Discrete Applied Mathematics 33 (1991) 109-128
North-Holland

A semi-systolic decoder for the PDSC-73 error-correcting code

Kevin Karplus and Habib Krit

Board of Studies in Computer Engineering, University of California at Santa Cruz, Santa Cruz, CA 95064, USA

Received 30 June 1989
Revised 15 December 1989

Abstract

Karplus, K. and H. Krit, A semi-systolic decoder for the PDSC-73 error-correcting code, Discrete Applied Mathematics 33 (1991) 109-128.

This paper presents a semi-systolic architecture for decoding cyclic linear error-correcting codes at high speed. The architecture implements a variant of Tanner's Algorithm B, modified for simpler and faster implementation. The main features of the architecture are low computational complexity, a simple, regular arrangement of cells for easy layout, short critical paths, and a high clock rate.

A prototype chip has been designed to decode a 73-bit perfect difference set code. This $4600\mu m \times 6800\mu m$ chip should achieve 25MHz decoding in $2\mu m$ n-well cMOS.

The success of the implementation illustrates the value of using technology dependent constraints and cost measures to guide the design of algorithms and architectures.

Keywords. Error correction, perfect difference set code, systolic architecture, decoder for error correction, codecs, decoding algorithm, threshold decoding, Tanner's algorithm.

1. Introduction

This paper presents an architecture for decoding linear cyclic error-correcting codes using Tanner's algorithm B [5]. The purpose of the paper is to present our architecture, and to illustrate how VLSI concerns affected the design of the decoder.

One of our key points is that it is important to look at the constraints and costs of the implementation technology. For VLSI chips in cMOS, pinout is limited, regularity is more important than size, and interconnection costs are high compared to computation costs. We used silicon area as our primary cost measure, with two abstract cost measures as estimators for the area: transistor counts and shift register bit counts. Using these cost measures rather than traditional operation counts guided us in making significant improvements in the decoding algorithm and architecture.

To illustrate both our architecture and our design methods, we describe a prototype chip design for one cyclic block code—the (73,45) code based on perfect difference sets.

Two approaches to decoding error-correcting codes have become standard. One uses convolutional codes with small constraint lengths and Viterbi's optimal decoding algorithm [8]. The other uses BCH codes and Berlekamp's algorithm [1]. Viterbi's algorithm is the preferred choice when the error rate is high and soft-decision information is available, as it uses the soft-decision information optimally. Berlekamp's algorithm is preferred at low error rates, where using a code with a large minimum distance is more important than using soft-decision information.

Our technique allows us to use soft-decision information with a block code that has a large minimum distance. The key to our approach is an efficient soft-decision algorithm developed by Tanner [5,6] which gives us reasonably good performance over a wide range of error rates that overlaps the Viterbi and Berlekamp ranges.

We developed a semi-systolic array that implements the computation more efficiently than the fully parallel architecture presented in Tanner's paper [5, p. 541]. Our architecture takes advantage of the structure of the code to produce a simple, regular layout. The regularity of the architecture makes pipelining for high speed easier and reduces the low-level design effort. Because the layout structure is related to the code structure in a very simple way, it should be easy to build an automatic layout tool to generate encoders and decoders directly from the specification of a cyclic code.

Under a research grant from Ford Aerospace, we designed and built a prototype chip to decode a (73,45) code based on perfect difference sets. Our chip is intended for satellite communications, where high data rates are needed over moderately noisy channels. Ford Aerospace would eventually like to design a single-chip decoder that is space qualified and that can decode 80 megabits per second or more. Our own goals for this prototype were more modest, in that we ignored radiation hardening and other concerns of the space environment, and are happy with 20-30 megabits per second from a cheap $2\mu m$ cMOS process. Also, to save fabrication costs, our decoder consists of three identical small chips, rather than a single larger chip.

This paper will discuss the code we use (Section 2), the decoding algorithm (Section 3), the semi-systolic architecture (Section 4), the hardware complexity (Section 5), other decoding algorithms (Section 6), and the chip itself (Section 7).

2. The code

The cyclic code chosen by Ford Aerospace was a perfect difference set code of length 73 (PDSC-73), with a rate of 0.616 (45 data bits) and a minimum distance of 10 [9]. The difference set polynomial, which defines the first row of the parity matrix, is not the same one as in Weldon's original paper, but the code has essentially the same structure. Our difference set polynomial is

$$\theta(x)=x^{57}+x^{56}+x^{45}+x^{37}+x^{31}+x^{27}+x^{24}+x^{22}+1.$$

Difference set codes are particularly attractive for Tanner's algorithm, as one can prove that hard-decision decoding can be done in one iteration [5, Theorem 6]. The results in this paper would have been more striking if the code had been one for which good soft decoding techniques were not known—Massey's threshold decoding algorithm [4] works well on PDSC-73. As we will see in the next section, applying Tanner's algorithm to cyclic codes is very similar to using Massey's algorithm. Unfortunately, the code was dictated by the particular application for which we were asked to design a chip—not chosen for its convenience in illustrating our techniques.

The architecture we describe in this paper is designed for cyclic codes, that is, block codes in which each row of the parity matrix is the cyclic left shift of the row above it. Our future work includes applying our design method to decoders for other codes, including convolutional and quasi-cyclic codes. To make the case for our techniques stronger, we will choose some codes that are not known to be easily decodable by other methods.

To keep costs down, the prototype chip is hard wired to decode a single code, but a programmable version to handle a variety of different cyclic codes could be designed using the same techniques. As we will see in Section 5.4, such a programmable decoder would be about four times as big as our single-code decoder.

3. The algorithm

3.1. Tanner's algorithm

Tanner's algorithm [5, Algorithm B, p. 541] uses soft-decision information in the form of a reliability value for each bit received over the noisy channel. Each iteration of the algorithm involves two steps: first, we compute the parity equations for a word of the code, then we update the reliability of each bit, based on the results of the parity computations. The reliability of a bit is increased for each parity equation it satisfies, and decreased for each one it violates.

The size of the change in a bit's reliability induced by a particular parity equation is the lowest of the reliabilities of all the other bits involved in that parity equation. If all the other bits are highly reliable, we will make a large change in the bit, but if one of the others is unreliable, we will make only a small change. All the updates from the various parity equations are summed to get the complete update for a bit's reliability. When the reliability of a bit becomes negative, the bit's value is changed and its reliability negated.

To compute a bit's change in reliability induced by one parity equation, we need to know the lowest reliability of all the other bits involved in the parity equation. At first glance, it might seem that a parity equation involving m bits would require m registers for the reliabilities for each parity equation. Tanner points out that only two are needed: the lowest and the second lowest [7]. All bits except one will be updated using the lowest of the m reliabilities (min_1). The one remaining bit, which originally had the lowest reliability, will be updated using the second lowest (min_2).

Our simulations show that three iterations of the algorithm give the best bit error rates for PDSC-73. Increasing the number of iterations continues to reduce the word error rate, as more words are decoded correctly. Unfortunately, increasing the number of iterations also causes more errors to appear in incorrectly decoded words, increasing the bit error rate. In a packet switching environment, where bad words must be retransmitted, we might prefer the lower word error rate given by more iterations, but for the intended application, minimizing the bit error rate was judged to be more important.

3.2. Changes to Tanner's algorithm

The above description of the algorithm is simpler than the one presented in [5]. In Tanner's version, a *bit processor* is associated with each bit of the code word, and a *subcode processor* with each subcode. There is a bi-directional link from each bit processor to the subcodes it is used in. For the linear codes we are using, the subcodes are all simple parity checks, and each subcode is one parity equation. For PDSC-73, each of the 73 bits is used in nine parity equations, and so there are 657 of the bi-directional links.

Each link has two registers associated with it. The register R_{ij} holds the information transmitted from bit i to subcode processor j, and R'_{ij} contains the information transmitted back from subcode j to bit i. Tanner's algorithm also keeps the original input data ($V_i(0)$) as a base for corrections in each iteration, thus requiring another 73 registers. For the algorithm implemented exactly as described in Tanner's paper, we would need a total of 1387 registers for R_{ij}, R'_{ij}, and $V_i(0)$.

We made three simplifications to reduce the number of registers:

- By using only parity checks as subcodes, we can reduce the number of R'_{ij} from 657 to 146.
- By transmitting the same data to all of a bit processor's subcode processors, we can reduce the number of R_{ij} from 657 to 73.
- By updating the results of the previous iteration rather than the originally received data, we can eliminate the 73 registers needed to store $V_i(0)$.

These three simplifications reduce the number of registers from 1387 to 219. (Note: this count includes only registers used for the computation, not registers used for pipelining or interfacing. Section 5 has a more thorough analysis of the size of our architecture.)

Tanner deserves credit for the first simplification. He pointed out that using simple parity checks makes the complicated formula given for R'_{ij} [5, p.541] equivalent to computing the minimum reliability of all bits in parity equation j other than bit i, and assigning the appropriate sign based on whether the parity equation is satisfied. That means that the magnitude of R'_{ij} takes on only two different values for parity equation j (min_{1j} and min_{2j}), and so only two registers are needed per subcode processor, independent of the degree.

The second simplification makes a subtle, but significant, change in the way the bit reliabilities are updated. When updating register R_{ij}, we use information from all the parity equations, not excluding equation j as the original algorithm did. That is, we have replaced Tanner's update formula

$$R_{ij}(t) = \sum_{l \in J_i} R'_{il} - R'_{ij} + V_i(0)$$

with

$$R_{ij}(t) = \sum_{l \in J_i} R'_{il} + V_i(0),$$

making R_{ij} independent of j. Because the same value can be sent to each subcode of a bit, instead of one register per link, we need only one register per bit processor, so that only 73 registers are needed for R_{ij}, not 657.

The third simplification changes the update formula to avoid having to store $V_i(0)$:

$$R_{ij}(t) = \begin{cases} V_i(0) & \text{if } t = 0, \\ \sum_{l \in J_i} R'_{il} + R_{ij}(t-1) & \text{if } t > 0. \end{cases}$$

With this change, only the first iteration uses $V_i(0)$, and so the same registers can be used for $V_i(0)$ and R_{ij}, saving us another 73 registers.

Simulations at Ford Aerospace showed that all these modifications caused only a trivial loss of coding gain (about 0.2dB), while our estimates of hardware size and complexity showed that the chip would be significantly easier to design.

The resulting algorithm is quite similar to Massey's threshold decoding algorithm [4]. One variant of Massey's algorithm uses the minimum reliability of all bits other than the orthogonalized bit as the weight of a composite parity check. The appropriately signed weights of the composite parity checks are added to determine the new weight of the orthogonalized bit. This is essentially the same computation as our variant of Tanner's algorithm, but performed on the orthogonalized matrix, rather than the original cyclic parity matrix. One of the chief advantages of Tanner's approach for cyclic codes is that the code need not be orthogonalizable.

Although the orthogonalized matrix is smaller than the original square matrix, its structure is not as regular. From a hardware designer's standpoint, the regularity of the original square parity matrix offers more opportunities for optimizations than the irregular orthogonalized matrix. Some of the hardware differences between the two approaches are discussed in Section 6.1.

3.3. *Algorithmic complexity*

For a cyclic code with m bits in each parity equation and n bits in each word, our algorithm requires nm exclusive-ors, $2nm$ $\min(a, b)$ computations, and nm additions for each iteration of the algorithm. This amounts to m exclusive-ors, $2m$ minimum computations, and m additions per bit. Because the minimum distance of a difference set code is $m+1$, the computation required per bit in our decoding technique grows only linearly with minimum distance, not exponentially as it does with Viterbi decoding.

The architecture described in Section 4 computes one bit per cycle, and so the amortized computation per bit can be used as a lower bound on the size of the hardware. PDSC-73 uses nine bits in each parity equation ($m = 9$), requiring 9 exclusive-ors, 18 minimum-of-two circuits, and 9 adders. If we had chosen the next larger perfect difference set code, PDSC-273, we would have had a minimum distance of 18 with only 17 exclusive-ors, 34 minimum-of-two circuits, and 17 adders. (PDSC-273 has 191 data bits, giving a rate of about 0.70.) In general, the number of functional units grows linearly with the number of bits in each parity equation, which is one less than the minimum distance for perfect difference codes.

The above analysis looks only at the number of computations needed for the decoding, but a guiding principle in VLSI design is that the cost of memory and communication often exceeds the cost of the computation. Section 5 will give a more accurate way of estimating hardware complexity.

4. The architecture

Before we joined the Ford Aerospace project, the hardware had been envisioned as a fully parallel implementation, with 73 parity processors, 73 update processors, and 657 parallel interconnections between them. This architecture was to be a direct implementation of the bipartite graph algorithm described in [5, p. 539]. The design was capable of decoding a 73-bit word in either 3 or 6 cycles, depending on how much pipelining was done, and on how wide the 657 interconnects were. Although such a processor would decode very rapidly, it was too large to fit on a single chip, impossible to partition into pieces with few connections, difficult to load with the initial data, and generally impractical to build and test.

4.1. *The semi-systolic structure*

Three observations were critical for the development of our new architecture:

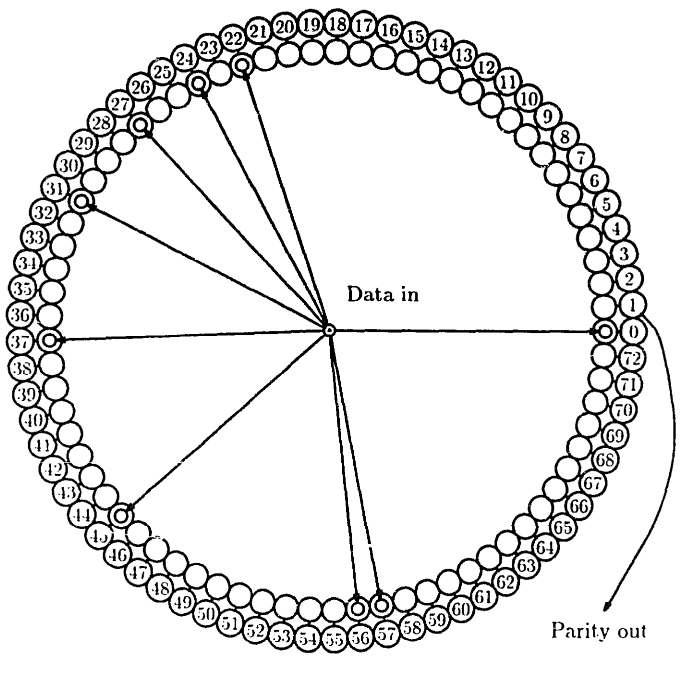
- Ford Aerospace wanted a decoder for a serial channel. This means that the decoder receives data serially, one bit (plus soft-decision information) per clock cycle, and so we do not need a throughput higher than one bit per cycle. The originally proposed decoder was needlessly fast.

- The number of computations needed for Tanner's algorithm is quite small, and so the hardware should be small. If we are decoding one bit per cycle, the amount of hardware we use should be based on the amortized computation cost described in Section 3.3.
- The cyclic structure of the code was not being used by the parallel architecture—the proposed scheme would have worked just as well for any linear code with a sparse parity matrix.

The combination of serial input, one bit per cycle throughput, and the cyclic structure of the parity matrix suggested that a systolic array of 73 cells connected as a cycle would be appropriate. The final design is actually semi-systolic, in that it has a broadcast bus in addition to the nearest-neighbor connections of a systolic array. The broadcast buses transmit information to only 9 of the 73 units in the array (see Fig. 1), and so the loading on the buses is not severe, imposing no performance limitation on the system.

We use two arrays: one for computing the parity equations, the other for updating the bit reliabilities. The communication between the parity and update processors is serial, with one data or parity bit and the associated soft-decision information being transmitted on each cycle.

Each array holds 73 partially computed results. On each clock cycle, one new piece of information is received, and all partial results that depend on it are updated. All the partial results are then shifted one position cyclically (clockwise in Fig. 1).



Parity processor

Fig. 1. Conceptual view of the parity processor for PDSC-73 code, showing the ring of parity values, the output shift register, and the broadcast bus for the data bits. The XOR processors are marked with an extra circle. The output shift register is represented by the outer ring of circles. The numbers indicate the positions of the parity values at the beginning of a new word.

Because each row and column of the parity matrix is a shifted version of the previous one, the movement of the partial results cancels the shifting in the matrix to make the same positions in the systolic array active on each cycle. Consequently, the computational circuitry need be put only in the nine active positions, with the other 64 positions containing simple shift register elements. The input is broadcast only to the nine active positions, and so the broadcast does not represent a significant performance bottleneck, as it might have, if all 73 positions had to receive the data.

The parity processor consists of a 73-stage shift register connected in a ring. Nine of the shift register elements have XORs and comparators for computing the parity values and the two lowest reliabilities. These elements correspond to the nine 1's in the first column of PDSC-73's parity matrix. The direct relationship between the difference set polynomial (the 1's in a row or column of the matrix) and the architecture can be seen in Fig. 1.

At the beginning of each word, the parity values are set to zero. Then they rotate around the ring as the data bits arrive. On each cycle, the parity values that use the currently broadcast data bit are in the active elements of the shift register. At the end of a word, the 73 parity values are transferred in parallel to another shift register for serial readout (see Fig. 1).

The update processor has a similar structure, with data bits circulating around a ring of 73 elements as parity bits are broadcast. To initialize the ring, the values output by the previous iteration are transferred in parallel from an input shift register that accumulated them as they were received. At the end of each word, the updated data bits are transferred in parallel to a shift register for serial readout. A block diagram showing both the parity and the update processors is shown in Fig. 2.

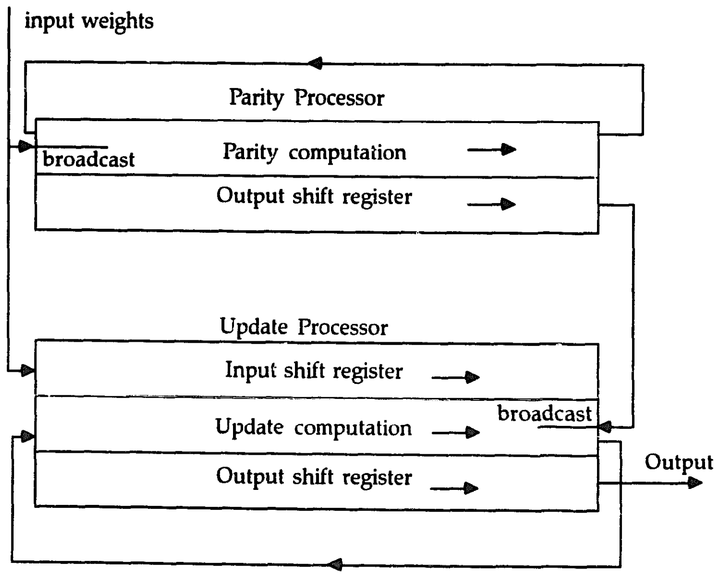


Fig. 2. Block diagram for one iteration of the decoder, showing the shift registers of the parity and update processors.

One attractive feature of the architecture is that the decoder can be partitioned between update and parity processors with only a few wires crossing the partition. The processors can be internally partitioned by cutting only a few more wires, which makes it easy to map the architecture to different technologies with different levels of integration. We designed the chip for the cheapest available VLSI technology (2μm cMOS), in which we could easily fit independent update and parity processors on one 40-pin package.

4.2. Number representation

Soft-decision information is usually treated theoretically as a real-numbered value from $-\infty$ to $+\infty$. For a binary channel, the number is usually the logarithm in some base of the ratio of the likelihoods of the two possible symbols. To simplify our hardware, we chose to represent the soft-decision information in sign magnitude form. The sign can be interpreted as a guess at the value of the transmitted bit, and the magnitude is the reliability of the guess. With this interpretation, the reliability is always a positive number.

In the parity processor, the sign bits and the magnitudes are handled independently. XORs are used on the sign bits to compute the parity, and comparators used on the magnitudes to find the two lowest reliability values involved in each parity equation.

In the update processor, one of these two minimum reliabilities must be added to or subtracted from the reliability of each participating data bit, depending on the broadcast parity bit. Using two's complement representation makes addition and subtraction of reliabilities easier. During the reliability computation, the data bit values are not changed. Just before the update processor outputs a bit, we combine the old bit value and the sign of the new reliability to get the new bit value. If the data bit value is changed, its new reliability is negated to keep the output reliability positive.

If we use the traditional interpretation of sign magnitude numbers, we have two representations for zero (+0 and −0), wasting one of the quantization levels—with w bits for the magnitude, we have only $2^{w+1}-1$ different reliabilities. With $(w+1)$-bit two's complement numbers, however, we have all 2^{w+1} different levels, but they are not symmetrically arranged. To remedy both problems, we chose a scheme that assumes the existence of a hidden one-bit (with weight 1/2) after the least significant digit. Thus for two bits of magnitude information, we would have the numbers −3.5, −2.5, −1.5, −0.5, +0.5, +1.5, +2.5, and +3.5 in both the sign magnitude and the two's complement notations. With either notation a number can be negated simply by complementing all its bits. To translate either way between the notations, we complement all the magnitude bits when the sign bit is a 1.

Addition of numbers in the new scheme poses a slight problem, as we do not want to represent the extra digit explicitly. We can compensate for omitting the extra bit by adding 1 using a high carry in into some of the adders. We have nine adders,

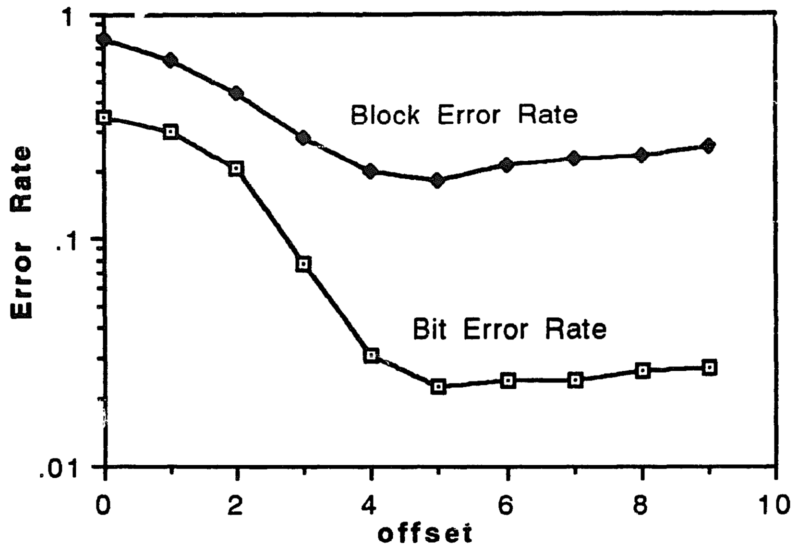


Fig. 3. The output bit error rate as a function of the number of high carry ins in the update processor. The input bit error rate is approximately 0.066.

which gives us the potential to add any number between 0 and 9 as an offset. Because all ten numbers that we add have a hidden 0.5, the intuitive offset to use is 5. Adding a larger offset makes it less likely for the final sum to be negative, thus making it less likely for the data bit to be changed. Adding a smaller offset has the opposite effect.

We chose the optimal offset by simulating the decoding algorithm on 1000 vectors and plotting the output bit error rate as a function of the offset value. The input error rate was chosen so that the average number of errors per word would be slightly less than half the minimum distance (BER $\approx$ 0.066). As can be seen in Fig. 3, performance improves dramatically when increasing the offset from 0 to 5, and gradually decreases above 5. For offsets less than four, the algorithm is clearly increasing the bit error rate.

The average number of bad bits in the incorrectly decoded words is about the minimum distance between code words for offsets of 4 or more, but is far larger for smaller offsets (see Fig. 4). A possible interpretation is that for large offsets, the algorithm converges to a nearby code word, but for small offsets the algorithm does not converge.

5. Hardware complexity

The three main factors affecting hardware cost are the labor costs of doing the detailed design work, the silicon area required for fabrication, and the achievable speed. We chose a highly regular, pipelined architecture to keep the design time low and the speed high. Most of our remaining implementation effort was spent in finding ways to reduce the silicon area.

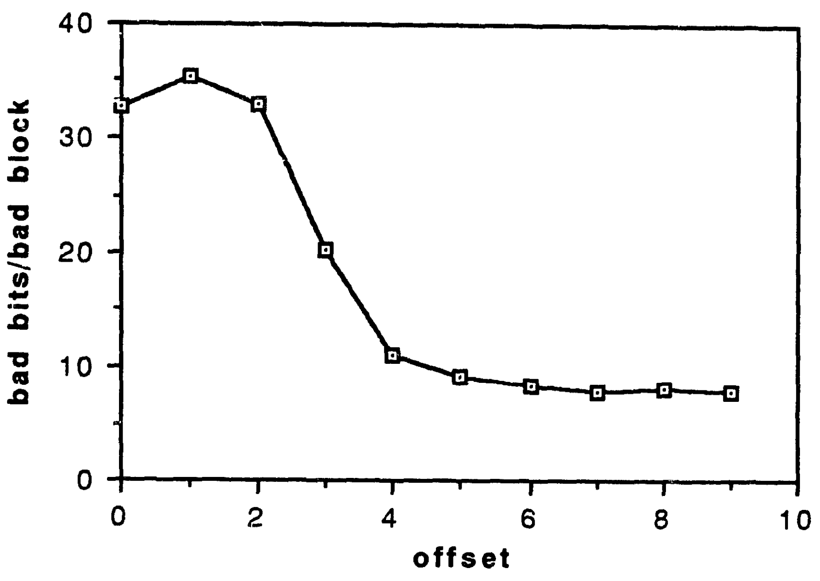


Fig. 4. The average number of bad bits in each incorrectly decoded word as a function of the offset. The input words had an average of 4.85 bad bits in each word. For offset ≥ 4, the bad words seem to be nearby code words. For offset ≤ 3, the algorithm does not always converge.

Once a code is chosen, the length of the shift registers and the number of active units in the parity and update processors are fixed. The main parameters left to choose are the number of soft-decision bits used in the computation (w) and the number of extra guard digits in the adders (g). The guard digits are needed to prevent overflow when adding reliabilities in the update processor (see Section 5.2).

In Section 3.3, we estimated the number of functional units (XORs, adders, and comparators) needed for one iteration of Tanner's algorithm. Because so little computation circuitry is needed, the traditional operation counts are poor estimates of the area needed for the implementation. By using transistor counts as area estimators for the various circuits, we projected that most of the chip area would be used for the shift registers that pipeline the data to the functional units. Because our regular interconnection strategy made the wiring grow roughly proportional to the transistor count, and because our transistor sizes were uniform, the transistor counts worked surprisingly well as area estimates.

If we use w bits of soft-decision information, the active (XOR) cells of the parity processor take $102w + 38$ transistors each, and the inactive (NOP) cells take $48w + 24$ transistors each, for a total count of $3990w + 1878$ transistors in the parity processor. Although the XOR cells are almost three times as large as the NOP cells, only nine XOR cells are needed, and so their total contribution to the parity processor size is at most 25%.

Because the transistor counts are dominated by the shift registers' contribution, much of our design effort was spent in finding ways to reduce this contribution. Two approaches were used: eliminating shift registers that weren't needed (see Section 3.2) and reducing the width of the essential registers.

5.1. Counting shift registers

Because the length of all the shift registers is n ($n = 73$ for PDSC-73), our main concern is the width of the various registers. The widths can be expressed in terms of the number of bits in the reliability information (w) and the number of guard bits in the adders (g).

The parity processor needs two one-bit wide shift registers for the parity bit (one for the computation and one for output) and four w-bit wide shift registers for the minimum reliabilities min_1 and min_2—two for the computation and two for output. The parity processor thus requires $(4w+2)n$ bits of shift register memory.

The update processor needs one $(w+1)$-bit wide shift register to delay the output of the previous iteration while parities are being computed, a $(w+g+1)$-bit wide shift register for computing the new reliability values, and a $(w+1)$-bit wide shift register for output. (Note: g is the number of guard bits needed when adding several numbers, as explained in Section 5.2.) The update processor thus requires $(3w+g+3)n$ bits of shift register memory.

The total number of shift register bits in both processors is $(7w+g+5)n$.

If we had not made the third change in Section 3.2, we would have needed to keep track of the original data in the update processor pipeline, requiring three extra shift registers of width $w+1$, a 44% increase in the shift register memory needed. We have not done detailed counts for the architectures one would get without the first two changes discussed in Section 3.2, but the memory requirements are clearly far larger, particularly when the necessary serial to parallel and parallel to serial converters are included.

5.2. Clipping and scaling in the update processor

The data path in the update processor needs to be wider than in the parity processor, because we need to avoid overflow as we add 10 numbers. One approach is to add 4 guard bits, so that overflow can never happen. But the extra hardware for keeping four guard digits is not efficiently used, as the sums will become large only when we are already certain about the value of a bit.

Another inconvenience of having four guard digits is that, after each update step, the new reliabilities of the data bit have to be scaled back down to w bits, so that the parity processor of the next iteration can accept them. The scaling can be accomplished by a simple shift, but then we lose valuable information for bits that have low reliability, while keeping more information than we need for bits that are unlikely to change.

Using clipping to throw out some information for highly reliable bits is one solution to the problem of losing information on the less reliable bits by too vigorous scaling. Computing the new reliabilities using a full four guard bits, then clipping and scaling the numbers, would still use update processor space inefficiently, as the high order bits of the adders would be used only for highly reliable data. A more efficient scheme is to use clipping adders with only $g<4$ extra bits and doing a final scaling by 2^{-g}, thus avoiding overflow with fewer guard bits. Our simulations showed that this approach had almost no loss of performance, and using clipping adders was significantly smaller than adding extra guard bits.

If we use clipping adders and g extra bits, the sum cells of the update processor take $118w+48g+62$ transistors each, and the NOP cells take $48w+12g+36$ transistors each, for a total update processor size of $4134w+1200g+2862$. Combining the update and parity processor counts gives us a total estimate of $8124w+1200g+4740$ transistors.

5.3. Picking minimal values for w and g

Clearly, we want to choose both w and g as small as possible without sacrificing decoder performance. Making w and g small has the further advantage of reducing the carry propagate time in the adders. Because this propagation time is a major component of the critical delay path, any reductions in it let us increase our clock speed.

Based on simulations done at Ford Aerospace, we determined that $w=3$ and $g=1$ were the smallest values for which coding gain was not significantly degraded. These choices gave us an estimated count of 13,848 transistors in the parity processor, and 16,464 transistors in the update processor. On the final chip, the parity processor takes up 5.02 square millimeters, and the update processor 6.69. The relative sizes can be seen in Fig. 5.

5.4. Hardware costs for other cyclic codes

If we wanted a programmable decoder for many different cyclic codes, we would have to make each cell of the parity processor capable of acting as an XOR cell, and each cell of the update processor capable of acting as a sum cell. The resulting design would be about four times as large as our single code design.

For hard wired decoders of cyclic block codes, we expect that the area of a decoder will still depend primarily on the space needed for the $(7w+g+5)n$ shift register elements. Because w and g are small constants independent of n, the number of shift register elements grows linearly with the word length, which grows quadratically with the minimum distance d for difference set codes. This means that the overall hardware cost grows roughly quadratically with d, not linearly as estimated by using traditional operation count measures in Section 3.3.

Our chip shows that the hardware cost is fairly small for $n=73$, and should still be reasonable for $n=273$. Decoders for the next larger perfect difference set code ($n=1057$, $d=34$) should be feasible to build, but the chips may be so large that yield would be poor. Although the PDSC-73 decoder is nowhere near chip size limitations, much larger chips may have some performance limitations because of clock distribution problems (see Section 7.1).

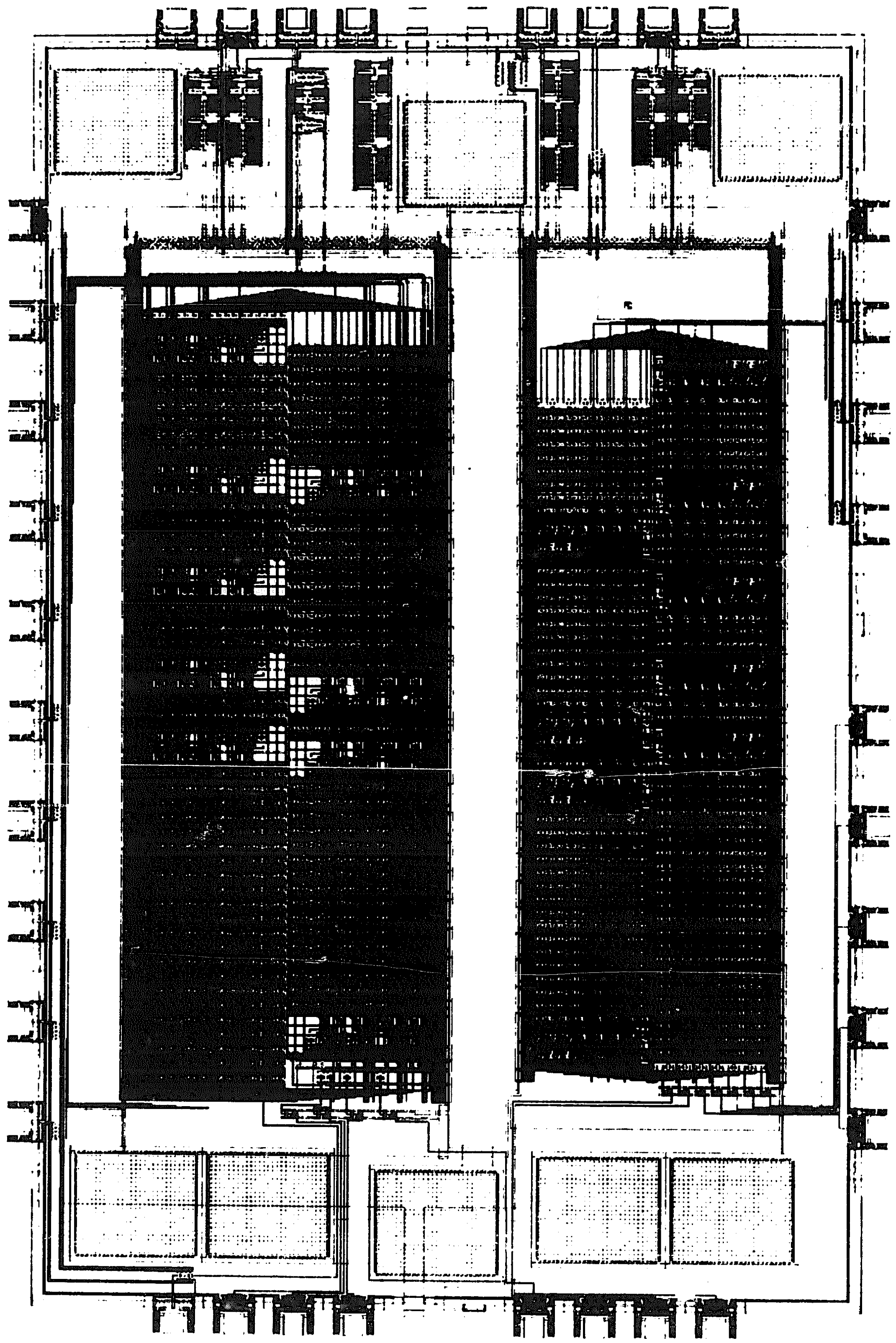


Fig. 5. Layout of the 2μm n-well cMOS PDSC-73 chip. The small blocks at the top are for buffering the master and derived clocks.

6. Comparison with other decoders

6.1. Comparison with Massey's threshold decoder

Massey shows how to construct hard-decision decoders for cyclic codes that can be one-step orthogonalized [4,91]. Using a similar circuit for PDSC-73 with the orthogonalization shown in [3, Tables 3–4, p. 134], we would need 28 XORs (9 for the syndrome, 19 to generate the composite checks), two one-bit wide shift registers (one with 73 bits, the other with 28 bits), and a 5-of-9 majority circuit. The critical delay path for this hard-decision decoder passes through the majority circuit and a five-input XOR tree, and is likely to be about the same speed as our soft-decision decoder.

Algorithm B can also be used for hard-decision decoding. In fact, Tanner proved that one iteration of the hard-decision algorithm can correct $(d-1)/2$ bits [5, Theorem 6]. Because all bits have the same reliability on the first iteration, no minimums need to be computed, and a simple increment/decrement circuit can be used to update the bits. For PDSC-73, the parity processor would have 9 XORs and 73 one-bit registers, and the update processor would have 9 increment/decrement circuits and 73 3-bit registers. The Algorithm B hard-decision decoder should be somewhat larger than a Massey hard-decision decoder, because of the need for more memory. But it should be faster than Massey's decoder, because of the shorter critical path.

To do soft-decision decoding with Massey's original algorithm would require several multipliers, adders, and a threshold unit. A standard simplification is to take the weight of each composite parity check as the minimum reliability of all bits in the parity check except the one that is being updated. With this simplification, we could compute for each xor the minimum weight of the contributions to the xor, then add the signed weights to determine the new weight for the current bit. The computation to be done is almost identical to the computation for our technique, except that the orthogonalized parity matrix is used, instead of the original cyclic one.

A straightforward implementation of this variant of Massey's threshold decoder would require 19 comparators, 19 XORs, and 8 adders. This is almost the same as the number of functional units our method uses, but Massey's algorithm uses less memory. The irregularity of the orthogonalized array makes Massey's decoder harder to lay out, and much harder to pipeline. Because of the difficulty in pipelining the decoder, the clock cycle will be determined by the time needed to propagate through the entire tree of comparators and adders, not just through a pair of comparators or a single adder. Therefore, it seems that Massey's technique would yield a smaller, but much slower decoder.

The main weakness of Massey's technique is that it only applies to orthogonalizable codes. We look forward to designing a decoder for a code that cannot be decoded by Massey's method.

6.2. Comparison with Chase's method

To emphasize the simplicity and small size of our method, we would like to compare it with another decoding algorithm that uses soft-decision information on block codes—Chase's method [2]. Chase's method uses soft-decision information to decide which bits of a received word are least reliable, change some of them, and do hard-decision decoding. Several different combinations of bits are changed, the distance of each decoded word from the original input is measured, and the closest fit is chosen as the result. The full algorithm tries all error patterns with weight $\leq d/2$, which, for PDSC-73, would require

$$\binom{73}{0}+\binom{73}{1}+\binom{73}{2}+\binom{73}{3}+\binom{73}{4}+\binom{73}{5}=16{,}173{,}662$$

hard-decision decodings. A reduced version of the algorithm tries all error patterns on the least reliable $d/2$ bits, requiring only 32 hard-decision decodings.

We could build a hard-decision decoder for a difference set code using one iteration of our architecture with $w=0$ and $g=3$. Such a decoder would be about a tenth the size of the three-iteration soft-decision decoder. If, instead, we use a majority decoder to do hard-decision decoding [4, p. 91], the size may be reduced some more, but is certainly no smaller than a thirtieth of the three-iteration soft-decision decoder.

With these figures, it looks like 32 hard-decision decoders takes up at least as much space as our soft-decision decoder. But Chase's method requires more than just hard-decision decoders—it requires circuitry for choosing the $d/2=5$ least reliable bits, modifying them, and measuring the distance between the received word and the decoded word. Each distance measurement requires about half the circuitry of our update processor, and the bit selection looks at least as complex as our parity processor. Thus Chase's method seems to require at least 4 times as much area as our three-iteration soft-decision decoder. Because of the greater complexity (as well as greater size), we have not investigated VLSI implementations of Chase's algorithm further.

7. The chip

We have designed a cMOS chip that implements both the parity processor and the update processor for the PDSC-73 code on a $4600\mu m \times 6800\mu m$ die (see Fig. 5), fitting a complete three-iteration decoder on only th[illegible] chips. Using conservative design in $2\mu m$ n-well cMOS, we expect to decode at 25 MHz.

Because we chose $w=3$, the parity processor needs 15 pins, and the update processor 20. The parity processor needs 4 input pins for the data bit and soft-decision information; 7 output pins for the parity bit, min_1, min_2; 2 pins (one input and one output) for the *start* signal that marks the beginning of each word; and one pin for each of clock, power, and ground. The update processor needs 4 input pins for the results of the previous iteration, 7 input pins for the information from the parity processor, 4 output pins for the updated values, a clock pin, an input and output pin for the *start* bit, one power, and one ground.

We had originally intended to put the processors on separate dies, packaging each in a 28-pin package. Changes in the price schedule by our silicon broker (MOSIS) made it cheaper to put both processors on a single chip in a 40-pin package. For ease in design verification and improved yield, we gave each processor independent pins (except for the master clock).

By tying the outputs of the parity processor to the inputs of the sum processor internally, a commercial version of the chip could be produced in a 14-pin package. Such a chip should still be testable, as the shift registers inherent in the architecture make all internal bits highly observable and controllable. Because the die size is quite small, a full three-iteration decoder could be produced on one chip with good yields.

7.1. Clock distribution

Our main speed limitation is in generating and distributing our two nonoverlapping internal clocks. We use single-phase clocking off chip, for ease of interfacing, and generate the two-phase clock internally. All the shift registers are controlled by the clock signals, thus putting large capacitive loads on the clocks. One of the clock signals has a load of 50pF, and the others are almost as large, requiring large buffers to drive the signals quickly.

Unfortunately, rapid charging and discharging of such large capacitances generates large current spikes on the power supplies. The inductance of the package leads converts these current spikes into large voltage spikes (proportional to dI/dT), propagating noise throughout our system. We could reduce the capacitance on the clock lines by going to a cMOS process with smaller geometries, but peak current would remain about the same, and dI/dT would increase, thus making our problem worse.

We can limit the noise by spreading the current spike. If the current spike is triangular, lengthening the spike by a factor of s decreases the voltage noise by a factor of s^2. The current spike can be spread by deliberately introducing clock skew and by using undersized drivers to reduce the edge speed. Splitting the large drivers into several smaller ones, one for each clock line in the layout, may be an effective technique for controlling both the edge speed and the clock skew.

Other approaches to limiting the noise are to reduce the lead inductance with better packaging or to put bypass capacitors on the chip.

7.2. Ease of testing

We did not add any extra hardware to our chip to make it more testable. The shift registers inherent in the design already make the controllability and observability good. Almost any random test pattern will test all the shift register cells for stuck at and bridging faults, and the XORs are similarly easy to test. We cannot, however, rely on random testing for the comparators, as the probability of getting all 1's for min_2 would be 2^{-2mw} (that is, 2^{-54} for PDSC-73).

Our testing technique for the comparators in the parity processor and the adders in the update processor relies on the *start* signal that marks the beginning of a word. When the *start* signal is active, it causes similar activity in both processors. In the parity processor, the parity bits and associated reliabilities (min_1 and min_2) are transferred in parallel from the computation shift register to the output shift register, and the computation shift register is re-initialized. In the update processor, the new values in the computation register are transferred to the output register, and the next set of input values are transferred to the computation register.

The *start* signal normally becomes active once every 73 bits to indicate that the current bit is the first bit of a new word. Having 73 shifts between *start* pulses causes the data to go through all nine processing elements, making it difficult to pinpoint faults, and possibly masking some of them. By asserting *start* on two consecutive clock cycles, we allow the data on the computing row to shift only once before transferring to the output shift register, thus ensuring that each bit goes through at most one processing element.

For example, in the parity processor, we can test outputs of the min_1 comparators by asserting *start* while inputting to the chip the reliability we want to see on the min_1 outputs. If we then send a normal word (*start* followed by 72 more clock cycles), we should see all one's on the min_1 outputs, except on the 9 cycles that correspond to the 9 active elements, where we should see the value that we input.

Exhaustive testing (all 2^{2w} input patterns) of the min_1 comparators can be done by repeating the following three steps:

(1) Send a partial word of at least 22 cycles with a constant reliability, to set up all the values in the min_1 shift register. The word should have *start* asserted only on the first cycle.

(2) Send the second reliability value for a single cycle with *start* still off.

(3) Assert *start* for one cycle, then clock out min_1 values until all 9 active positions have been observed.

Similar testing procedures exist for testing the min_2 comparators in the parity processor and the adders in the update processor.

7.3. Results from testing fabricated chips

Our first silicon returned with some minor errors in the design that were not caught in the simulation. The errors were missing contacts between wires that had the same name—magic's circuit extractor assumed they were connected elsewhere.

For the second silicon, we added the missing contacts and tried to minimize the power supply noise by putting large bypass capacitors on the chip, avoiding the lead inductance. We designed the capacitors to have a small series resistance, so that they could provide the needed current quickly enough. The total capacitance is about 3150pF, and the RC time constant was designed to be less than 0.5nsec. Carelessness by the capacitor designer and inadequate handling of wells by magic's circuit extractor resulted in power ground shorts that were not detected in simulation.

The third silicon had a slightly different bypass capacitor design and automated checks were done of all well and substrate contacts. The bypass capacitors can be seen as the large rectangles at the top and bottom of Fig. 5. Initial checks show the chip working up to 10MHz, at which point our current test setup fails to capture the results. Preliminary results indicate that the chips continue to work up to 25MHz. We are currently redoing our test setup to do more extensive tests at these higher speeds.

8. Conclusions and future work

The design of the decoder was driven by VLSI considerations:

- Using a semi-systolic architecture rather than a parallel one makes the pin count low enough to fit the decoder on a chip.
- Most of the data movement in the decoder uses nearest neighbor connections, minimizing the area and delay of on chip wiring.
- The regularity of the architecture makes layout straightforward, reducing design costs.
- Changes to the algorithm and architecture were evaluated using transistor counts and shift register bit counts, rather than operation counts.

We believe that considerations like these are becoming more important in hardware design, and that algorithm designers would do well to look beyond the traditional complexity models.

Although we are pleased with the chip we have designed, a lot more work can be done. For example, Tanner's algorithm works on a large class of codes, but the semi-systolic architecture presented here is only for cyclic block codes. A similar architecture with multiple rings should work for quasi-cyclic block codes. The convolutional codes that can be derived from quasi-cyclic block codes should also have simple architectures, using pipelines instead of rings. We are working on architectures for quasi-cyclic and convolutional codes with better error-correcting properties than PDSC-73, and we hope to obtain similar decoding speed.

Ideally, we would like to build a module generator that automatically produces a decoder from a specification of the code and a few decoder parameters. Part of such a module generator would be a redesign of the clock circuitry of the chip to get high performance with less noise, particularly for larger code sizes. By redesigning our clock driver and switching to a 1.2μm process, we should get 60–70MHz operation for the current code. With more effort, we could redesign the chip in gallium arsenide technology, for even greater speeds.

Acknowledgement

The architecture and chip design research was supported by a grant from Ford Aerospace and Communication Corporation. A patent is being applied for, and people interested in using the techniques should contact Ford Aerospace for licensing information. We would like to thank Michael Tanner for explanations of his algorithm, Paul Monte for simulating algorithm variants, Bindesh Patel for assistance in chip testing and redesign work for the second silicon, and Information Systems Institute for running the MOSIS silicon broker service. The architecture design is primarily the work of Kevin Karplus, but the logic design, algorithm tuning, and layout are primarily the work of Habib Krit.

References

[1] E.R. Berlekamp, Algebraic Coding Theory (McGraw-Hill, New York, 1968).

[2] D. Chase, A class of algorithms for decoding block codes with channel measurement information, IEEE Trans. Inform. Theory 18 (1972) 170–182.

[3] G.C. Clark Jr and J.B. Cain, Error-Correction Coding for Digital Communications (Plenum Press, New York, 1981).

[4] J.L. Massey, Threshold Decoding, MIT Press Series 20 (MIT Press, Cambridge, MA, 1963).

[5] R.M. Tanner, A recursive approach to low-complexity codes, IEEE Trans. Inform. Theory 27 (1981) 533–547.

[6] R.M. Tanner, Error-correcting coding system, United States Patent 4, 295, 218 (1981).

[7] R.M. Tanner, VLSI implementation of high speed decoders—possible architectures for intermediate technologies, Tech. Rept., Ford Aerospace (1986).

[8] A.J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans. Inform. Theory 13 (1967) 260-269.

[9] E.J. Weldon, Difference-set cyclic codes, Bell System Tech. J. 45 (1966) 1045–1055.