Encoded Arithmetic for Reversible Logic

(preliminary version)

AKHILESH TYAGI
Department of Computer Science
Atanasoff Hall, Iowa State University
Ames, IA 50011-1041

Abstract

The CCD based implementations of reversible logic consume constant amount of energy per switching event which depends only on the charge packet size and not on the interconnect length. Within this model of computation, it seems possible to leverage data encoding to reduce number of switching events for the computation resulting in lower overall computation energy. We explore the applicability of encoding for different datapath functions in this paper. We also develop a lower bound on switching count in a model similar to the traditional VLSI model of computation. A notion of reversible communication complexity is also developed.

1 Introduction

Many recent reversible logic implementation proposals, most notably Merkle's [Mer93], use charge packets to represent the logic state. The switches operate in a gradual manner (also known as electroid model [Hal92] or adiabatic switching [KA92]) once the two ends have been established at the same logic state. The switching energy of the switching transition in these models can be traded for the speed of switching almost arbitrarily. Another interesting aspect of these switching systems based on charge packets is that the switching energy is not proportional to the length of the interconnect any more. The amount of charge needed to establish a charge packet is not a function of the capacitance of the output node. This is a significant variant over the VLSI complexity theory model where the amount of charge transferred for a switching event is taken to be proportional to the length of the wire being driven. In this traditional VLSI model of computation, encoding of information is used as an adversary in the lower bound arguments. When n information bits need to be transmitted between two parts of a computing system, the amount of switching can be reduced by expanding the n bits of information into $m > n$ physical bits. Tyagi [Tya88] shows that the average switching in this case is given by $\Omega\left(\frac{n}{\log(m/n)}\right)$. Hence an exponential encoding (accomplished by a decoder) results in $O(1)$ switching. It has been shown Tyagi [Tya88], [Tya89], Aggarwal *et al.* [ACR88] that encoding of information, although reduces the total switching, still results in higher switching energy for majority of interesting functions. This is because by expanding information (encoding by introducing redundancy) the area of the implementation also increases which results in longer wire lengths. The effect of reduced switching due to encoding is overwhelmed by the increased wire lengths resulting in higher overall energy consumption. The independence of switching energy per switching event and wire lengths in the charge packet based implementations suggests the use of encoded arithmetic so that the number of switching events can be reduced. This results in a further reduction in switching energy. This paper addresses the benefits derivable from encoded arithmetic. In particular, all the functions with the communication complexity [Yao79, Yao81] of $O(1)$ do not benefit at all from encoding. The encoding results in higher area at no reduction in total switching. An adder, an ALU or a counter are examples of this type of function. However, transitive functions [Vui83] which include shifting, integer multiplication and matrix multiplication, can benefit significantly from encoding. For instance, an n-bit barrel shifter takes area $O(n^2)$, time $O(\log n)$ and results in $O(n \log n)$ switching events. When these n bits are encoded into $m > n$ bits s.t. $m \leq 2^n$ the area goes up to $O(m^2)$, time stays at $O(\log n)$ (assuming unit time per switching event) and the switching count comes down to $\frac{n \log n}{\log(m/n)}$. Integer multiplication exhibits similar set of savings in switching count. Note that the savings in switching count come at a significantly increased cost in area. There is also the cost of encoding and decoding both in terms of additional switching count and area. However, such an encoding and decoding need be performed only once if every datapath component is designed to work with encoded data.

Another issue explored in this paper is to develop a

0-8186-6715-X/94 $4.00

rudimentary complexity theory for the VLSI model with unit switching energy cost which reflects the realities of charge packet based computing systems. In particular, the attributes with significant change in lower bounds in this model are energy E and switching count. Note that the lower bound arguments for area and AT^2 are still valid. We demonstrate a lower bound of $\Omega(I \log I)$ on the switching count of a non-encoded computation of a function with communication (information) complexity of I. This establishes a tight lower bound of $\Omega(n \log n)$ on the switching count of shifting and integer multiplication (in fact on all transitive function computations). This proof can then be extended to show an $\Omega\left(\frac{n \log n}{\log(m/n)}\right)$ lower bound on the switching count of encoded arithmetic. The switching energy lower bound is also given by $\Omega(I \log I)$.

The third part of this work introduces the notion of *reversible communication complexity*. Yao [Yao79] introduced the notion of communication complexity for a function f of n input bits as follows. Given a partition of n input bits into two equal-sized sets I_L and I_R, and assuming that both the sides know the truth table for the function f, what is a lower bound on number of bits that need to be exchanged between two sides before one of them can decide on the function value? The communication complexity (or *information* complexity $I(f, n)$) of a function f characterizes the complexity of its implementation. This definition of communication complexity can be modified so that it models the number of bits exchanged between the two sets assuming that the computation is to be reversible. In general, the reversible communication complexity of any function with n input bits can be shown to be at most $2n$. If all the input bit combinations were equally likely and we dealt with a model consisting of Fredkin gates then a lower bound of n on the reversible communication complexity can also be derived. We have not been able to formulate or derive any non-trivial lower bounds on reversible communication complexity so far.

2 Preliminaries

This section provides some previously established results and a brief description of the model. The VLSI model of computation is adopted more or less from Thompson [Tho79]. Most features of this model are likely to be true for a charge packet based CCD like technology. The essential features of the model are: computation graph is embedded in a Cartesian grid; there are a constant number of communication layers in this grid; processing gates are embedded at the grid crossings with zero or unit area; the initial data values are localized to some constant area. Note that the Cartesian grid alongwith the finite number of routing layers assumption also stipulates that the fanin to each gate is limited to some constant.

Let us formalize the notion of switching count. We say that in a bit sequence $a_1, a_2, \ldots, a_l$, there is an *alternation* at position j if $a_j \neq a_{j+1}$ for $1 \leq j \leq l-1$. Let δ_j be 1 if $a_j \neq a_{j+1}$ and 0 otherwise. Then the *total alternation* for a l-bit sequence is given by $\sum_{j=1}^{l-1} \delta_j$. The following lemma from Tyagi [Tya89] provides a combinatorial result about average switching count in transmitting k information bits with the code space of $k' > k$ physical bits.

Lemma 1 *Let k' and k be two positive integers such that $k' \geq k$. The average number of alternations in transmitting a k' bit encoding of k information bits, $A(k', k) \geq k/4 \log(4k'/k)$.*

PROOF SKETCH: The proof is based on a greedy constructive method. Let $S_{k',i}$ denote the set of all the $k'-$bit strings with exactly i 1's. Thus $S_{2,0}$ contains only 00 and $S_{2,1}$ contains 01 and 10. There are $\binom{k'}{i}$ strings in $S_{k',i}$. We prove in [Tya89] that a set of the form $S_l = \cup_{i=0}^{l} S_{k',i}$, for $l \leq k'/2$, has the smallest average Hamming distance for its size. For $l = k/4 \log(4k'/k)$ this set also contains 2^k distinct codes and hence the result. □

3 Encoded Arithmetic

Encoding can be used to save on switching as proven in Lemma 1. The intuition behind this is that if only 2^n distinct events need to be transmitted, by choosing a selected subset of 2^n, m-bit vectors for $m > n$ (a subset with small average pair-wise Hamming distance) to represent these 2^n events the average switching count can be reduced. However, we still need at least one switching event since we cannot physically create fractional switching events. This is the reason why there is no apparent benefit from encoding in a function such as adder. An adder needs to transmit only one bit (a carry bit) between its bit slices. By encoding the n input bits of an adder into a larger number of bits m, the one bit communication requirement between bit slices still remains. We only add extra area of encoding without any switching count benefit in this case. This observation holds for all the functions with $O(1)$ communication complexity. There might be marginal improvement in switching count for a function which transmits say 3 bits between its bit slices. But, in most cases, this saving might not be worth the increase in area.

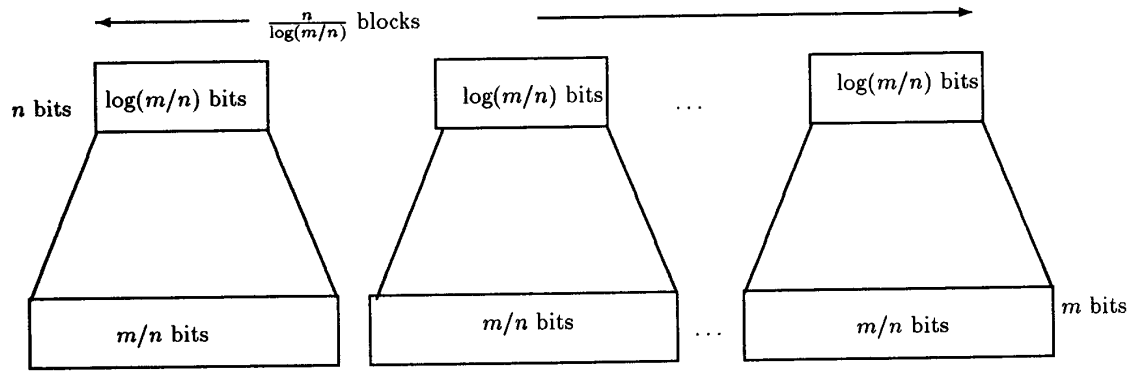

Figure 1: The Canonical $n \longrightarrow m$ Encoding Scheme

Encoding scheme: The canonical encoding we use for the datapath functions is as follows (shown in Figure 1. To encode n bits into $m > n$ bits, consider $\frac{n}{\log(m/n)}$ blocks of $\log(m/n)$ bits. Each such block is sent through a $\log(m/n) \longrightarrow (m/n)$ decoder resulting in approximately m encoded bits. The output of the encoder consists of $\frac{n}{\log(m/n)}$ blocks of (m/n) bits each. Each of these blocks contains exactly one 1 and all the rest of the bits are 0. The pairwise Hamming distance of any two such output words then is at most $\frac{2n}{\log(m/n)}$. This is because the Hamming distance of any two output blocks (of (m/n) bits) is at most 2. This type of encoding can be accomplished with at most $\frac{2n}{\log(m/n)}$ switching count using $\frac{n}{\log(m/n)}$ copies of $\log(m/n) \longrightarrow (m/n)$ decoders. However, this would require gates with fanin of n. Hence, a realization of such an encoder as a traditional $\log(m/n) \longrightarrow (m/n)$ decoder along the lines of decoder design in Mead-Conway [MC80] requires exactly $\log(m/n)$ switchings with $\frac{2m\log(m/n)}{n}$ area. Hence the total switching count of $n \longrightarrow m$ encoding is n with an area requirement of $2m$. Note this already precludes encoding in any datapath function where the maximum switching savings are of the order of 1 per input bit (such as adder) if the encoding were to be done only for that datapath function.

We can similarly estimate decoding energy for $m \longrightarrow n$ conversion. In $(m/n) \longrightarrow \log(m/n)$ decoding each of the $\log(m/n)$ output bits is an $\vee$ of $(m/2n)$ input bits. This can be designed similar to the *or-plane* of a PLA requiring $\frac{2m\log(m/n)}{n}$ area. Since roughly half of the $\log(m/n)$ bits are expected to switch each needing (m/n) switchings to generate, the total switching is $\frac{m\log(m/n)}{n}$. The switching can be reduced at the cost of area, but it cannot be lowered beyond $(\log(m/n))^2$ due to constant fanin restriction. The first case results in $2m$ area with m switching for $m \longrightarrow n$ decoding. The second design can reduce the switching to $n\log(m/n)$.

Datapath Functions: Now let us consider the impact of this encoding/decoding on the design of several datapath functions. Let us first consider an adder. We can illustrate the problems and savings involved with the example of a carry-ripple adder. Consider 8-bit addition. Say each block of 2-bits has been encoded into 4 bits. Hence the addition of A = 0010 0010 and B = 1001 0001 corresponds to adding encoded values: A' = 0001 0100 0001 0100 and B' = 0100 0010 0001 0010. Note that 0001 corresponds to a zero block, or in general, the position of 1 in a k bit block say i for $0 \leq i \leq (k-1)$ encodes the decimal value i. Addition of 2 such 4-bit blocks results in a 8-bit unary block, for instance 0100 + 0010 = 00001000. This requires shifting all the zeros to the left of the bit 1 in one addend to the less-significant bit positions of the other addend. This suggests a bit-serial implementation which is slow, but also saves on area. The number of switchings in this case can be as high as the number of bits in each encoded block, *i.e.*, (m/n) in the general case. On the other extreme, a higher area design would have m/n $\wedge$ gates per output position i to consider all the additive factors of the number i. In the constant fanin model, the output values of these m/n $\wedge$ gates need to be *or*ed requiring $\log(m/n)$ switchings for exactly one output bit position per block. But, recall that the original carry-ripple adder would have $O(\log(m/n)$ switchings for this block in any case. Hence there are no real switching savings despite the area penalty in this case. This would provably be the case for all datapath functions with information (communication) complexity of $O(1)$. Information complexity is the number of bits that need be exchanged between any partition of input bits into two evenly sized sets for the function computation to be correct. This is described in more detail in Section 5. For instance, for addition there exists a partition of input bits with $a_1, b_1, a_2, b_2, \ldots, a_{n/2}, b_{n/2}$

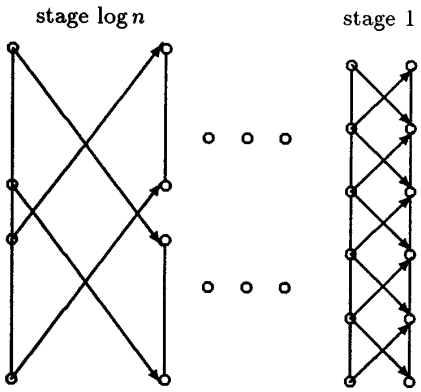

Figure 2: The $\log n$ Time Barrel Shifter

and $a_{(n/2)+1}, b_{(n/2)+1}, a_{(n/2)+2}, b_{(n/2)+2}, \ldots, a_n, b_n$ such that exactly one bit (the carry bit) need be exchanged between the two sets.

Lemma 2 *All the type-1 datapath functions, the functions with $O(1)$ information complexity, do not benefit from encoding of input.*

The proof would need to argue that any encoding still results in one switching per bit position which is zero gain in switching count.

Another difficulty encountered in the encoded addition outlined above is the variation in the number of encoded bits per block. For a consistent encoding, we wish to retain the same number of encoded bits per block (some m/n). This is specially important if we wish to encode and decode only once during the computation. But the result block of addition of two (m/n)-bit blocks has length $(2m/n)$. Fortunately, these blocks can be rearranged to restore the consistency of m/n encoded bits per block. However, it requires a carry propagation through $\log(m/n)$ adjacent blocks resulting in at least $\log(m/n)$ switchings per re-arrangement. Once again, this rearrangement need be performed only once per addition.

Let us consider shifting next. The barrel shifter design of Figure 2 requires $n \log n$ switchings for an area of $O(n^2)$. This shifter design can be extended in the way. The n data bits are encoded into m bits as described in our canonical encoding scheme, *i.e.*, each block of adjacent $\log(m/n)$ data bits is encoded into (m/n) bits. Note that the $\log n$ control bits for the shifter need not be encoded. Now each of the $\log n$ columns has m bits passing through it. Each block of $\log(m/n)$ bits from the non-encoded design (m/n encoded bits) is connected to the corresponding block of the next stage. This shifter only requires $\frac{n \log n}{\log(m/n)}$ switching at an increased area requirement of m^2.

Now let us consider integer multiplication. The integer multiplication has two parts to it. Consider the traditional shift and add multiplier. First, for the row multiply step, each encoded block of the multiplicand is multiplied by the current block of the multiplier (instead of a bit of a multiplier). This is unary multiplication of two (m/n) bit blocks each with exactly one 1. This involves replicating the string of 0s preceding the 1 in multiplicand block for each 0 but one preceding the 1 in the multiplier block. This step can be performed with (m/n) switchings per block multiplication for a bit-serial computation with a method reminescent of block additions. The best achievable switching count would be $\log(m/n)$ per block multiplication with a higher area cost. The (m/n) switching cost block multiplication costs even more in switchings than the unencoded multiplication. The $\log(m/n)$ block multiplication however has a total switching cost of $\frac{n^2}{\log(m/n)}$ which is $\log(m/n)$ factor better than the unencoded *shift-and-add* multiplier.

The pipelined $n \times n$ array multiplier also exhibits similar savings in switching count. Here as in the barrel shifter case both the n-bit multiplier and n-bit multiplicand are encoded into m bits each. The array consists of $\frac{n}{\log(m/n)} \times \frac{n}{\log(m/n)}$ (m/n) encoded block multipliers. Each block can be designed to work with $\log(m/n)$ switching count resulting in overall switching of $\frac{n^2}{\log(m/n)}$.

The variance in the size of encoded blocks is a problem with multiplication as well. The result of each (m/n)-encoded bit block multiplication has length $(m/n)^2$, which is too long. The length of the block multiplication result for a consistent encoding should only be m/n. In this case, the rearrangment depends only on the resulting block (self-contained) and can be accomplished within a constant number of switchings per block.

In general, for datapath functions with $\Omega(n)$ information complexity, such as shifter and multiplier, the switching count savings of a multiplicative factor $n/\log(m/n)$ can be achieved.

Lemma 3 *All the type-2 datapath functions, the functions with $\Omega(n)$ information complexity, can save a $\frac{n}{\log(m/n)}$ factor from switching count by using $n \longrightarrow m$ bit encoding for $m \geq n$.*

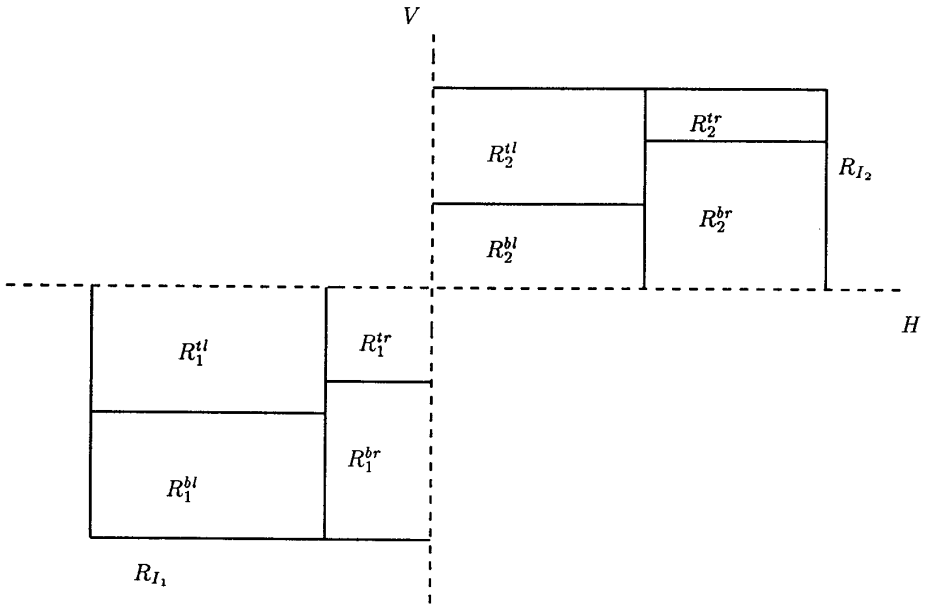

Figure 3: The Cutting Scheme for ET Lower Bounds

4 Switching Count Lower Bounds

In order to derive a lower bound of $\Omega(I \log I)$ on the un-encoded switching count of a function with information complexity I, we need a variant of the cutting lemma from Tyagi [Tya89]. The cutting lemma there asserts the existence of two rectangles each containing a significant fraction of n input bits, say $n/61$, such that the two rectangles are separated by distance equaling at least the smaller perimeter of the two rectangles. Figure 3 shows such a cut. This was needed to claim that the switching wires also have certain length. In the charge packet model, however, the wire lengths play no role in switching energy. We need to assert a lower bound on the distinct switching counts. The basic framework still is to separate the input bits into two large enough sets such that at least I information bits are to be exchanged between the two sets. Each such cut provides I switching count. We need roughly $\log I$ such distinct cuts so that the information needs of these cuts are disjoint. That should lead to an $\Omega(I \log I)$ lower bound. The following lemma proves the existence of $\log I$ such cuts.

Lemma 4 *A VLSI circuit C computing a function f with information complexity I can be cut into $\log I$ distinct partitions of n/c bits each for a constant $c > 2$.*

PROOF SKETCH: Use the two dimensional cutting technique of Tyagi [Tya89] repeatedly $\log n$ times. Let the top-level cut be level-0 cut. Cut each of these rectangles containing at least $n/81$ input bits again in the same way. The union of all the cuts in all the four rectangles from level-0 forms the level-1 cut. In general cutting level-i rectangles and taking the union of all these cuts results in level-$(i+1)$ cut. We can form $\log I$ (or $\log n$ for $I = \Omega(n)$) such cuts. At that point the resulting rectangles contain at least 1 bit each. Figure 4 shows such a collection of cuts. □

This lemma establishes the cut required for the lower bound of $\Omega(I \log I)$ on the switching count in the un-encoded case. Along with Lemma 1, the same lower bound translates into $\Omega((I \log I)/\log(m/I))$ lower bound when an I to m bit encoding is used.

Theorem 1 *An unencoded computation of a function with information complexity I requires $\Omega(I \log I)$ switchings.*

PROOF SKETCH: Construct the $\log I$ cuts given by Lemma 4 as shown in Figure 4 for the circuit C to compute this function. Each of the level i cuts provides a

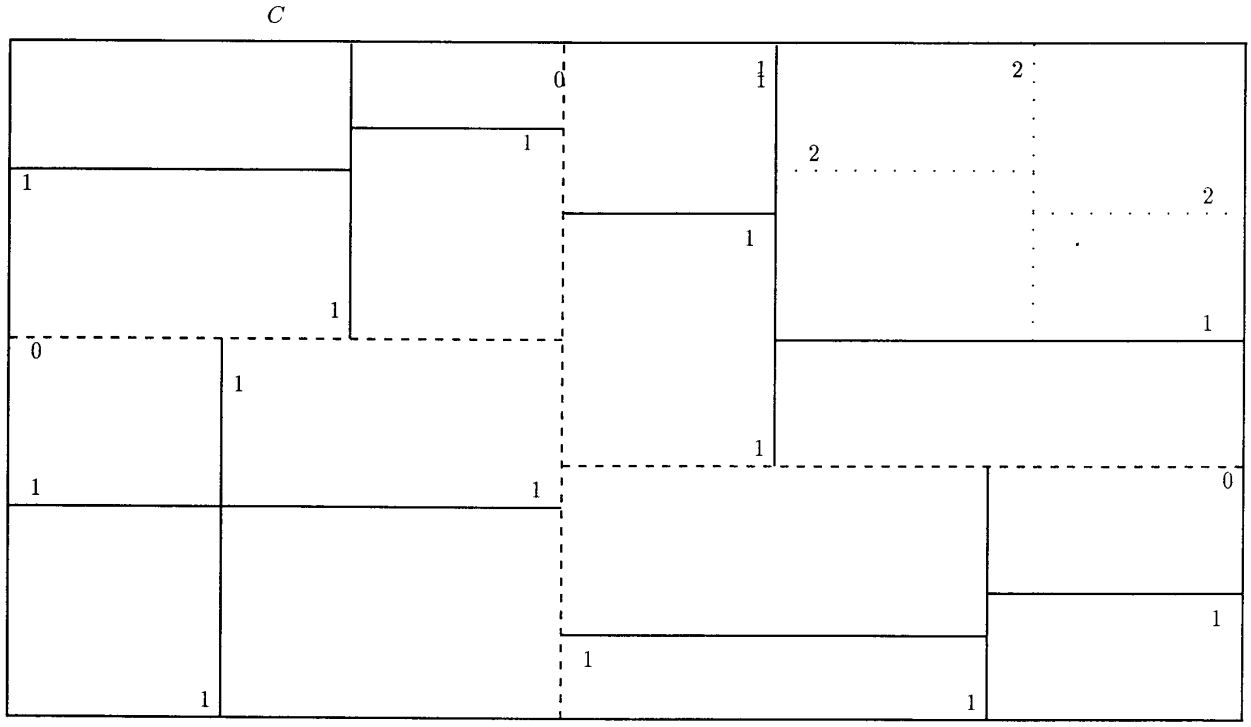

Figure 4: The $\log n$ step Cutting Scheme

balanced partition of the input bits and hence has $\Omega(I)$ information flow through it. In unencoded case, $O(I)$ physical bits get used for this purpose, which result in $\Omega(I)$ switchings by Lemma 1. Hence the total switching is $\Omega(I \log I)$. □

Theorem 2 *A computation of a function with information complexity I that encodes all the data through $n \longrightarrow m$ encoding requires $\Omega\left(\frac{I \log I}{\log(m/I)}\right)$ switchings.*

The proof is similar to Theorem 1 except that switching at each cut now is $\Omega(I/(\log(m/I)))$ and hence the result.

5 Reversible Communication Complexity

Yao describes the following communication model in [Yao79]. There are two computing agents A and B cooperating to compute a *Boolean* valued function $f(x_1, x_2, \ldots, x_n)$ whose argument is n bits long. A knows the values of roughly half the input bits $X_A = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{\lfloor n/2 \rfloor}}\}$ and B knows the values of the other half of the input bits $X_B = \{x_{j_1}, x_{j_2}, \ldots, x_{j_{\lceil n/2 \rceil}}\}$. Both A and B know the complete truth table for the function f. Let M be the set of $2^{\lfloor n/2 \rfloor}$ values of the input to A and let N be the set of $2^{\lceil n/2 \rceil}$ values of B's input bits. Then a function f can be specified in a tabular truth-table form. The entry in the ith row and the jth column in this table denoted by $f(i,j)$ corresponds to the value of the function $f(x_{i_1}, x_{i_2}, \ldots, x_{i_{\lfloor n/2 \rfloor}}, x_{j_1}, \ldots, x_{j_{\lceil n/2 \rceil}})$, where i and j are the values of the words formed with the bits in X_A and X_B respectively. A cross product $P \times Q$, $P \subseteq M$, $Q \subseteq N$ is known as a *rectangle*. Note that for $P \times Q$ to be a rectangle, the elements of P and Q need not be consecutive integers. If the value of the function f is constant over a rectangle, then it is known as a *monochromatic rectangle*. A partition of $M \times N$ into k disjoint monochromatic rectangles $S_1 \times T_1$, $S_2 \times T_2, \ldots, S_k \times T_k$ is known as a *$k-$decomposition* of f. Let $d(f)$ be the minimum k such that a $k-$decomposition of f exists.

The objective is to measure the amount of information that needs to be exchanged between A and B, for the function value to be computed. A and B can alternately send a bit in $\{0,1\}$ to each other based on a certain predetermined protocol, until one of them has enough information to determine the output value. Since the circuits we consider are deterministic, let us deal with the case where the algorithm to determine the value of the communication bits is deterministic. Let $p_A : \{0,1\}^{\lfloor n/2 \rfloor} \times \{0,1\}^* \longrightarrow \{0,1\}$ be the protocol function of A, where the first argument of length $\lfloor n/2 \rfloor$ is the input value of the bits in X_A and the second argument specifies the history of communication until that step. We can similarly define the protocol function for B; $p_B : \{0,1\}^{\lceil n/2 \rceil} \times \{0,1\}^* \longrightarrow \{0,1\}$. A protocol $p = (p_A, p_B)$ is completely specified by p_A and p_B, the communication algorithms of A and B. The computation starts with A sending the bit $a_1 = p_A(x_{i_1}, x_{i_2}, \ldots, x_{i_{\lfloor n/2 \rfloor}}, \epsilon)$ to B, where ϵ is the null string representing the beginning of communication. B responds with the bit $b_1 = p_B(x_{j_1}, x_{j_2}, \ldots, x_{j_{\lceil n/2 \rceil}}, a_1)$. The communication continues with A sending $a_2 = p_A(x_{i_1}, x_{i_2}, \ldots, x_{i_{\lfloor n/2 \rfloor}}, a_1 b_1)$. This process stops when either A or B has enough information to decide the function value. Without loss of generality, let A decide the value of f first, after receiving the bit b_l from B. It then sends a halt signal 'H' to B. The communication history at this point is given by $a_1, b_1, a_2, b_2, \ldots, a_l, b_l$. The number of bits exchanged in this computation is $2l$. The communication complexity, $c(p)$ for the protocol $p = (p_A, p_B)$ is defined to be the maximum number of bits exchanged over all the input values for computing f under the protocol p. In particular, $c(p)$ is expressed as a function, $g(n)$, of the input size n. Thus over all the 2^n input values, a computation of a function f under the protocol p will never need to exchange more than $g(n)$ bits. The worst case communication complexity, then, is given by the minimum value of the communication complexity $c(p)$ over all the protocols p. For such a two-way protocol, Yao defines the *2-way communication complexity* to be $C(f, A \Leftrightarrow B) = \min_p\{c(p) \mid \text{the protocol } p \text{ computes function } f\}$.

When one considers reversible computations, Yao's definition can be extended trivially by insisting on the protocols p_A and p_B being reversible, which results in $c^r(p)$. The minimum of $c^r(p)$ over all reversible protocols p provides the reversible communication complexity of a function.

Note that the reversible communication complexity I^r better characterizes the complexity of a reversible implementation. Hence the derivations of lower and upper bounds on the reversible communication complexity of interesting functions constitutes a useful and interesting exercise. We only have a trivial theorem that shows that in order not to lose information about the n input bits, the information about all of them needs to be exchanged for the computation to be reversible.

Theorem 3 *The reversible information complexity of a function with n input bits is $\Omega(n)$.*

Note that sending all the input bits also provides a trivial upper bound of $O(n)$. If we insist on reversing the computation then the n bits might have to be exchanged back leading to a total of $2n$ bits. Are there functions

for which some of the bits are so correlated that they not all of them need be sent back? In that case the communication complexity falls somewhere in between n and $2n$. We have not answered any of these interesting questions.

6 Conclusions

Reversible logic can be implemented with a technology where energy per communication/computation event does not depend on the length of the interconnect (or is constant). In such cases, by expanding the numbe of bits in the data beyond the minimum necessary, the number of switchings can be decreased. We have explored this technique partially in this paper. The functions with low communication requirements between their input bit partitions tend not to benefit from this encoding. Adder is such a function. However, functions that need to exchange a large number of bits between their input bit partitions can save energy by using encoded arithmetic. In general transitive functions [Vui83] which include datapath functions such as shifting and integer multiplication can save energy from encoding. We have sketched some solutions for encoded arithmetic problems. We intend to actually design and layout some of these encoded and un-encoded arithmetic functions to get more realistic rather than asymptotic comparisons. We also presented a lower bound on the switching count of VLSI computations.

Another interesting direction to explore would be to see if residual arithmetic is a better candidate for reversible logic. The computations are very localized in this arithmetic as there are no carries. Perhaps each of these modular blocks can be encoded? There are many questions that need to be answered in the choice of a good arithmetic system for reversible logic.

References

[ACR88] A. Aggarwal, A. K. Chandra, and P. Raghavan. Energy Consumption in VLSI Circuits. In *Proceedings of ACM Symposium on Theory of Computing*, pages 205–216. ACM-SIGACT, 1988.

[Hal92] J. S. Hall. An Electroid Switching Model for Reversible Computer Architectures. In *Proceedings of the Workshop on Physics and Computation, PhysComp '92*, pages 237–247. IEEE Computer Society Press, 1992.

[KA92] J. G. Koller and W. C. Athas. Adiabatic Switching, Low Energy Computing, and the Physics of Storing and Erasing Information. In *Proceedings of the Workshop on Physics and Computation, PhysComp '92*, pages 267–270. IEEE Computer Society Press, 1992.

[MC80] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, Mass., 1980.

[Mer93] R. C. Merkle. Reversible Electronic Logic Using Switches. *Nanotechnology*, 4:21–40, 1993. Also appears in Proc. of the Workshop on Physics and Computation, PhysComp '92, IEEE Computer Society Press.

[Tho79] C.D. Thompson. Area-Time Complexity for VLSI. In *Proceedings of ACM Symposium on Theory of Computing*, pages 81–88. ACM-SIGACT, 1979.

[Tya88] A. Tyagi. *The Role of Energy in VLSI Computations.* PhD thesis, Department of Computer Science, University of Washington, Seattle, 1988. *Available as UWCS Technical Report Number 88-06-05.*

[Tya89] A. Tyagi. Energy-Time Trade-Offs in VLSI Computations. In *Proceedings of the Ninth Conference on Foundations of Software Technology & Theoretical Computer Science*, pages 301–311. Lecture Notes in Computer Science #405, Springer-Verlag, 1989. a revised version to appear in IEEE Trans on Computers.

[Vui83] J. Vuillemin. A Combinatorial Limit to the Computing Power of VLSI Circuits. *IEEE Transactions on Computers*, pages 294–300, March 1983.

[Yao79] A.C. Yao. Some Complexity Questions Related to Distributed Computing. In *Proceedings of ACM Symposium on Theory of Computing*, pages 209–213. ACM-SIGACT, 1979.

[Yao81] A.C. Yao. The Entropic Limitations on VLSI Computations. In *Proceedings of ACM Symposium on Theory of Computing*, pages 308–311. ACM-SIGACT, 1981.