4th Workshop on

Reversible Computation (RC)

Preliminary Proceedings

July 2nd-3rd, 2012
Copenhagen, Denmark

RC

http://www.reversible-computation.org

Garbageless Reversible Implementation of Integer Linear Transformations

Stéphane Burignat[1], Kenneth Vermeirsch[1], Alexis De Vos[1,2], and Michael Kirkedal Thomsen[3]

[1]Department of Electronics and Information Systems and [2]Imec v.z.w., Universiteit Gent, B - 9000 Gent, Belgium
{sburigna, Kenneth.Vermeirsch, alex}@elis.ugent.be
[3]Department of Computer Science, University of Copenhagen, DK - 2100 Copenhagen, Denmark
michael@kirkedal.dk

Abstract. Discrete linear transformations are important tools in information processing. Many such transforms are injective and therefore prime candidates for a physically reversible implementation into hardware. We present here reversible digital implementations of different integer transformations on four inputs. The resulting reversible circuit is able to perform both the forward transform and the inverse transform. Which of the two computations that actually is performed, simply depends on the orientation of the circuit when it is inserted in a computer board (if one takes care to provide the encapsulation of symmetrical power supplies). Our analysis indicates that the detailed structure of such a reversible design strongly depends on the prime factors of the determinant of the transform: a determinant equal to a power of 2 leads to an efficient garbage-free design.

1 Introduction

Linear transforms are important in analysis and compression of audio signals, images, video, and much more. A common property of most of these transforms is the existence of an inverse transform, meaning that they, in theory, are lossless reversible functions.

An important transform is the Fourier transform. Its fast discrete implementation is widely used in digital signal processing. For the Fourier transform there exists an inverse transform and it has therefore been researched in a reversible context [1]. A problem with the Fourier transform is the use of non-integer and complex values, that in numerical computations result in rounding of fixed-point or floating-point numbers and thus a lossy coding. To avoid complex-number arithmetic, the Fourier transform is often replaced by a discrete cosine transform (DCT), for which real-number arithmetic is sufficient. However, it still suffers from rounding errors due to floating-point arithmetic. Integer transforms, which are invertible integer approximations of a real-valued linear transform such as the DCT, can solve this last problem.

For implementation in reversible computing the 'lifting scheme' [2] is a powerful tool. The lifting scheme decomposes an injective computation into a series of 'reversible updates' [3]. Designs throughout this paper are based on reversible logic as described by Fredkin and Toffoli [4]. A key element in our designs is the V-shaped reversible binary adder introduced by Vedral et al. [5] and improved by Cuccaro et al. [6]. Detailed descriptions of these can be found in [7, 8]; detailed tests and measurements of adiabatic calculations in a fabricated adder can be found in [9]. The actual physical circuit is implemented using reversible dual-line pass-transistor CMOS technology [8–10].

2 Approximating the Discrete Cosine Transform

The 4×4 discrete cosine transform is defined by

$$C = \begin{pmatrix} a & a & a & a \\ b & c & -c & -b \\ d & -d & -d & d \\ c & -b & b & -c \end{pmatrix} , \tag{1}$$

where

$$\begin{aligned} a &= \tfrac{1}{2}\cos(0) = \tfrac{1}{2} &&= 0.500 \\ b &= \tfrac{1}{\sqrt{2}}\cos(\pi/8) = \tfrac{\sqrt{2+\sqrt{2}}}{2\sqrt{2}} &&\approx 0.653 \\ c &= \tfrac{1}{\sqrt{2}}\cos(3\pi/8) = \tfrac{\sqrt{2-\sqrt{2}}}{2\sqrt{2}} &&\approx 0.271 \\ d &= \tfrac{1}{\sqrt{2}}\cos(\pi/4) = \tfrac{1}{2} &&= 0.500 . \end{aligned}$$

Its matrix determinant is

$$\det(C) = 8ad(c^2 + b^2) = 1 . \tag{2}$$

An integer transform is obtained by successively multiplying by a constant scalar α and rounding [11]:

$$H_\alpha = \mathrm{round}(\alpha C) .$$

We have $\det(H_\alpha) \approx \det(\alpha C) = \alpha^4 \det(C) = \alpha^4$.

Choosing α equal to unity, yields the matrix

$$H_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & -1 \\ 1 & -1 & -1 & 1 \\ 0 & -1 & 1 & 0 \end{pmatrix} ,$$

with determinant equal to 8, i.e. a value deviating surprisingly much from $\alpha^4 = 1$. We see how the rounding step strongly influences the value of the determinant.

Choosing α equal to 2, gives the transformation

$$H_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix},$$

with determinant equal to 16, i.e. exactly equal to α^4. This is the well-known Walsh–Hadamard matrix, applied in data encryption, signal processing and data compression algorithms (such as MPEG-4) [12].

Given an invertible matrix (a matrix with non-zero determinant), it is possible to automatically generate a lifting scheme using decomposition. One algorithm for this is described in detail by De Vos and De Baerdemacker [13]. However, for transformation matrices with built-in symmetries, like the cosine transform, such standard decomposition is far from optimal. More efficient decompositions into scaling, swapping, and lifting matrices are possible [14]. Figure 1 shows a decomposition of the Walsh–Hadamard matrix, involving only four scalings (all with scaling factor 2), eight liftings (all with lifting factor ± 1) and four swaps. The circuit generates no garbage and has an appealing modular structure: it is built from four 2×2 Hadamard matrices

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

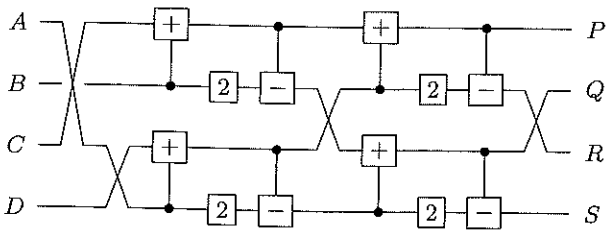


Fig. 1. Optimal lifting scheme for the Walsh–Hadamard 4×4 matrix.

The fact that the circuit does not generate garbage numbers is a most valuable property. It means the circuit can be used in reverse direction (i.e. from right to left in Fig. 1) in order to perform the inverse operation:

$$H_2^{-1} = \frac{1}{4} H_2 .$$

3 The H.264 Transform

For many applications, the Walsh–Hadamard is considered as too 'crude' approximation of the cosine transform. Values of α larger than 2 yield increasingly

more precise approximations to the real-valued DCT, but the resulting matrix coefficients will contain more diverse prime factors. Therefore, often the compromise choice $\alpha = 2.5$ is used. After multiplication by 2.5, matrix (1) becomes

$$\begin{pmatrix} 1.25 & 1.25 & 1.25 & 1.25 \\ 1.63 & 0.68 & -0.68 & -1.63 \\ 1.25 & -1.25 & -1.25 & 1.25 \\ 0.68 & -1.63 & 1.63 & -0.68 \end{pmatrix}. \tag{3}$$

Subsequent rounding yields

$$H_{2.5} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}, \tag{4}$$

with determinant 40, i.e. a value close to $\alpha^4 = \left(\frac{5}{2}\right)^4 = \frac{625}{16} \approx 39.063$. The choice of α is motivated by the absence of true multiplications in the resulting transformation. Indeed, conventional implementation of the factors of 2 in (4) is trivial both in hardware and in software, by merely using a bit shift operation.

The H.264 transform [11] is used in the MPEG-4/AVC video format. Figure 2 shows a decomposition with only four scalings, eight liftings and four swaps [14].

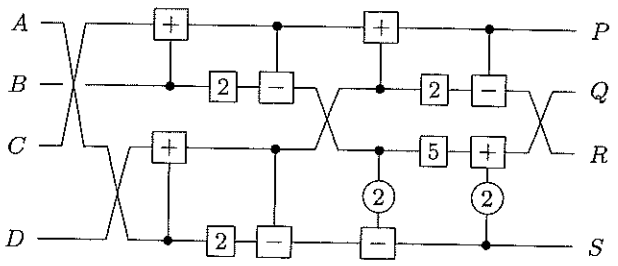
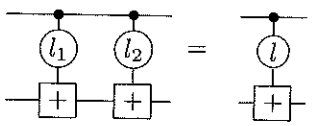


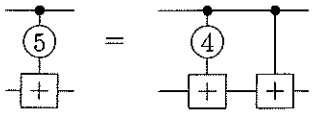
Fig. 2. Optimal lifting scheme for the H.264 discrete cosine transformation.

In contrast to the example of Sect. 2, we face here the problem of a scaling factor 5, i.e. a scaling that is different from a power of 2, the so-called 'perfect' coefficients [15]. Whereas scaling factors equal to a power of 2 can be implemented as a bit shift, other scaling factors cannot.

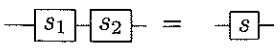
A lifting factor different from a power of 2 is no problem, because lifting factors add up:



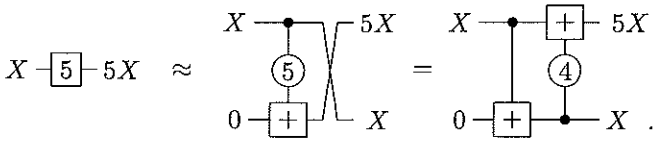
with $l = l_1 + l_2$. For example, the lifting factor 5 can be implemented with two 'perfect' liftings:



In contrast, a scaling factor different from a power of 2 is a problem, because scaling factors multiply:



with $s = s_1 s_2$. If the prime factorization of a scaling factor s contains prime factors different from 2, we need to convert the scaling into a lifting, by introducing a preset input and a garbage output:



The extra (preset) input line and extra (garbage) output line in fact means that we are embedding the 4×4 matrix within a 5×5 matrix. While the former has a determinant with an odd prime factor, the latter has a determinant which is a power of 2. Applying the above scaling-to-lifting transformation is embedding matrix (4) in the matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & \mathbf{0} \\ 2 & 1 & -1 & -2 & \mathbf{4} \\ 1 & -1 & -1 & 1 & \mathbf{0} \\ 1 & -2 & 2 & -1 & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{0} \end{pmatrix},$$

with determinant equal to -8, i.e. a 'perfect' value.

Instead of replacing only the scaling factor 5 by a lift, it is advantageous to replace the whole block consisting of the non-perfect scaling factor together with both the preceding lift and the succeeding lift giving the embedding of matrix (4) in

$$\begin{pmatrix} 1 & 1 & 1 & 1 & \mathbf{0} \\ 2 & 1 & -1 & -2 & \mathbf{0} \\ 1 & -1 & -1 & 1 & \mathbf{0} \\ 1 & -2 & 2 & -1 & \mathbf{-2} \\ \mathbf{0} & \mathbf{1} & \mathbf{-1} & \mathbf{0} & \mathbf{1} \end{pmatrix}, \tag{5}$$

with determinant equal to 8 and the lifting scheme shown in Fig. 3. Whereas the former embedding yields intermediate results ranging from -5 to 5 times the input data, the latter embedding restricts all intermediate and final data to the range from -3 to 4 times the input data. Thus reducing the increase in dynamic range of the signals is a valuable improvement.

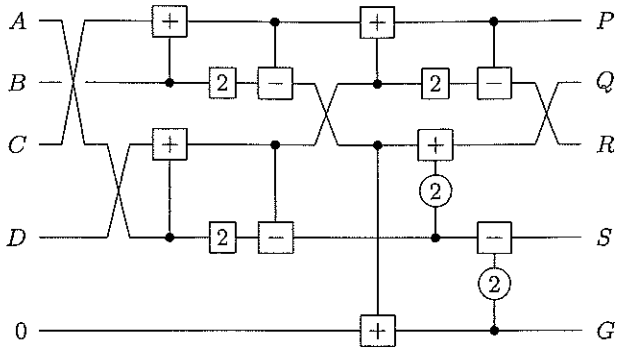


Fig. 3. Lifting scheme for the H.264 transform with simple multipliers but with garbage.

We end this section by noting that the appearance of a scaling factor different from 2^k is a direct and inevitable consequence of the fact that the determinant of the transform matrix does not have the form $\pm 2^k$. Indeed, as the determinant of any product of several matrices equals the product of the determinants of the separate matrices, the determinant of the whole circuit equals the product of the determinants of all the subcircuits. Because

- a swap has determinant -1,
- a lifting has determinant 1 (irrespective of its lifting factor l), and
- a scaling has determinant equal to its scaling factor s,

the full circuit has determinant $(-1)^m \prod_j s_j$, where m is the number of swaps in the circuit and s_j are the scaling factors of the circuit. For example, we can immediately see from Fig. 2 that its determinant equals $(-1)^4 \times 2 \times 2 \times 2 \times 5 = 40$, in accordance with the determinant of (4).

4 Implementing H.264

Our H.264 prototype coder is designed using four 3-bit unsigned integer inputs (A, B, C, and D) and four 6-bit signed integer outputs (P, Q, R, and S). The schematic implementation of the linear transform, given in (5) and Fig. 3, is designed in the *Cadence* computer-aided design environment, applying reversible dual-line pass-transistor CMOS style [8].

The design consists of two 4-bit reversible binary adders, two 4-bit reversible binary subtractors, one 5-bit adder, one 5-bit subtractor, one 6-bit adder, one 6-bit subtractor, and ten Feynman gates. Each reversible adder and subtractor being composed of $48w - 32$ transistors (where w is the bit width of the data: either 4, 5, or 6) and each Feynman of 8 transistors; the whole circuit thus contains 1648 transistors.

The prototype silicon chip contains a total of 1704 transistors (852 n-MOS transistors and 852 p-MOS transistors): the 1648 transistors for the actual coder plus 56 transistors for a small diagnostic circuits (that enables probing of intermediate results). It has been designed and fabricated in the standard CMOS technology of the foundry *ON Semiconductor*. The length of the transistors is 0.35 μm; the width is either 0.5 μm (n-MOS) or 1.5 μm (p-MOS). The threshold voltages are 0.6 volt (n-MOS) and -0.6 volt (p-MOS). The complete coder circuit fits into a rectangle of 610 μm $\times$ 300 μm. Figure 4 shows the prototype. One easily recognizes four 4-bit Cuccaro blocks (at the left-hand side), two 5-bit Cuccaro blocks (at the right-bottom part), and two 6-bit Cuccaro blocks (at the right-top part). The complete chip (bonding pads included) measures 2.2 mm $\times$ 2.2 mm = 4.8 mm^2 and is encapsulated in a 68-JLCC package.

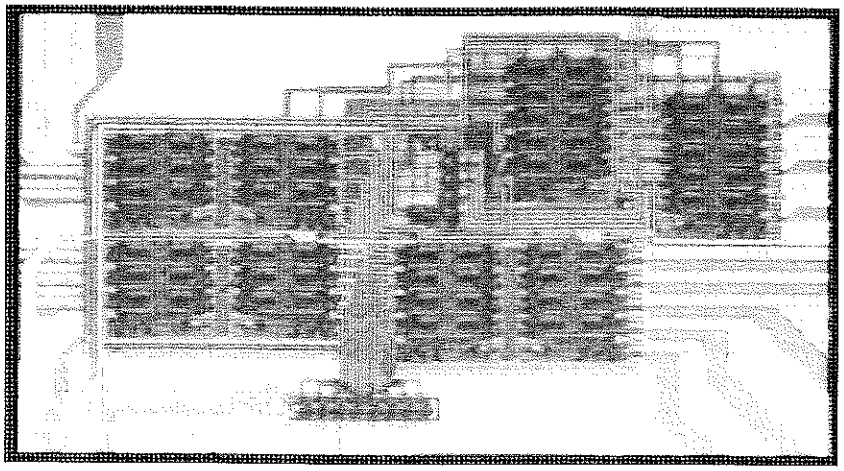
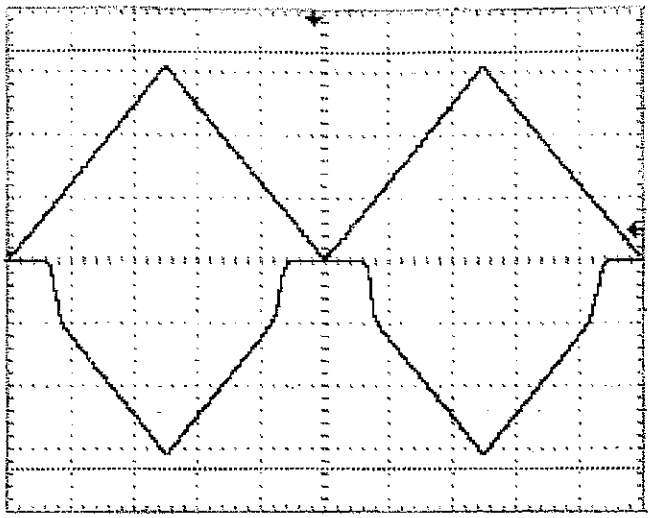


Fig. 4. Microscope view of the integrated circuit (680 μm $\times$ 380 μm).

Figure 5 shows a measurement of the functioning of the circuit. Positive voltages represent a logic 1; negative voltages represent a logic 0. Such so-called adiabatic signals are provided to each one of the inputs. We here see the input bit $A_0 = 1$. The resulting output bit P_4 turns out to be 0. In this example, the input vector $(A, B, C, D) = (111, 011, 000, 011)$ is used. Several other vectors have been tested, yielding positive as well as negative (coded in two's complement) output numbers.

5 Avoiding Garbage

The implementation of the discrete matrix is optimal with respect to the number of lifting steps and transistors. Unfortunately, as mentioned in Sect. 3, reversible implementation of the H.264 transform is hampered by the fact that its determinant equals $40 = 2^3 \times 5$, instead of a power of 2. The prime factor 5 is responsible



Vertical scale = 500 mV/div.; horizontal scale = 5 ms/div.

Fig. 5. Oscilloscope view of input bit $A_0 = 1$ and output bit $P_4 = 0$, for the input vector $A = 7$, $B = 3$, $C = 0$, and $D = 3$ (resulting in the output vector $P = 13$, $Q = 11$, $R = 7$, and $S = -2$).

for the creation of the garbage output G. This garbage datum makes it difficult to use of a same chip for both coding and decoding. In order to decode, i.e. to apply the inverse coding

$$H_{2.5}^{-1} = \frac{1}{20} \begin{pmatrix} 5 & 4 & 5 & 2 \\ 5 & 2 & -5 & -4 \\ 5 & -2 & -5 & 4 \\ 5 & -4 & 5 & -2 \end{pmatrix},$$

knowledge of the numbers P, Q, R, and S is insufficient to recover the values of A, B, C, and D. We need the extra knowledge of the value of G, an information that is not necessarily available. Standard solutions for this problem exist [16], uncomputing garbage, however at the expense of much extra hardware.

Therefore, it is worth investigating whether, especially for reversible hardware, an appropriate choice of the factor α (instead of $\alpha = 5/2$) would be advantageous. According to $\det(H_\alpha) \approx \alpha^4$, a choice $\alpha = 2^n$ looks promising. We therefore try the numbers 4 and 8. They result in the integer transforms

$$H_4 = \begin{pmatrix} 2 & 2 & 2 & 2 \\ 3 & 1 & -1 & -3 \\ 2 & -2 & -2 & 2 \\ 1 & -3 & 3 & -1 \end{pmatrix} \quad \text{and} \quad H_8 = \begin{pmatrix} 4 & 4 & 4 & 4 \\ 5 & 2 & -2 & -5 \\ 4 & -4 & -4 & 4 \\ 2 & -5 & 5 & -2 \end{pmatrix},$$

respectively. Unfortunately, because of the rounding of the matrix entries, the former matrix has determinant $320 = 2^6 \times 5$ (instead of $256 = 2^8$), whereas the

latter matrix has determinant $3712 = 2^7 \times 29$ (instead of $4096 = 2^{12}$). Thus, straightforward application of $\alpha = 2^n$ is no improvement. According to (2), we need a, d, and $b^2 + c^2$ each to be a power of 2. Well, unfortunately there exists no primitive Pythagorean triple $(b, c, 2^m)$.

A possible solution, is sticking to the choice $\alpha = 2.5$, but replacing the rounding step by a 'rounding up' step for the 1.63 entries of the matrix (3), but a 'rounding down' step for the 0.68 entries of the matrix, resulting in

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 0 & 0 & -2 \\ 1 & -1 & -1 & 1 \\ 0 & -2 & 2 & 0 \end{pmatrix}, \tag{6}$$

with determinant $32 = 2^5$. This, of course, has the disadvantage of lacking any international normalization. Nevertheless we give here its optimal reversible implementation in Fig. 6. Because it generates no garbage, the same circuit can be operated in both directions, without any garbage uncomputing circuitry.

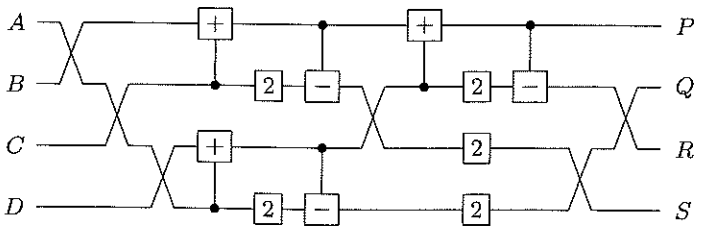


Fig. 6. Optimal lifting scheme for linear transform (6).

Finally, combining $\alpha = 8$ with 'creative rounding' suggests

$$\begin{pmatrix} 4 & 4 & 4 & 4 \\ 5 & 2 & -2 & -5 \\ 4 & -4 & -4 & 4 \\ 1 & -6 & 6 & -1 \end{pmatrix}, \tag{7}$$

with determinant $4096 = 2^{12}$. This solution has the following advantage compared to (6): its four successive rows display the traditional 0, 1, 2, and 3 sign changes. Matrix (7) also is a more accurate approximation of αC, if we use $\sum_{i=1}^{4} \sum_{j=1}^{4} |A_{ij}|$ as norm for a matrix A, according to [17]. Figure 7 shows the corresponding circuit. All scaling factors (i.e. 2, 2, 4, 8, and 32) can be implemented by appropriate bit shifts. All lifting factors (i.e. 5 and 6) can be decomposed into powers of 2 ($5 = 4 + 1$ and $6 = 4 + 2$, respectively) and thus can equally be implemented as bit shifts. From left to right, the circuit performs the transformation (7); from right to left, this same circuit performs the inverse

transformation

$$\frac{1}{64}\begin{pmatrix} 4 & 6 & 4 & 2 \\ 4 & 1 & -4 & -5 \\ 4 & -1 & -4 & 5 \\ 4 & -6 & 4 & -2 \end{pmatrix}.$$

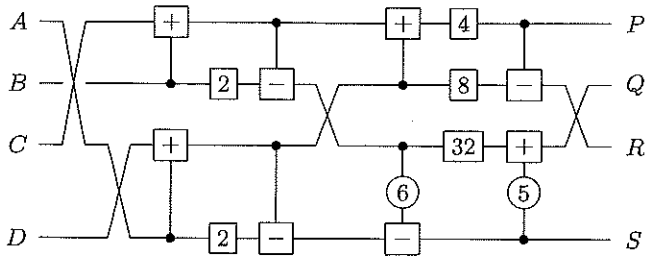


Fig. 7. Optimal lifting scheme for linear transform (7).

6 Conclusion

We have demonstrated that a single digital circuit can both perform a coding and a decoding operation. It suffices that the determinant of the coding matrix is of the form $\pm 2^k$. In analogy with the expression 'perfect scaling coefficient' introduced by Bruekers and van den Enden [15], we call such determinants 'perfect determinants'. They allow the implementation of an integer linear transform without creation of garbage numbers.

Acknowledgment

The authors would like to thank the *Danish Council for Strategic Research* for the support of this work in the framework of the *MicroPower* research project. They are also grateful to the *Invomec* division of *Imec v.z.w.* (Leuven, Belgium) and the *Europractice* consortium for the help with the prototyping of the chip.

References

1. M. Skoneczny, Y. Van Rentergem, and A. De Vos : Reversible Fourier transform chip. *Proceedings of the 15th International Conference on Mixed Design of Integrated Circuits and Systems*, Poznań (2008), pp. 281–286.

2. W. Sweldens : The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis* vol. 3 (1996), pp. 186–200.
3. H. Axelsen, R. Glück, and T. Yokoyama : Reversible machine code and its abstract processor architecture. In: Computer Science – Theory and Applications. *Springer Lecture Notes in Computer Science* vol. 4649 (2007), pp. 56–69.
4. E. Fredkin and T. Toffoli : Conservative logic. *International Journal of Theoretical Physics* vol. 21 (1982), pp. 219–253.
5. V. Vedral, A. Barenco, and A. Ekert : Quantum networks for elementary arithmetic operations. *Physical Review A* vol. 54 (1996), pp. 147–153.
6. S. Cuccaro, T. Draper, S. Kutin, and D. Moulton : A new quantum ripple-carry addition circuit. `arXiv:quant-ph/0410184v1`.
7. M. Thomsen and H. Axelsen : Parallelization of reversible ripple-carry adders. *Parallel Processing Letters* vol. 19 (2009), pp. 205–222.
8. A. De Vos : Reversible computing. *Wiley–VCH*, Weinheim (2010).
9. S. Burignat and A. De Vos : Test of a majority-based reversible (quantum) 4 bit ripple-adder in adiabatic calculation. *Proceedings of the 18th International Conference on Mixed Design of Integrated Circuits and Systems*, Gliwice (2011), pp. 368–373.
10. A. De Vos : Reversible computer hardware. *Electronic Notes in Theoretical Computer Science* vol. 253 (2010), pp. 17–22.
11. H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky : Low-complexity transform and quantization in H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology* vol. 13 (2003), pp. 598–603.
12. Hadamard transform. `Wikipedia` (2012).
13. A. De Vos and S. De Baerdemacker : Decomposition of a linear reversible computer: digital versus analog. *International Journal of Unconventional Computing* vol. 6 (2010), pp. 239–263.
14. A. De Vos, S. Burignat, and M. Thomsen : Reversible implementation of a discrete linear transformation. *International Journal of Multiple-valued Logic and Soft Computing* vol. 18 (2012), pp. 25–35.
15. F. Bruekers and A. van den Enden : New networks for perfect inversion and perfect reconstruction. *IEEE Journal on Selected Areas in Communications* vol. 10 (1992), pp. 129–137.
16. T. Yokoyama, H. Axelsen, and R. Glück : Optimizing reversible simulation of injective functions. *International Journal of Multiple-valued Logic and Soft Computing* vol. 18 (2012), pp. 5–24.
17. A. Baker : Matrix Groups. *Springer*, London (2002).