Reversible implementation of a discrete integer linear transformation

Alexis De Vos
Vakgroep elektronika en informatiesystemen
Imec and Universiteit Gent
Sint Pietersnieuwstraat 41
B - 9000 Gent
Belgium
Email: alex@elis.ugent.be

Stéphane Burignat
Vakgroep elektronika en informatiesystemen
Universiteit Gent
Sint Pietersnieuwstraat 41
B - 9000 Gent
Belgium
Email: research@burignat.eu

Michael Kirkedal Thomsen*
Dept. of Computer Science
University of Copenhagen
Universitetsparken 1
DK - 2100 Copenhagen
Denmark
Email: michael@kirkedal.dk

***Abstract*—Discrete linear transformations form important steps in processing information. Many such transformations are injective and therefore are prime candidates for a physically reversible implementation into hardware. We present here the first steps towards a reversible digital implementation of two different integer transformations on four inputs: The Haar wavelet and the H.264 transform.**

I. Introduction

Transforms are today an important tool used for 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.

Today, one of the most known and most used transforms is the Fourier transform that in its fast, discrete implementation is widely used in *e.g.* digital signal processing. For the Fourier transform there exists an inverse transform and it has therefore been researched in a reversible context. Most known perhaps is Shor's factorization algorithm [1] that makes use of a quantum Fourier transform, but the Fourier transform has also been implemented in classical reversible CMOS logic circuits [2]. 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. Approximation algorithms can solve the loss of information but time consuming multiplications are unavoidable.

Today, much focus has moved from the Fourier transform to *wavelet* transforms [3], [4] that are faster than the Fourier transform[1] and allow mapping integers to integers [5]. For implementation in reversible computing the *lifting scheme* [6] is a powerful tool and its use in wavelets was quickly acknowledged [7]. The lifting scheme decomposes an injective computation into a series of *reversible updates* [8].

In this work, as an example, we focus on the wavelet used for the *H.264 video coding standard* as described by Malvar *et al.* [9]. This coding, used in the AVC video format, is designed such that it only requires simple integer arithmetic. A reversible design using the lifting scheme is proposed and the schematic for the physical implementation is presented. In order to outline the advantages and disadvantages of the H.264 wavelet, we first relate it to the simpler *Haar wavelet*.

Diagrams and designs throughout this paper are based on reversible logic as described by Fredkin and Toffoli [10]. One key element in our design is the V-shaped reversible binary adder designed by Vedral *et al.* [11] that has later been improved [12], [2]. Detailed descriptions of these can be found in the previously mentioned literature or [13]. The actual physical implementation is done using reversible dual-line pass-transistor CMOS technology [14].

We begin in Section II by describing the Haar transform and its design by a lifting scheme. Section III shows the H.264 transform and design, followed by its implementation in Section IV. Finally, in Section V, we conclude and look at future work.

II. The Haar Wavelet

The very first wavelet has been credited to Haar[2] and is a simple discrete wavelet. It is a special case of the *Daubechies wavelets* [3] and has the *orthogonal basis* of row-vectors, defined by the following 4×4 Haar matrix:

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix} . \tag{1}$$

The applied matrix of the discrete transforms must be orthogonal (*i.e.* the basis of row-vectors must be *orthonormal*), which is achieved by normalizing each row-vector such that

$$H_t = \text{diag}(\frac{1}{2}, \frac{1}{2}, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})\, H \ . \tag{2}$$

For a 4×4 input matrix X, the transformed Y is calculated by $H_t\, X\, H_t^T$ (where H^T is the transpose of H) giving that

$$Y = \text{diag}(\frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2})\, H\, X\, H^T \ . \tag{3}$$

*At the time of writing visiting Vakgroep elektronika en informatiesystemen, Universiteit Gent.

[1]Wavelet transforms have generally a computational time of $\mathcal{O}(n)$ compared to the Fourier transforms $\mathcal{O}(n \log n)$, where n is the size of the data.

[2]Alfréd Haar proposed this function in 1909, long before the term and theory of wavelets was used. It has later been recognized as a wavelet.

The transformation can thus be calculated using only integer arithmetic, as the diagonal matrix in practice is not applied.

A. Decomposition into Lifting Scheme

As mentioned in the introduction, the lifting scheme [6] can be used for implementing wavelets, but it is also a very powerful tool for reversibly implementing linear transformations. 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 Baerdemacker [15].

First, as the determinant of H is different from unity, we decompose H into a diagonal matrix D and a special matrix G (*i.e.* a matrix with unit determinant):

$$H = \begin{pmatrix} D_{11} & 0 & 0 & 0 \\ 0 & D_{22} & 0 & 0 \\ 0 & 0 & D_{33} & 0 \\ 0 & 0 & 0 & D_{44} \end{pmatrix} G\ , \tag{4}$$

where $\det(D) = D_{11}\,D_{22}\,D_{33}\,D_{44} = \det(H) = 8$. When decomposing an $n \times n$ matrix this way, the standard procedure is choosing all diagonal elements equal to 1, except D_{nn} which equals $\det(H)$. This will finally yield one scaling factor and $n^2 - 1$ lifting factors, as the matrix G is decomposed into $2n - 1$ lifting matrices:

$$\begin{aligned} G = &\begin{pmatrix} 1 & L_{12} & L_{13} & L_{14} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ L_{21} & 1 & L_{23} & L_{24} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L_{31} & L_{32} & 1 & L_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ L_{41} & L_{42} & L_{43} & 1 \end{pmatrix} \\ &\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & R_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & R_{23} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & R_{12} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \ , \end{aligned} \tag{5}$$

complemented by zero to $n-1$ permutation matrices. If we use more than one non-identity scaling factor D_{jj}, we may use the extra degrees of freedom in order to obtain the more integer entries in the decomposition. Here, we have a maximum of four scaling factors (*i.e.* D_{11}, D_{22}, D_{33}, and D_{44}) and fifteen lifting factors (*i.e.* L_{12}, L_{13}, L_{14}, L_{21}, L_{23}, L_{24}, L_{31}, L_{32}, L_{34}, L_{41}, L_{42}, L_{43}, R_{34}, R_{23}, and R_{12}.)

Even if we restrict ourselves to positive integer factors, there exist no less than 20 different factorizations $D_{11}\ D_{22}\ D_{33}\ D_{44}$ of the number 8 and each choice strongly influences the matrix G and thus its decomposition. The number of non-unit scaling factors ranges from 1 to 3; the number of non-zero lifting factors ranges from 10 to 12. Because we restrict ourselves to integer scaling factors, all lifting factors are rational. The denominators of these rationals range from 1 to 16. An example of a decomposition with a small number of lifting coefficients, each with a small denominator, is as follows:

$$\begin{aligned} H = &\begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\ &\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{6}$$

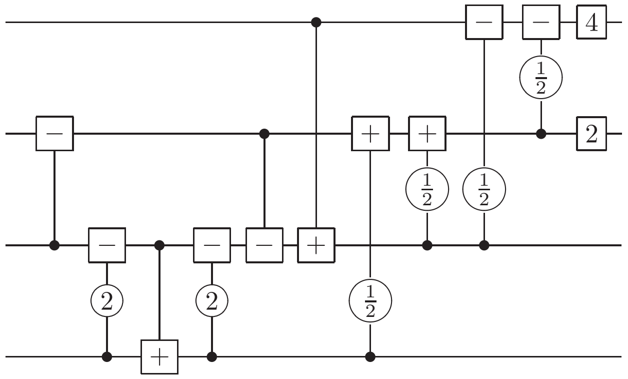


Fig. 1. Possible automatic decomposition of the Haar 4×4 matrix with a column vector input using the algorithm presented in [15], chosen for its simple lifting factors.

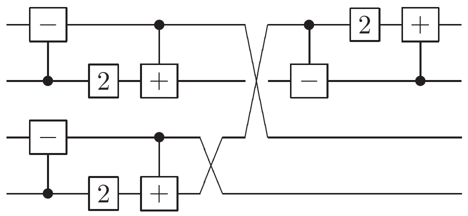


Fig. 2. Optimal lifting scheme for the Haar 4×4 matrix of a column vector input.

with two scaling factors (i.e. 2 and 4) and ten lifting factors (i.e. -2, -1, $-1/2$, $1/2$ and 1 (all numbers appearing twice)). Fig. 1 shows the corresponding logic circuit.

Although we have here the choice with the fewest and simplest lifting factors among the twenty possibilities, this decomposition is still far from optimal. Many more decompositions into scaling, swapping, and lifting matrices are possible. *E.g.* applying more than one diagonal matrix can yield an optimal solution[3]: Fig. 2 shows a decomposition containing two diagonal matrices, leading to only 3 scalings, 6 liftings and 1 permutation. The circuit also has a more modular structure.

III. The H.264 Transform

More recent is the H.264 transform [9] that is used in the MPEG-4/AVC video format. This discrete cosine transformation has the integer approximation given by:

$$H_{264} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix} . \tag{7}$$

Again the determinant is different from unity (it amounts to 40) and restricting ourselves to positive integer factors, there exist no less than 80 different positive integer factorizations $D_{11}\ D_{22}\ D_{33}\ D_{44}$. In the automatic lifting scheme generation,

[3]An informal proof of optimality can be made by starting with the identity matrix and showing the least number of row operations needed to generate a matrix without zeroes. We can call this an inverse Gaussian elimination. Gaussian elimination can also be used for decomposition of the matrix into lifting steps.

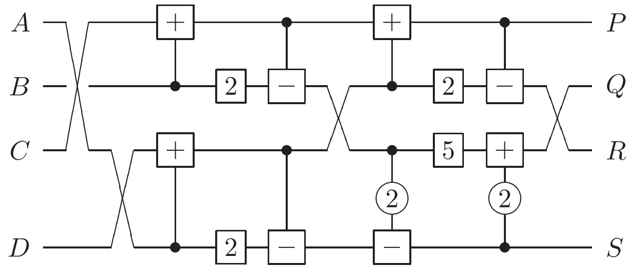


Fig. 3. Optimal lifting scheme for the H.264 discrete cosine transformation.

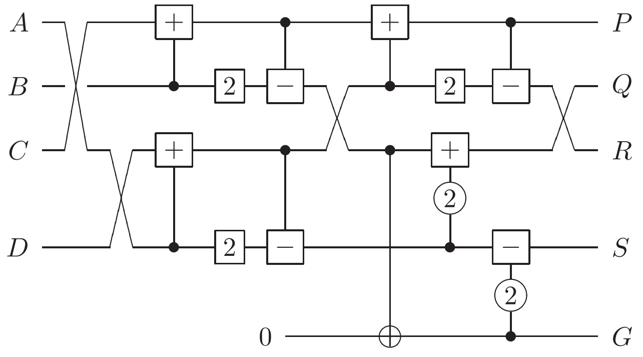
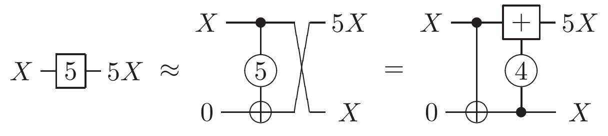


Fig. 4. Lifting scheme for the H.264 discrete cosine transformation with the simple multiplication but with garbage.

the number of non-unit scaling factors ranges from 1 to 4, the number of non-zero lifting factors ranges from 11 to 15, and the denominators of the lifting factors range from 1 to as high as 7480.

Applying more than one diagonal matrix yields a better solution and Fig. 3 shows a decomposition with only 4 scalings, 8 liftings and 3 permutations.

In contrast to the example of Sec. II, we face here the problem of a scaling factor that is different from a power of 2, the so-called perfect coefficients [16]. Whereas scaling factors equal to a power of 2 can be implemented as a bit shift, other scaling factors cannot. To have a simple implementation in reversible logic we need to introduce a preset input and a garbage output:



An extra (preset) input line and an extra (garbage) output line in fact means that we are embedding the 4×4 matrix within a 5×5 matrix. The former having 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 (7) in the matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 2 & 1 & -1 & -2 & 4 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -2 & 2 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 \end{pmatrix}, \qquad (8)$$

with determinant equal to -8, *i.e.* a perfect coefficient. Instead of replacing only the scaling factor 5 by a lift, it is advantageous to replace the whole block consisting of the scaling factor together with the preceding lift and succeeding lift giving the embedding of matrix (7) in

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 2 & 1 & -1 & -2 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -2 & 2 & -1 & -2 \\ 0 & 1 & -1 & 0 & 1 \end{pmatrix}, \qquad (9)$$

with determinant equal to 8 and the lifting scheme shown in Fig. 4. 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.

IV. Implementation of H.264

In order to limit the number of input and output bits such that the H.264/AVC coder may be embedded in a 68-*pin package*, our chip is designed using 3-bit unsigned integer inputs and 6-bit signed integer outputs, all of which are both represented by the signal and its negation. Fig. 5 presents a simplified diagram of the detailed schematic implementation of the linear transform given in (9) and Fig. 4, designed in *Cadence computer-aided design environment*.

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 word length of the data: either 4, 5, or 6) and each Feynman of 8 transistors, the whole chip thus contains 1648 transistors for this particular design.

First, in Box 1 (see Fig. 5) we describe the addition/scaling-by-2/subtraction step that is used twice in this box. Starting from 3-bit unsigned integers, all four inputs are zero-extended to 4 bits. Then the first stage of adders calculate the 4-bit sum of two (originally 3-bit) numbers by updating one of them. Next, the multiplication-by-two of the non-updated number is done by a bit-roll up. Remember that this 4-bit number was zero-extended from a 3-bit integer and thus its most significant bit is equal to 0. Finally, using the 4-bit numbers, we subtract the sum from the product. It is not yet necessary to extend the number representation when we consider the difference as a signed 4-bit integer. The result of Box 1 is therefore two unsigned 4-bit sums and two signed 4-bit differences. Using such mixed representation of the binary integers reduces the number of necessary transistors.

Now, in Box 2 a calculation similar to the two in Box 1 (with an extra bit-size, see Fig. 5) is performed on the two 4-bit unsigned sums, resulting in a 5-bit unsigned number and a 5-bit signed number. In order to have homogeneous outputs, the numbers are extended to 6-bit (signed) numbers using a zero- and a sign-extend[4], respectively. Also, one of the two

[4]An n-bit two's complement signed number is *sign-extended* to a $n+1$-bit number by copying the most significant bit, using a single Feynman gate.

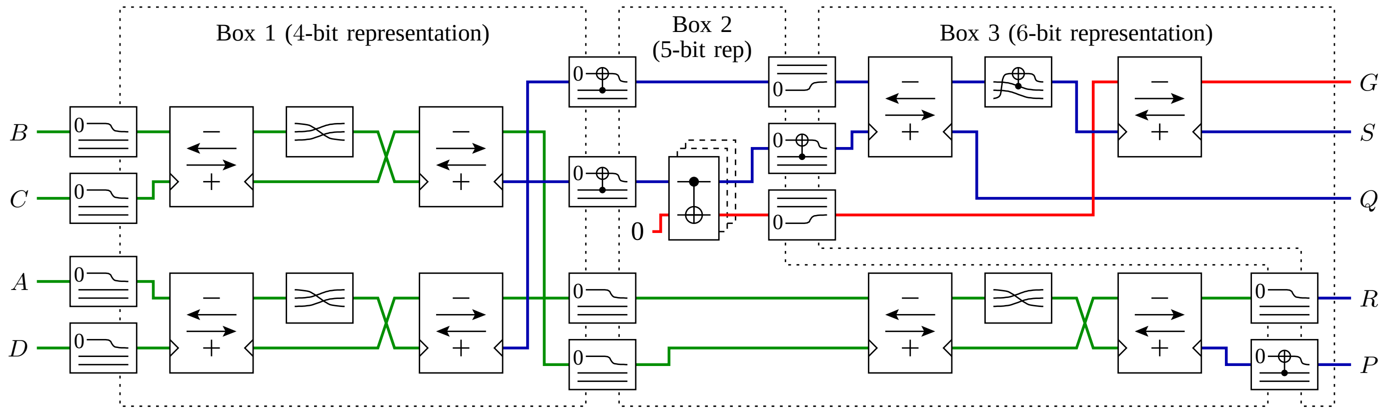


Fig. 5. Diagram outlining the *Cadence* schematic of the H.264 discrete cosine transformation. The green wires carry unsigned data, blue wires carry signed data, and red wires carry signed preset/garbage. The outputs are $P = A+B+C+D$, $Q = 2A+B-C-2D$, $R = A-B-C+D$, $S = A-2B+2C-D$, and the garbage $G = 2B - 2C$.

differences is sign-extended and copied to a garbage line using 5 Feynman gates.

In Box 3, the copied value is sign-extended to a 6-bit number and the double of the other difference (this value is both sign-extended one bit and multiplied by 2)[5] is added. Finally, the double of the copy is subtracted from the remaining difference with the correct sign-extension.

All this results in the expected four 6-bit signed integer outputs (P, Q, R, S) and a garbage output (G). By adding extra logic, it is possible to uncompute this garbage.

V. Conclusion and Future Work

In this paper we have shown the initial steps for implementing the H.264/AVC transform on a physical chip using reversible dual-line pass-transistor CMOS technology. The implementation of the discrete wavelet matrix is optimal with respect to the number of lifting steps and uses only 1648 transistors.

The paper has mainly focused on the encoding of signals but decoding is just as important. In theory decoding is the inverse transformation, but in practice, when transforms are used for signal compression, a lossy step is purposely added after encoding. This additional step deletes less important signal information, by rounding of numbers. Therefore, decoding is done by applying another matrix [9] and this matrix is dependent on the amount of information one deletes. The rounding and decoding matrix as presented in [9] actually leads to a lossy (non-reversible) decoding as integer rounding can occur. Solving this problem is important for the design of a single circuit where reversibility is used to both compute encoding and decoding.

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.

[5]Multiplication-by-2 is done by adding a 0 as the least significant bit.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.
[2] M. Skoneczny, Y. Van Rentergem, and A. De Vos, "Reversible Fourier transform chip," in *Proceedings of the 15th International Conference on Mixed Design of Integrated Circuits and Systems*, Poznań, 2008, pp. 281–286.
[3] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 41, no. 7, pp. 909–996, 1988.
[4] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 7, pp. 674–693, July 1989.
[5] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, 1998.
[6] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.
[7] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–269, May 1998.
[8] H. B. Axelsen, R. Glück, and T. Yokoyama, "Reversible machine code and its abstract processor architecture," in *Computer Science – Theory and Applications. Proceedings*, ser. LNCS, V. Diekert, M. V. Volkov, and A. Voronkov, Eds., vol. 4649. Springer-Verlag, 2007, pp. 56–69.
[9] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 598–603, July 2003.
[10] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3-4, pp. 219–253, 1982.
[11] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, pp. 147–153, July 1996.
[12] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," *arXiv:quant-ph/0410184v1*, 2005.
[13] M. K. Thomsen and H. B. Axelsen, "Parallelization of reversible ripple-carry adders," *Parallel Processing Letters*, vol. 19, no. 1, pp. 205–222, June 2009.
[14] A. De Vos, "Reversible computer hardware," in *Electronic Notes in Theoretical Computer Science*, vol. 253, 2010, pp. 17–22, Proceedings of the Workshop on Reversible Computation (RC 2009), York.
[15] A. De Vos and S. De Baerdemacker, "Decomposition of a linear reversible computer: digital versus analog," *International Journal of Unconventional Computing*, vol. 6, 2010.
[16] F. Bruekers and A. van den Enden, "New networks for perfect inversion and perfect reconstruction," *Selected Areas in Communications, IEEE Journal on*, vol. 10, no. 1, pp. 129–137, Jan 1992.