finite groups, In *Proceedings of the 29'th Midwest Symposium on Circuits and Systems*, Elsevier Publishing Company (1986), pp 972–975.
12. S. S. Magliveras and N. D. Memon, Algebraic Properties of Cryptosystem PGM, in *Journal of Cryptology*, **5** (1992), pp 167–183.
13. T. Horváth, S. Magliveras, Tran van Trung, A Parallel Permutation Multiplier for a PGM Crypto-chip, *Advances in Cryptology - CRYPTO'94*, Springer-Verlag 1994, pp 108–113.
14. T. Horváth, Secret-key Cryptosystem TST, *Ph.D. thesis, Institut for Experimental Mathematics, University of Essen, Germany, to be published in 1999.*