VLSI Complexity of Coding

Abbas El Gamal[1] Jonathan W. Greene[2] King F. Pang[3]
Information Systems Laboratory
Electrical Engineering Department
Stanford University
Stanford, CA 94305 USA

Abstract

Under a general VLSI model, it is shown that the area A, computation time T, and pipeline period P for any integrated circuit that encodes or decodes an (n,Rn,t) binary t-error-correcting code must satisfy the lower bound $AT^2 \geq AP^2 = \Omega(R^2nt)$. This bound also holds for an (n,Rn,P_e)-code if t is replaced by $\log(n/P_e)$, where P_e is the average probability of decoding error.

An encoder for linear codes can be implemented using a circuit with $A = O(Rn^2\log(Rn))$ and $T = O(\log(Rn))$. For binary primitive BCH codes of constant rate, $AT^2 = \Omega(n^2/\log n)$; a decoding algorithm can be implemented on a circuit with $A = O(n^2\log^2 n)$ and $T = O(\log^2 n \log\log n)$.

1. Introduction

Error correcting codes are widely used for protecting data communicated over a noisy channel or stored in a noisy computer memory. Much effort has been expended on the design of codes with good error correcting properties, and several classes of such codes are already known. Efficient algorithms for decoding several classes of codes (e.g. BCH codes) have also been developed [1,2]. However, the complexity of the required encoding and decoding circuitry is still a major obstacle to the application of many codes and to the use of coding in many systems. This is very likely to change with the advent of very-large-scale-integrated (VLSI) circuit technology, where hundreds of thousands of devices can be integrated on a single circuit. A VLSI coding circuit will be faster, cheaper, smaller, more reliable and consume less power.

The purpose of this paper is to investigate the complexity of VLSI circuits that encode or decode any particular code. The results, which are based on the VLSI model introduced by Thompson [3], are in the form of bounds on AT^2 or AP^2, where A is the area of the circuit, T is the computation time, and P is the period for pipelined implementations.

Decoding complexity for ensembles of codes has been investigated by Savage [4] and by Berlekamp *et al* [6]. In [4], Savage demonstrated that for large block lengths, with probability near one, a randomly chosen code requires a decoder with a number of logic and memory elements exponential in the block length. In [6], Berlekamp *et al.* proved that the following problem is *NP-Complete*:

> Given: any parity-check code matrix $\mathbf{H}$ and a sequence $\mathbf{y}$.
>
> Find: the decoding of $\mathbf{y}$ according to the code specified by $\mathbf{H}$.

Lower bounds on the number of gates and delay required to decode a particular code have also been established by Savage [5]. However, our approach is different. As in [3], we relate area and time to the information flow among circuit elements. In addition, we employ two new key ideas. First, we use cuts that change in time as well as in space. Second, we partition the circuit into many small blocks and guarantee that at least one block has an exact number of inputs (or outputs), less than or equal to a certain number of outputs (or inputs), and small perimeter (Section 3).

We consider codes for use on a *binary discrete memoryless channel* (DMC), consisting of binary input alphabet $\mathcal{X}=\{0,1\}$, binary output alphabet $\mathcal{Y}=\{0,1\}$, and a probability transition matrix $p(y \mid x)$. When a codeword $\mathbf{x} \in \{0,1\}^n$ is transmitted, the received sequence $\mathbf{Y} \in \{0,1\}^n$ is determined according to a transition probability function $p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{n} p(y_i \mid x_i)$. We further assume that there exist a $p>0$ and some $y \in \{0,1\}$ such that $p \leq p(y \mid x)$ for all $x \in \{0,1\}$. Without loss of generality, we assume that this is true for $y=0$. Our results can readily be extended to non-binary channels as well.

In coding theory, a code is specified by its length n, its rate R, and the number t of errors it is guaranteed to correct. An (n,Rn,t)-code consists of a set of information sequences $\{0,1\}^{Rn}$, an encoding function

$$f: \{0,1\}^{Rn} \rightarrow \mathcal{X}^n$$

1. Work partially supported under DARPA Contract MDA 903-79-C-0680.

2. Work partially supported by an NSF Graduate Fellowship and under DARPA Contract MDA 903-79-C-0680.

3. Work partially supported under DARPA Contract MDA 903-79-C-0680 and NSF Grant ECS83-00988.

and a decoding function

$$g: \mathcal{Y}^n \rightarrow \{0,1\}^{Rn}$$

such that for every information sequence $\mathbf{u} \in \{0,1\}^{Rn}$,

$$d(f(\mathbf{u}),\mathbf{y}) \leq t \text{ implies } g(\mathbf{y})=\mathbf{u}.$$

(Here $d(\cdot,\cdot)$ denotes the Hamming distance, or number of positions in which the sequences differ).

In information theory, the quality of a code is customarily specified by the average probability of decoding error

$$P(\text{error}) = \sum_{\mathbf{u}} 2^{-Rn} P(g(\mathbf{Y}) \neq \mathbf{u} \mid \mathbf{u} \text{ sent }).$$

An (n,Rn,P_e)-code for a given binary DMC consists of encoding and decoding functions such that $P(\text{error}) \leq P_e$. Note that the probabilities are only used to state the requirements on the deterministic functions $f(\cdot)$ and $g(\cdot)$. Once we fix a function, we can evaluate its complexity using a worst case input.

In Section 2, we describe the VLSI model used in this paper. Lower bounds for the VLSI complexity of encoding and decoding are derived in Section 3. Theorem 2 is the VLSI-model equivalent of the lower bounds in [5]. The use of communication, or *information flow*, arguments allows us to prove the bounds of Theorems 3 and 4, which, for codes with good error-correction capabilities, are stronger. Theorems 3 and 4 apply to encoders as well.

The succeeding sections of the paper give upper bounds. Section 4 describes an encoding scheme for linear codes which comes very close to the lower bound of Theorem 3 when t is proportional to n. In Section 5, we examine the decoding complexity of the well-known BCH codes and give a decoder design with complexity reasonably close to the lower bound.

The notations $O(m)$, $\Theta(m)$, and $\Omega(m)$ are defined as follows. There exist constants c_i independent of m such that for all sufficiently large m: $O(m) \leq c_1 m$; $c_2 m \leq \Theta(m) \leq c_3 m$; $\Omega(m) \geq c_4 m$. Also, $o(m)$ denotes a quantity such that $o(m)/m \rightarrow 0$. $1(\cdot)$ denotes the indicator function. All logarithms are to the base 2 unless otherwise indicated by a subscript. The greatest integer less than or equal to x is denoted by $\lfloor x \rfloor$. The least integer greater than or equal to x is denoted by $\lceil x \rceil$.

2. The Model for VLSI Complexity

A circuit can be implemented in one of several VLSI technologies (e.g. nMOS, CMOS, ECL, Josephson Junction). In order to obtain complexity results that are technology independent, we use the following model, which is a simplified version of the model in [7] (see also [8, Ch.1], [9]). Results obtained from this model only differ by a constant factor from those of the actual implementation.

Given some function, a circuit to compute it must be laid out on a grid of unit squares according to the following assumptions:

i) Wires run along grid edges, and can cross at grid points. Only one wire may occupy an edge. Each grid point can accommodate a logic element, a wire crossing or connection, or an input/output pin. Each wire runs from a single element output to one or more element inputs.

ii) A logic element can compute any boolean function of at most three inputs. This limitation on fan-in results from the grid structure. A logic element can only store one bit at a time.

iii) For every set of input values, the output values must agree with the function being implemented. Each input bit appears exactly once at exactly one pin. Each output bit appears at least once at some pin. The time and pin at which an input or output appears are predetermined and are independent of the input values (i.e. the system is where- and when-determinate [8]). There can be at most one bit input or output at a pin at a given time.

iv) The area A of the circuit is defined as the area of the smallest rectangle enclosing the circuit. Note that if the circuit is convex, the area of the bounding rectangle is never more than twice that of the circuit itself.

v) Only one bit can be propagated through a wire or transmitted through an input/output pin per unit time. This unit time includes the time needed for a logic element to compute the bit, and for unlimited fanout. The *computation time* T is defined as the number of unit times between the appearances of the first input bit and the last output bit. For a *pipelined* circuit, another figure of merit is the *period* P, defined as the number of unit times between the appearance of the first input bit of one problem instance and the appearance of the first input bit of the next instance.

vi) For upper bounds only, we assume that the circuit operates synchronously. During a clock cycle, each element computes its output(s) and these propagate along the wires to the appropriate element inputs for use during the next cycle.

Remarks:

1. The model in [3] assumes for upper bound arguments that the time required to propagate a signal along a wire of length l is $\Theta(\log l)$ and that a driver of size $\Theta(1) \times \Theta(l)$ is required. These assumptions would increase T and P by modest factors of $\log n$ for the encoder described in Section 4, and $\log n \log\log n$ for the BCH decoder in Section 5. The orders of growth of A are not affected.

2. The model assumptions can be relaxed to allow any constant number of logic elements, wire crossings or pins at each grid point, or any constant number of wires on an edge of the grid. It has been shown that the effect of this

is merely to reduce the area of the circuit by a constant factor [3, p. 36-38].

3. Since any chip with computation time T can be a pipelined chip with $P=T$, a lower bound on P is *a fortiori* a bound on T. See [9] for further discussion of pipelined circuits.

The complexity measure AT^2 (or AP^2) appears naturally when the information flow argument (Section 3) is applied. It is a pleasant surprise that many functions permit an area-time tradeoff such that for various implementations, the product AT^2 is close to the lower bound. Examples are multiplication, the discrete Fourier transform, and sorting [8].

3. Lower Bounds

First, in Theorems 1 and 2 we give some simple bounds for decoding complexity. The arguments are similar to those of Savage [5]. These bounds are most useful for codes with only weak error correction capability, such as Hamming codes and repetition codes. We then describe a method of partitioning a circuit and the information flow arguments that are used to prove the main results of this section, Theorems 3 and 4.

Theorem 1: Any circuit that performs a decoding function for an (n,Rn,t)-code must satisfy: $T\geq\log_3(t+1)$, $AT\geq AP\geq n$, and so $AT^2\geq n\log_3(t+1)$.

Proof: Suppose there is an output bit that depends on only j received bits. If $j\leq t$, then all j received bits can be zero no matter what sequence $\mathbf{x}$ is transmitted. The resulting value of the output bit is in error for half of the possible information sequences $\mathbf{u}$. Thus we must have $j\geq t+1$.

Since each logic element has fan-in at most 3, computation of an output requires at least $\log_3(j)$ levels of logic. Hence $T\geq\log_3(t+1)$.

During every decoding, n bits must be input to the decoder. Since a grid point can contain only one input pin and a pin can accept only one input per unit time, $AT\geq AP=n$. □

A similar theorem can be demonstrated for (n,Rn,P_e)-codes.

Theorem 2: Any circuit that performs a decoding function for an (n,Rn,P_e)-code with $P_e<1/2$ must satisfy: $T\geq\log_3(\log_p(2P_e))$, $AT\geq AP\geq n$, and so $AT^2\geq n\log_3(\log_p(2P_e))$.

Proof: Suppose there is an output bit that depends on only j received bits. No matter what sequence is transmitted, there is probability at least p^j that each of the j bits is a zero. The resulting value of the output bit is in error for half of the possible information sequences $\mathbf{u}$. Thus $P_e\geq P(\text{error})\geq(1/2)p^j$, which implies $j\geq\log_p(2P_e)$. The rest of the argument is the same as in Theorem 1. □

Area-time lower bounds are usually proved by partitioning the circuit into two blocks with roughly equal numbers of outputs, and such that only a limited amount of information can flow between the blocks each period (e.g. [3,7,8]). The $L\times W$ rectangle enclosing the circuit is partitioned by a cut like (a) in Fig. 1. Since each wire crossing the cut can transfer at most one bit per unit time, only $(W+1)P$ bits can cross the cut in P time units.

However, this approach fails for our purposes on two counts. First, cuts like (a) cannot partition the outputs arbitrarily because each pin can be the source of several outputs sequentially. For example, cut (b) in the figure may have too few outputs on the left, but when the cut is moved to the other side of pin (c) too many outputs may have been added. Second, we need to partition the circuit into many blocks, not just two.

Our solution to the first problem is to allow cuts to move once during the period and then return to their original position at the end of the period. This way the outputs can be divided exactly as required. Such a cut is more easily visualized as a cut in a three-dimensional $L\times W\times P$ region, such as cut (d) in Fig. 2. As before, at most $(W+1)P=O(WP)$ bits per period can flow along the wires crossing the cut. (This value corresponds to the area of the cut-planes parallel to the time axis). However, an additional bit can be passed in each direction by being stored at the single pin that is transferred from block to block. (These two additional bits correspond to the area of the cut-planes that are perpendicular to the time axis). The total information flow across the cut per period is therefore at most $(W+1)P+2$ bits.

Our solution to the second problem is to partition the circuit into several blocks as required by first partitioning it into a slices by cuts across the width and then partitioning each slice into b blocks by cuts in the perpendicular direction, as shown in Fig.3. The total information flow into all the blocks is the sum of the numbers of bits transferred across each of the cuts. The following lemma holds.

Lemma: Consider a partition of a VLSI circuit into ab blocks as shown in Fig.3. Let aW and bL differ by no more than a constant factor. Then the average number of bits transferred per block each period is $O(P\sqrt{A/(ab)})$, where $LW=A$.

Proof: The total number of bits transferred along wires is $O(aWP+b(L+a)P)$. The number of bits transferred by storage at transferred pins is $O(ab)$. Note that $\Theta(aW)=\Theta(bL)=\Theta(\sqrt{abWL})$. Then the average number of bits transferred per block is $O((aWP+bLP+abP+ab)/(ab))$ $=$ $O((P\sqrt{abWL}+abP)/(ab))$ $=$ $O(P(\sqrt{A/(ab)}+1))$. If $A\geq ab$, this is $O(P\sqrt{A/(ab)})$. If not, we clearly still have $AP\geq ab$, and also the obvious fact that the number of bits transferred along wires is $O(LWP)$. Then the average number of bits transferred per block is $O((LWP+ab)/(ab))$ $=$ $O(AP/(ab))$ $=$ $O(P\sqrt{A/(ab)})$ since $A<ab$, giving the same result. □

The proof of the next theorem relies on the following general argument. Given a partition of the type depicted in Fig.3, we first demonstrate the existence of a block with a certain number of outputs, no more than a certain number of inputs, and that can receive only about an average share of the total information flow. Then we show that the numbers of inputs and outputs are such that the amount of information the block receives, and so the total information flow, must be large. Finally we use the relation between the total information flow and A and P established in the Lemma to prove the desired lower bound.

Theorem 3: Any circuit that performs a decoding or encoding function for an (n,Rn,t)-code must satisfy: $AT^2 \geq AP^2 = \Omega(R^2nt)$.

Proof: We begin with a decoder. Let $a = \left\lceil \sqrt{2nL/(tW)} \right\rceil$ and $b = \left\lceil \sqrt{2nW/(tL)} \right\rceil$. We partition the circuit into $ab \geq 2n/t$ blocks, as in Fig.3, in such a way that each of the resulting blocks contains at least $\lfloor Rn/(ab) \rfloor = \Theta(Rt)$ of the Rn outputs. This is insured by using cuts of the type depicted in Fig.2. By the above Lemma, the average number of bits entering a block from its neighbors per period is $O(P\sqrt{A/(ab)}) = O(P\sqrt{At/n})$. Clearly, more than half the blocks receive at most twice the average number of bits. Likewise, more than half the blocks contain at most twice the average number of inputs, that is, at most $2(n/(ab)) \leq t$. We conclude that there is at least one block that: (i) contains at most t inputs; (ii) contains $\Theta(Rt)$ outputs; and (iii) receives $O(P\sqrt{At/n})$ bits of information per period from neighboring blocks.

Since there are at most t inputs to the block, they can all be set to zero by errors and the decoder must still output whatever information sequence $\mathbf{u}$ is sent. To determine the proper outputs, $\Theta(Rt)$ bits of information must enter the block; more precisely, there is some information sequence which is only decoded properly after $\Theta(Rt)$ bits have entered the block. If this information sequence and set of errors recur again and again, $\Theta(Rt)$ bits must flow into the block during each period. As stated above, only $O(P\sqrt{At/n})$ bits can. Thus $AP^2 = \Omega(R^2nt)$.

The argument for an encoder is similar, except that the roles of inputs and outputs are interchanged. Some block will have t or fewer outputs, which can all be changed to zero by errors. Nevertheless, the $\Theta(Rt)$ inputs to this block must be accurately reconstructed at the corresponding outputs of the decoder. This requires that, for at least one input $\mathbf{u}$, $\Theta(Rt)$ bits flow out the block's boundaries and through other encoder blocks so that they are available at the decoder. □

Again, there is a similar result for (n,Rn,P_e)-codes.

Theorem 4: Any circuit that performs a decoding or encoding function for an (n,Rn,P_e)-code with $P_e < 1/2$ must satisfy: $AT^2 \geq AP^2 = \Omega(R^2n\log(n/P_e))$.

Proof: The general idea of the argument for the decoder is as follows. We partition the circuit into blocks as before and identify a set of blocks that receive only about the average number of bits from their neighbors. With probability $>2P_e$, at least one of these blocks has all its inputs equal to zero. We claim that in this event, the block's output bits can be determined only if enough information is received. Suppose instead that the amount of information that can flow into the block is one bit less than the number of outputs. Then the correct outputs are computed for only half the values of $\mathbf{u}$, and so $P(\text{error}) > P_e$, a contradiction. Details of this proof can be found in [16].

4. An Encoder Circuit for Linear Codes

The operation of encoding a linear code can be realized by multiplying the information sequence $\mathbf{u} \in \{0,1\}^{Rn}$ with an $Rn \times n$ matrix G, the *Generator Matrix*. Hence, the task of designing an area-time efficient encoder can be reduced to that of designing a circuit for multiplying an Rn vector with a constant $Rn \times n$ matrix.

Our proposed circuit for computing $\mathbf{u} \cdot G$ consists of an array of n processors. Each processor is a complete binary tree performing a multiplication of $\mathbf{u}$ by a fixed column of G. The area of each processor is $O(Rn\log(Rn))$ and the computation time is $O(\log(Rn))$. Thus we have proved the following

Theorem 5: There exists a circuit which computes $\mathbf{u} \cdot G$, where $\mathbf{u} \in \{0,1\}^{Rn}$ and G is a constant $Rn \times n$ matrix with $A = O(n^2R\log(Rn))$, $T = O(\log(Rn))$. □

For the class of binary cyclic codes, encoding amounts to multiplying two polynomials, the information polynomial $u(x)$ of degree Rn and the generator polynomial $g(x)$ of degree $n(1-R)$. By using the multiplication circuit described in Appendix 2 of [16], this can be achieved in area $O(n^2)$ and time $O(\log n)$. Alternatively, a systematic cyclic code can be generated by the following procedure : The coefficients of $u(x)$ is output as the first Rn bits. The remaining $(1-R)n$ bits are generated as the remainder of $\frac{x^{n(1-R)}}{g(x)}$. As shown in Appendix 2 of [16], polynomial division has the same area and time complexities as multiplication. Therefore, encoding for cyclic codes can be achieved in $AT^2 = O(n^2\log^2 n)$.

5. Decoding of BCH Codes

In this section, we consider an implementation of a decoder for a binary primitive BCH Code. A binary primitive BCH code is a cyclic code with block length $n = 2^m - 1$ for some integer m and generator polynomial

$$g(x) = LCM\{M^{(b)}(x), M^{(b+1)}(x), \ldots, M^{(b+\delta-2)}(x)\}$$

where $M^{(i)}(x)$ is the primitive polynomial of α^i (α being a primitive element of $GF(2^m)$), b is an arbitrary positive integer and $\delta \geq 2t+1$.

The AT^2 complexity of decoding BCH codes can be lower bounded by Theorem 3 and an asymptotic result on t for BCH codes of constant rates [10], and upper bounded by an implementation of the Euclid's algorithm decoding method [11]. These bounds are proved in the following theorem :

Theorem 6: Any decoder for an (n,k,t) binary primitive BCH code with constant rate must satisfy $AT^2 = \Omega(\frac{n^2}{\log n})$. Furthermore, there exists an implementation of such a decoder with $A = O(n^2 \log^2 n)$ and $T = O(\log^2 n \log\log n)$.

Proof: We first prove the lower bound. Berlekamp [10] has shown that a binary primitive BCH of constant rate corrects $t = \Theta(\frac{n}{\log n})$ errors. Substituting this into Theorem 3, we obtain

$$AT^2 = \Omega(\frac{n^2}{\log n}).$$

For the upper bound, we consider the decoding scheme based on Euclid's algorithm. A block schematic of our proposed implementation is shown in Fig.4. It consists of three steps.

Step 1: Computation of the Syndrome Polynomial. Let $\mathbf{x}$ be the transmitted sequence, $\mathbf{e}$ be the error sequence and $\mathbf{y}$ be the received sequence. They are related by $\mathbf{y} = \mathbf{x} + \mathbf{e}$. Let α be a primitive element of $GF(2^m)$. The syndrome polynomial $\sum_{i=1}^{2t} S_i x^{i-1}$ has coefficients given by

$$S_j = \sum_{i=0}^{n-1} \alpha^{ij} y_i , \quad j = 1,...,2t$$

Hence S_j can be computed as the appropriate $2t$ components of an n-point Fourier transform [2]. Using the butterfly network [8], this can be achieved in area A_1 and time T_1 where

$$A_1 = L_1 \times W_1 = O(n\log n) \times O(n\log n) \quad \text{and}$$
$$T_1 = O(\log n \log\log n).$$

The factors of $\log n$ in each dimension of area and $\log\log n$ in time are due to the fact that multiplications are performed on elements in $GF(2^m)$ rather than integers. Details of this operation are given in Appendix 3 of [16].

Step 2: Calculation of Error Locator Polynomial by Euclid's Algorithm. Since we are only considering binary codes, the decoding operation is completed as soon as we have succeeded in determining the error positions. We define the error locator polynomial $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$ as in [2], where it was shown that (Key Equation)

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}.$$

It is developed in [2] that $\Omega(x)$ and $\Lambda(x)$ can be determined by applying Euclid's Algorithm to the pair of polynomials $s^{(0)}(x) = x^{2t}$ and $t^{(0)}(x) = S(x)$. Successive members of the remainder sequence are computed by

$$\begin{bmatrix} s^{(r)}(x) \\ t^{(r)}(x) \end{bmatrix} = \begin{bmatrix} A_{11}^{(r)}(x) & A_{12}^{(r)}(x) \\ A_{21}^{(r)}(x) & A_{22}^{(r)}(x) \end{bmatrix} \begin{bmatrix} s^{(0)}(x) \\ t^{(0)}(x) \end{bmatrix}.$$

When this reduction reaches a point where $deg(t^{(r-1)}(x)) \geq t$ and $deg(t^{(r)}(x)) \leq t-1$, $\Lambda(x)$ is given by $A_{22}^{(r)}(x)$. Using Euclid's Algorithm in the straightforward manner, this reduction can take $O(t)$ steps. A systolic implementation of this algorithm has been proposed by Brent and Kung [12]. It requires area $O(t)$ and time $O(t)$. A more time-efficient way is to apply the HGCD (Half GCD) algorithm in [13]. (This algorithm is also used by Justesen [14] to speed up his Reed-Solomon decoder).

$HGCD(a_o(x), a_1(x))$ returns the pair of polynomials $(b_o(x), b_1(x))$ such that $deg(b_o(x)) \geq k$ and $deg(b_1(x)) \leq k$ where $k = \lceil deg(a_o(x))/2 \rceil$. It also returns the matrix $A = \begin{bmatrix} A_{11}(x) & A_{21}(x) \\ A_{12}(x) & A_{22}(x) \end{bmatrix}$ such that $\begin{bmatrix} a_o(x) \\ a_1(x) \end{bmatrix} = A \cdot \begin{bmatrix} b_o(x) \\ b_1(x) \end{bmatrix}$. After performing $HGCD(x^{2t}, s(x))$, one of the resulting remainders has degree $\leq t$. Hence at most one more division step is required to obtain a polynomial with degree $\leq t-1$ in the remainder sequence, which is the terminating condition in the reduction for determining $\Lambda(x)$.

The HGCD algorithm includes two recursive calls to itself, involving polynomials whose degrees are $\leq k$. There is also one polynomial division step with polynomials of degrees $\leq \lfloor 3k/2 \rfloor$. In addition, there is a multiplication of three 2×2 matrices and a multiplication of a 2×2 matrix by a 2 vector (the elements of the matrices and the vector are polynomials with degrees $\leq k$). In the Appendix, we examine the complexity of HGCD in detail and prove that it requires area $A_H = L_H \times W_H = O(t\log n) \times O(t\log n)$ and time $T_H = O(\log^2 t \log\log n)$. At most one extra division step is required after HGCD. The area and time complexities of this additional step are negligible compared to those of HGCD. Hence the complexity of Euclid's algorithm is

$$A_2 = O(t^2 \log^2 n) \text{ and } T_2 = O(\log^2 t \log\log n).$$

Step 3: Calculation of Error positions. If α^{-i} is a root of $\Lambda(x)$, then the ith position of the received sequence is in error. One way to determine the roots of $\Lambda(x) = \sum_{i=0}^{t} \lambda_i x^i$ is to evaluate it over all elements of $GF(2^m)$. That is, computing

$$\sum_{i=1}^{t} \lambda_i \alpha^{-ij} \quad \text{for } j = 1,...,n$$

This is equivalent to an n-point inverse Fourier Transform [2], with $\leq t$ nonzero inputs. Again, with the butterfly network, this can be achieved in area A_3 and time T_3 given by

$$A_3 = L_3 \times W_3 = O(n\log n) \times O(n\log n) \text{ and}$$
$$T_3 = O(\log n \log\log n).$$

Summing up the area and time complexities of the three steps,

$$A = O(n^2\log^2 n) + O(t^2\log^2 n) + O(n^2\log^2 n)$$
$$T = O(\log n\log\log n) + O(\log^2 t\log\log n) + O(\log n\log\log n)$$

Using Berlekamp's result that $t = \Theta(\frac{n}{\log n})$ for a BCH code with constant rate, we obtain

$$A = O(n^2\log^2 n) \text{ and } T = O(\log^2 n\log\log n).$$

□

6. Conclusions

Using a general VLSI model, we have obtained bounds on the complexity of encoding and decoding circuits. The lower bounds are nearly achieved by an implementation of Euclid's algorithm decoding method for BCH codes.

The lower bounding technique employs a new partitioning idea, which can be used to establish AT^2 lower bounds for other functions, e.g. the class of l-independent functions introduced by Grigoryev [17]. The proof of Theorem 4 involves a line of reasoning very similar to that used in [15] to prove lower bounds on the complexity of a certain defect-tolerant VLSI array. This is not surprising, since the flow of wires needed to replace defective elements is analogous to the flow of information in a decoder needed to correct erroneous bits.

Appendix Complexity of the HGCD Algorithm

In this appendix, we shall use the term polynomial(s) to refer to polynomial(s) over $GF(2^m)$. We first assume that the area for a circuit that performs division of two polynomials of $deg \leq i$ is $A_D(i) = W_D(i) \times L_D(i)$ and that it takes time $T_D(i)$ (polynomial multiplication, being no more costly than division, is charged the same as the latter) and derive the cost of performing HGCD in terms of these values. Upper bounds for $A_D(i)$ and $T_D(i)$ can be found in Appendix 2 of [16].

The block schematic of a circuit that performs $HGCD(a_o(x),a_1(x))$ (where the degrees of both polynomials are $\leq i$) is shown in Fig.5. It consists of the following steps:

a) Comparing the degrees of $a_o(x)$ and $a_1(x)$ and performing the transformations

$$a_o(x) = b_o(x)\cdot x^k + c_o(x)$$
$$a_1(x) = b_1(x)\cdot x^k + c_1(x)$$

where $k = \lfloor deg(a_o(x))/2 \rfloor$ and the degrees of $b_o(x)$, $b_1(x)$, $c_o(x)$ and $c_1(x)$ are $\leq \lceil deg(a_o(x))/2 \rceil$. It can be easily shown that is achievable in area $A = O(i\log i) \times O(\log i)$ and time $O(\log i)$.

b) A recursive call to $HGCD(b_o(x),b_1(x))$.

c) A multiplication of a 2×2 matrix with a 2-vector of polynomials. This requires area $O(A_D(i))$ and time $O(T_D(i))$.

d) A polynomial division that produces the quotient $e(x)$ and the remainder $f(x)$. This again requires area $O(A_D(i))$ and time $O(T_D(i))$.

e) A polynomial transformation

$$e(x) = h_1(x)\cdot x^l + h_3(x)$$
$$f(x) = h_2(x)\cdot x^l + h_4(x)$$

where $l = \lfloor deg(e(x))/2 \rfloor$ and the degrees of $h_1(x)$, $h_2(x)$, $h_3(x)$ and $h_4(x)$ are all $\leq \lceil deg(e(x))/2 \rceil$. This requires no more area and time than step a).

f) A recursive call to $HGCD(h_o(x),h_1(x))$.

g) Multiplying three 2×2 matrices of polynomials (of degrees $\leq i$). This can be achieved in area $O(A_D(i))$ and time $O(T_D(i))$.

Transportation of polynomials between registers and computation elements in the various steps can be effected by a data bus of $i\log n$ bits wide and this only adds $O(i\log n)$ to each dimension of the circuit. Note also that steps (a)-(c) and (e)-(g) are essentially identical (a matrix-vector multiplication can obviously be performed on a matrix-matrix multiplication circuit). Hence, an execution of $HGCD(a_o(x),a_1(x))$ actually involved two passes through the circuit shown in Fig.5.

Let $A_H(i) = L_H(i) \times W_H(i)$ and $T_H(i)$ be the area and time complexities of the HGCD implementation of size i (i.e. the two polynomials have degrees $\leq i$). We have the recursion

$$W_H(i) \leq W_H(i/2) + c_1 i\log n + c_2 W_D(i)$$

where c_1 and c_2 are constants. It is shown in Appendix 2 of [16] that $W_D(i) \geq c_3 i\log n$ for a constant c_3. Hence the above recursion becomes

$$W_H(i) \leq W_H(i/2) + c_4 W_D(i), \quad c_4 \text{ being a constant.}$$

It is clear that $W_H(1)$ is constant. Hence the recurrent equation has solution $W_H(i) = O(W_D(i))$. Similarly, the upper bounds for $L_H(i)$ and $T_H(i)$ are

$$L_H(i) = O(L_D(i)) \text{ and}$$
$$T_H(i) = O(T_D(i)\log i) \text{ respectively.}$$

Therefore the area of the circuit is

$$A_B(i) = O(A_D(i)).$$

In Appendix 2 of [16], we show that $A_D(i) = O(i^2\log^2 n)$ and $T_D(i) = O(\log i \log\log n)$. Substituting $i = t$, we have

$$A_B(t) = O(t^2\log^2 n)$$
$$T_B(t) = O(\log^2 t \log\log n).$$

References

[1] W.W. Peterson and E.J. Weldon, Jr., *Error-Correcting Codes*, 2nd Ed., MIT Press, 1972.

[2] R.E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1983.

[3] C. Thompson, *A Complexity Theory for VLSI*, PhD Thesis, Carnegie-Mellon University Tech. Rep. CMU-CS-80-140, Aug. 1980.

[4] J.E. Savage, 'Complexity of Decoders : I -- Classes of Decoding Rules,' *IEEE Trans. Info. Th.*, vol.IT-15, no.6, Nov. 1969, pp.689-695.

[5] ----, 'The Complexity of Decoders -- Part II : Computational work and Decoding time,' *IEEE Trans. Info. Th.*, vol.IT-17, no.1, Jan. 1971, pp.77-85.

[6] E.R. Berlekamp, R.J. McEliece and H.C.A. van Tilborg, 'On the Inherent Intractability of certain Coding Problems,' *IEEE Trans. Info. Th.*, vol.IT-24, no.3, May 1978, pp.384-386.

[7] R.P. Brent and H.T. Kung, 'The Area-Time Complexity of Binary Multiplication,' *J. ACM*, vol.28, no.3, July 1981, pp. 521-534.

[8] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1983.

[9] J. Vuillemin, 'A Combinatorial Limit to the Computing Power of VLSI Circuits,' *Proc. 21st IEEE Symp. Foundations of Comp. Sci.*, 1980, pp. 294-300.

[10] E.R. Berlekamp, 'Long Primitive Binary BCH Codes have Distance $d \approx 2n\ lnR^{-1}/\log n$,' *IEEE Trans. Info. Th.*, vol. IT-18, no.8, May 1972, pp.415-426.

[11] Y. Sugiyama, M. Kasahara, S. Hirasawa and T. Namekawa, 'A Method for Solving Key Equation for Decoding Goppa Codes,' *Info. and Contr.*, vol.27, Jan 1975, pp.87-99.

[12] R.P. Brent and H.T. Kung, 'Systolic VLSI Arrays for Polynomial GCD Computation,' CMU CS Dept. Report, May 1982.

[13] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[14] J. Justesen, 'On the Complexity of Decoding Reed-Solomon Codes,' *IEEE Trans. Info. Th.*, vol.IT-22, no.1, March 1976, pp.237-238.

[15] J.W. Greene and A. El Gamal, 'Configuration of VLSI Arrays in the Presence of Defects,' Section 3, to appear, *J. ACM.*

[16] A. El Gamal, J.W. Greene and K.F. Pang, 'VLSI Complexity of Coding,' submitted to *IEEE Trans. Info. Th.*

[17] D. Grigoryev, 'An Application of Separability and Independence notions for proving Lower Bounds on circuit complexity,' *Notes of Scientific Seminars, Steklov Math. Inst.*, vol.60, 1976, pp.35-48. (In Russian. Translation of Section 2 obtained from J. Savage.)

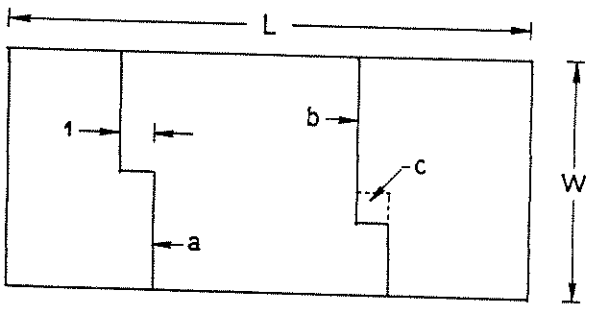


Fig.1 Cuts that are fixed in time.

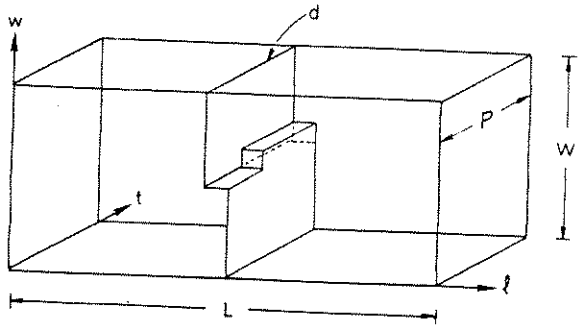


Fig.2 A cut that moves in time. During each period, one pin is transferred from the right side of the cut to the left side and then back.

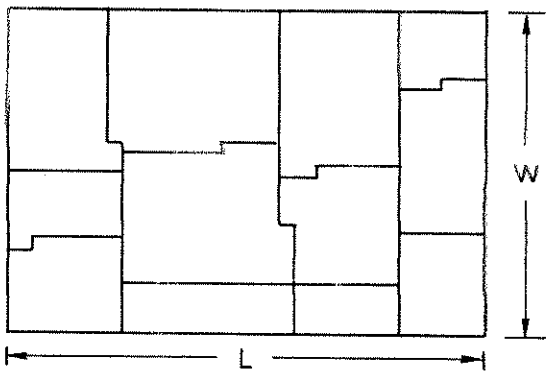


Fig.3 Partition of a circuit into blocks, $a = 4$ and $b = 3$

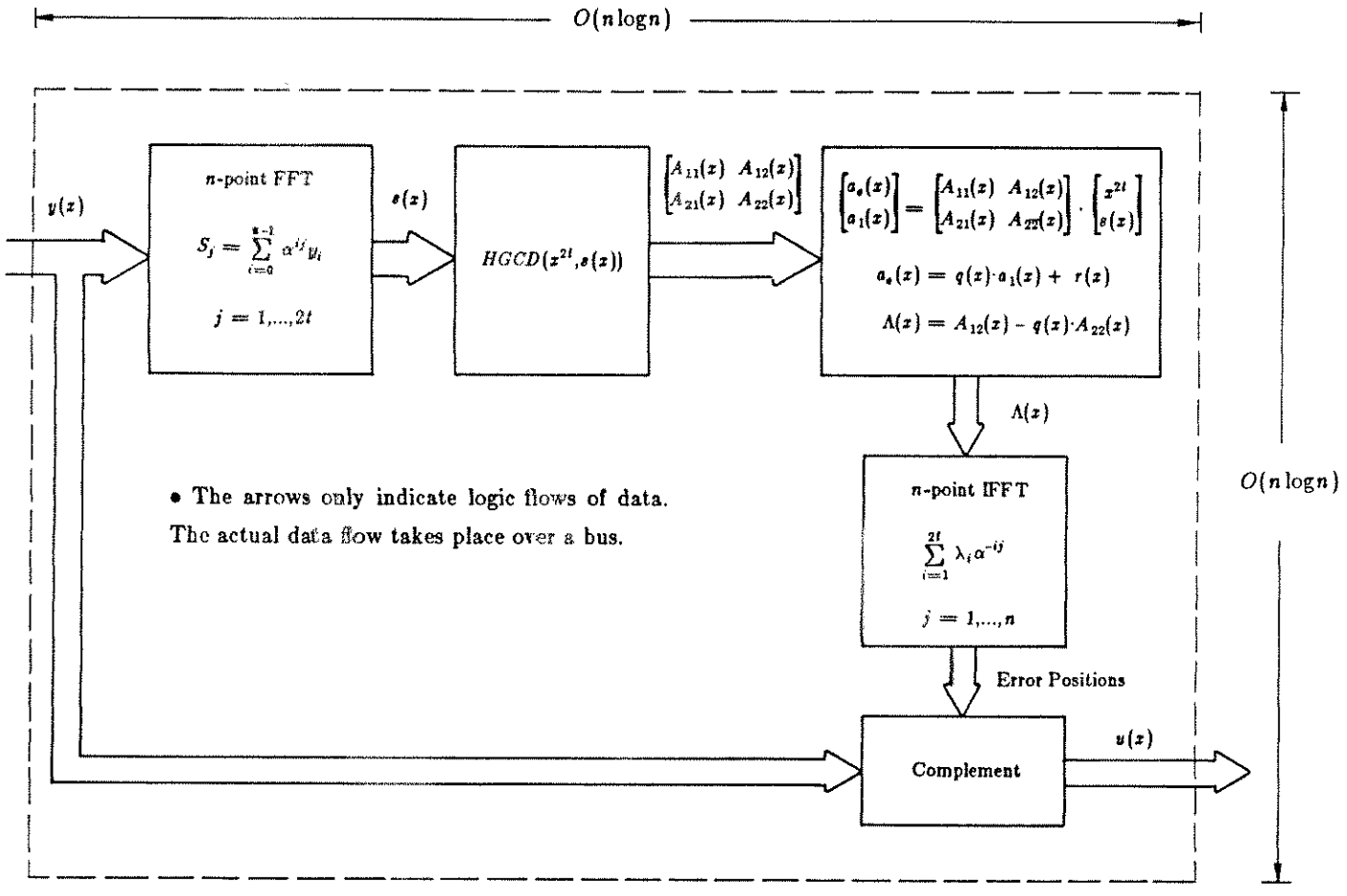
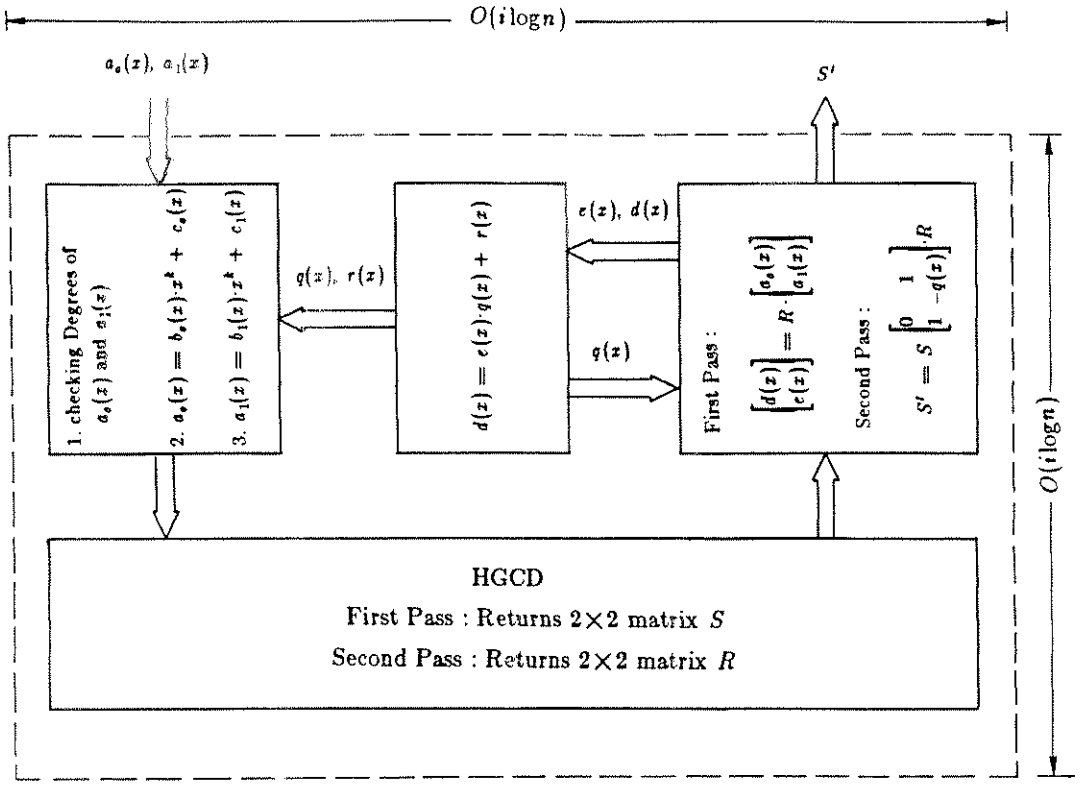


Fig.4 BCH Decoder



Notes: 1. An execution of $HGCD(a_0(x), a_1(x))$ requires two passes through the circuit.

2. The arrows only indicate logic flows of data. The actual data flow takes place over a bus.

Fig.5 HGCD System