Implementing an arbitrary reversible logic gate

2005 J. Phys. A: Math. Gen. 38 3555

(http://iopscience.iop.org/0305-4470/38/16/007)

INSTITUTE OF PHYSICS PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND GENERAL

J. Phys. A: Math. Gen. **38** (2005) 3555–3577 doi:10.1088/0305-4470/38/16/007

Implementing an arbitrary reversible logic gate

Yvan Van Rentergem[1], **Alexis De Vos**[1] **and Leo Storme**[2,3]

[1] Imec v.z.w. and Vakgroep elektronika en informatiesystemen, Universiteit Gent, Sint Pietersnieuwstraat 41, B-9000 Gent, Belgium
[2] Vakgroep zuivere wiskunde en computeralgebra, Universiteit Gent, Krijgslaan 281, B-9000 Gent, Belgium

Received 14 September 2004, in final form 25 February 2005
Published 6 April 2005
Online at stacks.iop.org/JPhysA/38/3555

Abstract

The $(2^w)!$ reversible logic gates of width w, i.e. reversible logic gates with w inputs and w outputs, together with the action of cascading, form a group **R**. We define a subgroup **K**, consisting of the SWITCHED gates. There are $[(2^{w-1})!]^2$ such gates. They partition the group **R** into $2^{w-1}+1$ double cosets. This allows us to decompose an arbitrary reversible gate into the cascade of three gates: a SWITCHED gate, an upside-down simple control gate and a second SWITCHED gate. We present an algorithm to perform this factorization, and thus provide a method of implementing an arbitrary reversible gate into hardware. The algorithm can be used to automate the implementation of a reversible function in future (c-MOS) technologies, realizing low-cost computing.

PACS numbers: 02.10.Ab, 02.20.−a, 03.67.Lx, 84.30.Bv

1. Introduction

Reversible computing [1] is useful both in classical and in quantum computing. Originally, Landauer and Bennett [2–5] introduced reversible computing in order to avoid the generation of the amount $kT\log(2)$ of heat, each time a bit of information is lost during a computational process. By avoiding any energy loss associated with logical irreversibilities, classical reversible circuits can, in principle, be made asymptotically lossless [6, 7], in the limit of arbitrarily large time delays. Conversely, reversible computing can be applied as part of a quantum computer [8–11]. Indeed, most gate libraries for quantum computation consist, for a large part, of (classical) reversible building blocks [12]. In the present paper, we will focus on the application of reversible computing in low-power digital electronics. Because of frictional phenomena, each computational step is accompanied by a heat generation Q of the order CV_t^2, where C is the capacitance of the logic gate and V_t the threshold voltage of its switches (e.g. its transistors). We can write C as $\epsilon_0\epsilon A/t$ where ϵ_0 is the vacuum permittivity and ϵ is the dielectric constant of the insulating material. Furthermore, A is the surface area of the

[3] The third author thanks the Fund for Scientific Research—Flanders (Belgium) for a Research Grant.

logic gate and t the dielectric's thickness. With present-day orders of magnitude ($\epsilon \approx 5$, $A \approx (100\ \text{nm})^2$, $t \approx 10$ nm, and $V_t \approx 0.3$ V), we get $Q \approx 4 \times 10^{-18}$ J, i.e. 4 attojoules. This is three orders of magnitude larger than the Landauer quantum, which (at $T = 300$ K) amounts to 3×10^{-21} J, i.e. 3 zeptojoules. If Moore's law will continue to be valid in the near future, a further reduction of Q will make the Landauer effect non-negligible and will make reversible computer architectures attractive.

For the study of reversible computing, we consider all reversible logic gates of width w, i.e. with w binary inputs $A_1, A_2, \ldots, A_w$ and w binary outputs $P_1, P_2, \ldots, P_w$. The truth table of such logic gate corresponds to a permutation of the 2^w rows $(0, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 1), \ldots, (1, 1, \ldots, 1, 1)$ of the table. Therefore, there is a one-to-one relationship between the words $(A_1, A_2, \ldots, A_w)$ and the words $(P_1, P_2, \ldots, P_w)$. This allows us to write the inverse truth table of a given reversible truth table. The inverse logic gate 'undoes' what the original gate does. In other words, whereas a given reversible gate is calculating 'forwards', its inverse calculates 'backwards'.

Each reversible logic gate having its own inverse, the reversible logic gates form a group (which we will denote by $\mathbf{R}$) isomorphic to the symmetric group $\mathbf{S}_{2^w}$. The order of this group is

$$r(w) = (2^w)!.$$

Given a particular reversible gate (either by its truth table or by its permutation), the question arises how to implement it into hardware. We distinguish only two kinds of hardware:

- 'active' devices, in particular switches, and
- 'passive' devices, in particular interconnections, i.e. wirings.

We will assume that the passive devices are free of cost, whereas the active ones have a price. For example in low-power electronics, an interconnection is merely a metal wire, whereas a switch needs two silicon transistors. There are basically two reasons for these simplifying assumptions:

- First, there is the hardware cost, i.e. fabrication cost of the chip, which is proportional to the silicon area. Simple metal wirings need less silicon area than transistors.
- Next, there is the operation cost, i.e. the power cost during use of the chip. A metal interconnection can be approximated by an RLC transmission line. According to the Athas principle [13], the power dissipation (and thus the heat generation) per computational step within such line can be made arbitrarily small, provided we accept sufficient delay time. A transistor however, because of its threshold voltage V_t cannot be regarded as an ideal switch with an RC load. Therefore, the heat dissipation within it is at least CV_t^2, however slowly we perform the elementary computation step.

In modern and future deep-submicron logic, interconnections become more significant, both in length (and thus in fabrication cost) and in power cost. However, it turns out that the number of metal wires is closely related to the number of transistors. In pass-transistor logic design, the number of interconnections is proportional to the number of transistors. Therefore, even in the deep-submicron world (i.e. the nano world), the transistor count is a valuable cost function.

We will assume three axioms:

- Each Boolean variable X is accompanied by its inverse $\overline{X}$ ($= \texttt{NOT}\ X$). We say we use 'dual line logic'.
- A variable and its inverse can be interchanged or swapped (at no cost).
- Two different variables X and Y can be interchanged (at no cost).

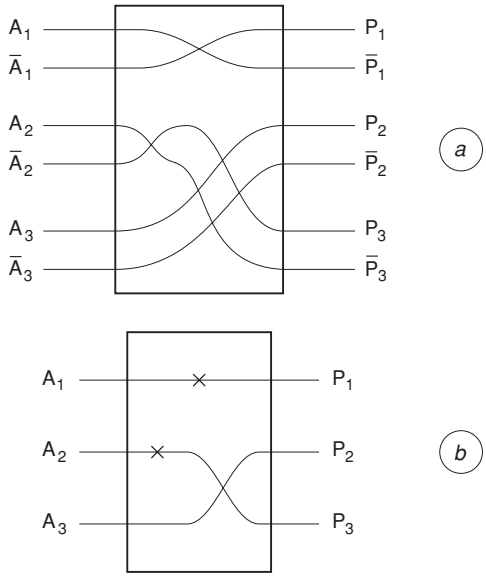

Figure 1. A selective gate of width 3: (*a*) implementation, (*b*) schematic.

Table 1. Truth table of two reversible gates of width 3: (a) selective gate, (b) simple control gate.

(a) Selective gate		(b) Simple control gate	
$A_1A_2A_3$	$P_1P_2P_3$	$A_1A_2A_3$	$P_1P_2P_3$
0 0 0	1 0 1	0 0 0	0 0 1
0 0 1	1 1 1	0 0 1	0 0 0
0 1 0	1 0 0	0 1 0	0 1 1
0 1 1	1 1 0	0 1 1	0 1 0
1 0 0	0 0 1	1 0 0	1 0 0
1 0 1	0 1 1	1 0 1	1 0 1
1 1 0	0 0 0	1 1 0	1 1 1
1 1 1	0 1 0	1 1 1	1 1 0

As a consequence, all selective reversible gates can be implemented at no cost. We call a reversible gate selective iff each of its outputs P_j equals either an input A_k or the inverse $\overline{A_m}$ of an input. Table 1 shows an example: the truth table of the selective gate obeying $P_1 = \overline{A_1}$, $P_2 = A_3$, and $P_3 = \overline{A_2}$. We see how the output rows are the permutation `(1, 7, 4, 6) (2, 5, 3, 8)` of the input rows. Figure 1 shows its hardware implementation, as well as its short-hand schematic. Note that a small cross indicates an inversion or `NOT` gate.

Thus the implementation (or synthesis) problem is solved for these gates. They form a subgroup of order

$$i(w)e(w) = 2^w w!.$$

Indeed, this subgroup of $\mathbf{R}$ is the semi-direct product of the subgroup of inverters and the subgroup of exchangers [14] and therefore is isomorphic to $\mathbf{S}_2^w : \mathbf{S}_w$. Table 2 shows the number of selective gates, the number of inverters and the number of exchangers. For the exchangers, we use the semicolon notation [15]. For example, the permutation `(1;2)` denotes the exchange

Table 2. Some special types of reversible logic gates, the number p of different gates of each type and the maximum number s of switches in the implementation of such gate.

Gate type	p	s
Inverter	2^w	0
Exchanger	$w!$	0
Selective gate	$w!2^w$	0
Simple control gate	$2^{2^{w-1}}$	$(w-1)2^w$
Compact control gate	$2^{w-1}+1$	$4(w-1)$
Arbitrary reversible gate	$(2^w)!$	$\frac{4}{3}(4^w-3w-1)$

(or swap) of bits A_1 and A_2. Thus `(1;2)` is a particular permutation of the w columns of the truth table. As any reversible gate of width w corresponds to a permutation of the 2^w rows of its truth table, the column permutation `(1;2)` corresponds to a particular row permutation. Indeed, we have

`(1;2) = (`2^{w-1}`-`2^{w-2}`+1,`2^{w-1}`+1) (`2^{w-1}`-`2^{w-2}`+2,`2^{w-1}`+2) ... (`2^{w-1}`,`2^{w-1}`+`2^{w-2}`).`

The special exchanger `(1;2;...;w)` we will call the cyclic exchanger and denote by z.

Now, the synthesis problem has to be solved for all remaining reversible gates. We will proceed by induction: we will assume that we have an algorithm for implementing an arbitrary reversible gate of width $w-1$ and will construct an algorithm for implementing an arbitrary reversible gate of width w. There exist only two reversible gates of unitary width: the follower and the inverter. Both can be implemented into hardware. This fact guarantees that the recursive construction can be completed in a finite number of steps.

2. Simple control gates

The subgroup of simple control gates was introduced by De Vos, Storme and Desoete [15, 16], in the framework of ultra-low power electronics:

$$P_i = A_i \qquad \text{for all} \quad i \in \{1, 2, \ldots, w-1\}$$
$$P_w = f(A_1, A_2, \ldots, A_{w-1}) \ \texttt{XOR} \ A_w,$$

where f denotes an arbitrary Boolean function of $w-1$ Boolean arguments. The reader can easily verify that, if we cascade two identical such gates, all outputs equal their corresponding inputs. This proves that such a gate is reversible and moreover equals its own inverse.

Table 1b shows an example for $w = 3$ and $f(A_1, A_2) = \overline{A_1} + A_2$, i.e. where the control function f is the `OR` of the bits $\overline{A_1}$ and A_2. The permutation, performed by the gate, is `(1,2)(3,4)(7,8)`. Figure 2 shows its schematic implementation, as well as its hardware implementation. The latter contains eight switches. Switches with a label X are closed iff $X = 1$. Note that each switch needs either three or four metal interconnects, according to its connection (either in series or in parallel). As there are an equal number of series and parallel wirings, the interconnect count equals $\frac{7}{2}$ of the switch count.

Note that a small square in the schematic indicates a control. In the special case that $f(A_1, A_2, \ldots, A_{w-1})$ equals the `AND` function $A_1 A_2 \ldots A_{w-1}$, we speak about the `CONTROLLED`$^{w-1}$ `NOT` gate. Its permutation is `(`2^w`-1,`2^w`)`. In this particular case, the little squares in figure 2(b) are replaced by little circles.

These simple control gates form a group. As there exist 2^{2^q} different Boolean functions of q Boolean variables, the group is of order $2^{2^{w-1}}$ (isomorphic to $\mathbf{S}_2^{2^{w-1}}$). Any simple control gate can be implemented with the help of $(w-1)2^w$ switches (or less).

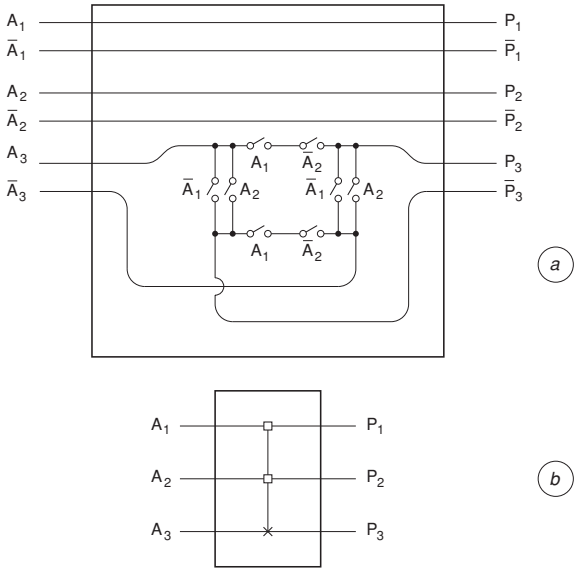

Figure 2. A simple control gate of width 3: (*a*) implementation, (*b*) schematic.
Note. For sake of clarity, the 'vertical' interconnections between the controlling lines (A_1, $\overline{A_1}$, A_2 and $\overline{A_2}$) and the controlled switches are not shown explicitly in (*a*).

We define here a special class of simple control gates: the compact control gates. A simple control gate is compact iff its control function is compact. A compact Boolean function of q Boolean arguments can be written as a combination of ORs and ANDs with only q letters. See appendix A. As a consequence, such gate can be implemented with only $4(w-1)$ switches (or less). We note that the compact control gates do not form a subgroup of the group of simple control gates. They merely form a set of generators of this group. Table 2 shows the number of simple control gates, the number of compact control gates as well as their respective switch counts.

We conclude this section by introducing 'upside-down simple control gates'. These gates very much resemble the above-described simple control gates. However, not the lowermost bit (i.e. A_w) is controlled by the others (i.e. by $A_1, A_2, \ldots,$ and A_{w-1}), but the uppermost bit (i.e. A_1) is controlled by the others (i.e. by $A_2, A_3, \ldots,$ and A_w):

$$P_1 = f(A_2, A_3, \ldots, A_w) \text{ XOR } A_1$$
$$P_i = A_i \qquad \text{for all} \quad i \in \{2, 3, \ldots, w\}.$$

The reader can easily verify that these form a group, conjugate to the group of simple control gates. Indeed any upside-down simple control gate can be written as the cascade zcz^{-1}, where c is a simple control gate and z is the cyclic exchanger.

3. SWITCHED gates

We define a SWITCHED gate of width w (with binary inputs $A_1, A_2, \ldots, A_w$ and binary outputs $P_1, P_2, \ldots, P_w$) as follows:

> The output bit P_1 equals the input bit A_1, whereas $P_2, P_3, \ldots, P_w$ are the outputs of some reversible gate of width $w-1$ with inputs $A_2, A_3, \ldots, A_w$: if $A_1 = 0$ then of gate g' else of gate g''.

Figure 3(*a*) shows the schematic representation, for the example $w = 4$.

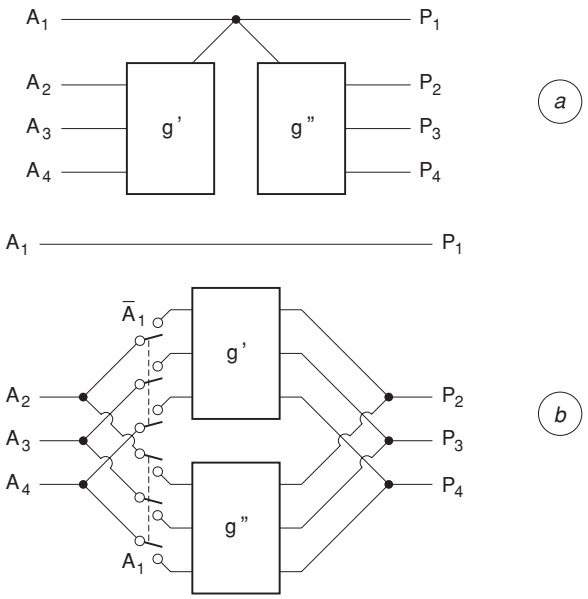

Figure 3. SWITCHED gate of width 4: (*a*) schematic, (*b*) layout.

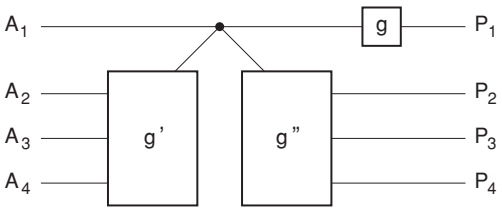

Figure 4. A 4-bit SWITCHED gate, followed by an arbitrary 1-bit reversible gate.

As there exist $(2^{w-1})!$ different reversible gates of width $w-1$, there exist

$$s(w) = [(2^{w-1})!]^2$$

different SWITCHED gates of width w. The cascade of two SWITCHED gates (one with subgates g_1' and g_1'', the other with subgates g_2' and g_2'') is itself a SWITCHED gate (with subgates $g_1'g_2'$ and $g_1''g_2''$). Therefore the SWITCHED gates form a group, which we will denote by **K**. It is isomorphic to the direct product $\mathbf{S}_{2^{w-1}}^2$. We note that it is an 'almost maximal' subgroup of $\mathbf{S}_{2^w}$. Indeed, there exists precisely one group that simultaneously is a proper subgroup of **R** and a proper supergroup of **K**. According to the O'Nan–Scott theorem [17], this 'intermediate' group is isomorphic to the wreath product group $\mathbf{S}_{2^{w-1}}$ wr $\mathbf{S}_2$ and has order $2s(w)$. Thus, we have the following chain of maximal subgroups:

$$\mathbf{S}_{2^{w-1}} \times \mathbf{S}_{2^{w-1}} \subset \mathbf{S}_{2^{w-1}} \text{ wr } \mathbf{S}_2 \subset \mathbf{S}_{2^w}.$$

The intermediate group is the normalizer of **K** (i.e. the largest subgroup of **R** in which **K** is normal). Figure 4 represents an arbitrary gate of this group: it is a SWITCHED gate, combined with a 1-bit gate g (necessarily either a 1-bit follower or a 1-bit inverter).

Note that several notorious reversible gates are special SWITCHED gates. The FREDKIN gate is the SWITCHED gate of width 3 with

- g' the 2-bit follower, and
- g'' the 2-bit exchanger (also referred to as the SWAP gate).

The CONTROLLED NOT gate is the SWITCHED gate of width 2 with

- g' the 1-bit follower, and
- g'' the 1-bit inverter.

The CONTROLLED^k NOT has a recursive definition: it is the SWITCHED gate of width $k+1$ with

- g' the k-bit follower, and
- g'' the CONTROLLED^{k-1} NOT.

Also the simple control gates of section 2 can be defined in a recursive way: the simple control gate of width w, with control function $f(A_1, A_2, \ldots, A_{w-1})$, can, thanks to the Shannon decomposition [18], be interpreted as the SWITCHED gate of width w with

- g' the simple control gate with control function $f(0, A_2, \ldots, A_{w-1})$, and
- g'' the simple control gate with control function $f(1, A_2, \ldots, A_{w-1})$.

It is clear that, if we are able to implement both the gate g' and the gate g'' with a finite number of switches (say s' and s'', respectively), then we can implement the SWITCHED either-g'-or-g'' gate with $s' + s'' + 4(w-1)$ switches, the $4(w-1)$ 'additional' switches all being controlled by the bit A_1. See figure 3(b) for the case $w = 4$, where the upper three switches are closed iff $A_1 = 0$, the lower three being closed iff $A_1 = 1$. The figure displays $2(w-1)$ switches; not shown in the figure are the dual lines $\overline{A_i}$ and $\overline{P_i}$. As also $\overline{A_2}$, $\overline{A_3}$ and $\overline{A_4}$ have to be switched by the value of A_1, the six switches in figure 3(b) in fact represent twelve switches in the physical implementation. For arbitrary w, there are thus $4(w-1)$ physical switches. Also not shown explicitly in figure 3(b) are the 'vertical' control wires between the 'horizontal' controlling lines (A_1 and $\overline{A_1}$) and the controlled switches. The total number of metal interconnections is ten times the number of switches. Now, if we denote by $\sigma(w)$ the maximum number of switches necessary to implement an arbitrary reversible gate of width w, we can conclude that the SWITCHED gates of width w can be manufactured with $2\sigma(w-1) + 4(w-1)$ switches (or less).

We finally note that the group **K** of SWITCHED gates is not a normal subgroup of **R**. The minimal normal subgroup of **R** containing **K** (i.e. the normal closure of **K**) is the entire group **R** of all reversible gates. The maximal normal subgroup of **R** contained in **K** (i.e. the core of **K**) is the trivial subgroup **1** consisting merely of the w-bit follower.

The subgroup **K** has $\frac{1}{2}C_{2^w}^{2^w/2} = (2^w)!/2[(2^{w-1})!]^2$ conjugate subgroups. Among these, there are w conjugate subgroups (1;j)**K**(1;j), where (1;j) denotes the exchanger gate interchanging column A_1 and column A_j of the truth table. The simplest example of a subgroup conjugate to **K** is (1;2)**K**(1;2). Figure 5(b) shows an arbitrary gate of this conjugate group. We see in figure 5(b) how A_2 takes over the role of A_1 in figure 5(a). Of course, the two subgroups **K** and (1;2)**K**(1;2) are not disjoint. They have an intersection, necessarily a subgroup of both. A gate which is simultaneously member of **K** and (1;2) **K** (1;2), necessarily has its output P_1 equal to A_1 and its output P_2 equal to A_2. The reader can easily verify that the other outputs $(P_3, P_4, \ldots, P_w)$ are functions of the inputs $(A_3, A_4, \ldots, A_w)$ which can be controlled by both A_1 and A_2. In figure 5(c), we see four subsubgates between (A_3, A_4) and (P_3, P_4). Which of these four gates is actually valid depends on the value of (A_1, A_2). Therefore, the intersection of **K** and (1;2) **K** (1;2) is simply isomorphic to the product group $\mathbf{S}^4_{2^{w-2}}$ of order $[(2^{w-2})!]^4$.

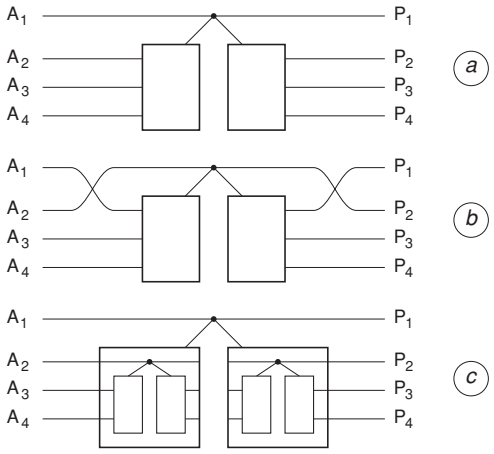

Figure 5. Three subgroups of the group of reversible gates: (*a*) the subgroup **K** of `SWITCHED` gates, (*b*) its conjugate subgroup `(1;2)` **K** `(1;2)` and (*c*) the intersection of both.

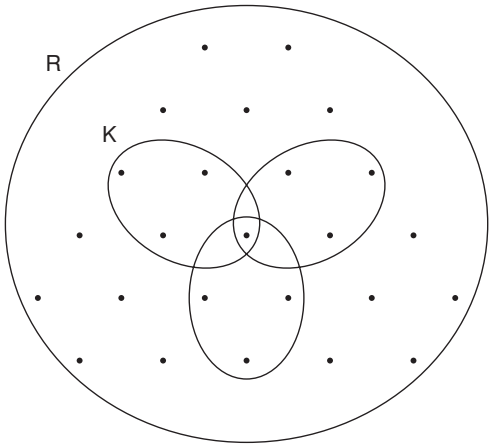

Figure 6. The group **R** of reversible gates of width 2 and its three conjugate subgroups **K**, `(1,3)` **K** `(1,3)` and `(2,3)` **K** `(2,3)`.

Figure 6 shows the case of $w = 2$: the full group **R** of $4! = 24$ reversible gates, the subgroup **K** of $(2!)^2 = 4$ `SWITCHED` gates and the latter's two conjugate groups of the same order.

4. Double cosets

Let **G** be an arbitrary group and **H** an arbitrary (proper) subgroup of **G**. Then we can partition the elements of **G** into equivalence classes called double cosets. The double coset of an arbitrary element g of **G** consists of all elements $h_1 g h_2$, where both h_1 and h_2 are arbitrary elements of **H**. Assume we can build all elements of **H**, as well as one particular element g of **G**. Then we can build all elements of the double coset of g by simply constructing the appropriate cascade of h_1, g and h_2. Thus, in order to be able to hardwire all elements of **G**,

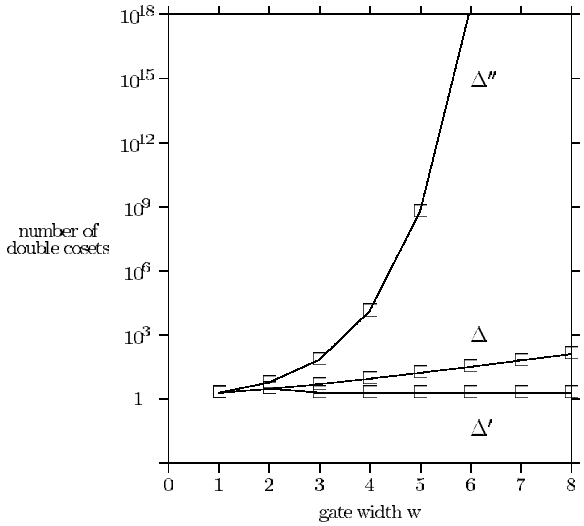

Figure 7. Number of double cosets in which the group **R** of reversible gates is partitioned by the subgroup **K** of SWITCHED gates.

it suffices to have a hardware implementation of

- all elements of **H** and
- one element of each double coset.

The latter element is called the representative of that particular double coset.

The number of elements of a group is called the order of the group. Let γ be the order of **G** and η the order of **H**. Let Δ be the number of double cosets in which **G** is partitioned by **H**. Then, in order to be able to construct all γ elements of **G**, it is sufficient to have an implementation of

- the η elements of **H** and
- the Δ representatives of the Δ double cosets of **H** in **G**.

These two sets overlap, as one of the double cosets of **H** is **H** itself. Thus the union of these two sets contains $\eta + \Delta - 1$ elements. We call this new set the library of the hardware implementation. This strategy is particularly attractive if $\eta + \Delta - 1$ is much smaller than γ. This requires an appropriate choice of **H**: not too large (otherwise η is large), but also not too small (otherwise Δ is large). We will now follow this strategy with **G** equal to **R** and **H** equal to **K**.

The group **K** of SWITCHED gates partitions the group **R** of reversible gates into a number (say Δ) of double cosets. The number Δ is a function of the width w and therefore is denoted by $\Delta(w)$. The reader can easily verify that $\Delta(1) = 2$ and $\Delta(2) = 3$. The computer algebra package GAP [19, 20] (and in particular its command DoubleCosets) tells us that $\Delta(3) = 5$ and $\Delta(4) = 9$. Appendix B gives a lower bound Δ' and an upper bound Δ'' on Δ. Figure 7 shows the resulting large uncertainty on the number of double cosets generated by the SWITCHED subgroup.

We will now perform the double coset decomposition of the group **R** and so determine $\Delta(w)$ for arbitrary w. It is a special case of the decomposition of the symmetric group $\mathbf{S}_{a+b}$ into double cosets by its subgroup $\mathbf{S}_a \times \mathbf{S}_b$. See appendix C. For convenience, we will number the Δ double cosets as numbers 0, 1, 2, ..., i.e. starting from the 'zeroth' set. The zeroth double

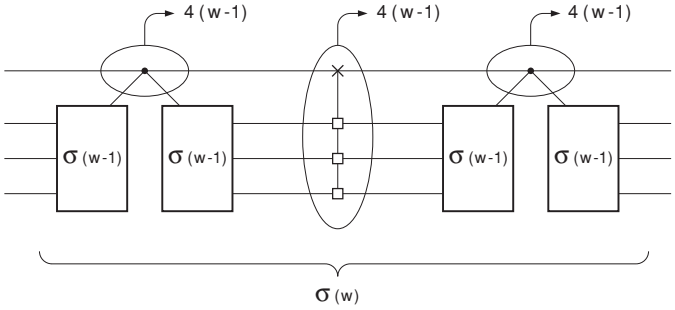

Figure 8. An arbitrary reversible gate of width w (here $w = 4$) with the position of its various switches.

coset is the double coset of `()`, i.e. the subgroup itself. The first double coset is the double coset of the upside-down `CONTROLLED`$^{w-1}$ `NOT` gate. In appendix D, we calculate the size of this particular double coset. It is $[(2^{w-1})!]^4/[(2^{w-1}-1)!]^2$. The second double coset is the double coset of the upside-down `CONTROLLED`$^{w-2}$ `NOT` gate. In appendix D, we subsequently construct all double cosets, until all gates of **R** are exhausted. This yields an analytical expression for Δ, the number of double cosets:

$$\Delta(w) = 2^{w-1} + 1.$$

The number $\Delta(w)$ is shown in figure 7.

We conclude that an arbitrary reversible gate g can be decomposed as $k_1 r k_2$, where k_1 and k_2 stand for two particular `SWITCHED` gates and r is the representative of the appropriate double coset. The latter is an upside-down `SWITCHED` gate of a very particular type, i.e. an upside-down compact control gate, needing only $4(w-1)$ switches. The arbitrary reversible gate thus needs only

$$\sigma(w) = 4\sigma(w-1) + 12(w-1)$$

switches. Figure 8 gives a 'geographic' representation of this recursion formula. For the solution of this equation, see appendix E, with $a = 4$, $b = 12$, $c = -12$ and $\sigma(1) = 0$. We obtain

$$\sigma(w) = \tfrac{4}{3}(4^w - 3w - 1). \tag{1}$$

This result has been added to table 2.

5. The synthesis algorithm

From the previous section, we know that an arbitrary gate g can be built from a `SWITCHED` gate k_1, an upside-down compact control gate r and a second `SWITCHED` gate k_2. Now the following question arises: given the truth table of an arbitrary gate g, which gates k_1, r and k_2 do we have to cascade? From appendix D, we know that there are quite some degrees of freedom in the construction of the decomposition. Thus the procedure below is far from unique.

Table 3a gives an example of a truth table representing an arbitrary reversible 3-bit gate (out of the $8! = 40{,}320$ different such gates which actually exist). Between the w input columns A_i and the w output columns P_i of table 3a, we introduce w extra columns F_i and w extra columns J_i. Figure 9 shows the 'physical' interpretation of these F and J bits.

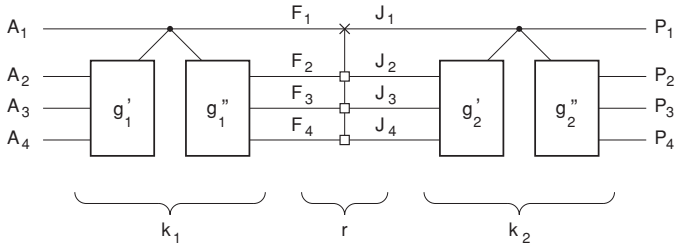

Figure 9. Decomposition of arbitrary reversible gate.

Table 3. Decomposition of a truth table: (a) original table and (b) extended table.

(a) Original table		(b) Extended table			
$A_1A_2A_3$	$P_1P_2P_3$	$A_1A_2A_3$	$F_1F_2F_3$	$J_1J_2J_3$	$P_1P_2P_3$
0 0 0	1 1 1	0 0 0	0 0 1	1 0 1	1 1 1
0 0 1	1 1 0	0 0 1	0 1 0	1 1 0	1 1 0
0 1 0	1 0 0	0 1 0	0 1 1	1 1 1	1 0 0
0 1 1	0 0 0	0 1 1	0 0 0	0 0 0	0 0 0
1 0 0	1 0 1	1 0 0	1 0 0	1 0 0	1 0 1
1 0 1	0 1 0	1 0 1	1 0 1	0 0 1	0 1 0
1 1 0	0 0 1	1 1 0	1 1 0	0 1 0	0 0 1
1 1 1	0 1 1	1 1 1	1 1 1	0 1 1	0 1 1

The procedure for filling in these additional $2w$ columns consists of two steps:

- first we fill in F_1 equal to A_1 and J_1 equal to P_1;
- then we compare column J_1 with column F_1:
  (a) among the first 2^{w-1} rows, on those rows where $J_1 = F_1$, we give $(F_2, F_3, \ldots, F_w) = (J_2, J_3, \ldots, J_w)$ the values $(0, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 1)$, ...
  (b) among the last 2^{w-1} rows, on those rows where $J_1 = F_1$, we give $(F_2, F_3, \ldots, F_w) = (J_2, J_3, \ldots, J_w)$ the same values $(0, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 1)$, ...
  (c) on the remaining rows of the first 2^{w-1} rows, we give $(F_2, F_3, \ldots, F_w) = (J_2, J_3, \ldots, J_w)$ the remaining values ..., $(1, 1, \ldots, 1, 0)$, $(1, 1, \ldots, 1, 1)$.
  (d) on the remaining rows of the last 2^{w-1} rows, we give $(F_2, F_3, \ldots, F_w) = (J_2, J_3, \ldots, J_w)$ the same remaining values ..., $(1, 1, \ldots, 1, 0)$, $(1, 1, \ldots, 1, 1)$.

Table 3b shows the result.

The former step guarantees that both the gate between $(A_1, A_2, \ldots, A_w)$ and $(F_1, F_2, \ldots, F_w)$ and the gate between $(J_1, J_2, \ldots, J_w)$ and $(P_1, P_2, \ldots, P_w)$ are SWITCHED gates. The latter step guarantees that $(F_2, F_3, \ldots, F_w)$ always equals $(J_2, J_3, \ldots, J_w)$ and therefore that the gate between $(F_1, F_2, \ldots, F_w)$ and $(J_1, J_2, \ldots, J_w)$ is an upside-down simple control gate. As

- F_1 equals J_1 for the minterm $J_2J_3 \ldots J_{w-1}J_w$ equal $00 \ldots 00$, $00 \ldots 01$, ...
- F_1 equals $\overline{J_1}$ for the minterm $J_2J_3 \ldots J_{w-1}J_w$ equal ..., $11 \ldots 10$, $11 \ldots 11$,

the control function is a compact function.

The middle part of table 3b results in the control function of the control gate: see table 4c. The left part of table 3b leads to the truth tables of both g'_1 and g''_1: see tables 4a and b. The right part of table 3b leads to the truth tables of both g'_2 and g''_2: see table 4d and e.

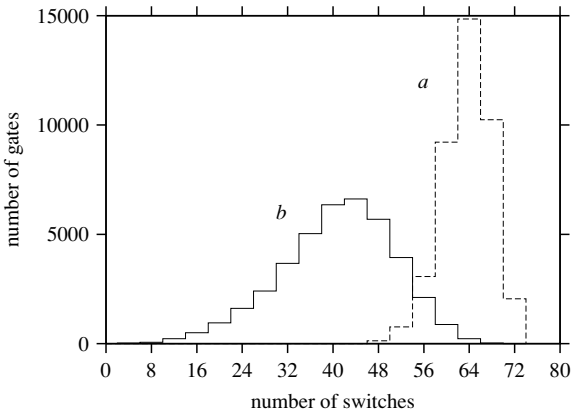

Figure 10. Switch count distribution for $w = 3$: (*a*) according to basic procedure and (*b*) according to improved procedure.

Table 4. Five small truth tables: (a) g_1', (b) g_1'', (c) r, (d) g_2' and (e) g_2''.

(a) g_1'		(b) g_1''		(c) r		(d) g_2'		(e) g_2''	
A_2A_3	F_2F_3	A_2A_3	F_2F_3	$F_1F_2F_3$	J_1	J_2J_3	P_2P_3	J_2J_3	P_2P_3
0 0	0 1	0 0	0 0	0 0 0	0	0 0	0 0	0 0	0 1
0 1	1 0	0 1	0 1	0 0 1	1	0 1	1 0	0 1	1 1
1 0	1 1	1 0	1 0	0 1 0	1	1 0	0 1	1 0	1 0
1 1	0 0	1 1	1 1	0 1 1	1	1 1	1 1	1 1	0 0
				1 0 0	1				
				1 0 1	0				
				1 1 0	0				
				1 1 1	0				

Then we apply the above procedure to each of these four subtables. Each of them gives rise to one control gate and four smaller subgates. At this stage, the circuit consists of one control gate of width w, four control gates of width $w - 1$ and $4^2 = 16$ subsubgates of width $w - 2$. We proceed further according to these lines, until we have 4^{w-1} subtables, all of width 1, i.e. until all subtables represent merely 1-bit followers or 1-bit inverters.

We applied the above procedure to all 40 320 reversible truth tables of width 3, using the computer algebra package `Maple`. Figure 10 shows the resulting distribution of the switch number s. Depending on the compact control functions distilled by the procedure, s can have different values, ranging from 48 to 72. The worst value ($s = 72$) is a direct consequence of equation (1), whereas the lowest value ($s = 48$) is explained as follows: in the best case the control functions of all upside-down control gates equal the constant function $f_0 = 0$ and therefore need no switches, leading to $\frac{8}{9}(4^w - 3w - 1)$ (see appendix E, with $a = 4, b = 8$ and $c = -8$). The average value of s turns out to be $\frac{1336}{21} \approx 63.6$. One can easily make this number smaller by making the procedure somewhat more sophisticated. For example we can foresee that whenever $g' = g''$, then these two gates and the corresponding switches are replaced by a single gate without switches. The figure also shows the switch count produced by such refined procedure. Now the switch count s ranges from 0 to $\sigma(w)$. For $w \geqslant 3$, the distribution falls to zero even before $s = \sigma(w)$. In our example (i.e. $w = 3$), the average s is

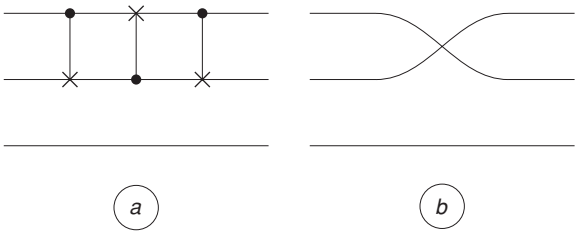

Figure 11. Implementation of the 3-bit reversible gate `(3,5)(4,6)`: (*a*) by artificial intelligence and (*b*) by natural intelligence.

now only $\frac{17\,131}{420} \approx 40.8$ anymore. The procedure can, of course, be refined further, if necessary. For example the automatic procedure implements the 3-bit reversible gate `(3,5)(4,6)` as in figure 11(*a*), needing 12 switches, whereas the reader can easily verify that the circuit in figure 11(*b*), needing zero switches, does the job equally well. The automatic replacement of figure 11(*a*) by 11(*b*) is possible by programming 'template matching'. For this subject, the reader is referred to the detailed discussions by Miller *et al* [21] and by Shende *et al* [12, 22, 23].

6. Discussion

Other authors recently have presented algorithms for synthesizing arbitrary reversible gates. Many approaches require extensive searching. In the present section, we will compare our results only with those of other methods that, just like ours, synthesize in a straightforward way, avoiding any extensive search.

6.1. Dueck and Maslov

Dueck and Maslov [24] have presented a synthesis consisting of a cascade of gates, all of the type *ece*, where *e* is an exchanger of type `(j;w)`, with $1 \leqslant j \leqslant w$, and *c* is a simple control gate with control function

$$f(A_1, A_2, \ldots, A_{w-1}) = \widetilde{A_1}\widetilde{A_2}\ldots\widetilde{A_{w-1}},$$

where the tilde has the following meaning: $\widetilde{X}$ equals either X or $\overline{X}$ or 1. In other words, f is a minterm of the arguments $A_1, A_2, \ldots, A_{w-1}$ or a minterm of a subset of these. Their synthesis needs no more than $w2^w$ such simple control gates. As the hardware implementation of a simple control gate with such special control function does not need more than $4(w-1)$ switches, the Dueck and Maslov algorithm needs $4(w-1)w2^w$ switches or less. Figure 12 compares this worst-case number to our worst-case switch count $\sigma(w)$. For $w \leqslant 6$, our synthesis method turns out to be slightly more efficient, whereas for $w \geqslant 7$, the Dueck–Maslov algorithm is more efficient.

6.2. Miller, Maslov and Dueck

Miller, Maslov and Dueck [21] have presented a variant of the Dueck–Maslov algorithm. It also uses a string of *ece* gates; it again uses control functions of the form $\widetilde{A_1}\widetilde{A_2}\ldots\widetilde{A_{w-1}}$, but now $\widetilde{X}$ stands for either X or 1. Thus all simple control gates are `CONTROLLED`^{k} `NOT` gates (with $0 \leqslant k \leqslant w-1$).

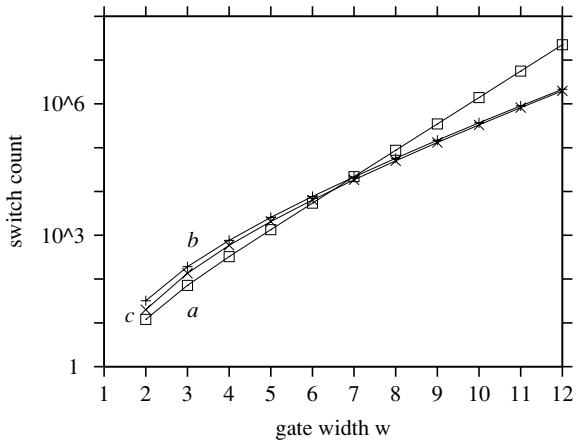

Figure 12. Maximum number of switches in the implementation of a w-bit reversible gate: (a) according to the present algorithm, (b) according to the Dueck–Maslov algorithm and (c) according to the Miller–Maslov–Dueck algorithm.

The Miller–Maslov–Dueck synthesis needs at most $(w-1)2^w+1$ gates, which each need at most $4(w-1)$ switches. This results into a maximum of $4(w-1)[(w-1)2^w+1]$ switches. This amount is displayed in figure 12(c). We see how the Miller–Maslov–Dueck synthesis uses slightly less switches than the Dueck–Maslov synthesis.

7. Conclusion

We have introduced group theory as a possible design strategy for synthesizing a reversible gate. This leads us to an algorithm which can be used to automate the implementation of an arbitrary reversible function in future (c-MOS) technologies, and thus realizing low-cost computing.

Among the group $\mathbf{R}$ of $(2^w)!$ reversible logic gates of width w, we have distinguished two special types:

- the $[(2^{w-1})!]^2$ `SWITCHED` gates of width w, which form a subgroup $\mathbf{K}$, and
- the $2^{w-1}+1$ compact control gates of width w, which do not form a group.

We have derived what is called in group theory [25] 'the double coset space $\mathbf{K}\backslash\mathbf{R}/\mathbf{K}$ of $\mathbf{R}$ relative to $\mathbf{K}$'. It consists of $2^{w-1}+1$ sets. Because there is a 1-to-1 relationship between compact Maitra terms and double cosets, the compact control gates, put upside down, form a complete set of double coset representatives for $\mathbf{K}\backslash\mathbf{R}/\mathbf{K}$. Thus, we have proven that any reversible gate can be constructed by cascading three gates:

- a first `SWITCHED` gate,
- an upside-down compact control gate and
- a second `SWITCHED` gate.

We have also given an explicit algorithm in order to determine, for a given reversible gate, which two `SWITCHED` gates and which compact control gate we need. The procedure leads to an explicit hardware implementation, consisting of $\frac{4}{3}(4^w-3w-1)$ controlled switches (or less). This solves the synthesis problem of the reversible logic gate.

It is well known [26, 27] that any combinatorial network with n_{in} binary inputs and n_{out} binary outputs can be embedded within a reversible truth table by merely adding the

appropriate number of garbage output columns and preset input columns. Such reversible truth table will have an appropriate width w satisfying $\max(n_{\text{in}}, n_{\text{out}}) \leqslant w \leqslant n_{\text{in}} + n_{\text{out}}$. Once the table is constructed, it can be implemented into hardware by the method presented here. As a conclusion, we can say that any Boolean operation (of arbitrary complexity) can be implemented by the above presented algorithm into a hardware circuit that is logically reversible.

Appendix A. Compact functions

There exist 2^{2^q} functions of q Boolean variables $X_1, X_2, \ldots, X_q$. All can be written as an OR of minterms:

$$f(X_1, X_2, \ldots, X_q) = \epsilon_{00\ldots0}\overline{X_1}\,\overline{X_2}\ldots\overline{X_q} + \epsilon_{00\ldots1}\overline{X_1}\,\overline{X_2}\ldots X_q + \cdots + \epsilon_{11\ldots1}X_1X_2\ldots X_q.$$

The values of the 2^q coefficients ϵ fully determine the function f. We call a function compact if the last j coefficients equal 1 and all the remaining $2^q - j$ coefficients equal 0, with j an arbitrary integer satisfying $0 \leqslant j \leqslant 2^q$. There exist $2^q + 1$ such functions:

$$\begin{aligned}
&f_0 = 0\\
&f_1 = X_1X_2\ldots X_{q-1}X_q\\
&f_2 = X_1X_2\ldots X_{q-1}\overline{X_q} + X_1X_2\ldots X_{q-1}X_q\\
&f_3 = X_1X_2\ldots\overline{X_{q-1}}X_q + X_1X_2\ldots X_{q-1}\overline{X_q} + X_1X_2\ldots X_{q-1}X_q\\
&\ldots\\
&f_{2^q-1} = \overline{X_1}\,\overline{X_2}\ldots\overline{X_{q-1}}X_q + \overline{X_1}\,\overline{X_2}\ldots X_{q-1}\overline{X_q} + \ldots + X_1X_2\ldots X_{q-1}X_q\\
&f_{2^q} = \overline{X_1}\,\overline{X_2}\ldots\overline{X_{q-1}}\,\overline{X_q} + \overline{X_1}\,\overline{X_2}\ldots\overline{X_{q-1}}X_q + \ldots + X_1X_2\ldots X_{q-1}X_q.
\end{aligned}$$

Many of these expressions can be simplified to some short combination of ORs and ANDs:

$$\begin{aligned}
&f_0 = 0\\
&f_1 = X_1X_2\ldots X_{q-1}X_q\\
&f_2 = X_1X_2\ldots X_{q-1}\\
&f_3 = X_1X_2\ldots X_{q-2}(X_{q-1} + X_q)\\
&\ldots\\
&f_{2^q-1} = X_1 + X_2 + \cdots + X_{q-1} + X_q\\
&f_{2^q} = 1.
\end{aligned}$$

We will now prove that any compact function can be written as an expression containing OR operators, AND operators and at most q letters X_k (with $1 \leqslant k \leqslant q$). The proof is by full induction.

Let us assume the theorem holds for $q = Q - 1$. We have to demonstrate that it also holds for $q = Q$. Any function of Q variables can be decomposed according to the Shannon decomposition:

$$f(X_1, X_2, \ldots, X_Q) = X_1 f(1, X_2, \ldots, X_Q) + \overline{X_1} f(0, X_2, \ldots, X_Q).$$

If f is compact, then there are three possibilities:

- If f has less than 2^{Q-1} terms in its minterm expansion, then $f(1, X_2, \ldots, X_Q)$ is compact and $f(0, X_2, \ldots, X_Q)$ equals 0, such that $f(X_1, X_2, \ldots, X_Q) = X_1 f(1, X_2, \ldots, X_Q)$ is written with at most $1 + (Q - 1) = Q$ letters.
- If f has exactly 2^{Q-1} terms in its minterm expansion, then $f(1, X_2, \ldots, X_Q) = 1$ and $f(0, X_2, \ldots, X_Q) = 0$, such that $f(X_1, X_2, \ldots, X_Q) = X_1$ is written with one letter.

- If f has more than 2^{Q-1} terms in its minterm expansion, then $f(1, X_2, \ldots, X_Q)$ equals 1 and $f(0, X_2, \ldots, X_Q)$ is compact, such that $f(X_1, X_2, \ldots, X_Q) = X_1 + \overline{X_1} f(0, X_2, \ldots, X_Q) = X_1 + f(0, X_2, \ldots, X_Q)$ is written with at most $1 + (Q - 1) = Q$ letters.

We additionally note that the theorem holds for $q = 1$, as there exist only three compact functions of one variable X_1:

$$f_0(X_1) = 0 \qquad f_1(X_1) = X_1 \qquad f_2(X_1) = 1.$$

All three can be written by using one letter (or less). This makes the proof complete.

There exists a similar theorem for functions with the dual property: functions with the first j coefficients ϵ equal to 1 and all remaining $2^q - j$ coefficients ϵ equal to 0: they can be written as an expression containing `OR` operators, `AND` operators and at most q overlined letters $\overline{X_k}$.

We finally note that, except for f_0, all compact functions can be written as Maitra terms [28–30]:

$$f_i = X_1 \,\&\, (X_2 \,\&\, (\ldots X_{q-1} \,\&\, (X_q \,\&\, 1) \cdots)) \qquad \text{for} \quad 1 \leqslant i \leqslant 2^q,$$

where `&` stands for 'either `AND` or `OR`'. Alternatively, all but the last compact functions can be interpreted as follows as Maitra terms:

$$f_i = X_1 \,\&\, (X_2 \,\&\, (\ldots X_{q-1} \,\&\, (X_q \,\&\, 0) \cdots)) \qquad \text{for} \quad 0 \leqslant i \leqslant 2^q - 1.$$

In Maitra's original paper [28], the symbol `&` has the meaning 'any of the 16 Boolean functions of two Boolean variables'. In the paper by Sarabi *et al* [29], however, the symbol `&` stands for 'either `AND` or `OR` or `XOR`'. Mishchenko and Perkowski [31] consider three different interpretations of the infix operator `&`:

- any two-input Boolean function
- two-input function restricted to `AND`, `OR` and `XOR`
- two-input function restricted to `AND` and `OR`.

Here, we apply the last meaning: `&` stands for 'either `AND` or `OR`', for the simple reason that an `AND` can be hardwired as a series connection of switches, whereas an `OR` can be implemented as a parallel connection of switches. In order to avoid any confusion because of the proliferation of definitions, we will call Maitra terms applying exclusively `AND` and `OR` functions compact Maitra terms.

Appendix B. Bounds on the number of double cosets

A subgroup **H** (of order η) of a group **G** (of order γ) partitions the group **G** into γ/η left cosets, all of size η. It analogously partitions **G** into γ/η right cosets, all of size η. The same subgroup **H** partitions **G** into double cosets. However, these are usually not of equal size. Each double coset has a size which is a multiple of η, the smallest possible size being η itself, the largest possible size being η^2. The smallest possible size is always present, as **H** itself is one of the double cosets; the size η^2 is not always present. Not only the size distribution of double cosets is difficult to determine, even the number of sets is difficult to predict. We can, however, easily put some bounds on the number. Let Δ be the exact number of double cosets. Then Δ is largest in case all double cosets have minimum size η and is smallest if one double coset is of size η and all others of maximum size η^2. Taking into account that Δ is an integer, we finally deduce

$$1 + \left\lceil \frac{\gamma - \eta}{\eta^2} \right\rceil \leqslant \Delta \leqslant \frac{\gamma}{\eta},$$

where $\lceil x \rceil$ stands for the ceiling of x, i.e. the smallest integer greater than or equal to x. Note that γ/η always is an integer (because of Lagrange's theorem) and is called the index of $\mathbf{H}$ in $\mathbf{G}$. The above lower bound we will denote by Δ', whereas the above upper bound we will denote by Δ''.

Appendix C. A theorem on double cosets

We consider the symmetric group $\mathbf{G} = \mathbf{S}_n$, where n is an arbitrary integer. It consists of all permutations of the n numbers $\{1, 2, \ldots, n\}$. Its order is $n!$. We treat n as the sum of two other natural numbers: $n = a + b$. We consider the permutations of the a numbers $\{1, 2, \ldots, a\}$, as well as the permutations of the b numbers $\{a + 1, a + 2, \ldots, n\}$. We define the subgroup $\mathbf{H}$ as the direct product $\mathbf{S}_a \times \mathbf{S}_b$. Its order is $a!b!$. We will now prove the following theorem:

> The subgroup $\mathbf{S}_a \times \mathbf{S}_b$ partitions the group $\mathbf{S}_{a+b}$ into $\min(a, b) + 1$ double cosets.

Merely for convenience, we will assume here that $a \leqslant b$. We will demonstrate that the following permutations form a complete set of representatives: `()`, `(a,n)`, `(a-1,n-1)(a,n)`, ..., and `(1,n-a+1)(2,n-a+2)...(a,n)`.

Let us consider the permutation

$$r_j = \texttt{(a-j+1,n-j+1)(a-j+2,n-j+2)...(a,n)},$$

where j is an arbitrary integer satisfying $0 \leqslant j \leqslant a$. The double coset of r_j consists of all permutations written as $h_1 r_j h_2$, where h_1 and h_2 are two arbitrary elements of the subgroup $\mathbf{H}$. This double coset consists of all permutations π which map i on π_i (with $1 \leqslant i \leqslant n$) such that $\{\pi_1, \pi_2, \ldots, \pi_a\}$ contains $a - j$ numbers from $\{1, 2, \ldots, a\}$ and j numbers from $\{a + 1, a + 2, \ldots, n\}$, and where $\{\pi_{a+1}, \pi_{a+2}, \ldots, \pi_n\}$ contains j numbers from $\{1, 2, \ldots, a\}$ and $b - j$ numbers from $\{a + 1, a + 2, \ldots, n\}$. In other words, this double coset consists of all permutations of $\{1, 2, \ldots, n\}$, where exactly j numbers have moved from somewhere within $\{1, 2, \ldots, a\}$ to somewhere within $\{a + 1, a + 2, \ldots, n\}$ and exactly j have moved from somewhere within $\{a + 1, a + 2, \ldots, n\}$ to somewhere within $\{1, 2, \ldots, a\}$. Conversely, permutations that exchange j' numbers between these two parts of $\{1, 2, \ldots, n\}$, with $j' \neq j$, do not belong to the double coset of r_j.

In order to determine the size of the double coset of r_j without any double counting, we can e.g. proceed in the following three steps:

- we choose the j elements $\{x_1, x_2, \ldots, x_j\}$ from $\{1, 2, \ldots, a\}$ which will be moved to $\{a + 1, a + 2, \ldots, n\}$, and analogously we choose the j elements $\{y_1, y_2, \ldots, y_j\}$ from $\{a + 1, a + 2, \ldots, n\}$ which will be moved to $\{1, 2, \ldots, a\}$;
- we permute the numbers $x_1, x_2, \ldots, x_j$ with the numbers $a - j + 1, a - j + 2, \ldots, a$ by means of an appropriate member of $\mathbf{S}_a$ (i.e. by means of a permutation that maps x_1 to $a - j + 1$, x_2 to $a - j + 2$, etc), and analogously we permute the numbers $y_1, y_2, \ldots, y_j$ with the numbers $n - j + 1, n - j + 2, \ldots, n$ by means of an appropriate member of $\mathbf{S}_b$, and finally we perform the permutation r_j;
- we permute the numbers $1, 2, \ldots, a$ by means of an arbitrary member of $\mathbf{S}_a$, and analogously we permute the numbers $a + 1, a + 2, \ldots, n$ by means of an arbitrary member of $\mathbf{S}_b$.

The first step yields $C_j^a C_j^b$ different possibilities; the second step yields no degrees of freedom; the third step gives $a!b!$ different possibilities. Multiplication of these independent possibilities finally yields the desired number: $(a!)^2(b!)^2/[(j!)^2(a - j)!(b - j)!]$. Note that this size is

smaller than $(a!b!)^2$, the square of the order of **H**, a fact which demonstrates that indeed we have eliminated 'double countings'.

Taking all values for j, from 0 to a, gives $a + 1$ double cosets. The union of all elements of these $a + 1$ sets exactly exhausts all elements of **G**. This is a consequence of the identity

$$\sum_{j=0}^{a} \frac{a!b!}{(a-j)!j!j!(b-j)!} = \frac{(a+b)!}{a!b!},$$

which is deduced from a special case ($n = p = a$ and $m = b$) of the Gradshteyn and Ryzhik [32] formula 0.156.1:

$$\sum_{k=0}^{a} \frac{a!}{k!(a-k)!} \frac{b!}{(a-k)!(b-a+k)!} = \frac{(a+b)!}{a!b!},$$

by performing the substitution $k = a - j$.

Appendix D. The size of the double cosets

For convenience, we number the Δ double cosets as numbers $0, 1, 2, \ldots$, i.e. starting from number zero. We will apply the theorem of appendix C, for the special case $n = 2^w$ and $a = b = 2^{w-1}$.

D.1. The zeroth double coset

We choose a very simple representative r_0 of the zeroth double coset: the w-bit follower `()`. We count the number of different cascades $k_1 r k_2$, where k_1 and k_2 are two arbitrary members of **K**. As $k_1 r k_2$ equals $k_1 k_2$, and as **K** forms a group, this product equals some third `SWITCHED` gate k_3. Thus there are 'only' $s(w) = [(2^{w-1})!]^2$ gates in this zeroth double coset.

We close the present subsection by noting that the representative $r_0 =$ `()` of the zeroth double coset can be interpreted as the upside-down simple control gate with control function $f = 0$, i.e. f equal to the zeroth Maitra term f_0.

D.2. The first double coset

We choose a representative r_1 of the first double coset: the `CONTROLLED`$^{w-1}$ `NOT` gate in which the bits $F_2, F_3, \ldots,$ and F_w control F_1, i.e. the `CONTROLLED`$^{w-1}$ `NOT` gate upside down. See figure 9 (with its $w - 1$ little squares replaced by little circles). Its permutation notation is `(2`$^{w-1}$`, 2`^{w}`)`. We count the number of different cascades $k_1 r_1 k_2$, where k_1 and k_2 are two arbitrary members of **K**. According to appendix C, there are 'only' $[(2^{w-1})!]^4/[(2^{w-1}-1)!]^2$ different cascades.

We note that the upside-down `CONTROLLED`$^{w-1}$ `NOT` gate is the upside-down simple control gate of width w, where the bit F_1 is controlled by means of the control function $f(F_2, F_3, \ldots, F_w)$ equal to the minterm $F_2 F_3 \ldots F_w$, which can also be written as the Maitra term $f_1(F_2, F_3, \ldots, F_w)$. The first double coset contains more upside-down simple control gates. Indeed, the reader can easily verify that any upside-down simple control gate with control function equal to a minterm is an element of the set and thus can play the role of r_1.

D.3. The second double coset

We choose a representative r_2 of the second double coset: the `CONTROLLED`$^{w-2}$ `NOT` gate in which bits $F_2, F_3, \ldots,$ and F_{w-1} control F_1, i.e. the `CONTROLLED`$^{w-2}$ `NOT` gate upside down.

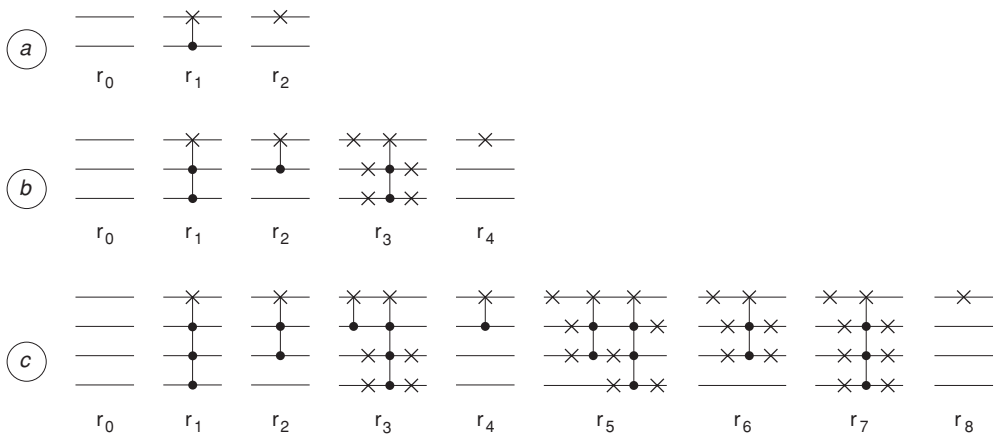

Figure 13. The representatives of the double coset decomposition: (*a*) for $w = 2$, (*b*) for $w = 3$ and (*c*) for $w = 4$.

Its permutation notation is $(2^{w-1}\text{-}1, 2^w\text{-}1)(2^{w-1}, 2^w)$. According to appendix C, there are 'only' $[(2^{w-1})!]^4/[2!(2^{w-1}-2)!]^2$ different cascades of the form $k_1 r_2 k_2$.

We note that the upside-down CONTROLLED^{w-2} NOT gate is the upside-down simple control gate of width w with control function $f(F_2, F_3, \ldots, F_w)$ equal to $F_2 F_3 \ldots F_{w-1}$ and thus to a sum of two minterms: $F_2 F_3 \ldots F_{w-1} F_w + F_2 F_3 \ldots F_{w-1} \overline{F_w}$, and thus to the Maitra term $f_2(F_2, F_3, \ldots, F_w)$. The second double coset contains more upside-down simple control gates. Indeed, any upside-down simple control gate with control function equal to the sum of two minterms is an element of the set.

D.4. The jth double coset

One can proceed further along these lines. Two simple control gates with an equal number j of minterms in their control functions belong to the same double coset. Conversely, any simple control gate with k minterms ($k \neq j$) in its control function does not belong to the double coset defined by a simple control gate with j minterms in its control function.

Thus, for the representative r_j of the jth double coset, we can choose any upside-down control gate with a control function consisting of j minterms. In particular, we choose the upside-down control gate with control function equal to the jth compact Maitra term f_j:

$$r_j = (2^{w-1}\text{-}j\text{+}1, 2^w\text{-}j\text{+}1)(2^{w-1}\text{-}j\text{+}2, 2^w\text{-}j\text{+}2) \ \ldots \ (2^{w-1}, 2^w).$$

Figure 13 shows this set of representatives for the cases $w = 2, w = 3$, and $w = 4$. For a change, each simple control gate is drawn here not as a compact control gate but as a decomposition into CONTROLLED^i NOTs (with i ranging from 0 to $w-1$). The last representative is the NOT gate applied to $F_1 = A_1$.

According to appendix C, the jth double coset has size

$$\frac{[(2^{w-1})!]^4}{[j!(2^{w-1}-j)!]^2}.$$

Figure 14 shows the different sizes for the case $w = 4$. The sum of all these sizes exactly yields the order $(2^w)!$ of the whole group **R**. The $2^{w-1} + 1$ different compact Maitra terms, each one giving rise to a single double coset, thus suffice to enumerate the whole space of double cosets.

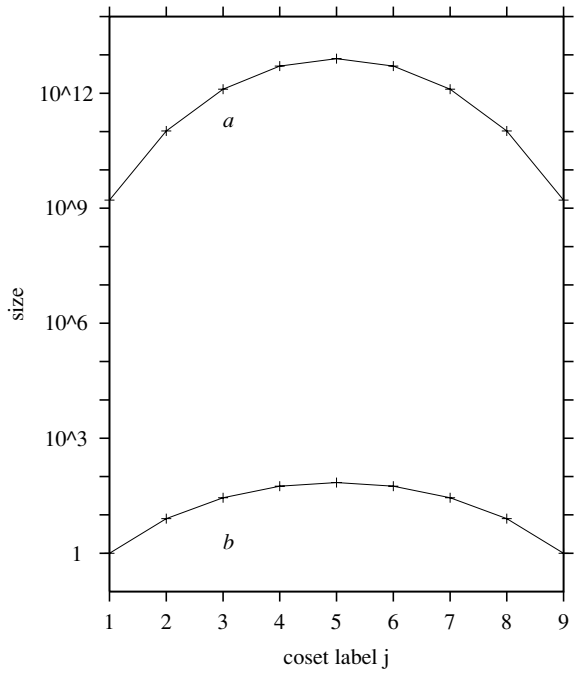

Figure 14. The number of elements in each of the nine double cosets in the case $w = 4$: (*a*) total number of gates and (*b*) the number of upside-down simple control gates.

By applying a reasoning analogous to section C.3, one can prove that any control function with j terms in its minterm expansion can play the role of representative r_j. There are $C^j_{2^{w-1}}$ such functions. Figure 14(*b*) shows how many upside-down simple control gates can be found in each of the double cosets. The number of upside-down compact control gates in each double coset is exactly 1.

Figure 13 shows how the representative r_j can be constructed. Figure 15(*a*) gives a more systematic implementation of the representatives in figure 13: the jth representative consists of

- the cyclic exchanger z,
- a `SWITCHED` gate with 2^{w-1} gates of width 1, the first $2^{w-1} - j$ being 1-bit followers, the last j being 1-bit inverters and
- the inverse z^{-1} of the cyclic exchanger.

Such hardware implementation needs $4(2^w - w - 1)$ switches. See appendix E, with $a = 2, b = 4$ and $c = -4$. Figure 15(*b*) illustrates how such gate can be simplified, such that its number of switches is only $2w(w - 1)$. Figure 16 gives the implementation as a compact control gate, needing only $4(w - 1)$ switches. In contrast to figures 15(*a*) and (*b*), figure 16 also displays explicitly the dual lines $\overline{F_i}$ and $\overline{J_i}$. The labels near the switches tell which Boolean variable has to equal 1 in order to close the switch. The reader can easily recognize twice the implementation of the Maitra term

$$f_3(F_2, F_3, F_4) = F_2(F_3 + F_4)$$

and twice the implementation of its inverse

$$\overline{f_3(F_2, F_3, F_4)} = \overline{F_2} + (\overline{F_3}\ \overline{F_4}).$$

In electronics, one speaks of two pull-up and two pull-down networks.

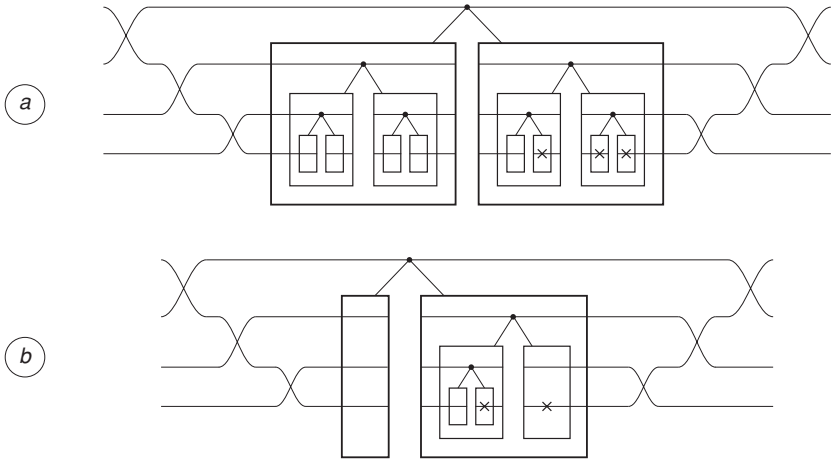

Figure 15. The representative r_3 of the third double coset in the case $w = 4$: (*a*) systematic implementation as SWITCHED gate (with 44 switches) and (*b*) simplified implementation as SWITCHED gate (with only 24 switches).

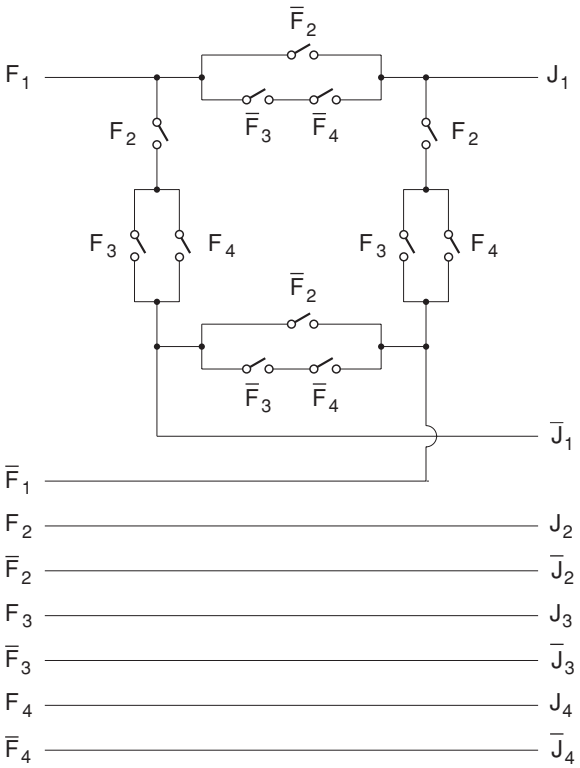

Figure 16. The representative r_3 of the third double coset in the case $w = 4$: implementation as compact control gate (with only 12 switches).
Note. For sake of clarity, the 'vertical' interconnections between the controlled switches and their controlling lines (F_2, $\overline{F_2}$, F_3, $\overline{F_3}$, F_4 and $\overline{F_4}$) are not shown explicitly.

Appendix E. Recursion relation

We consider the recursion relation

$$\sigma(w) = a\sigma(w-1) + bw + c,$$

where a, b and c denote constants. We introduce the new unknown $\tau(w)$ by postulating

$$\sigma(w) = a^w \tau(w),$$

yielding the new recursion relation

$$\tau(w) = \tau(w-1) + (bw+c)\left(\frac{1}{a}\right)^w$$

and the solution

$$\begin{aligned}\tau(w) &= \tau(1) + \sum_{i=2}^{w}(bi+c)\left(\frac{1}{a}\right)^i \\ &= \tau(1) + b\sum_{i=2}^{w} i\left(\frac{1}{a}\right)^i + c\sum_{i=2}^{w}\left(\frac{1}{a}\right)^i\end{aligned}$$

or

$$\tau(w) = \tau(1) + b\frac{2a^w - a^{w-1} - (w+1)a + w}{a^w(a-1)^2} + c\frac{a^{w-1}-1}{a^w(a-1)},$$

and thus

$$\sigma(w) = a^{w-1}\sigma(1) + b\frac{2a^w - a^{w-1} - (w+1)a + w}{(a-1)^2} + c\frac{a^{w-1}-1}{a-1}.$$

In the special case that $c = -b$, the expression simplifies to

$$\sigma(w) = a^{w-1}\sigma(1) + b\frac{a^w - (a-1)w - 1}{(a-1)^2}.$$

References

[1] Markov I 2003 An introduction to reversible circuits *Proc. Int. Workshop on Logic and Synthesis (Laguna Beach, May 2003)* pp 318–9

[2] Landauer R 1961 Irreversibility and heat generation in the computing process *IBM J. Res. Dev.* **5** 183–91

[3] Bennett C 1982 The thermodynamics of computation—a review *Int. J. Theor. Phys.* **21** 905–40

[4] Bennett C and Landauer R 1985 The fundamental physical limits of computation *Sci. Am.* **253** 38–46

[5] Bennett C 1988 Notes on the history of reversible computation *IBM J. Res. Dev.* **32** 16–23

[6] Wayner P 1994 Silicon in reverse *Byte* **19** 67–74

[7] De Vos A 2003 Lossless computing *Proc. IEEE Workshop on Signal Processing (Poznań, Oct 2003)* pp 7–14

[8] Feynman R 1985 Quantum mechanical computers *Opt. News* **11** 11–20

[9] Feynman R 1996 *Feynman Lectures on Computation* ed A Hey and R Allen (Reading: Addison-Wesley)

[10] Berman G, Doolen G, Mainieri R and Tsifrinovich V 1998 *Introduction to Quantum Computers* (Singapore: World Scientific)

[11] Nielsen M and Chuang I 2000 *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press)

[12] Shende V, Prasad A, Markov I and Hayes J 2003 Synthesis of reversible logic circuits *IEEE Trans. Comput.-aided Design Integr. Circuits Syst.* **22** 710–22

[13] Athas W, Svensson L, Koller J, Tzartzanis N and Chou E 1994 Low-power digital systems based on adiabatic-switching principles *IEEE Trans. Very Large Scale Integr. Syst.* **2** 398–407

[14] Storme L, De Vos A and Jacobs G 1999 Group theoretical aspects of reversible logic gates *J. Univ. Comput. Sci.* **5** 307–21

[15] De Vos A, Raa B and Storme L 2002 Generating the group of reversible logic gates *J. Phys A: Math. Gen.* **35** 7063–78
[16] Desoete B and De Vos A 2002 A reversible carry-look-ahead adder using control gates *Integr. the VLSI J.* **33** 89–104
[17] Liebeck M, Praeger C and Saxl J 1987 A classification of the maximal subgroups of the finite alternating and symmetric groups *J. Algebra* **111** 365–83
[18] Devadas S, Ghosh A and Keutzer K 1994 *Logic Synthesis* (New York: McGraw-Hill)
[19] Schönert M 1992 GAP *Comput. Algebra Nederland Nieuwsbrief* **9** 19–28
[20] http://www-gap.dcs.st-and.ac.uk/˜gap
[21] Miller D, Maslov D and Dueck G 2003 A transformation based algorithm for reversible logic synthesis *Proc. 40th Design Automation Conference (Annaheim, June 2003)* pp 318–23
[22] Shende V, Prasad A, Markov I and Hayes J 2002 Synthesis of optimal reversible logic circuits *Proc. Int. Workshop on Logic and Synthesis (New Orleans, June 2002)* pp 125–30
[23] Shende V, Prasad A, Markov I and Hayes J 2002 Reversible logic circuit synthesis *Proc. Int. Conf. on Computer-Aided Design (San Jose, Nov. 2002)* pp 353–60
[24] Dueck G and Maslov D 2003 Reversible function synthesis with minimum garbage outputs *Proc. 6th Int. Symp. on Representation and Methodology of Future Computing Technologies (Trier, March 2003)* 154–61
[25] Maslen D and Rockmore D 2000 Double coset decompositions and computational harmonic analysis on groups *J. Fourier Anal. Appl.* **6** 349–88
[26] Fredkin E and Toffoli T 1982 Conservative logic *Int. J. Theor. Phys.* **21** 219–53
[27] Maslov D and Dueck G 2004 Reversible cascades with minimal garbage *IEEE Trans. Comput.-aided Design Integr Circuits Syst.* **23** 1497–509
[28] Maitra K 1962 Cascaded switching networks of two-input flexible cells *IRE Trans. Electron. Comput.* **11** 136–43
[29] Sarabi A, Song N, Chrzanowska–Jeske M and Perkowski M 1994 A comprehensible approach to logic synthesis and physical design for two-dimensional logic arrays *Proc. Design Automation Conference 94 (San Diego, June 1994)* pp 321–6
[30] Lee G 1997 Logic synthesis for cellular architecture FPGA using BDD *Proc. Asia and South Pacific Design Automation Conference 97 (Chiba, Jan. 1997)* pp 253–8
[31] Mishchenko A and Perkowski M 2002 Logic synthesis of reversible wave cascades *Proc. Int. Workshop on Logic and Synthesis (New Orleans, June 2002)* pp 197–202
[32] Gradshteyn I and Ryzhik I 1994 *Table of Integrals, Series, and Products* 5th edn (Boston: Academic)