UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

Apple Inc.,
Petitioner

v.

California Institute of Technology,
Patent Owner.

Case: IPR2017-00700

DECLARATION OF JAMES A. DAVIS, PH.D. REGARDING U.S. PATENT NO. 7,421,032

Apple 1004

TABLE OF CONTENTS

Page

I, James A. Davis, Ph.D., declare as follows:

1. My name is James A. Davis.

I. BACKGROUND

2. I am a Professor of Mathematics at the University of Richmond in Richmond, Virginia.

3. I received a B.S. in Mathematics (with honors) from Lafayette College in 1983 and an M.S. and Ph.D. in Mathematics from the University of Virginia in 1985 and 1987, respectively.

4. After receiving my doctorate, I taught for one year at Lafayette College before accepting a position at the University of Richmond as an Assistant Professor of Mathematics in 1988. I became an Associate Professor of Mathematics in 1994 and a Full Professor of Mathematics in 2001.

5. Since joining the faculty of the University of Richmond in 1988, I have been engaged in research in Coding Theory, Algebra, and Combinatorics. My research has appeared in journals such as IEEE Transactions on Information Theory, the Journal of Combinatorial Theory Series A, Designs, Codes, and Cryptography, the Proceedings of the American Mathematical Society, and the Journal of Algebra.

6. I have made several major contributions to the field of coding theory in wireless communication and sequence design. I co-discovered the connection

between sequences with good power control and Reed-Muller codes, an important step in making OFDM communication practical. I co-discovered a technique for constructing difference sets that has been applied to constructions of bent functions. I co-wrote the paper on the non-existence of Barker arrays.

7. I was a co-Principal Investigator of a $1.5 million National Science Foundation grant designed to engage undergraduates in long-term research projects in mathematics.

8. I have taught mathematics courses in Calculus, Statistics, Linear Algebra, Abstract Algebra, Coding Theory, and Cryptography, among others. I have directed 12 honors projects and 76 summer research experiences for undergraduates in the general area of Coding Theory and Combinatorics.

9. I spent two years (academic years 1995-96 and 2000-01) working at Hewlett-Packard Laboratories in Bristol, England. I was in a communications lab during this time, an industrial research lab focused on applications of Coding Theory to wireless communication and storage devices. I am co-inventor on 16 patents based on my work during this time.

10. I served as Chair of the Department of Mathematics and Computer Science 1997-2000.

11. I have authored or co-authored over 50 peer-reviewed academic publications in the fields of Coding Theory, Combinatorics, Finite Geometry, and Algebra.

12. A copy of my curriculum vitae is attached as Appendix A.

13. I have reviewed the specification and claims of U.S. Patent No. 7,421,032 (the "'032 patent"; Ex. 1001). I have been informed that the '032 patent is a continuation of U.S. Patent No. 7,116,710, which claims priority to provisional applications filed on May 18, 2000 and August 18, 2000.

14. I have also reviewed the following references, all of which I understand to be prior art to the '032 patent:

- L. Ping, W. K. Leung, N. Phamdo, "Low Density Parity Check Codes with Semi-random Parity Check Matrix." Electron. Letters, Vol. 35, No. 1, pp. 38-39, published on January 7, 1999 ("Ping"; Ex. 1003.)
- D. J. C. MacKay, S. T. Wilson, and M. C. Davey, "Comparison of Constructions of Irregular Gallager Codes," IEEE Trans. Commun., Vol. 47, No. 10, pp. 1449-54, published in Oct. 1999 ("MacKay"; Ex. 1002.
- D. Divsalar, H. Jin, and R. J. McEliece, "Coding Theorems for 'Turbo-like' Codes," Proc. 36th Allerton Conf. on Comm., Control

and Computing, Allerton, Illinois, pp. 201-10, March, 1999 ("Divsalar"; Ex. 1017.)

- Luby, M. et al., "Practical Loss-Resilient Codes," STOC '97, pp. 150-159, published in 1997 ("Luby97"; Ex. 1008.)
- Pfister, H. and Siegel, P, "The Serial Concatenation of Rate-1 Codes Through Uniform Random Interleavers," 37th Allerton Conf. on Comm., Control and Computing, Monticello, Illinois, published on or before September 24, 1999 ("Pfister"; Ex. 1022.)

15. I am being compensated at my normal consulting rate for my work.

16. My compensation is not dependent on and in no way affects the substance of my statements in this Declaration.

17. I have no financial interest in Petitioners. I similarly have no financial interest in the '032 patent.

II. LEGAL PRINCIPLES

18. I have been informed that a claim is invalid as anticipated under Pre-AIA 35 U.S.C. § 102(a) if "the invention was known or used by others in this country, or patented or described in a printed publication in this or a foreign country, before the invention thereof by the applicant for patent." I have also been informed that a claim is invalid as anticipated under Pre-AIA 35 U.S.C. § 102(b) if "the invention was patented or described in a printed publication in this or a

foreign country or in public use or on sale in this country, more than one year prior to the date of the application for patent in the United States." Further I have been informed that a claim is invalid as anticipated under Pre-AIA 35 U.S.C. § 102(e) if "the invention was described in … an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent …." It is my understanding that for a claim to be anticipated, all of the limitations must be present in a single prior art reference, either expressly or inherently.

19. I have been informed that a claim is invalid as obvious under Pre-AIA 35 U.S.C. § 103(a):

> if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which [the] subject matter pertains.

20. I understand that a claimed invention would have been obvious, and therefore not patentable, if the subject matter claimed would have been considered obvious to a person of ordinary skill in the art at the time that the invention was made. I understand that when there are known elements that perform in known ways and produce predictable results, the combination of those elements is probably obvious. Further, I understand that when there is a predictable variation

and a person would see the benefit of making that variation, implementing that predictable variation is probably not patentable. I have also been informed that obviousness does not require absolute predictability of success, but that what does matter is whether the prior art gives direction as to what parameters are critical and which of many possible choices may be successful.

III. OVERVIEW OF THE TECHNOLOGY

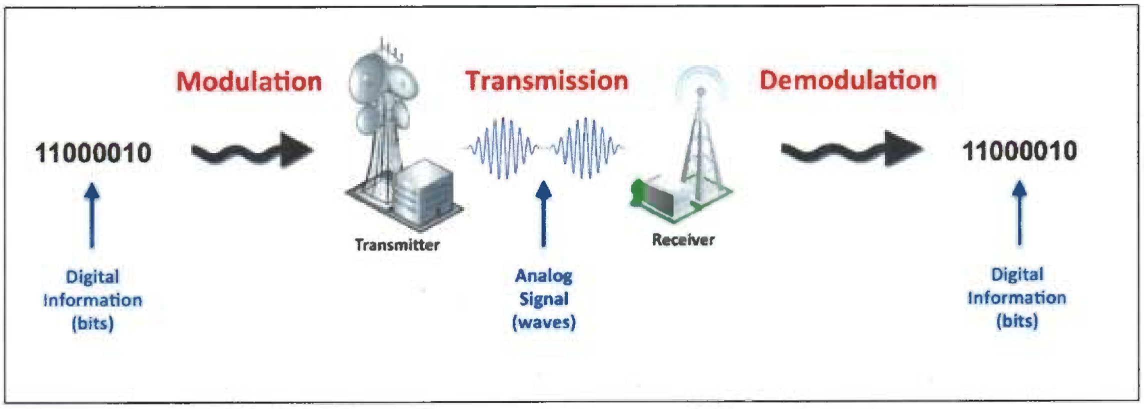
21. The '032 patent relates to the field of channel coding and error-correcting codes. This section provides an introduction to channel coding and error-correcting codes, highlighting developments relevant to the '032 patent.

A. Error-Correcting Codes in General

22. Most computing devices and other digital electronics use bits to represent information. A bit is a binary unit of information that may have one of two values: 1 or 0. Any type of information, including, for example, text, music, images and video information, can be represented digitally with bits.

23. When transmitting binary information over an analog communication channel, the data bits representing the information to be communicated (also called "information bits") are converted into an analog signal that can be transmitted over the channel. This process is called *modulation*. The transmitted signal is then received by a device and converted back into binary form. This process, in which a

received analog waveform is converted into bits, is called *demodulation.* The steps of modulation and demodulation are illustrated in the figure below:



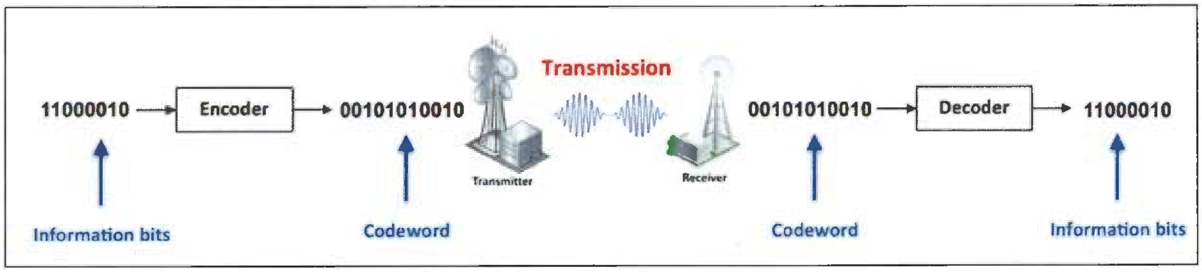
Modulation, Transmission, and Demodulation

24. Transmission over physical channels is rarely 100% reliable. The transmitted signal can be corrupted during transmission by "noise" caused by, for example, obstructions in the signal path, interference from other signals, or electrical/magnetic disturbances. Noise can cause bits to "flip" during transmission: for example, because of noise, a bit that was transmitted as a 1 can be corrupted during transmission and demodulated as 0, and vice versa.

25. Error-correcting codes were developed to combat such transmission errors. Using the bits representing the information to be communicated (called "information bits") an error-correcting code generates "parity bits" that allow the receiver to verify that the bits were transmitted correctly, and to correct transmission errors that occurred.

26. Bits are encoded by an encoder, which receives a sequence of information bits as input, generates parity bits with an encoding algorithm, and outputs a sequence of encoded bits called a *codeword*. The codeword produced by the encoder is then modulated and transmitted as an analog signal.

27. At the receiver, the signal is demodulated, and the codeword is passed to the *decoder*, which uses a decoding algorithm to recover the original information bits.



Encoding and Decoding

28. Error-correcting codes work by adding redundant information to the original message. Due to redundancy, the information represented by a given information bit is spread across multiple bits of the codeword. Thus, even if one of those bits is flipped during transmission, the original information bit can still be recovered from the others.

29. As a simple example, consider an encoding scheme, called "repeat-three," that outputs three copies of each information bit. In this scheme, the information bits "1 0 1" would be encoded as "111 000 111." Upon receipt, the

decoder converts instances of "111" into "1" and instances of "000" into "0" to produce the decoded bits "1 0 1," which match the original information bits.

30. Suppose a bit is flipped during transmission, changing "000" to "010." The decoder can detect a transmission error, because "010" is not a valid "repeat-three" codeword. Using a "majority vote" rule, the decoder can infer that the original information bit was a 0, correcting the transmission error. Thus, due to the redundancy of the codeword, no information was lost due to the transmission error.

31. Error-correcting codes are either *systematic* or *non-systematic*. (Ex. 1020, p. 12, "Also, a [binary convolutional encoder] can be systematic or non-systematic."; *id.* pp. 12-13, Figures 2.1 and 2.2, showing systematic and non-systematic encoders.) In a systematic code, both the parity bits and information bits are included in the codeword. (Ex. 1020, p. 14, "a systematic encoder is one for which the encoder input (the data) forms a substring of the output (the codword)"; Ex. 1021, pp. 6, 229.) In a non-systematic code, the encoded data only includes the parity bits.

32. Systematic and non-systematic codes had been known in the art for decades prior to May 18, 2000, the claimed priority date of the '032 patent.

B. Coding Rate

33. Many error-correcting codes encode information bits in groups, or *blocks* of fixed length *n*. An encoder receives a *k*-bit block of information bits as

input, and produces a corresponding *n*-bit codeword. The ratio *k/n* is called the *rate* of the code. Because the codeword generally includes redundant information, *n* is generally greater than *k*, and the rate *k/n* of an error-correcting code is generally less than one.

C. Performance of Error-Correcting Codes

34. The effectiveness of an error-correcting code may be measured using a variety of metrics.

35. One tool used to assess the performance of a code is its *bit-error rate* (BER.) The BER is defined as the number of corrupted information bits divided by the total number of information bits during a particular time interval. For example, if a decoder outputs one thousand bits in a given time period, and ten of those bits are corrupted (meaning they differ from the information bits originally received by the encoder), then the BER of the code during that time period is (10 bit errors) / (1000 total bits) = 0.01 or 1%.[1]

[1] Note that as used herein, BER refers to the *information* BER, which measures the percentage of bits that remain incorrect *after* decoding. This is different than the *transmission* BER, which measures the percentage of bits that are incorrect when *received* by the decoder, but before the decoding process begins.

36. The BER of a transmission depends on the amount of noise in the communication channel and the strength of the transmitted signal (meaning the power that is used to transmit the signal). An increase in noise tends to increase the error rate and an increase in signal strength tends to decrease the error rate. The ratio of signal strength to noise, called the "signal-to-noise ratio," is often used to characterize the channel over which the encoded signal is transmitted. The signal-to-noise ratio can be expressed mathematically as E_b/N_0, in which E_b is the amount of energy used to transmit each bit of the signal, and N_0 is the density of the noise. The BER of an error-correcting code is often measured for multiple values of E_b/N_0 to determine how the code performs under various channel conditions.

37. Error-correcting codes may also be assessed based on their computational complexity. The complexity of a code is a rough estimate of how many calculations are required for the encoder to generate the encoded parity bits and how many calculations are required for the decoder to reconstruct the information bits from the parity bits. If a code is too complex, it may be impractical to build encoders/decoders that are fast enough to use it.

D. LDPC Codes, Turbo Codes, and Repeat-Accumulate Codes

38. In 1963, Robert Gallager described a set of error correcting codes called Low Density Parity Check ("LDPC") codes. Gallager described how LDPC codes provide one method of generating parity bits from information bits using a

matrix populated with mostly 0s and relatively few 1s, as explained in more detail below. (*See* Gallager, R., *Low-Density Parity-Check Codes*, Monograph, M.I.T. Press, 1963; Ex. 1005.)

39. Gallager's work was largely ignored over the following decades, as researchers continued to discover other algorithms for calculating parity bits. These algorithms included, for example, convolutional encoding with Viterbi decoding and cyclic code encoding with bounded distance decoding. In many cases, these new codes could be decoded using low-complexity decoding algorithms.

40. In 1993, researchers discovered "turbocodes," a class of error-correcting codes capable of transmitting information at a rate close to the so-called "Shannon Limit" – the maximum rate at which information can be transmitted over a channel. A standard turbocoder encodes a sequence of information bits using two "convolutional" coders. The information bits are passed to the first convolutional coder in their original order. At the same time, a copy of the information bits that have been reordered by an interleaver is passed to the second convolutional coder. The figure below shows the structure of a typical turbocoder. *See* Berrou *et al.*, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," *ICC '93*, Technical Program, Conference Record 1064, Geneva 1993 (Ex. 1006)

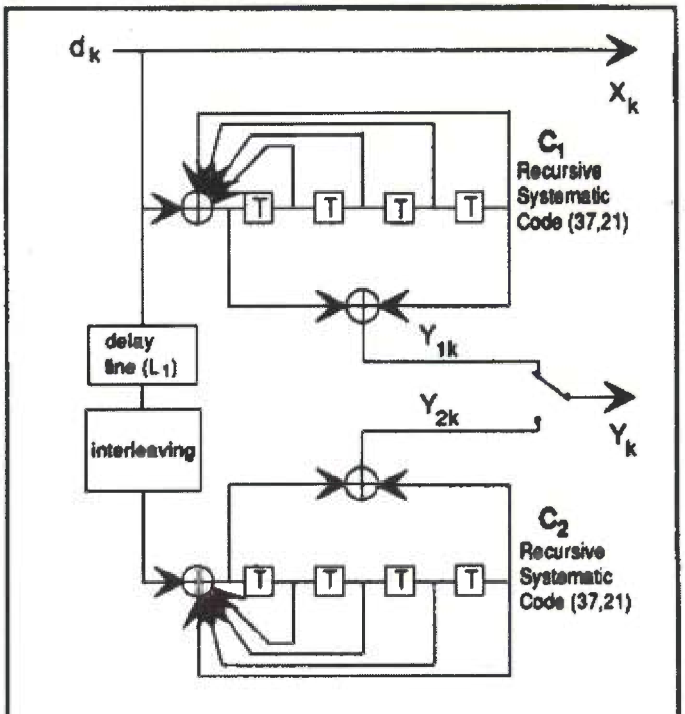


Fig. 2 Recursive Systematic codes with parallel concatenation.

41. The main drawback of convolutional codes is that they do not perform well over channels in which errors are clustered tightly together. Turbo codes overcome this deficiency by encoding the input bits twice. The input bits are fed to a first convolutional encoder in their normal order to produce a first set of parity bits. The same input bits are also reordered by an interleaver and then encoded by a second convolutional encoder to produce a second set of parity bits. The two sets of parity bits together with the information bits form a single codeword. Using a turbo code, a small number of errors will not result in loss of information unless

the errors happen to fall close together in both the original data stream and in the permuted data stream, which is unlikely.

42. In 1995, David J. C. MacKay rediscovered Gallager's work from 1963 relating to low-density parity-check (LDPC) codes, and demonstrated that they have performance comparable to that of turbocodes. *See* MacKay, D. J. C, and Neal, R. M. "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letters*, vol. 32, pp. 1645-46, 1996 (Ex. 1016.) This rediscovery was met with wide acclaim. Turbocodes and LDPC codes have common characteristics: both codes use pseudo-random permutations to spread out redundancy, and both use iterative decoding algorithms.

43. In 1995 and 1996, researchers began to explore "concatenated" convolutional codes. *See* Benedetto, S. et al., *Serial Concatenation of Block and Convolutional Codes*, 32.10 Electronics Letters 887-8, 1996 (Ex. 1007.) While turbo codes use two convolutional coders connected in parallel, concatenated convolutional codes use two convolutional coders connected in series: the information bits are encoded by a first encoder, the output of the first encoder is interleaved, and the interleaved sequence is encoded by a second, convolutional code. In such codes, the first and second encoders are often called the "outer coder" and the "inner coder," respectively.

44. In 1998, researchers developed "repeat-accumulate," or "RA codes," by simplifying the principles underlying turbocodes. (*See* Ex. 1017.) In RA codes, the information bits are first passed to a repeater that repeats (duplicates) the information bits and outputs a stream of repeated bits (the encoder described above in the context of the "repeat three" coding scheme is one example of a repeater). The repeated bits are then optionally reordered by an interleaver, and are then passed to an accumulator where they are "accumulated" to form the parity bits.

45. Accumulation is a running sum process whereby each input bit is added to the previous input bits to produce a sequence of running sums, each of which represents the sum of all input bits yet received. More formally, if an accumulator receives a sequence of input bits i_1, i_2, i_3, … i_n, it will produce output bits o_1, o_2, o_3, … o_n, such that:

$$\begin{aligned} o_1 &= i_1 \\ o_2 &= i_1 \oplus i_2 \\ o_3 &= i_1 \oplus i_2 \oplus i_3 \\ &\vdots \\ o_n &= i_1 \oplus i_2 \oplus i_3 \oplus \cdots \oplus i_n \end{aligned}$$

Where the ⊕ symbol denotes modulo-2 addition. Accumulators can be implemented simply, allowing accumulate codes to be encoded rapidly and cheaply.

E. Mathematical Representations of Error-Correcting Codes

1. *Linear Transformations*

46. Coding theorists often think of error-correcting codes in linear-algebraic terms. For example, a k-bit block of information bits is a k-dimensional vector of bits, and an n-bit codeword is an n-dimensional vector of bits (a "dimensional vector" of bits is a sequence of bits). The encoding process, which converts blocks of information bits into codewords, is a linear transformation that maps k-dimensional bit vectors to n-dimensional bit vectors. This transformation is represented by an $n \times k$ matrix **G** called a *generator matrix*. For a vector of information bits $\boldsymbol{u}$, the corresponding codeword $\boldsymbol{x}$ is given by:

$$\boldsymbol{x} = \mathbf{G}\boldsymbol{u} = \mathbf{G} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_k \end{bmatrix} = \begin{bmatrix} \sum_i^k g_{1,i} u_i \\ \sum_i^k g_{2,i} u_i \\ \vdots \\ \sum_i^k g_{n,i} u_i \end{bmatrix}$$

The n-dimensional vectors that can be written in the form $\mathbf{G}\boldsymbol{u}$ are valid codewords.

47. Most n-dimensional vectors are *not* valid codewords. It is this property that allows a decoder to determine when there has been an error during transmission. This determination is made using an $(n - k) \times n$ matrix **H**, called a *parity check matrix*. Using a parity check matrix, a vector $\boldsymbol{x}$ is a valid codeword if

and only if **H***x* = **0**. If the parity check shows that **H***x* does not equal 0, then the decoder detects a transmission error (much like the "010" example in the "repeat three" code above).

48. Each of the *n* - *k* rows of the parity-check matrix **H** represents an equation that a valid codeword must satisfy. For example, consider a codeword **x** and a parity check matrix **H** given as follows:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

49. If **x** is a valid codeword, the product *H***x** must be equal to **0**, so we have:

$$\mathbf{H}\boldsymbol{x} = \begin{bmatrix} x_3 + x_4 \\ x_1 + x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \mathbf{0}$$

As this equation shows, the first row of **H** represents the constraint that $x_3 + x_4 = 0$, and the second row of **H** represents the constraint that $x_1 + x_2 = 0$. If the vector **x** satisfies both of these constraints, it is a valid codeword. In practice, parity-check matrices often have hundreds or thousands of rows, each of which represents an equation of the form $x_a + x_b + \ldots + x_z = 0$, similar to those shown in the above example. These equations are called *parity-check* equations.

2. *Repeating and Permuting as Examples of LDGM Codes*

50. As explained above, the encoding process can be represented using a *generator matrix*. This is true of all linear codes, including the "repeat," "interleave," and "accumulate" phases of the "repeat-accumulate" codes discussed above. More specifically, the "repeat" and interleave" phases can be represented using generator matrices whose entries are mostly zeros, with a relatively low proportion of 1s. Such generator matrices are called "low-density generator matrices" (LDGMs). The following is a generator matrix that repeats each information bit twice:

$$G_{\text{REPEAT2}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

51. Using this generator matrix, encoding a sequence of input bits "101010101" would result in the encoded sequence "110011001100110011" (the number of times each input bit is repeated is determined by the number of "1s" in the column corresponding to that input bit). In the 18×9 matrix above, 11.1% of

the entries are 1s, and the rest are 0s. The relative scarcity of 1s means that $\mathbf{G}_{\text{REPEAT2}}$ is a low-density generator matrix (LDGM).

52. Permutation is another linear transform that is represented by a low-density generator matrix. For example, the matrix below is a permutation matrix that will output bits in a different order than bits are input:

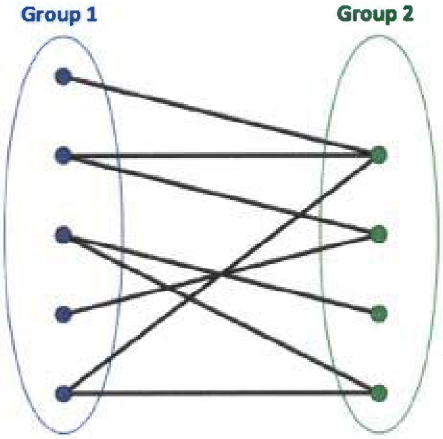
$$\mathbf{G}_{\text{SHUFFLE}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

53. Permutation matrices, like $\mathbf{G}_{\text{SHUFFLE}}$ above, have exactly one 1 per row and one 1 per column. The order of the output bits is determined by the position of each of these 1s within its row/column. The matrix above, for example, would transform input bits i_1, i_2, i_3, i_4, i_5, i_6, i_7, i_8 into the output sequence i_2, i_1, i_7, i_5, i_4, i_8, i_6, i_3. Permutation matrices are square, with dimensions $n \times n$, because the input sequence and the output sequence have the same length. With only one 1 per row/column of permutation matrices, the density of 1s is $1/n$ ($\mathbf{G}_{\text{SHUFFLE}}$, shown above, has a density of 1/8, or 12.5%). With linear codes, which often operate on blocks of more than a thousand bits at a time, a permutation matrix might comprise

0.1% 1s or fewer, and are therefore very low-density generator matrices (LDGMs).[2]

3. *Tanner Graphs*

54. Another popular mathematical representation of error-correcting codes is the "Tanner Graph." Tanner graphs are bipartite graphs wherein nodes can be divided into two groups. Every edge connects a node in one group to a node in the other (meaning no two nodes in the same group are connected by an edge). A bipartite graph is shown below:



A Simple Bipartite Graph

55. A Tanner graph includes one group of nodes called *variable nodes* that correspond to the information and parity bits, and a second group of nodes

[2] In fact, for a given *n* and *k*, there are no viable codes whose generator matrices have a lower density than repeat-codes or permute-codes. Both of these codes have matrices with exactly one 1 per row – a matrix with lower density would necessarily have at least one row without any 1s at all, which would result in *every* codeword having a 0 in a particular bit position, conveying no information about the original message.

called *check nodes* that represent constraints that parity and information bits must satisfy. In particular, when a set of variable nodes are connected to a particular check node, it means that the information/parity bits corresponding to those variable nodes must sum to 0.

56. These two mathematical descriptions of linear codes – one using matrices, one using Tanner graphs – are two different ways of describing the same thing. Matrices and Tanner graphs are two different ways of describing the same set of linear codes, in much the same way that "0.5" and "½" are two different ways of describing the same number. Every generator matrix corresponds to a Tanner graph, and vice versa.

F. Irregularity

57. Irregular LDPC codes were first introduced in a 1997 paper by Luby. *See* Luby, M. *et al.*, "Practical Loss-Resilient Codes," *STOC '97*, 1997 (Ex. 1008). The paper showed that irregular codes perform better than regular codes on certain types of noisy channels.

58. *Regular* codes are codes in which each information bit contributes to the same number of parity bits, while *irregular* codes are codes in which different information bits or groups of information bits contribute to different numbers of parity bits. This is supported by the Board's construction of "irregular" in prior proceedings. (*See*, *e.g.*, IPR2015-00060, Paper 18, p. 12.) Irregularity can also be

defined in terms of Tanner graphs. A regular code is one with a Tanner graph in which each variable node corresponding to an information bit is connected to the same number of check nodes. Irregular codes are those with Tanner graphs in which some variable nodes corresponding to information bits are connected to more check nodes than others. These two formulations of irregularity are different (and well-known) ways of describing the same concept.

59. After Luby's initial paper describing irregularity, the same team of researchers published a second paper, expanding the theory of irregularity in error-correcting codes. (*See* Luby, M. *et al.*, "Analysis of Low Density Codes and Improved Designs Using Irregular Graphs," *STOC '98*, pp. 249-59, published in 1998.) (Ex. 1009.) At the time, both of these Luby papers were widely read by coding theorists, and motivated extensive research into irregularity.

60. For example, the 1998 Luby paper was the first reference cited in a paper by Frey and MacKay titled "Irregular Turbocodes," presented at the 1999 Allerton Conference on Communications, Control, and Computing. (*See* Ex. 1002.) The second author, MacKay, was the same researcher who had rediscovered LDPC codes in 1995. In this paper, the authors applied the concept of irregularity to turbo codes by explaining how to construct irregular turbo codes in which some information bits connect to more check nodes than others. The experimental results

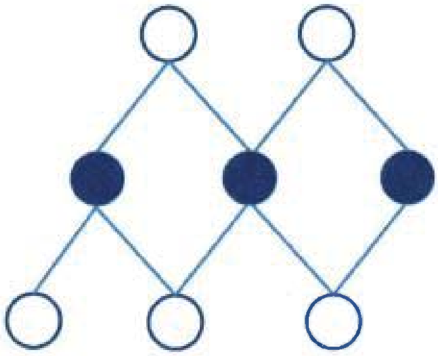
presented in the Frey paper demonstrated that these irregular turbo codes perform better than the regular turbocodes that were known in the art. (*See* Ex. 1002.)

61. By May 18, 2000, the claimed priority date of the '032 patent, it was common knowledge that the performance of error-correcting code could be improved with irregularity.

G. Message Passing and Belief Propagation Decoders

62. After the encoded bits are transmitted, they are received by a decoder, which attempts to correct any errors that occurred during transmission and reconstruct the original message. One type of decoder that was well-known in the art by the time the '032 patent was filed is called a "message passing decoder." Message passing decoding works by passing messages back and forth between variable nodes (representing information and parity bits) and check nodes (representing parity constraints, as described above) according to a Tanner graph. Using the information contained in the messages, each variable node is, over time, able to determine the true value of its corresponding bit. (*See, e.g.*, Kschischang and Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219-230, 1998; Ex. 1024.)

63. This message passing procedure can be illustrated using the example Tanner graph shown below, in which variable nodes are represented by open circles, and check nodes are represented by filled circles:



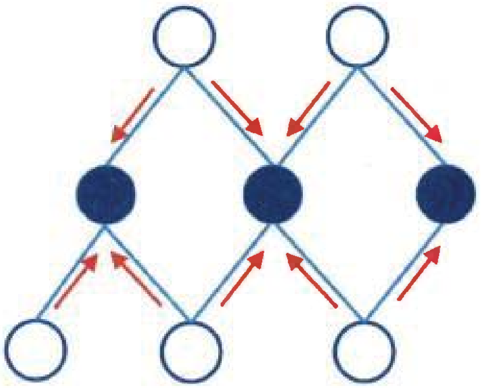
At the start of the algorithm, each variable node forms an initial estimate as to the value of its corresponding bit, based on the information provided by the demodulator. Some demodulators make a "hard" decision as to the value of each bit during demodulation (*e.g.*, a if a "hard" demodulator determines that a given bit is more likely to be a 1 than a 0, it tells the decoder that the bit is a 1). Other demodulators provide "soft" information to the decoder, representing the relative likelihood that the bit has a given value based on the signal received by the demodulator (*e.g.*, a "soft" demodulator might indicate that the probability that a given bit is equal to 1 is 0.63, or 63%). The message-passing algorithm can be used with either type of demodulator, and treats both hard- and soft-demodulation information as probabilities (*i.e.*, the message-passing algorithms treat the values received from the demodulator as probabilities of 1.0 and 0.0, where a hard-decision that a bit is a 1 is represented by a 1.0, and a hard decision that a bit is a 0 is represented by a probability of 0.0).[3] However, due to the possibility of a transmission error and other imperfections in the transmission process, many nodes can have a relatively low level of confidence that their initial estimates are correct. For example, a variable node might initially have a 60% level of confidence that the value of its corresponding bit is a 1.[4]

64. After the initial estimates have been formed, message passing proceeds in *iterations*, each of which has two phases. In the first phase of an

[3] In practice, the two possible values received from a "hard" demodulator may be treated by the decoder as probabilities that are *close* to 1.0 and 0.0 (*e.g.*, 0.95 and 0.05), recognizing that there is a non-zero probability that the demodulator may be wrong.

[4] A bit has only two possible values, *i.e.*, 1 or 0. So, if the probability that a bit has a value of 1 equals p, then the probability that the bit has a value of 0 is equal to 1-p. Thus, the probability that a bit is equal to 1 and the probability that a bit is equal to 0 can both be conveyed using a single message. For example, if a node has 60% confidence that its bit is equal to 1, then it necessarily has 40% confidence that its bit is equal to 0. A message of "60%" conveys all the information that a node needs to send in a single iteration of the decoding algorithm.

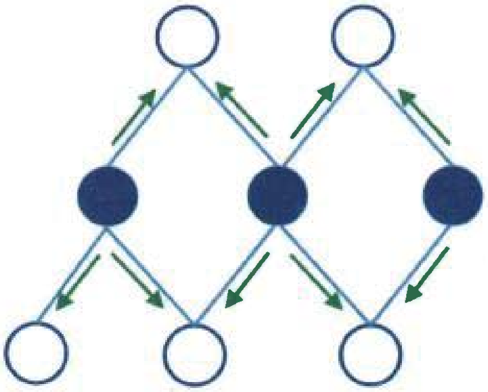
iteration, each variable node computes a message and sends its message (red arrows below) to the check nodes to which it is connected:



Phase 1: Variable Nodes Pass Messages to Check Nodes

A message sent from a variable node to a check node represents the variable node's confidence that its corresponding bit is a 1 (in the first iteration, this value is based on information received from the demodulator, but this confidence value will be refined as the algorithm proceeds, as described below.)

65. In the next phase of the iteration, each check node computes a message for each of the variable nodes to which it is connected, and sends each message (green arrows below) to the appropriate variable node:



Phase 2: Check Nodes Pass Messages to Variable Nodes

A message sent from a check node to a variable node represents the check node's confidence, based on the messages the check node has received from *other* variable nodes, that the value of the variable node's bit is 1.

66. Each variable node, based on the messages it receives from the check nodes to which is it connected, adjusts its own estimate that its value is 1. A message of 50% represents maximum uncertainty about the value of a bit. A message of 0% represents the highest possible belief that the node's associated bit has a value of zero and a message of 100% represents the highest possible belief that the node's associated bit has a value of one. For example, a variable node might start with 60% confidence that its value is 1, but if it receives messages from check nodes that indicate a lower level of confidence, it will adjust its own confidence estimate downward. At the same time, if it receives messages from check nodes that indicate a higher level of confidence, it will adjust its own confidence estimate upward.

67. Accordingly, with each iteration, each variable node refines its confidence level in the value of its associated bit. With each successive iteration of the message passing algorithm, the confidence level that each variable node has a value of 1 should either approach 100% (which indicates that the node's value is indeed 1) or 0% (which indicates that the node's value is actually 0). Once the

confidence level of every variable node is sufficiently close to 100% or 0%, the algorithm terminates.

68. There are many varieties of message passing algorithms used for decoding, and these algorithms are described using a variety of terms, including, for example, "belief propagation" decoding, the "sum-product algorithm," decoding by "probability propagation," decoding a code defined by a "Tanner graph," and decoding a code defined by a "factor graph." It would have been obvious to one of ordinary skill to use the 'message-passing' techniques described above in a conventional implementation of any of these algorithms. In particular, these techniques would comprise iteratively sending parallel messages back and forth between variable nodes and check nodes. These operations are described in several publications prior to the '032 patent (*e.g.*, "Low Density Parity Check Codes", monograph, M.I.T. Press, 1963; Ex. 1005.)

IV. OVERVIEW OF PRIMARY PRIOR ART REFERENCES

H. Ping

69. L. Ping, W. K. Leung, N. Phamdo, "Low Density Parity Check Codes with Semi-random Parity Check Matrix," *Electron. Letters*, Vol. 35, No. 1, pp. 38-39 ("Ping", Ex. 1003) was published in January 1999, more than a year before the filing of the provisional application to which the '032 patent claims priority. I understand that Ping is thus prior art to the '032 patent under 35 U.S.C. § 102(a)

and (b). Ping was not considered by the Patent Office during prosecution of the '032 patent.

70. Ping discloses an LDPC code with two stages or coding blocks: an outer LDGM coder followed by an inner coder that is an accumulator. In Ping's outer coder, a generator matrix is applied to a sequence of information bits to produce sums of information bits. In Ping's inner coder, parity bits are set equal to accumulated sums of information bits. Ping thus teaches LDPC codes that are also accumulate codes. (*See* Ex. 1003 at 38.)

71. Ping's Equation (4) illustrates that Ping's encoding process involves two distinct types of encoding operations – *i.e.*, a two-stage encoding method. This is a *logical* division of the Ping encoding process into two stages – a first stage and a second stage (note that the terms "first" and "second" here do not imply that the two stages cannot be performed in parallel; whether to perform the two stages sequentially or simultaneously are design choices that are left to the implementer). The first stage (the red box below) operates as a mod-2 summation ("Σ") of information bits and the second stage (the green box below) operates as a mod-2 summing of a previously calculated parity bit with the summation of information bits.[5] The first stage (red box) results in a summation, which is then used to

[5] The summations of Ping's first stage output parity bits and Ping's second stage also outputs parity bits.

compute the final sum (green box) in the second stage. The second stage is recursive because each successive parity bit, p_i, is calculated using the previous parity bit, p_{i-1}.

$$p_i = \boxed{p_{i-1} + \boxed{\sum_j h^d_{ij} d_j}}$$

Ex. 1003, p. 38 (annotated)

72. Ping's full Equation (4) is shown below (highlighting added). The left portion of Equation (4) shows how to compute the first parity bit, p_1, and the right portion of the equation shows how to compute successive parity bits, p_2, p_3, and so on, based on prior parity bits and information bits:

$$p_1 = \boxed{\sum_j h^d_{1j} d_j} \quad \text{and} \quad p_i = p_{i-1} + \boxed{\sum_j h^d_{ij} d_j} \pmod 2$$

73. In the above equation, the terms h^d_{ij} (which are part of the summation highlighted in red) are elements of a matrix $\mathbf{H}^{\mathbf{d}}$ described in Ping, where each particular h^d_{ij} equals either 0 or 1 and "d" represents an information bit (d_j is the j^{th} information bit). (Ex. 1003, p. 38.)

74. Ping's matrix $\mathbf{H}^{\mathbf{d}}$ is comprised of entries h^d_{ij} as follows:

$$\mathbf{H^d} = \begin{bmatrix} h^d_{1,1} & h^d_{1,2} & h^d_{1,3} & \cdots & h^d_{1,k} \\ h^d_{2,1} & h^d_{2,2} & h^d_{2,3} & \cdots & h^d_{2,k} \\ h^d_{3,1} & h^d_{3,2} & h^d_{3,3} & \cdots & h^d_{3,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ h^d_{n-k,1} & h^d_{n-k,2} & h^d_{n-k,3} & \cdots & h^d_{n-k,k} \end{bmatrix}$$

As Ping explains, $\mathbf{H^d}$ is a portion of parity-check matrix **H**, which is comprised of two parts, $\mathbf{H^d}$ and $\mathbf{H^p}$, where "$\mathbf{H} = [\mathbf{H^p}, \mathbf{H^d}]$" (Ex. 1003, p. 38.)

75. **Ping's First Stage:** The first stage in implementing Ping's Equation (4) is illustrated by red highlighting above and produces summations. That is, Ping's first stage is computing the summation terms shown in Ping's Equation (4). The equations below describe computation of those summation terms:

$$\sum_j h^d_{1j} d_j = h^d_{1,1} d_1 + h^d_{1,2} d_2 + \cdots + h^d_{1k} d_k$$
$$\sum_j h^d_{2j} d_j = h^d_{2,1} d_1 + h^d_{2,2} d_2 + \cdots + h^d_{2k} d_k$$
$$\vdots$$

76. Similarly, the equation for the i_{th} summation, $\sum_j h^d_{ij} d_j$, for all possible values of "*i*" is:

$$\sum_j h^d_{ij} d_j = h^d_{i1} d_1 + h^d_{i2} d_2 + \cdots + h^d_{ik} d_k$$

77. These equations can be conceptualized in terms of the $\mathbf{H}^{\mathbf{d}}$ matrix as follows:

- The first summation, $\sum_j h_{1j}^d d_j$, is produced by multiplying terms in the first row of the $\mathbf{H}^{\mathbf{d}}$ matrix and the information bits, and then summing those products;
- The second summation, $\sum_j h_{2j}^d d_j$, is produced by multiplying terms in the second row of the $\mathbf{H}^{\mathbf{d}}$ matrix and the information bits, and then summing those products;
- And so on.

78. The summations, $\sum_j h_{ij}^d d_j$, are not only a conceptual way of understanding Ping's equation. Rather, Ping teaches actually computing those summations. Specifically, Ping states, "[I]t requires very little memory to store $\mathbf{H}^{\mathbf{d}}$ in the encoder if $\mathbf{H}^{\mathbf{d}}$ is sparse (this can be ensured using small *t*)." One of ordinary skill would have known that Ping's encoder would not store the $\mathbf{H}^{\mathbf{d}}$ matrix in the encoder unless it were used. That $\mathbf{H}^{\mathbf{d}}$ matrix supplies the values of h_{ij}^d for the above equations for computing the summations, $\sum_j h_{ij}^d d_j$. That is the only purpose for which Ping's encoder uses the h_{ij}^d values. Therefore, one of ordinary skill would have known that the summations, $\sum_j h_{ij}^d d_j$, are actually computed as the first stage of computing the parity bits per Ping's Equation (4).

79. **Ping's Second Stage:** Once the summation, $\sum_j h_{ij}^d d_j$, has been calculated, Ping then teaches a second stage – an accumulation operation – to calculate the parity bits p_i using the value of the summation $\sum_j h_{ij}^d d_j$ and the value of the previous parity bit $p_{i\text{-}1}$. That second stage, which uses each summation as an input, is illustrated by the equations below.

$$p_1 = summation_1$$

$$p_2 = p_1 + summation_2$$

$$p_3 = p_2 + summation_3$$

$$\cdots$$

$$p_i = p_{i-1} + summation_i$$

In other words, for any value of "*i*" greater than one, $p_i = p_{i-1} + summation_i$.

80. While it is not necessary that all of the summations be calculated before beginning to compute the parity bits p_i, it is possible to implement the encoder taught by Ping this way, because each of the values of *summation*$_i$ is defined in terms of the information bits and $\mathbf{H}^\mathbf{d}$, both of which are provided as input to the encoder. The formulas $\sum_j h_{ij}^d d_j$ are *explicit* formulas that can be calculated based entirely on known values (*i.e.*, the values of the information bits and the matrix $\mathbf{H}^\mathbf{d}$).

81. The second stage of the Ping encoding operation is different it is defined by *recursive* formulas, which must be calculated in a particular order. For example, calculating the value of p_2 involves adding the second summation, *summation*$_2$, to the first parity bit p_1. In other words, to calculate p_2, we must first calculate both *summation*$_2$ and p_1. Similarly, before calculating p_3 we need to calculate both *summation*$_3$ and p_2, and so on. Finally, before calculating the final parity bit p_{n-k}, we need to have calculated *summation*$_{n-k}$ as well as all of the previous parity bits p_x, for all $x < n-k$, which in turn requires calculating *summation*$_x$ for all $x < n-k$.

82. Therefore, as the above equations show, the final operation of the second stage of Ping's encoding algorithm, calculating the final parity bit p_{n-k}, cannot be completed until all of the summations from the first stage, *summation*$_1$ through *summation*$_{n-k}$, have been calculated. Because of this, while the first and second stages do not have to be performed in sequential order, the first stage of Ping (computing the summations) must be completed before the second stage (computing the parity bits) can be finished. Thus, while the two stages of Ping can be performed in parallel, there will necessarily be a point in time near the end of the encoding process when the first ("summation") stage is complete and the second ("accumulation") stage is still underway.

83. **Ping's $\mathbf{H^d}$ Matrix**: As noted above, the individual elements of the $\mathbf{H^d}$ matrix (*i.e.*, the h_{ij}^{d} for all *i* and *j*) are all equal to either 1 or 0. This is so because Ping's codes are binary. (Ex. 1003, p. 38, "For simplicity we will consider only binary codes."; Equation (4) stating that computations are "(mod 2).") In Ping's $\mathbf{H^d}$ matrix, every column corresponds to an information bit (d_i) and every row corresponds to a summation ($\sum_j h_{ij}^{d} d_j$). This can be seen from the equations shown above for computing the summations in Ping's first step. For example, the first information bit, d_1, is always multiplied by an entry in the first column of the $\mathbf{H^d}$ matrix (*i.e.*, h_{i1}^{d} for all rows i). Similarly, the second information bit, d_2, is always multiplied by an entry in the second column of the $\mathbf{H^d}$ matrix (*i.e.*, h_{i2}^{d} for all rows *i*). Also, multiplying the *i*th row of the $\mathbf{H^d}$ matrix by the vector of information bits produces the *i*th summation ($\sum_j h_{ij}^{d} d_j$).

84. Because each column of the $\mathbf{H^d}$ matrix corresponds to a particular information bit, the number of 1s in a column determines the number of summations to which an information bit contributes. Ping refers to the number of 1s in a column as the "column weight" and uses the variable "*t*" to refer to it. (Ex. 1003, p. 38, "The resultant $\mathbf{H^d}$ has a column weight of *t*… (the weight of a vector is the number of 1s among its elements).") Ping gives an example in which "*t*=4", meaning that every information bit contributes to exactly four of the summations

($\sum_j h_{ij}^d d_j$), or four outer coder parity bits. Also, because each row of the $\mathbf{H}^{\mathrm{d}}$ matrix corresponds to a particular summation ($\sum_j h_{ij}^d d_j$), the number of 1s in a row determines the number of information bits that are summed to produce the summation. Ping refers to the number of 1s in a row as the "row weight." (Ex. 1003, p. 38, "The resultant $\mathbf{H}^{\mathrm{d}}$ has … a row weight of $kt / (n-k)$ (the weight of a vector is the number of 1s among its elements).")

I. MacKay

85. "Comparison of constructions of irregular Gallager codes," *IEEE Trans. Commun.*, Vol. 47, No. 10, pp. 1449-1454 ("MacKay"; Ex. 1002) was published in October 1999. I understand that MacKay accordingly qualifies as prior art under 35 U.S.C. § 102(a) and (b). MacKay was not considered by the Patent Office during prosecution of the '032 patent.

86. MacKay describes "[t]he excellent performance of irregular Gallager codes," and explores "ways of further enhancing these codes" (Ex. 1002, p. 1459). Specifically, MacKay investigates both regular and irregular Gallager codes with encoding algorithms that have low encoding complexity. As noted above, in 1995, David MacKay rediscovered Gallager's 1963 work on low-density parity-check (LDPC) codes. One reason why Gallager's LDPC codes had been overlooked for nearly thirty years was their perceived encoding complexity. While LDPC *parity-*

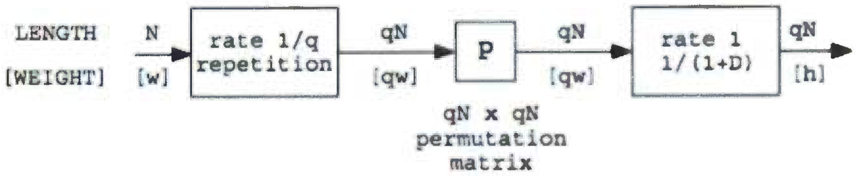
check matrices are low density, their *generator* matrices are not necessarily so, which made LDPC encoding impractical for 1960s-era computers.

87. By the 1990s, computing power had advanced significantly. Encoding Gallager codes was, by then, more computationally feasible. MacKay is therefore directed to implementing these codes using a general-purpose computer, in which a processor (or "CPU") performs calculations on values stored in "memory." (Ex. 1002, 1454.) To that end, MacKay investigates improving Gallager codes so that they can be rapidly encoded. (*See.* Ex. 1002, p. 1454, "we examine regular and irregular constructions which lend themselves to rapid encoding.") This included experimenting with irregularity, and constructing irregular codes with high performance. As MacKay concluded, "[t]he excellent performance of irregular Gallager codes is the motivation for this paper…." (Ex. 1002, p. 1449.)

J. Divsalar

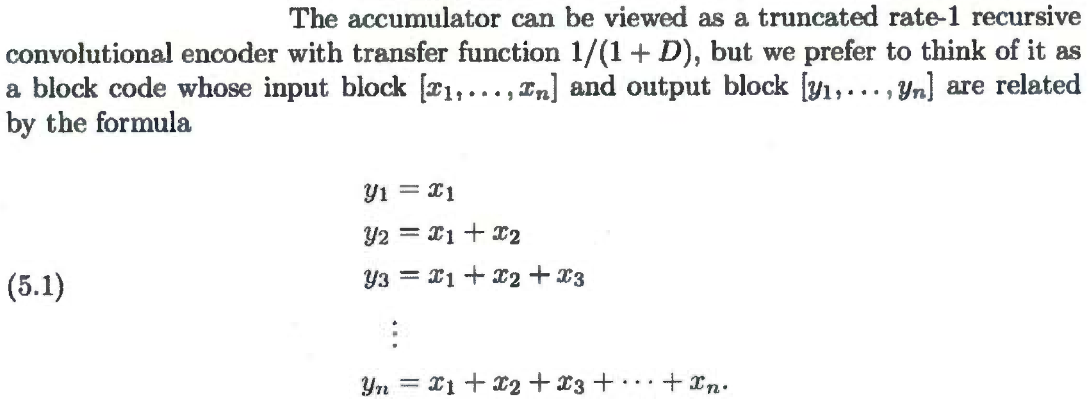
88. "Coding Theorems for 'Turbo-Like' Codes," by Divsalar *et al.*, was published in March 1999, in the *Proceedings of the 36th Allerton Conference on Communication, Control and Computing*. ("Divsalar," Ex. 1017.) As the Board found in the prior IPR proceeding for the '032 patent, Divsalar qualifies as prior art under 35 U.S.C. §102(b) because it was published "well before" the effective filing date of the '032 patent, which is May 18, 2000. (Ex. 1008, pp. 13-22; *see also* Ex. 1019, explaining that Divsalar was available to the public by March 30, 1999.)

89. Divsalar teaches "repeat and accumulate" codes, which it describes as "a simple class of rate 1/*q* serially concatenated codes where the outer code is a *q*-fold repetition code and the inner code is a rate 1 convolutional code with transfer function 1/(1 + D)." (Ex. 1017, p. 1.) Figure 3 of Divsalar shows an encoder for a repeat-accumulate code with rate *1/q*:



90. A block of *N* information bits enters the coder at the left and is provided to the repeater (labeled "rate 1/q repetition"). (*See id.*, p. 5.) The repeater duplicates each of the *N* information bits *q* times and outputs the resulting *N* × *q* repeated bits, which are then "scrambled by an interleaver of size *qN*" (*id.*, referring to the box labeled "P"). The scrambled bits are "then encoded by a rate 1 *accumulator*." (*Id.*, emphasis in original.)

91. Divsalar describes the accumulator as follows:



The accumulator can be viewed as a truncated rate-1 recursive convolutional encoder with transfer function $1/(1+D)$, but we prefer to think of it as a block code whose input block $[x_1, \ldots, x_n]$ and output block $[y_1, \ldots, y_n]$ are related by the formula

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= x_1 + x_2 \\ y_3 &= x_1 + x_2 + x_3 \\ &\vdots \\ y_n &= x_1 + x_2 + x_3 + \cdots + x_n. \end{aligned} \tag{5.1}$$

(Ex. 1017, p. 5.) Divsalar focuses on *binary* linear codes, in which data elements are bits, and thus the + symbols above are understood to denote modulo-2 addition. (Ex. 1017, p. 2, "Consider a binary linear (n,k) block code...")

92. Divsalar explains that RA codes have "very good" performance and that they can be efficiently decoded using a "message passing decoding algorithm." (Ex. 1017, pp. 9-10.)

93. Further, Divsalar introduces this paper as relevant to "turbo-like" codes, which Divsalar states include classical turbocodes, serially concatenated convolutional codes, and RA codes. "We call these systems 'turbo-like' codes and they include as special cases … the classical turbo codes," "the serial concatenation of interleaved convolutional codes," and "a special class of turbo-like codes, the repeat-and-accumulate codes." (Ex. 1017, p. 1.) Divsalar also uses turbo-like decoding methods in the decoder: "an important feature of turbo-like codes is the availability of a simple iterative, message passing decoding algorithm

that approximates ML decoding. We wrote a computer program to implement this 'turbo-like' decoding for RA codes with q = 3 (rate 1/3) and q = 4 (rate 1/4), and the results are shown in Figure 5." (*Id.*, p. 9.)

94. As explained below, Divsalar teaches all but one aspect of an IRA code: irregularity (the "**I**" in ***I***rregular ***R***epeat-***A***ccumulate). That is, Divsalar teaches regular repeat-accumulate (RA) codes rather than irregular repeat-accumulate codes as described in the '032 patent. A single modification to Divsalar – changing the repeat to irregular rather than regular – results in the irregular codes that Patent Owner claims to have invented.

K. Luby97

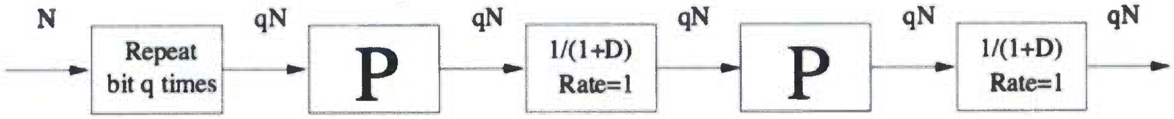
95. Luby, M. *et al.*, "Practical Loss-Resilient Codes," *STOC '97*, pp. 150-59, published in 1997 ("Luby97"; Ex. 1008) first introduced irregularity. Luby97 was published on or before June 9, 1998, and is therefore prior art to the '032 patent under 35 U.S.C. § 102. (Ex. 1008, p. 150.)

96. In addition to introducing irregularity, Luby97 describes receiving data to be encoded in a *stream* of data symbols (*e.g.*, bits), where the "*stream* of data symbols [] is partitioned and transmitted in logical units of *blocks*." (Ex. 1008, p. 150, emphasis added.) One of ordinary skill would have understood that information bits are often received in a real-time *stream*. The encoder waits until it

has received a certain number of information bits (*i.e.*, a "block"), which are then encoded all at once, producing a codeword of fixed size.

L. Pfister

97. Pfister, H. and Siegel, P., "The Serial Concatenation of Rate-1 Codes Through Uniform Random Interleavers," *37th Allerton Conf. on Comm., Control and Computing*, Monticello, Illinois, published on or before September 24, 1999 ("Pfister"; Ex. 1022) was presented by Paul Siegel at the Allerton Conference in September 1999. (*See* Ex. 1023, p. 3.) Pfister relates to codes constructed as a serial chain of rate-1 encoders and interleavers. (*See* Ex. 1022, pp. 1-2.) Pfister explicitly builds on Divsalar and introduces Divsalar as the starting point for his findings. (*Id.*, p. 4.) In particular, Pfister discusses a class of codes called "RAA (Repeat-Accumulate-Accumulate) codes," in which the outer code comprises a repeater, followed by two accumulators with Rate 1. (*Id.*, p. 1.) In particular, Pfister shows an RAA coder that uses two rate-1 accumulators:



Ex. 1022, p. 20

The boxes labeled "1/(1+D) Rate=1" represent accumulators that are chained together in series after the repeater (represented by the box labeled "Repeat bit q times"). Thus, the RAA codes of Pfister are simply Divsalar's Repeat-Accumulate codes with an extra accumulator added to the end. Pfister compares the performance of Divsalar's RA codes to RAA codes, and concludes that adding an extra accumulator improves the codes' performance. (*See id.*, p. 21.)

V. PERSON OF ORDINARY SKILL IN THE ART

98. A person of ordinary skill in the art at the time of the alleged invention of the '032 patent would have had a Ph.D. in mathematics, electrical or computer engineering, or computer science with emphasis in signal processing, communications, or coding, or a master's degree in the above area with at least three years of work experience in this field at the time of the alleged invention.

VI. OVERVIEW OF THE '032 PATENT

M. Claims

99. The '032 patent includes 23 claims, of which claims 1, 11, and 18 are independent. Claims 11-17 are challenged in the accompanying petition. Independent claim 11 is directed to an encoder that generates a sequence of parity bits from a collection of message bits in accordance with a particular Tanner graph. Independent claim 18 is directed to a device for decoding a data stream that has been encoded in accordance with the same Tanner graph. (Ex. 1001, 7:61-10:57.)

N. Summary of the Specification

100. The specification of the '032 patent is generally directed to irregular RA codes (or "IRA" codes). Figure 2 of the specification, below, shows the structure of an IRA encoder:

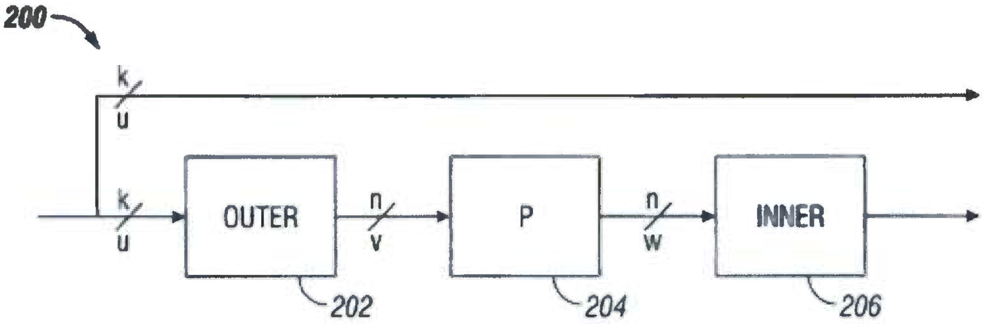


FIG. 2

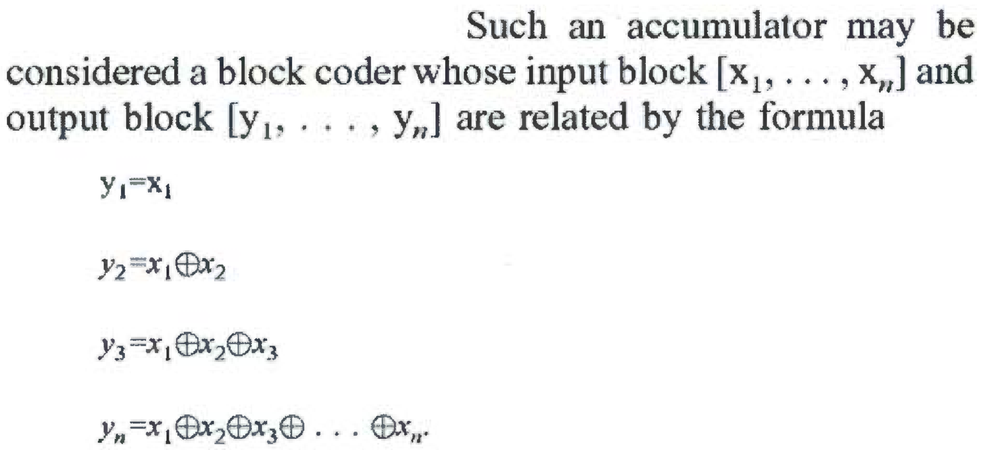
Ex. 1001, Fig. 2

101. Explaining this figure, the patent describes encoding data using an outer coder 202 connected to an inner coder 206 via an interleaver 204 (labeled "P") (Ex. 1001, 2:35-42.)

102. Outer coder 202 receives a block of information bits and duplicates each bit a given number of times, producing a sequence of repeated bits at its output. (Ex. 1001, 2:52-60.) The outer coder repeats bits irregularly – *i.e.*, it outputs more duplicates of some information bits than others. (*Id.*)

103. The repeated bits are passed to an interleaver 204, where they are scrambled. (Ex. 1001, 3:24-28.) The scrambled bits are then passed to the inner

coder 206, where they are accumulated to form parity bits. (*Id.*, 2:66-3:18.) According to the specification:



> Such an accumulator may be considered a block coder whose input block $[x_1, \ldots, x_n]$ and output block $[y_1, \ldots, y_n]$ are related by the formula
>
> $$y_1 = x_1$$
>
> $$y_2 = x_1 \oplus x_2$$
>
> $$y_3 = x_1 \oplus x_2 \oplus x_3$$
>
> $$y_n = x_1 \oplus x_2 \oplus x_3 \oplus \ldots \oplus x_n.$$

Ex. 1001 at 3:3-18

104. The specification describes systematic and non-systematic codes. In a systematic code, the encoder outputs a copy of the information bits in addition to the parity bits output by inner coder 206 (the systematic output is represented in Fig. 2. as an arrow running toward the right along the top of the figure). In a non-systematic code, the codeword output by the encoder is comprised of parity bits only.

VII. CLAIM CONSTRUCTION

O. "irregular"

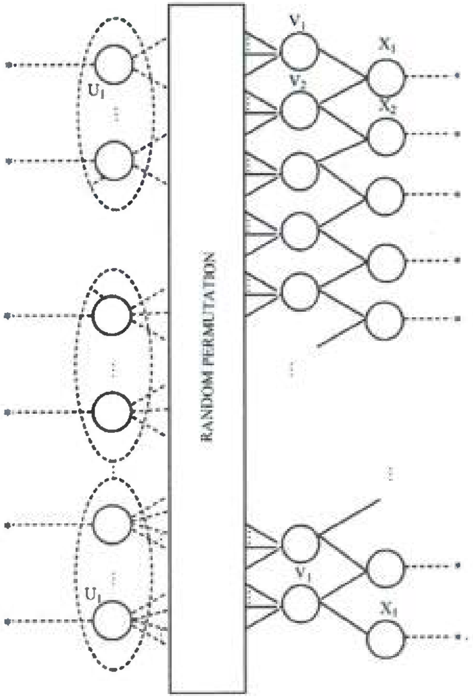
105. As explained above, in *regular* codes, each information bit contributes to the same number of parity bits. In *irregular* codes, different information bits or groups of information bits contribute to different numbers of parity bits.

106. Irregularity can also be defined in terms of Tanner graphs. A regular code has a Tanner graph in which each variable node corresponding to an information bit is connected to the same number of check nodes. An irregular code has a Tanner graph in which some variable nodes corresponding to information bits are connected to more check nodes than others.

107. These two formulations of irregularity are alternative (and well-known) ways of describing the same concept. I also understand that both are consistent with the Board's construction of "irregular" in prior proceedings. (IPR2015-00060, Paper 18, p. 11.)

P. "Tanner graph" (Claims 11, 18)

108. Claims 11 and 18 recite the following diagram, called a "Tanner graph":



109. I understand that in IPR2015-00060, the Board construed this Tanner graph term to mean: "a graph representing an [irregular repeat accumulate ("IRA")] code as a set of parity checks where every message bit is repeated, at least two different subsets of message bits are repeated a different number of times, and check nodes, randomly connected to the repeated message bits, enforce constraints that determine parity bits" along with the added constraint that "a parity bit is determined as a function of both information bits and other parity bits as shown by the configuration of nodes and edges of the Tanner graph." (IPR2015-00060, Paper 18, pp. 13-14.)

110. In the Tanner graph of independent claims 11 and 18, the parity bits are determined as a function of information bits and other parity bits, as captured

by the "added constraint" of the Board's prior construction. The '032 specification also supports the Board's prior construction, including the "added constraint." For example, per the equations shown in the '032 patent, the second parity bit (y_2) equals the sum of the first parity bit (y_1, which is itself equal to the first information bit x_1) and the second information bits (x_2); the third parity bit (y_3) equals the sum of the second parity bit (y_2) and the third information bit (x_3), and so on. (*See* Ex. 1001, 3:5-17).

IX. THE CHALLENGED CLAIMS ARE INVALID

Q. <u>Ground 1: Claims 11, 12, and 14-16 Are Obvious over Ping in View of MacKay and Further in View of Divsalar</u>

1. *Motivation to Combine Ping with MacKay and Divsalar*

(a) *Incorporating Irregularity into Ping*

111. A person of ordinary skill would have had the motivation to incorporate the irregularity disclosed in MacKay into Ping's code. Both references are directed to the same field of endeavor, the field of improving error-correcting codes, and both point toward one another by teaching how to improve the same type of code (a "Gallagher code").

112. Specifically, Ping is directed to "[l]ow density parity check codes with semi-random parity check matrix," and describes the recent "revived interest in the low density parity check (LDPC) codes originally introduced in 1962 by Gallager." (Ex. 1003, p. 38.) Ping credits this revived interest to the work of David MacKay,

which demonstrated the "very good performances" of such codes (*id.*, pp. 38, 39 (reference "[2]")), and then proceeds to describe an "LDPC code design" that "can achieve essentially the same performance as the standard LDPC encoding method with significantly reduced complexity." (*Id.*, p. 38.)

113. MacKay was published just a few months after Ping, is entitled "Comparison of Constructions of Irregular Gallager Codes," and demonstrates that "[t]he low-density parity check codes whose performance is closest to the Shannon limit are 'Gallager codes' based on irregular graphs." (Ex. 1002, p. 1449.) Like Ping, MacKay is focused on improving LDPC codes so that they can be encoded faster with less complexity. (*See id.*, p. 1450, "we examine regular and irregular constructions which lend themselves to rapid encoding").) Like Ping, MacKay measures performance improvement in terms of BER vs. E_b/N_0 performance. (*Compare* Ex. 1003, p. 39 *with* Ex. 1002, p. 1449 and Fig. 1.) And, notably, after comparing "regular" Gallager codes (like those of Ping) against "irregular Gallager codes" (*e.g.*, Ex. 1002, p. 1449 and Fig. 1), MacKay concludes that making an LDPC code irregular improves performance. (*Id.*, p. 1449, "The low-density parity check codes whose performance is closest to the Shannon limit are 'Gallager codes' based on irregular graphs."; *id.*, p. 1452, "Not only do these irregular codes outperform the regular codes, they require fewer iterations….")

114. Because MacKay teaches that irregular codes perform better than regular codes, one of ordinary skill would have had the motivation to incorporate irregularity into Ping. Doing so would have been straightforward. As explained above, Ping's outer LDPC code is regular because each column in Ping's generator matrix $\mathbf{H}^d$ contains the same number of 1s – exactly "*t*" 1s. Ping accordingly states that matrix "$\mathbf{H}^d$ has a column weight of *t* …." (Ex. 1003, p. 38.) As a result, each information bit in Ping's matrix contributes to the same number of outer coder parity bits (*i.e.*, "*t*" summations $\sum_j h_{ij}^d d_j$).

115. Like Ping, MacKay teaches matrices in which each information bit corresponds to a column, and where the weight of that column (the number of 1s contained in that column) represents the degree of the information bit. (Ex. 1002, p. 1450, "The parity check matrix of a code can be viewed as defining a bipartite graph with 'bit' vertices corresponding to the columns and 'check' vertices corresponding to the rows. Each nonzero entry in the matrix corresponds to an edge connecting a bit to a check. The profile specifies the degrees of the vertices in this graph.") Using "column weight" language identical to Ping, MacKay also teaches how to make LDPC matrices "irregular" by implementing a "*nonuniform* weight per column." (Ex. 1002, p. 1449, "The best known binary Gallager codes are *irregular* codes whose parity check matrices have *nonuniform* weight per column.") MacKay describes, as an example, a matrix that "has columns of weight

9 and of weight 3…." (*Id.*, p. 1451.) As a result, in MacKay's irregular matrix, different information bits contribute to different numbers of parity bits.

116. It would have been straightforward for a person of ordinary skill to change Ping's generator $\mathbf{H}^{d}$ matrix such that not all columns had the same weight – for example, setting some columns to weight 9 and others to weight 3, as taught by MacKay. (Ex. 1002, p. 1451.) This change would result in some information bits contributing to more outer LDPC parity bits than others, and would have made Ping's outer LDPC code irregular. This would have been an easy way for one of ordinary skill to incorporate the irregularity disclosed by MacKay into Ping. Additionally, MacKay's teaching that the best performing LDPC codes are irregular would also have made this modification obvious (and desirable) to try. (Ex. 1002, pp. 1449, 1454, "The excellent performance of irregular Gallager codes is the motivation for this paper….")

117. Also, one of ordinary skill would have had the motivation to combine Ping and MacKay because Ping's $\mathbf{H}^{d}$ matrix is described in terms very similar to the terms MacKay uses to describe irregularity with matrices. Because Ping's Equation (4) uses the $\mathbf{H}^{d}$ matrix to produce parity bits from information bits, it is a "generator matrix." (Ex. 1003, p. 38.) Ping's $\mathbf{H}^{d}$ matrix is also part of Ping's "parity check" matrix **H**. (*Id.*, "Accordingly, we decompose **H** into $\mathbf{H}=[\mathbf{H}^{p}, \mathbf{H}^{d}]$.") MacKay describes its irregularity in terms of the "parity check" matrix. (Ex. 1002,

p. 1450.) As explained below, MacKay's parity-check matrix and Ping's matrix $\mathbf{H}^{\mathbf{d}}$ both establish the same thing. Ping's Equation (4) provides a formula for calculating p_i:

$$p_i = p_{i-1} + \sum_j h_{ij}^d d_j$$

118. Adding p_i to both sides of the equation (mod 2) allows Ping's equation to be written as the following equation (the mod-2 sum of a bit with itself is 0):

$$0 = p_i + p_{i-1} + \sum_j h_{ij}^d d_j$$

119. MacKay's parity-check matrix also establishes constraints, as do all parity-check matrices, that sums of bits must equal zero. Accordingly, one of ordinary skill would have understood how to use MacKay's statements about parity-check matrices in Ping and would have had the motivation to do so.

(b) *Using Repetition in Ping*

120. As explained above, in the first stage of Ping's two-stage encoding procedure (Ping's outer LDPC coder), each information bit contributes to t summations $\sum_j h_{ij}^d d_j$, and Ping explicitly discloses an example in which "t=4." (Ex. 1003, p. 39.) One of ordinary skill would have understood that an obvious way to implement having each information bit contribute to t summations $\sum_j h_{ij}^d d_j$

would be to repeat each information bit *t* times. However, even if Ping standing alone is not understood to teach, or render obvious, repeating information bits, doing so would have been obvious in view of Divsalar's explicit teaching of repeating bits. Indeed, the use of a repeater in implementing an outer coder was common in the prior art, as evidenced by, for example, Frey and MacKay's "Irregular Turbocodes" (Ex. 1010, showing repetition of bits at the bottom of Figure 2) and Pfister (Ex. 1022, showing outer coder block labeled "Repeat bit q times" in Figure 1, reproduced above).

121. A person of ordinary skill would have had the motivation to use Divsalar's repetition in Ping (as modified with the irregularity of MacKay, as described above). Specifically, one of ordinary skill would have recognized that the repetition techniques taught by Divsalar provide a natural way to implement the codes of Ping. In one such implementation, each of Ping's information bits d_j is "repeated" once for each summation $\sum_j h_{ij}^d d_j$ in which it appears. This generates t "repeats" of each information bit, which are then used to compute the summations $\sum_j h_{ij}^d d_j$ required by Equation (4) of Ping, described above.

122. One of ordinary skill would have been further motivated to implement Ping using the repeater of Divsalar because this implementation would be both cost-effective and easy to build. A repeater is a simple component that can be built cheaply and would require relatively little space in an encoding circuit.

123. Additionally, both Ping and Divsalar relate to linear error-correcting codes and, specifically, both references describe encoders that perform an accumulation as the final stage of the encoding process. (*See* Ex. 1003, p. 38; *see also* Ex. 1017, p. 5.) Both references also teach encoding procedures that are recursive (Ex. 1017, p. 5, "The accumulator can be viewed as a truncated rate-1 *recursive* convolutional encoder …," emphasis added; *see also* Ex. 1003, p. 38, providing a recursive formula (Equation (4)) for generating the parity bits **p**.) These additional similarities between Ping and Divsalar would have provided additional motivation to one of ordinary skill to implement Ping's code using a repeater, as taught by Divsalar, because it would have been obvious from Divsalar that a repeater would work for Ping's purpose (including to produce the outer coder summations that are then further encoded by Ping's inner coder).

124. For these reasons, and those discussed more below, a person of ordinary skill would have had the motivation to incorporate the irregularity of MacKay and the repetition of Divsalar into the encoding methods of Ping. Also, as demonstrated below, the similarity and combinability of Ping, MacKay, and Divsalar are evidenced by the number of claim limitations that are simultaneously taught by these references.

2. *Claim 11*

125. As explained below, claim 11 is rendered obvious by the combination of Ping in view of MacKay and Divsalar.

(a) *"A device comprising ..."*

126. Ping, MacKay, and Divsalar each teach the preamble. Ping refers to an "encoder" comprising "memory," which a person of ordinary skill would have understood to be a "device." (Ex. 1003, p. 38.) Similarly, MacKay teaches that "both the *memory requirements and the CPU requirements at the encoder* of our fast-encoding codes are substantially smaller." (Ex. 1002, p. 1453, emphasis added). Divsalar likewise teaches an encoding device, as shown in Figure 3, reproduced above. (Ex. 1017, p. 5.)

(b) *"an encoder configured to receive a collection of message bits and encode the message bits to generate a collection of parity bits"*

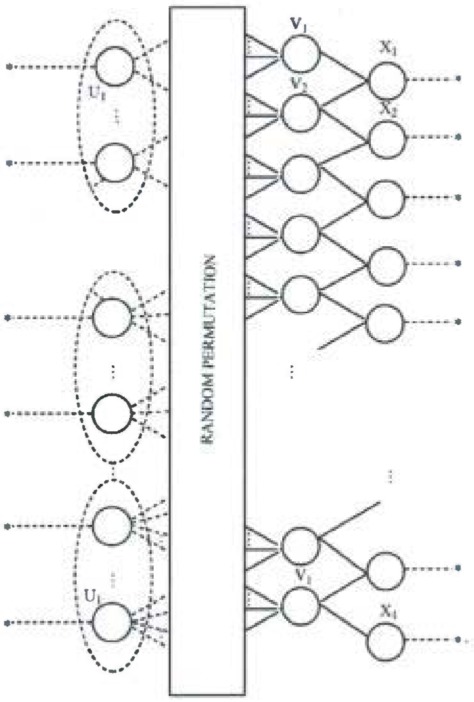
127. Ping teaches this limitation. As described in detail above, Ping provides equations from which the parity bits "$\mathbf{p} = \{p_i\}$ can easily be calculated from a given $\mathbf{d} = \{d_i\}$" (where "**p** and **d** contain the parity and information bits, respectively"). (Ex. 1003, p. 38.)

128. The term "message bits" is synonymous with "information bits." One of ordinary skill would have understood that Ping's "information bits" **d** are a "collection of message bits," and that, per Ping's Equation (4), those information

bits are encoded to generate “a collection of parity bits” **p** as the claim requires. (Ex. 1003, p. 38.)

(c) *“Tanner graph”*

129. Claim 11 requires that the parity bits “be generated in accordance with the following Tanner graph:”



(Ex. 1001, 8:65-9:35.)

130. As noted above, the Board previously construed the Tanner graph limitation to mean: “a graph representing an [irregular repeat accumulate (“IRA”)] code as a set of parity checks where every message bit is repeated, at least two different subsets of message bits are repeated a different number of times, and check nodes, randomly connected to the repeated message bits, enforce constraints

that determine parity bits" along with the added constraint that "a parity bit is determined as a function of both information bits and other parity bits as shown by the configuration of nodes and edges of the Tanner graph." (IPR2015-00060, Paper 18, pp. 13-14.) This limitation accordingly requires three elements:

(i) "a graph representing an [irregular repeat accumulate ("IRA")] code as a set of parity checks where every message bit is repeated, at least two different subsets of message bits are repeated a different number of times"

(ii) "check nodes, randomly connected to the repeated message bits, enforce constraints that determine parity bits"; and

(iii) "a parity bit is determined as a function of both information bits and other parity bits as shown by the configuration of nodes and edges of the Tanner graph."

Ping, in view of MacKay and Divsalar, teaches all three elements of the claimed Tanner graph, as the Board has previously construed this term. For ease of explanation, we demonstrate how Ping in view of MacKay and Divsalar teaches these elements in an order different than how they appear in the Board's construction.

(iii) "a parity bit is determined as a function of both information bits and other parity bits as shown by the configuration of nodes and edges of the Tanner graph"

131. As explained above, Ping teaches an LDPC-accumulate code in which the value of one parity bit is used to calculate the next parity bit:

$$\boxed{p_1 = \sum_j h^d_{1j} d_j} \quad \text{and} \quad \boxed{p_i = p_{i-1} + \sum_j h^d_{ij} d_j} \quad \boxed{(\text{mod } 2)} \qquad \textbf{(4)}$$

Ping: Equation (4) (Ex. 1003, p. 38)

132. Once the initial parity bit p_1 is calculated (blue box), Ping's Equation (4) can be used to calculate the second parity bit (orange box), which can be used to calculate the third parity bit (orange box), and so on. To make this clear, we expand Ping's Equation (4) below to show calculation of individual parity bits:

$$\begin{aligned}
p_2 &= p_1 + \sum_j h^d_{2j} d_j \\
p_3 &= p_2 + \sum_j h^d_{3j} d_j \\
p_4 &= p_3 + \sum_j h^d_{4j} d_j \\
&\vdots \\
p_i &= p_{i-1} + \sum_j h^d_{ij} d_j \\
&\vdots
\end{aligned}$$

Accordingly, each parity bit p_i (for all i greater than or equal to two) depends on the previous parity bit p_{i-1} and one of the summations $\sum_j h^d_{ij} d_j$. Similarly, each parity bit p_i (for all i greater than or equal to three) depends on the two previous parity bits p_{i-1} and p_{i-2} and one of the summations $\sum_j h_{ij}^d d_j$ (*e.g.*, p_3 depends upon p_2, and p_2 in turn depends on p_1).

133. Also, as explained in the Ping overview above, each of the summations ($\sum_j h_{ij}^d d_j$) (meaning the intermediate bits produced by the outer LDPC coder) is a sum of information bits, d_j. That is, each summation ($\sum_j h_{ij}^d d_j$) equals the sum of information bits d_j for which the corresponding values of h_{ij}^d are equal to one. (Ex. 1003, p. 38.)

134. Accordingly, Ping teaches that "a parity bit [such as the third parity bit, p_3] is determined as a function of both information bits and other parity bits" as required by part (iii) of the Board's prior construction of the Tanner graph.

(i) "a graph representing an [irregular repeat accumulate ("IRA")] code as a set of parity checks where every message bit is repeated, at least two different subsets of message bits are repeated a different number of times"

135. This element of the Board's prior Tanner graph construction has two requirements. First, it requires the code to be an irregular repeat accumulate (or "IRA") code. Second, when represented as a graph, the code must have a set of parity checks where every message bit is repeated, and at least two different subsets of message bits are repeated a different number of times. Ping, in view of MacKay and Divsalar, teaches both requirements.

(a) Ping, in View of MacKay and Divsalar, Teaches an Irregular Repeat Accumulate (IRA) Code

- *"Irregular"*

136. Ping's outer LDPC coder is regular. That is, in Ping each information bit contributes to exactly "*t*" outer coder parity bits. (Ex. 1003, p. 38.) However, as explained in the motivation to combine section above, one of ordinary skill would have had the motivation to use MacKay's irregularity in Ping, thus making Ping's outer LDPC encoder irregular. For example, it would have been obvious to use MacKay's specific example where some information bits contribute to nine parity bits and others contribute to three parity bits. (Ex. 1002, p. 1451.) Doing so produces an irregular code as recited in this limitation.

- *"Repeat"*

137. As explained in the motivation to combine section above, one of ordinary skill would have had the motivation to use Divsalar's repetition in Ping's outer LDPC encoder. Using Divsalar's repetition and MacKay's irregularity makes Ping's outer LDGM encoder an irregular-repeat encoder in which different subsets of message bits are repeated a different number of times. For example, using MacKay's specific "9/3" example, some bits would be repeated nine times and others would be repeated three times. (Ex. 1002, p. 1451.)

- *"Accumulate"*

138. As explained in the overview of Ping above, Ping's outer coder is an LDPC encoder and Ping's inner coder is an accumulator. Ping's inner coder is an

accumulator because each successive inner coder parity bit, p_i, is calculated by adding something to the previous parity bit, p_{i-1}. This is shown by Ping's Equation (4):

$$\begin{aligned}
p_1 &= \sum_j h^d_{1j} d_j \\
p_2 &= p_1 + \sum_j h^d_{1j} d_j = \sum_j h^d_{1j} d_j + \sum_j h^d_{2j} d_j \\
p_3 &= p_2 + \sum_j h^d_{3j} d_j = \sum_j h^d_{1j} d_j + \sum_j h^d_{2j} d_j + \sum_j h^d_{3j} d_j \\
&\vdots
\end{aligned}$$

Ping's teaching therefore matches how the '032 patent itself describes accumulation. (*See* Ex. 1001, 3:3-19.)

139. Therefore, the combination of Ping in view of Divsalar and MacKay teaches "an [irregular repeat accumulate ("IRA")] code where every message bit is repeated" and "at least two different subsets of message bits are repeated a different number of times," as required by the Board's prior construction.

(b) Ping, in View of MacKay and Divsalar, Teaches A "Graph Representing … Parity Checks …" as Construed by the Board

140. As explained above in the background, a Tanner graph includes one group of nodes (called variable nodes) that correspond to the information and parity bits, and a second group of nodes (called check nodes) that represent constraints that the information and parity bits must satisfy. Specifically, when a

set of variable nodes are connected to a particular check node, it means that the information and parity bits corresponding to those variable nodes must sum to zero.

141. A single code can be described either in terms of matrices or as a Tanner graph. MacKay, for example, describes its code in terms of both matrices and Tanner graphs. (Ex. 1002, p. 1450, "The parity check matrix of a code can be viewed as defining a bipartite [Tanner] graph with 'bit' vertices corresponding to the columns and 'check' vertices corresponding to the rows. Each nonzero entry in the matrix corresponds to an edge connecting a bit to a check.") These two mathematical descriptions of linear codes – one using matrices, and one using Tanner graphs – are merely two ways of describing the same thing – in much the same way that "0.5" and "½" describe the same number.

142. One of ordinary skill would therefore have understood the IRA code taught by Ping, in view of MacKay and Divsalar, to correspond to a Tanner graph with variable nodes and check nodes exactly as set forth in the Board's prior construction. Specifically, Ping's information bits ("**d**") and parity bits ("**p**") are variable nodes when represented as a graph. Similarly, Ping's parity check equations – each instance of Equation (4) for each of the parity bits $p_2, p_3, p_4 \ldots p_i$, as discussed above – constitute check nodes when represented as a graphic. Ping, whether alone or in view of Divisalar, teaches repeating each message bit (meaning

the information bits "**d**") for the reasons explained in §IX.A.1 above. And, as further explained in §IX.A.1 above, Ping in view of MacKay teaches at least two different subsets of message bits are repeated a different number of times. Applying MacKay's teaching to have "columns of weight 9 and of weight 3," for example, would result in two subsets of message bits: a first subset containing message bits that are repeated nine times and a second subset containing message bits that are repeated three times. (Ex. 1002, p. 1451.)

143. Ping, in view of MacKay and Divsalar, therefore teaches all aspects of element (i) of the Board's prior construction.

(ii) "check nodes, randomly connected to the repeated message bits, enforce constraints that determine parity bits"

144. This final element of the Board's construction has two requirements. First, the recited "check nodes" must "enforce constraints that determine parity bits." Second, the "check nodes" must also be "randomly connected to the repeated message bits." Ping teaches both requirements.

(a) Ping teaches "check nodes" that "enforce constraints that determine parity bits"

145. As is true of parity checks generally, Ping's parity checks enforce constraints that determine parity bits. Specifically, the constraint that Ping's parity checks enforce is the requirement that a particular collection of information bits and parity bits sum to zero. This is taught by Ping's Equation (4). As explained

for part (i) of the Tanner graph construction above, and taking the second parity bit as an example, Ping's Equation (4) can be written as follows:

$$p_i = p_{i-1} + \sum_j h^d_{ij} d_j$$

146. Adding p_i to both sides of the equation (mod 2) allows Ping's equation to be written as the following equation (the mod-2 sum of a bit with itself is 0):

$$0 = p_i + p_{i-1} + \sum_j h^d_{ij} d_j$$

147. Ping generates each parity bit (for example, p_2, p_3, p_4, … p_i) by performing an instance of Equation (4). In each instance, the constraint is the same: each particular collection of information bits and parity bits must sum to zero.

148. Additionally, each instance of Equation (4) constitutes a "check node" in the language of the Board's prior construction. As explained above for part (i) of the Tanner graph construction, a person of ordinary skill would have understood that the identical error-correcting code can be expressed using either linear algebra or a Tanner graph (just like "½" and "0.5" both express the same number). Ping's Equation (4), using linear algebraic notation, constitutes a "check node" when expressed in the form of a Tanner graph. Ping's check nodes accordingly "enforce

constraints that determine parity bits" precisely as required by the Board's prior construction of this term.

(b) Ping's "check nodes" are "randomly connected to the repeated message bits."

149. As explained above for part (i) of the Tanner graph construction, using Divsalar's repetition in Ping means that the check nodes require a collection of repeated information bits and parity bits to sum to zero, as illustrated by the equation below:

$$0 = p_i + p_{i-1} + \sum_j h_{ij}^d d_j$$

In this equation, repeated message bits are represented by the variable d_j. Because the repeated message bits are a variable in each instance of Ping's parity check Equation (4), Ping's check nodes are connected to the repeated message bits.

150. Additionally, Ping teaches that these connections are "random." As explained above, Ping teaches placing 1s into $\mathbf{H}^{\mathbf{d}}$ "randomly," thereby ensuring that the information bits d_j where $h_{ij}^d = 1$, are selected randomly. Because the 1s in Ping's $\mathbf{H}^{\mathbf{d}}$ matrix identify the information bits that contribute to each parity bit, those 1s represent connections in a Tanner graph. Ping teaches placing 1s into $\mathbf{H}^{\mathbf{d}}$ "randomly," thereby ensuring that the information bits d_j (where $h_{ij}^d = 1$) contributing to each parity bit p_i are chosen at random. (Ex. 1003, p. 38, "In each

sub-block $\mathbf{H}^{di}$, i=1, 2…t, we *randomly create* exactly one element 1 per column and $kt/(n\text{-}k)$ 1s per row." (emphasis added).) Because of this, in an implementation of Ping that uses repetition, as taught by, for example, Divsalar, this random selection satisfies the requirement of the claim that the check nodes are "randomly connected to the repeated message bits."

151. To summarize, the bits that must sum to zero for a given parity check are represented in a Tanner graph by connections between those bits and a check node. The "random" connections between the repeated information bits and the check nodes in the claimed Tanner graph correspond to the random placement of 1s in Ping's matrix $\mathbf{H}^{\mathbf{d}}$. Therefore, the codes of Ping meet the "randomly connected" requirement of claim 11. Also, when Ping is implemented using repetition, as taught by, for example, Divsalar, the check nodes are "randomly connected" to "repeated information bits."

152. Accordingly, as shown above, the combination of Ping in view of MacKay and Divsalar teaches every limitation of claim 11.

3. *Claim 12*

153. Claim 12 depends from claim 11, and adds "wherein the encoder is configured to generate the collection of parity bits as if a number of inputs into nodes v_i was not constant."

154. In claim 11, the nodes of the claimed Tanner graph shown immediately to the right of the "random permutation" box are labeled "*v*." The '032 patent explains, with reference to Figure 3, that these "*v*" nodes are check nodes. (Ex. 1001, 3:37-39, "The Tanner graph includes two kinds of nodes: variable nodes (open circles) and *check nodes (filled circles)*," emphasis added.) Each check node, v_i, determines the value of a parity bit such that the parity bits and information bits to which the check node is connected sum to zero. (*Id.*, 4:4-6.) As explained above, this limitation of claim 11 was obvious over the combination of Ping in view of MacKay and Divsalar.

155. MacKay teaches claim 12's additional "not constant" limitation as set forth below.

(a) *The number of inputs to a check node "v" is determined by the row weight of a matrix*

156. MacKay describes codes in terms of both parity-check matrices and Tanner graphs, stating: "The parity check matrix of a code can be viewed as defining a bipartite [Tanner] graph with 'bit' vertices corresponding to the columns and 'check' vertices corresponding to the rows. Each nonzero entry in the matrix corresponds to an edge connecting a bit to a check." (Ex. 1002, p. 1449.) Each row of MacKay's parity check matrix accordingly corresponds to a check node in a Tanner graph, and each nonzero entry ("1") in a row represents an input to the check node. (*Id.*)

157. Claim 12's requirement that the number of inputs into the check nodes is not constant means – when stated in terms of a parity check matrix – that the number of "1"s in rows of the parity check matrix is not constant. That is, a parity check matrix where some rows have more ones than other rows teaches the "not constant" limitation of claim 12.

(b) *In MacKay, the number of inputs to each check node "v" is not constant*

158. MacKay describes "irregular codes" that have parity check matrices "with both nonuniform ***weight per row*** and nonuniform weight per column." (Ex. 1002, p. 1449, emphasis added.) "Row weight" refers to the number of "1"s in a row. MacKay therefore teaches a parity check matrix where the number of "1"s in each row is not constant, corresponding to the claimed Tanner graph where the number of inputs into each check node is not constant. In particular, MacKay describes designing a code such that some rows of the parity check matrix have more ones than others. (Ex. 1002, pp. 1449-50, "First, we select a *profile* that describes the desired number of columns of each weight ***and the desired number of rows of each weight***," emphasis added).

159. Additionally, as explained above, each row of the parity check matrix corresponds to a check node in the Tanner graph. The check nodes of a Tanner graph each establish the constraint that the bits connected to the check node must sum to zero. Similarly, each row of the parity check matrix establishes a constraint

that the information bits and parity check bits appearing in that row must sum to zero, as shown by Ping's Equation (4):

$$p_i = p_{i-1} + \sum_j h^d_{ij} d_j$$

160. Adding p_i to both sides of the equation (mod 2) allows Ping's equation to be written as the following equation (the mod-2 sum of a bit with itself is 0):

$$0 = p_i + p_{i-1} + \sum_j h^d_{ij} d_j$$

161. The rows of MacKay's parity-check matrix accordingly also establish constraints, as do all parity-check matrices, that sums of bits must equal zero.

(c) *Ping in view of MacKay and Divsalar renders claim 12 obvious*

162. As demonstrated for claim 11 above, one of ordinary skill would have had the motivation to use MacKay's teaching of "nonuniform column weight" in the combination of Ping, MacKay, and Divsalar, thereby making the column weight of Ping's matrix $\mathbf{H}^{\mathbf{d}}$ nonuniform. (*See supra* at §IX.A.1.) Varying the column weight of Ping's $\mathbf{H}^{\mathbf{d}}$ matrix would result in some information bits contributing to more summations ($\sum_j h^d_{ij} d_j$) than other information bits.

163. Moreover, modifying Ping's $\mathbf{H}^d$ matrix such that it had both nonuniform ***row*** weights and nonuniform column weights would have been straightforward for a person of ordinary skill. Implementing nonuniform column weights in Ping's $\mathbf{H}^d$ matrix has been discussed in conjunction with claim 11 above. Additionally varying the ***row*** weight of Ping's $\mathbf{H}^d$ matrix would make the number of inputs into the check nodes variable, as required by claim 12.

164. For example, Ping's first two outer LDPC coder bits could be computed according to the following equations.[6]

$$\sum_j h^d_{1j} d_j = d_1 + d_2$$

$$\sum_j h^d_{2j} d_j = d_3 + d_4$$

165. Continuing this example, if the row weight of Ping's first row were five and the row weight of Ping's second row were two, the equations for the first two summations might reduce to, *e.g.*:

[6] Because the 1s in Ping's $\mathbf{H}^d$ matrix are randomly distributed, any particular summation is unlikely to equal the sum of these two particular information bits. But this example illustrates the concept that each summation equals the sum of two particular information bits.

$$\sum_j h^d_{1j} d_j = d_1 + d_2 + d_4 + d_8 + d_9$$

$$\sum_j h^d_{2j} d_j = d_3 + d_4$$

In this simple example, MacKay's teaching of nonuniform row weight is incorporated into Ping by changing three 0s to 1s in the first row of Ping's $\mathbf{H^d}$ matrix, yielding one row (the first row) with weight 5, and one row (the second row) with weight two. Modifying Ping's $\mathbf{H^d}$ matrix in this way, or other ways, to use MacKay's nonuniform row weight would have been obvious.

166. In summary, claim 12 would have been obvious to one of ordinary skill over the combination of Ping, Divsalar, and MacKay. As explained above, the additional limitation imposed by claim 12, meaning generating "a collection of parity bits as if a number of inputs into nodes v_i was not constant," is satisfied by a parity check matrix in which the number of 1s per row – the "row weight" – is not uniform. As explained more above, MacKay teaches parity check matrices with nonuniform row weight. Therefore, this teaching of MacKay satisfies the additional limitations recited by claim 12.

4. *Claim 14*

167. Claim 14 depends from claim 12, and it also requires that "the accumulator comprises a recursive convolutional coder." (Ex. 1001, 9:46-47.) Divsalar teaches an accumulator that "comprises a recursive convolutional coder."

(Ex. 1017, p. 5, "The accumulator can be viewed as a truncated rate-1 recursive convolutional encoder with transfer function 1/(1 +*D*)…"); *see also id.*, Figure 3.)

168. Ping's accumulator is also a recursive convolutional encoder. Ping's Equation (4) shows how to compute each parity bit, p_i, as a sum of the prior parity bit, p_{i-1}, and a summation ($\sum_j h^d_{ij} d_j$), and can be written as:

$$\begin{aligned} p_1 &= \sum_j h^d_{1j} d_j \\ p_2 &= p_1 + \sum_j h^d_{2j} d_j \\ p_3 &= p_2 + \sum_j h^d_{3j} d_j \\ &\vdots \\ p_i &= p_{i-1} + \sum_j h^d_{ij} d_j \\ &\vdots \end{aligned}$$

Ping therefore teaches the same accumulation as the '032 patent. (Ex. 1001, 3:3-18.)

169. As explained above, the combination of Ping, Divsalar, and MacKay teaches every claim limitation of claim *12*, from which claim 14 depends. Therefore, claim 14 is obvious over Ping in view of Divsalar and MacKay.

5. *Claim 15*

170. Claim 15 depends from claim 14, and also requires that "the recursive convolutional coder comprises a truncated rate-1 recursive convolutional coder."

(Ex. 1001, 9:48-50.) As explained above, the combination of Ping in view of Divsalar and MacKay teaches every claim limitation of claim 14. Divsalar also teaches that the accumulator is a "truncated rate-1 recursive convolutional coder." (Ex. 1017, p. 5.) Therefore, claim 15 is obvious over Ping in view of Divsalar and MacKay.

6. *Claim 16*

171. Claim 16 depends from claim 14, and also requires that "the recursive convolutional coder has a transfer function of 1/(l+*D*)." (Ex. 1001, 9:51-52.) As explained above, the combination of Ping in view of Divsalar and MacKay teaches every claim limitation of claim 14. Divsalar also teaches that the recursive convolutional coder has a transfer function of 1/(l+*D*). (Ex. 1017, p. 5, Figure 3.) In particular, in Figure 3, Divsalar depicts its accumulator as having a transfer function of "1/(1+D)" and Divsalar states in the text that "[t]he accumulator can be viewed as a truncated rate-1 recursive convolutional encoder with transfer function 1/(1+D)…." (Ex. 1017, p. 5.) Therefore, claim 16 is obvious over Ping in view of Divsalar and MacKay.

R. Ground 2: Claims 13 and 18-23 Are Obvious over Ping in View of MacKay, Divsalar, and Luby97

1. *Motivation to Combine Ping, MacKay, Divsalar and Luby97*

(a) *Irregularity and Repetition*

172. As explained above for Ground 1, one of ordinary skill would have had the motivation to use MacKay's irregularity and Divsalar's repetition in Ping.

(b) *Streams*

173. Ping discloses block codes in which the codeword "**c** = [**p**, **d**], where **p** and **d** contain the parity and information bits, respectively." (Ex. 1003, p. 38.) One of ordinary skill would have understood that it was obvious for Ping to receive information bits **d** in "a source data stream" and to output its parity bits **p** in a "transmission data stream." That is, it would have been obvious for information bits arriving in a stream to be collected into blocks and encoded per Ping's Equation (4) and for Ping's codewords, which contain parity bits, to be output sequentially in a stream.

174. Even if Ping is not understood to teach encoding bits in a "stream" standing alone, it would have been obvious to use "streams" as taught by Luby97 in Ping. As explained above, Luby97 describes receiving data to be encoded in a *stream* of data symbols (which could be, for example, bits), where the "*stream* of data symbols [] is partitioned and transmitted in logical units of *blocks*." (Ex. 1008, p. 150, emphasis added.) One of ordinary skill in the art would have understood that, in practice, information bits are often received in a real-time *stream*. The encoder waits until it has received a certain number of information bits (meaning a

"block" of information bits), which are then encoded all at once, producing a codeword of fixed size.

175. Coders that receive "streams" of bits that comprise one or more "blocks" of data were common, and would have been familiar to a person of ordinary skill. To the extent that "*streams*" were not obvious in view of Ping alone, it would have been obvious to use Luby97's teaching of "streams" in Ping to make the encoder capable of receiving and processing "streams" as opposed to just blocks. Accordingly, it would have been obvious to a person of ordinary skill to incorporate Luby97's "stream" teachings into the encoder of Ping (modified with the irregularity of MacKay and the repeater of Divsalar, as described above).

2. *Claim 13*

176. Claim 13 depends from claim 11 and also requires that the encoder comprise: (a) "a low-density generator matrix (LDGM) coder configured to perform an irregular repeat on message bits having a first sequence in a source data stream to output a random sequence of *repeats* of the message bits"; and (b) "an accumulator configured to XOR sum in linear sequential fashion a predecessor parity bit and '*a*' bits of the random sequence of repeats of the message bits." (Ex. 1001, 9:39-45.)

177. As explained above, the combination of Ping in view of MacKay and Divsalar discloses every claim limitation of claim 11. Ping in view of MacKay, Divsalar and Luby97 teach the additional limitations of claim 13.

(a) *"a low-density generator matrix (LDGM) coder configured to perform an irregular repeat on message bits having a first sequence in a source data stream to output a random sequence of repeats of the message bits"*

178. This limitation imposes three requirements: (i) a low-density generator matrix (LDGM) coder configured to perform an irregular repeat on message bits; (ii) the LDGM coder outputs a random sequence of repeats of the message bits; and (iii) the message bits have a first sequence in a source data stream. The prior art teaches each of these requirements.

(1) *a low-density generator matrix (LDGM) coder configured to perform an irregular repeat on message bits*

179. Ping's matrix $\mathbf{H}^d$ is a low-density generator matrix. Ping's matrix $\mathbf{H}^d$ has *n-k* ("n" minus "k") rows and *k* columns, where *k* is the number of information bits in a codeword, *n* is the total number of bits in a codeword, and *t* is a "preset integer." (Ex. 1003, p. 38.) Ping discloses an example in which $k = 30{,}000$ and the code rate, *k*/*n*, is 1/3. (*Id.*, p. 39 (Fig. 1 caption).) In a rate 1/3 code, three bits are output for every single information bit. So, in this example, the length of the codeword, *n*, is 90,000 bits. This corresponds to a generator matrix $\mathbf{H}^d$ with

60,000 rows, 30,000 columns, and therefore a total of 1,800,000,000 entries. Every column contains exactly "*t*" ones, and in Ping's example, "t=4." (*Id.*) Therefore, of the 1,800,000,000 entries, only 240,000 of the entries are ones (t(n□k)=4(60,000)=240,000). The ratio of ones to total entries in the $\mathbf{H}^{\mathbf{d}}$ matrix is 240,000/1,800,000,000=0.00013, meaning that only 0.013% of the entries are ones. Accordingly, $\mathbf{H}^{\mathbf{d}}$ is a very low density matrix.

180. Additionally, as explained in the context of claim 11, it would have been obvious to one of ordinary skill to implement Ping by using Divsalar's technique of repeating bits and MacKay's nonuniform column weights, which would result in some message bits being repeated more than others – thereby satisfying the "irregular repeat" limitation of claim 13.

(2) *the LDGM coder outputs a random sequence of repeats of the message bits*

181. As explained above in the context of claim 11, the positions of 1s in Ping's generator matrix $\mathbf{H}^{\mathbf{d}}$ are "randomly" distributed: "In each sub-block $\mathbf{H}^{\mathbf{d}i}$, i = 1, 2 … t, we <u>*randomly*</u> create exactly one element 1 per column and $kt/(n\text{-}k)$ 1s per row." (Ex. 1003, p. 38, emphasis added.) Because the 1s are randomly distributed in Ping's generator matrix $\mathbf{H}^{\mathbf{d}}$, the matrix would output "a random sequence" of bits, as claim 13 requires.

(3) *the message bits have a first sequence in a source data stream*

182. Luby97 describes receiving data to be encoded in a *stream* of data symbols (which could be, for example, bits), where the "*stream* of data symbols [] is partitioned and transmitted in logical units of *blocks*." (Ex. 1008, p. 150, emphasis added.) One of ordinary skill would have understood that, in practice, message bits are often received in a real-time *stream*. The encoder waits until it has received a certain number of message bits (a "block" of message bits), which are then encoded all at once, producing a codeword of fixed size.

183. Coders that receive "streams" of message bits were common, and would have been familiar to a person of ordinary skill. To the extent that "streams" were not obvious in view of Ping alone, it would have been obvious to use Luby97's "stream" teaching in Ping to make the encoder capable of receiving and processing "streams" as opposed to just blocks. Accordingly, it would have been obvious to a person of ordinary skill to incorporate the "stream" teachings of Luby97 into the encoder of Ping.

(b) *"an accumulator configured to XOR sum in linear sequential fashion a predecessor parity bit and 'a' bits of the random sequence of repeats of the message bits"*

184. As discussed above for claim 11, Ping generates the parity bits by summing the claimed "predecessor parity bit and 'a' bits of the random sequence of bits." That is, as shown in Equation (4), Ping computes the parity bit p_i by summing the claimed "predecessor parity bit," (p_{i-1}) and a summation $\sum_j h_{ij}^d d_j$. (Ex.

1003, p. 38.) As also explained for claim 11, this process uses "accumulation," or "an accumulator."

185. Also, as shown on the right side of Ping's Equation (4), the computations are performed "mod 2." Mod 2 (or modulo 2) arithmetic performs an "XOR" sum (meaning for both "mod 2" and "XOR," the following are true: $0\oplus0=0$; $0\oplus1=1$; $1\oplus0=1$; and $1\oplus1=0$).

186. Also, as explained above for claim 12, the number of information bits that are summed to produce each summation ($\sum_j h_{ij}^d d_j$) corresponds to the number of 1s in each row of Ping's matrix $\mathbf{H^d}$ and Ping discloses an example in which exactly two information bits are summed to produce each summation ($\sum_j h_{ij}^d d_j$). Accordingly, in this example, the claimed value of "a" is the constant two. Therefore, each summation, $\sum_j h_{ij}^d d_j$, is a sum of the claimed "'*a'* [2] bits of the random sequence of bits."

187. Also, as explained above in subsection (a) for claim 13 (*supra* at §IX.B.2), (i) the information bits used in the summation $\sum_j h_{ij}^d d_j$ are selected from a "random sequence" of information bits, and (ii) it would have been obvious to one of ordinary skill to implement Ping by using Divsalar's "repeating" technique such that the information bits used in the summation $\sum_j h_{ij}^d d_j$ were "repeats" of the information bits.

188. Finally, Ping computes the parity bits in "linear sequential fashion" as claimed. As shown by Ping's Equation (4), Ping computes the parity bit $p_{(i-1)}$ before computing the parity bit p_i, accordingly computing the parity bits in "sequential fashion" as claimed. Also, Ping's Equation (4) is "linear" as claimed. The only operations in Ping's Equation (4) are multiplication and addition, and both of those are linear operations.

3. *Claim 18*

189. As explained below, one limitation at a time, claim 18 is rendered obvious by Ping in view of MacKay, Divsalar, and Luby97.

(a) *"A device comprising:*

190. As explained above for claim 11, Ping, MacKay, and Divsalar each teach this preamble.

(b) *"a message passing decoder configured to decode a received data stream that includes a collection of parity bits"*

191. Divsalar teaches this limitation. Divsalar teaches that "an important feature of turbo-like codes is the availability of a simple iterative, *message passing decoding* algorithm that approximates ML decoding. We wrote a computer program to implement this 'turbo-like' decoding for RA codes with $q = 3$ (rate 1/3) and $q = 4$ (rate 1/4), and the results are shown in Figure 5." (Ex. 1017, p. 9, emphasis added.) One of ordinary skill would understand that "message passing

decoding" is performed by a device called a "decoder" that receives "a collection of parity bits."

192. MacKay also teaches this limitation. MacKay explains that Gallager codes "are asymptotically good and can be practically decoded with Gallager's sum-product algorithm." (Ex. 1002, p. 1449.) For additional information about the "sum-product" algorithm, MacKay directs the reader to "R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, 'Turbo decoding as an instance of Pearl's "*belief propagation' algorithm*."''" (Ex. 1002, p. 1454, reference [9], emphasis added). As explained above (*supra* at §IV.G), a person of ordinary skill would have recognized that the "sum-product algorithm" and "belief propagation" are types of a "message passing decoder," as required by claim 18. The fact that a "belief propagation" decoder is a type of "message passing decoder" is also confirmed by claim 22 of the '032 patent, which depends from claim 18 and further specifies that the "message passing decoder" is a "belief propagation decoder."

193. Luby97 also teaches this limitation. As Luby97 explains, "We have also implemented error-correcting codes that use our novel graph constructions and decode with *belief propogation* [sic] techniques. Our experiments with these constructions yield dramatic improvements in the error recovery rate. We will report these results in a separate paper." (Ex. 1008, p. 158, emphasis added.)

194. While Ping does not specify a particular decoding algorithm, it explains that "[t]he decoding algorithm follows that in [2]", where reference "[2]" is "MacKay, D. J. C, and Neal, R. M. 'Near Shannon Limit Performance of Low Density Parity Check Codes', *Electron. Lett.*, 1997." (Ex. 1016, p. 39.) The 1996 version of the 1997 MacKay reference cited by Ping describes "an approximate belief propagation algorithm." (Ex. 1002_ORIG, p. 1646.) In view of this citation, and the fact that one of ordinary skill would understand message passing algorithms to be a standard technique for decoding linear error-correcting codes, it would have been obvious to use the "message passing decoders" taught by Divsalar, MacKay, and Luby97 to decode the LDPC-accumulate codes of Ping.

195. Additionally, it would have been obvious to use a stream-based decoder to decode the LDPC-accumulate codes of Ping. As discussed above, Luby97 describes receiving data to be encoded in a *stream* of data symbols (which could be, for example, bits), where the "*stream* of data symbols [] is partitioned and transmitted in logical units of blocks." (Ex. 1008, p. 150, emphasis added.)

196. While this passage discusses transmitting data in "*logical* units of blocks," a person of ordinary skill in the art would understand that the bits in each logical block are not actually transmitted all at once, but rather sequentially, as a stream of encoded data bits, that are only grouped "logically" into units of blocks. Accordingly, they are also received sequentially by the decoder as a stream; a

person of ordinary skill in the art would understand that groups of codeword bits are not received simultaneously in blocks, but sequentially as a stream.

197. Even if a person of ordinary skill were to treat the codeword bits in each block as if they had been received simultaneously, they would have recognized that a sequence of blocks with a defined order, such as the logical blocks of encoded symbols transmitted by the Luby97 encoder, constitutes a stream.

198. Indeed, like the encoder of Luby97, the decoding algorithm described in the '032 specification also treats the codeword as a "logical block" containing parity bits. As the '032 patent explains, "***[b]efore decoding***, the messages $m(v\rightarrow u)$ and $m(u\rightarrow v)$ are initialized to be zero, and $m_0(u)$ is initialized to be the log-likelihood ratio based on the channel received information. If the channel is memoryless, i.e., each channel output only relies on its input, and y is the output of the channel code bit u, then $m_0(u)=\log(p(u=0|y)/p(u=1|y))$. ***After this initialization, the decoding process may run*** …" (Ex. 1001, 5:64-6:3 (emphasis added).) Here, the variable "u" represents a variable node of the Tanner graph, and the value "$m_0(u)$" represents the likelihood that the codeword bit corresponding to node u is a 0, based on "the channel received information." (Ex. 1001, 5:64-6:3.) As this passage explains, the information "$m_0(u)$" corresponding to all of the codeword

bits "u" must be received in order to complete the initialization phase, only after which the decoding process may begin.

199. Thus, just as the encoder described in Luby97 *transmits* codeword bits in "logical blocks," the preferred embodiment of claim 18 *receives* parity bits in "logical blocks," waiting until every parity bit in the block has been received before decoding can begin. A person of ordinary skill in the art would have understood that while each codeword is treated by the encoder and the decoder as a "logical block" containing parity bits, the bits in this "logical block" are not transmitted over the channel all at once. Instead, the bits in the codeword are transmitted sequentially over time, as a "stream" of parity bits.

200. It would therefore have been obvious to a person of ordinary skill to decode the LDPC-accumulate codes of Ping with a message passing decoder that is compatible with these teachings of Luby97 (especially in view of Luby97's teaching of a message passing decoder, as explained above). Specifically, it would have been obvious to one of ordinary skill to use a decoder that can receive encoded bits in a stream where the encoder that encoded those bits output them in a stream, as taught by Luby97.

(c) *"the message passing decoder comprising two or more check/variable nodes operating in parallel to receive messages from neighboring check/variable nodes and send updated messages to the neighboring variable/check nodes"*

201. As discussed above in the background section, a person of ordinary skill in the art would understand that "two or more check/variable nodes operating in parallel to receive messages from neighboring checking/variable nodes and send updated messages to the neighboring variable/check nodes" are features that would be obvious in view of any conventional message-passing decoder, including the message-passing decoders taught by Divsalar, MacKay, and Luby97 (and described in the earlier, 1996 version of the 1997 MacKay paper cited by Ping).

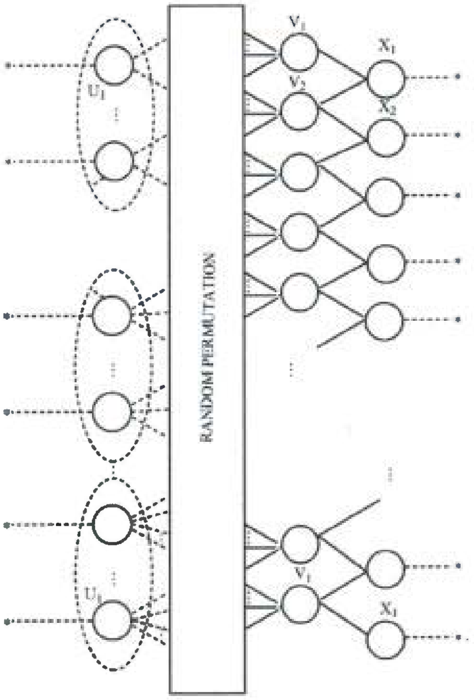
202. In particular, as discussed above, message passing decoding works by passing messages back and forth between variable nodes (representing information and parity bits) and check nodes (representing parity constraints, as described above) according to a Tanner graph. Using the information contained in the messages, each variable node is, over time, able to determine the true value of its corresponding bit. (*See, e.g.*, Kschischang and Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 2, pp. 219-30, 1998; Ex. 1024.)

203. As also explained above, there are a number of varieties of message-passing decoding algorithms, including, for example, "belief propagation" decoding, the "sum-product algorithm," decoding by "probability propagation," decoding a code defined by a "Tanner graph," and decoding a code defined by a "factor graph." It would have been obvious to one of ordinary skill to use the

"message-passing" techniques described above in a conventional implementation of any of these algorithms, all of which involve "two or more check/variable nodes operating in parallel to receive messages from neighboring checking/variable nodes and send updated messages to the neighboring variable/check nodes," as required by claim 18.

(d) *Tanner Graph*

204. Claim 18 requires that the "the message passing decoder [be] configured to decode the received data stream that has been encoded in accordance with the following Tanner graph:"



(Ex. 1001, 10:5-45.)

205. As explained above for claim 11, the combination of Ping, MacKay and Divsalar teaches encoding in accordance with this claimed graph. Additionally, as explained above, the combination of Ping, MacKay, Divsalar, and Luby97 teaches a message passing decoder configured to decode a received data stream that has been encoded in accordance with this claimed graph.

4. *Claim 19*

206. Claim 19 depends from claim 18, and also requires that "the message passing decoder [be] configured to decode the received data stream that includes the message bits." (Ex. 1001, 10:43-45.)

207. As explained above for claim 18, the combination of Ping in view of MacKay, Divsalar, and Luby97 teaches every claim limitation of claim 18, including "the message passing decoder [being] configured to decode the received data stream." Claim 19 adds that the received data stream "includes the message bits," which requires the code to be systematic. First, as explained above for claim 11, Ping teaches a systematic code in which the codeword includes the information bits. (Ex. 1003, p. 38 "Decompose the codeword **c** as c = [**p**, **d**]....") Also, Luby97 teaches transmitting a stream of data symbols (*e.g.*, bits) to a decoder, where the transmitted symbols constitute a "*systematic* code" consisting of "*message* symbols" and "*check* symbols." (Ex. 1008, p. 150, "Consider an application that sends a real-time stream of data symbols that is partitioned and transmitted in

logical units of blocks."; *id.*, p. 151, "We assume that the receiver knows the position of each received symbol within the stream of all encoding symbols."; *id.*, p. 152, "*systematic* code"; *see also id.*, p. 159, "We have also implemented error-correcting codes that use our novel graph constructions and decode with belief propogation techniques.".) Therefore, Ping and Luby97 both teach that the received data stream "includes the message bits," as claim 19 requires.

5. *Claim 20*

208. Claim 20 depends from claim 18, and also requires that "the message passing decoder [be] configured to decode the received data stream as if a number of inputs into nodes v_i was not constant. (Ex. 1001, 10:46-48.)

209. As explained above for claim 12, the combination of Ping in view of MacKay, and Divsalar teaches the limitation added by claim 20.

6. *Claim 21*

210. Claim 21 depends from claim 18, and also requires that "the message passing decoder [be] configured to decode in linear time at rates that approach a capacity of a channel." (Ex. 1001, 10:49-51.)

211. As explained above, the combination of Ping in view of MacKay, Divsalar, and Luby97 teaches every claim limitation of claim 18, including a "message passing decoder." Luby97 teaches the additional limitation of claim 21 that the message passing decoder be "configured to decode in linear time at rates

that approach a capacity of a channel." Luby97 designed codes "with simple and fast *linear-time* encoding and *decoding algorithms* that can transmit over lossy channels *at rates extremely close to capacity*." (Ex. 1008, p. 158, emphasis added; *see also id.*, p. 151, "We obtain very fast linear-time algorithms by transmitting just below channel capacity.") One of ordinary skill in the art would understand Luby97's "at rates extremely close to capacity" to mean "at rates that approach a capacity of a channel," as claim 21 requires. Accordingly, Luby97 discloses "the message passing decoder [being] configured to decode in linear time at rates that approach a capacity of a channel."

7. *Claim 22*

212. Claim 22 depends from claim 18, and also requires that "the message passing decoder comprises[] a belief propagation decoder." (Ex. 1001, 10:53-54.)

213. As explained above, the combination of Ping in view of MacKay, Divsalar, and Luby97 teaches every claim limitation of claim 18, including a "message passing decoder." Both MacKay and Luby97 teach the additional limitation of claim 22 that the message passing decoder comprise a "belief propagation decoder." MacKay explains that Gallager codes "are asymptotically good and can be practically decoded with Gallager's sum-product algorithm." (Ex. 1002, p. 1449.) For additional information about the "sum-product" algorithm, MacKay cites to "R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, Turbo

decoding as an instance of Pearl's *'belief propagation' algorithm*." (Ex. 1002, p. 1454, reference [9], emphasis added.)

214. Luby97 also teaches a belief propagation decoder: "We have also implemented error-correcting codes that use our novel graph constructions and decode with *belief propogation* [sic] techniques. Our experiments with these constructions yield dramatic improvements in the error recovery rate. We will report these results in a separate paper." (Ex. 1008, p. 158, emphasis added.)

8. *Claim 23*

215. Claim 23 depends from claim 18, and also requires that the message passing decoder be "configured to decode the received data stream without the message bits." (Ex. 1001, 10:56-57.)

216. Divsalar describes Figure 3 as introducing "a class of turbo-like codes." (Ex. 1017, p. 5.) In Divsalar's "turbo-like" coding system, no direct path exists between the input and output of the encoder, which means the input bits (information bits) are not being transmitted. As explained in the background section, such a coding system, where information bits are not being transmitted as part of the codeword, is a non-systematic code and both systematic and non-systematic codes were known for decades prior to the '032 patent. (Ex. 1020, p. 12, "Also, a BCI can be systematic or non-systematic."; *id.*, pp. 12-13, Figures 2.1 and 2.2 captions showing both systematic and non-systematic encoders; *see also* Ex.

1001, 4:19-25.) Therefore, it would have been obvious to make the code of Ping in view of MacKay, Divsalar and Luby97 a non-systematic code in view of the long-known nature of systematic codes and in view of Divsalar.

S. Ground 3: Claim 17 Is Obvious over Ping in View of MacKay, Divsalar, and Pfister

217. Claim 17 depends from claim 12, and also requires "a second accumulator configured to determine a second sequence of parity bits that defines a second condition that constrains the random sequence of repeats of the message bits." (Ex. 1001, 9:53-56.) As explained above, Ping in view of MacKay and Divsalar teaches every claim limitation of claim 12. Pfister discloses the additional limitation of claim 17.

218. As described above in the technology background section, Pfister teaches "RAA codes," (where "RAA" stands for "Repeat-Accumulate-Accumulate"). (Ex. 1022, p. 1.) The RAA codes of Pfister are simply Divsalar's Repeat-Accumulate codes with an extra accumulator added to the end. (Ex. 1022, p. 20.)

219. A person of ordinary skill would have had the motivation to incorporate the two-accumulator design of Pfister into Divsalar and/or Ping, both of which use a single accumulator stage. The references are directed to the same field (the field of error-correcting codes), and are specifically directed to error-

correcting codes that use rate-1 inner coders, *i.e.*, accumulators. (*See, e.g.*, Ex. 1017, Figure 3; *see also* Ex. 1022, pp. 1-2.)

220. Additionally, incorporating a second accumulator into Divsalar's or Ping's codes would have been obvious to one of ordinary skill. As explained above, Pfister's RAA codes are merely RA codes with an additional accumulator added to the end of the encoding chain in series. (*See* Ex. 1022, p. 20.) Indeed, Pfister compares the performance of Divsalar's RA codes to RAA codes, and concludes that adding an extra accumulator improves the performance of Divsalar's RA codes, providing an explicit suggestion to combine the extra accumulator of Pfister with the RA codes of Divsalar. (*See id.*, p. 21.) The same logic also suggests replacing Ping's single accumulator with two accumulator stages.

221. As explained above, the combination of Ping, Divsalsar and MacKay discloses every limitation of claim 12, from which claim 17 depends. Claim 17 is therefore obvious over Ping, Divsalar and MacKay, and further in view of Pfister.

X. AVAILABILITY FOR CROSS-EXAMINATION

222. In signing this declaration, I recognize that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I also recognize that I may be subject to cross-examination in the case and that cross-examination will take place

within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

XI. RIGHT TO SUPPLEMENT

223. I reserve the right to supplement my opinions in the future to respond to any arguments that the Patent Owner raises and to take into account new information as it becomes available to me.

XII. JURAT

224. I declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the full knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States code.

Dated: January 19, 2017

James A Davis

James A. Davis, Ph.D.
Richmond, Virginia

APPENDIX A

Curriculum Vita

James A. Davis 2214 Grainmill Court Richmond, VA 23233 (804) 289-8094 (W) (804) 747-5858 (H) jdavis@richmond.edu

EDUCATION:

Ph.D. Mathematics, University of Virginia, August, 1987. Dissertation: "Difference sets in abelian 2-groups"; Advisor: Dr. Harold Ward.

B.S. Mathematics with Honors, Magna cum laude, Lafayette College, May, 1983.

EMPLOYMENT:

Professor, University of Richmond, 2001-Present.

Associate Professor, University of Richmond, 1994-2001.

Assistant Professor, University of Richmond, 1988-1994.

Visiting Assistant Professor, Lafayette College, August 1987-1988.

Teaching Assistant, University of Virginia, August 1983-1987.

Summer Actuary Intern, Provident Mutual Life Insurance Company, Summer of 1982.

DESCRIPTION OF RESEARCH:

Use of algebraic and number theoretic tools to construct combinatorial structures in Design Theory and Coding Theory.

UNDERGRADUATE HONORS THESES DIRECTED:

Cameron-Libeler line classes and partial difference sets, Uthaipon Tantipongpipat, 2016.

Partitioning groups with difference sets, Becca Funke, 2016.

Nonexistence of non quadratic Kerdock sets in 6 variables, John Clikeman, 2016.

Difference sets in nonabelian groups of order 256, Taylor Applebaum, 2013.

Boolean functions with high second order nonlinearity, Cornel Bodea, 2010.

Crossover property of the nonperiodic autocorrelation function of quaternary sequences, Michael Pohl, 2007.

Relative difference sets in 2-groups: a group cohomological viewpoint, Brian Wyman, 2005.

Difference sets, symmetric designs, and coding theory: a look at the incidence matrices for several difference sets, Kate Nieswand, 2000.

On some new constructions of difference sets, Sarah Spence, 1997.

Generalizations of the mod4 construction of the Nordstrom-Robinson Code, Tim Frey, 1995.

Topics in cyclotomic and quadratic fields, Pam Mellinger, 1993.

A new view of McFarland difference sets, John Polhill, 1993.

SUMMER RESEARCH PROJECTS DIRECTED: Almost difference sets, David Clayton, University of Richmond Undergraduate Research Committee, Summer 2016.

Kerdock sets and constructions of bent functions via covering EBSs, John Clikeman and David Clayton, Summer 2016.

Covering EBS bent functions in 8 variables, Uthaipon Tantipongpipat, John Clikeman, Becca Funke, Tedi Aliaj, Sam Bell, David Clayton, Sami Malik, Alexandru Pana, University of Richmond Undergraduate Research Committee, Spider Research Fund, and Virginia Foundation for Independent Colleges as sponsors, Summer 2015.

Bent functions in 6 and 8 variables, Uthaipon Tantipongpipat, John Clikeman, Tedi Aliaj, Sam Bell, Hayu Gelaw, University of Richmond Undergraduate Research Committee, Spider Research Fund, and Virginia Foundation for Independent Colleges as sponsors, Summer 2014.

Bent functions and difference sets, Gavin McGrew, Uthaipon Tantipongpitat, Becca Funke, John Clikeman, Kevin Erb, Erin Geoghan, Richuan Hu, Ningxi Wei, Thomas Meyer, Tahseen Rabbani, Daniel Habibi, Apollo Jain, University of Richmond Undergraduate Research Committee and HHMI as sponsors, Summer 2013.

Boolean functions with high second order nonlinearity, Gavin McGrew, Dayton Steele, Katherine Marsten, Xi He, Shalleetta Hicks, Taryn Smith, Zach Davis, Timmy Albright, Irina Kiseeva, Kirstin Ladas, Alex Martin, Kate Pitchford, Summer 2011. Sponsored by the National Science Foundation, the Grainger Foundation, and the University of Richmond Undergraduate Research Committee.

Boolean functions with high second order nonlinearity, Gavin McGrew, Alex Martin, Kate Pitchford, Katherine Utz, Francesco Spadaro, Josh Wilson, Taryn Smith, summer 2010. Sponsored by the National Science Foundation and the University of Richmond Undergraduate Research Committee.

Sequences with the shared autocorrelation property, Corneliu Bodea, Calina Copos, Matt Der, David Oneal, Summer 2008. Sponsored by the National Science Foundation.

Sequences with the shared autocorrelation property, Calina Copos, Matt Der, David Oneal, and Lauren Vanson, Summer 2007. Sponsored by the National Science Foundation.

Reversible difference sets in 2-groups, Ryan Daut, Summer 2005. Sponsored by the National Security Agency.

A search for new partial difference sets, Peter Magliaro and Adam Weaver, Summer 2003. Sponsored by the National Security Agency.

Partial difference sets in new groups, Tiffany Broadbent and Ed Kenney, Summer 2002. Sponsored by the University of Richmond Undergraduate Research Committee, Collaborative Research Grant.

Difference sets in 2-groups and their associated codes,Mohammed Abouzaid and Jamie Bigelow, Summer, 2000. Sponsored by Hewlett-Packard.

Construction of New Asymptotic Classes of Binary Sequences Based on Existing Asymptotic Classes, Anthony Kirilusha and Ganesh Narayanaswamy, summer 1999. Sponsored by Hewlett-Packard.

Intersections of Golay codes with higher order Kerdock codes, Mohammed Abouzaid and Nick Gurski, summer 1999. Sponsored by Hewlett-Packard.

An examination of codewords with optimal merit factor, Mike Cammarano and Anthony Kirilusha, summer, 1998. Sponsored by Hewlett-Packard.

Octary codewords with peak power 3.2^m, Katie Nieswand and Kara Wagner, summer, 1998. Sponsored by Hewlett-Packard.

Using the Quantum Computer to break and Elliptic Curve Cryptosystem, Jodie Eicher and Yaw Opoku, summer, 1997. This won the Student Research Symposium Natural Science Award at the University of Richmond in spring, 1998. Sponsored by Hewlett-Packard.

Power control in Golay cosets, Mike Cammarano and Meredith Walker, summer, 1997. Sponsored by Hewlett-Packard.

Coding theory over Galois Rings, Sarah Spence and Brian McKeever, Summer, 1995. This won a poster presentation at the annual Joint Math meetings of the American Mathematical Society and the Mathematics Association of America in January, 1996.

PH.D. DISSERTATION DIRECTED: Constructing partial difference sets using Galois Rings, John Polhill, University of Virginia, 1999.

GRANTS:

National Security Agency, research award, $55,000, 2015-2017.

National Science Foundation, LURE, 2008-2011, $100,000

National Science Foundation, LURE, 2007-2011, $1,500,000.

National Security Agency Grant, salary, travel expenses, and support for undergraduate research, January 2003-December 2005.

University of Richmond Undergraduate Research Commmittee Collaborative Grant, salary and travel for me and two students, summer 2002.

Hewlett-Packard Grant for Leave of Absence, academic year 2000-2001, paying salary, moving expenses, cost of living expenses, and benefits.

Hewlett-Packard Grant for undergraduate summer research institute, May 1997-August 2000, to fund 4 students full time working on applied problems.

Hewlett-Packard sabbatical Grant, August 1995-August 1996, funded move to England, half salary, other expenses for the academic year.

National Security Agency Young Investigator Grant, June 1995-June 1997, summer salary and travel.

University of Richmond Summer Research Grant, Summer 1994, salary.

National Security Agency Young Investigator Grant, June 1992-June 1994, summer salary and travel.

Hewlett-Packard equipment grant (50 graphing calculators and supporting equipment), summer 1993.

University of Richmond Summer Research Grant, Summer 1989, salary.

CONSULTING:

Consultant for Hughes Corp, May-June 2014, patent dispute vs. Cal Tech.

Testifying Expert for Apple in Patent dispute, October 2011-June 2012. Deposed on April 10, 2012, testified at trial on June 12, 2012. Wrote Markman report (claims constructions), Invalidity report, Rebuttal non-infringement report.

Consultant for Hewlett-Packard, August 1996-August 2000. Worked on patent preparation, standards body submissions, basic research.

GRANTED PATENTS:

K.K. Smith, J. Jedwab, J.A. Davis, D. Banks and S.R. Wyatt, System for error correction coding and decoding, U.S. Patent No. 7418644, 26 Aug 2008.

D. Banks, J. Davis, and J. Jedwab, Identifying uncorrectable codewords in a Reed-Solomon decoder for errors and erasures, US Patent No. 7278086, 2 October 2007.

J. Davis, K. Eldredge, J. Jedwab, G. Seroussi, and K. Smith, MRAM with controller, US Patent No. 7266732, 4 September 2007.

S.M. Brandenberger, T. Munden, J. Jedwab, J. Davis, and D. Banks, System and method for configuring a solid-state storage device with error correction coding, U.S. Patent No. 7,210,077, 24 Apr 2007.

J. Jedwab, J.A. Davis and G. Seroussi, Method for error correction decoding in a magnetoresistive Solid-state storage device, U.S. Patent No. 7,149,949, 12 Dec 2006.

J.A. Davis, J. Jedwab, S. Morley, K.G. Paterson, F.A. Perner, K.K. Smith and S.R. Wyatt, Manufacturing Test for a fault tolerant magnetoresistive solid-state storage device, U.S. Patent No. 7,149,948, 12 Dec 2006.

J. Jedwab, J.A. Davis, K.G. Paterson and G. Seroussi, Manufacturing test for a fault tolerant magnetoresistive solid-state storage device, U.S. Patent No. 7107508, 12 Sep 2006.

J.A. Davis, J. Jedwab, S. Morley, and K.G. Paterson, Magnetoresistive solid-state storage device and Data storage methods for use therewith, U.S. Patent No. 7,107,507, 12 Sep 2006.

J.A. Davis, J. Jedwab, D. McIntyre, K.G. Paterson, F. Perner, G. Seroussi, K.K. Smith, and S. Wyatt, Error correction coding and decoding in a solid-state storage device, U.S. Patent No. 7036068, 25 April 2006.

F.A. Lerner, J. Jedwab, J.A. Davis, D. McIntyre, D. Banks, S. Wyatt, and K.K. Smith, Magnetic memory including a sense result category between logic states, U.S. Patent No. 6,999,366, 14 Feb 2006.

J.A. Davis, J. Jedwab, K.G. Paterson, and G. Seroussi, Method for error correction decoding in an MRAM device (historical erasures), U.S. Patent No. 6,990,622, 24 Jan 2006.
J.A. Davis, J. Jedwab, K.G. Paterson, G. Seroussi, and K.K. Smith, Data storage method for use in a magnetoresistive solid-state storage device, U.S. Patent No. 6,981,196, 27 Dec 2005.

J.A. Davis and J. Jedwab, Allocation of sparing resources in a magnetoresistive solid-state storage device, U.S. Patent No. 6,973,604, 6 Dec 2005.

J. Jedwab and J.A. Davis, Methods and apparatus for encoding and decoding data, European patent application No. 97 3 10252.8-2209; US patent 6,373,859, granted 16 April 2002.

J. Jedwab, J.A. Davis, and K.G. Paterson, Method and apparatus for decoding data, US patent 6,487,258, granted 26 November 2002.

M. Mowbray, J.A. Davis, K.G. Paterson, and S.E. Crouch, System and Method for Transmitting Data, US patent 6,119, 263, granted 12 September 2000.

HONORS AND AWARDS: University of Richmond Outstanding Mentor Award, 2004.

University of Richmond Distinguished Educator, 2000.

Roger Francis and Mary Saunders Richardson Chair of Mathematics, 1998-2010.

President's Fellowship for graduate study, University of Virginia, l983-86.

Phi Beta Kappa, l983.

Francis Shunks Downs Award for leadership, awarded by the President of Lafayette College, 1983.

PUBLICATIONS: Near-complete External Difference Families, with S. Huczynska and G. Mullen, *Designs, Codes, and Cryptography*, accepted June 2016.

Bi-Cayley normal uniform multiplicative designs, with L. Martinez and M.J. Sodupe, *Discrete Mathematics*, accepted April 2016.

A comparison of Carlet's nonlinearity bounds, with G. McGrew, S. Mesnager, D. Steele, and K. Marsden, *International Journal of Computer Mathematics*, accepted December 2015.

Linking systems in nonelementary abelian groups, with J. Polhill and W. Martin, *J. Comb. Th. (A),* Vol. 123 Issue 1, 92-103, April 2014.

A new product constructions for partial difference sets, with J. Polhill and K. Smith, *Designs, Codes, and Cryptography*, Vol. 68. 155-161, July 2013.

Shared Autocorrelation property of sequences, with C. Bodea, C. Copos, M. Der, and D. O'Neal, *IEEE Transactions on Information Theory*, Vol. 57 issue 6, 3805-3809, June 2011.

On the nonexistence of a projective (75,4,12,5) set in PG(3,7), with A. Chan and J. Jedwab, *Journal of Geometry*, vol. 97 pp. 29-44, 2010.

Novel Classes of Minimal Delay and Low PAPR Rate 1/2 Complex Orthogonal Designs, with S. Adams, N. Karst, M. K. Murugan, B. Lee, *IEEE Trans. on Inf. Th.*, Vol. 57 issue 4, 2254-2262, April 2011.

Difference set constructions of DRADs and association schemes, with J. Polhill, *J. Comb. Th. (A)*, Vol. 117 Issue 5, 598-605, July 2010.

G-perfect Nonlinear Functions, with L. Poinsot, *Designs, Codes and Cryptography* 46:1 January 2008

The design of the IEEE 802.12 coding scheme, with S. Crouch and J. Jedwab, *IEEE Transactions on Communications*, October 2007

Proof of the Barker array conjecture, with J. Jedwab and K. Smith, *Proceedings of the AMS*, Vol. 135, 2011-2018, 2007.

New amorphic association schemes, with Q. Xiang, *Finite Fields and their Applications*, Vol. 12, 595-612, 2006.

Negative Latin square type partial difference sets in nonelementary abelian 2-groups, with Q. Xiang, *J. London Math. Soc.*, Vol. 70 issue 1, 125-141, 2004.

Exponential numbers of inequivalent difference sets in Z_{2^{s+1}}^2, with D. Smeltzer, *Journal of Combinatorial Designs*, Vol. 11 number 4, 249-259, 2003.

A Family of partial difference sets with Denniston parameters in nonelementary abelian 2-groups, with Q.Xiang, *The European Journal of Combinatorics*, Vol. 00, 1-8, 2000.

Constructions of low rank relative difference sets in 2-groups using Galois rings, with Q.Xiang, *Finite Fields and their Applications 6*, 130-145, 2000.

A unified approach to difference sets with gcd(v,n)>1, with J. Jedwab, in T.Helleseth et al.,eds., NATO Science Series C, Difference Sets, Sequences and Their Correlation Properties, Vol. 542, Kluwer Academic Publishers, editor A. Pott and others, 85-112, Dordrecht.

Codes, correlations and power control in OFDM, with J.Jedwab and K.Paterson, in T.Helleseth et al., eds., NATO Science Series C, Difference Sets, Sequences and Their Correlation Properties, Vol. 542, Kluwer Academic Publishers, editor A. Pott and others, 113-132, Dordrecht.

A new family of relative difference sets in 2-groups, with J. Jedwab, *Designs, Codes, and Cryptography* Memorial Volume for Ed Assmus, Vol. 17, 305-312, 1999.

Peak-to-mean power control in OFDM, Golay complementary sequences and Reed-Muller codes, with

J.Jedwab, *IEEE Transactions on Information Theory*, Vol. 45, 2397-2417, 1999.

Some recent developments in difference sets, with J.Jedwab, in K. Quinn, B. Webb, F. Holroyd and C. Rowley, eds., Pitman Research Notes in Mathematics,Combinatorial Designs and their Applications, Addison Wesley Longman, 83-102, 1999.

A unifying construction for difference sets, with J.Jedwab, *Journal of Combinatorial Theory Series (A)*, Vol. 80 No.1, 13-78, Oct. 1997.

Hadamard difference sets in nonabelian 2-groups, with J.Iiams, *Journal of Algebra*, 199, 62-87, 1998.

Nested Hadamard difference sets, with J. Jedwab, *Journal of Statistical Planning and Inference*, vol. 62, 13-20, 1997.

Finding cyclic redundancy check polynomials for multilevel systems, with M.Mowbray and S.Crouch, *IEEE Transactions on Communications*, October, 1998, Vol. 46, Number 10, 1250-1253.

Peak-to-Mean power control and error correction for OFDM transmission using Golay sequences and Reed-Muller codes, with J.Jedwab, *Electronics Letters*, Vol. 33 No.4, 267-268, 13 February 1997.

New families of semi-regular relative difference sets, with J.Jedwab and M.Mowbray, *Designs, Codes, and Cryptography* 13, 131-146, 1998.

New semi-regular divisible difference sets, *Discrete Mathematics*, 188, 99-109, 1998.

Construction of relative difference sets in 2-groups using the simplex code, with S.Sehgal, *Designs, Codes, and Cryptography*, 11, 1-11, 1997.

Recursive construction for families of difference sets, with J. Jedwab, *Bulletin of the ICA*, vol. 13, p. 128, 1995, Research announcement.

Partial difference sets in p-groups, *Archiv Mathematik*, 63, 103-110, 1994.

A Survey of Hadamard difference sets, with J. Jedwab, In K.T. Arasu et.al., editors, *Groups, Difference Sets, and the Monster*, 145-156, de Gruyter, Berlin-New York, 1996.

Exponent bounds for a family of abelian difference sets, with K.T.Arasu, J.Jedwab, S.L. Ma, and R. McFarland, In K.T. Arasu et.al., editors, *Groups, Difference Sets, and the Monster*, 145-156, de Gruyter, Berlin-New York, 1996.

A nonexistence result for abelian Menon difference sets using perfect binary arrays, with K.T.Arasu and J.Jedwab, *Combinatorica*, 15 (3), 311-317, 1995.

A construction of difference sets in high exponent 2-groups using representation theory, with K. Smith, *Journal of Algebraic Combinatorics*, Vol. 3, 137-151, 1994.

A note on new semiregular divisible difference sets, with J. Jedwab, *Designs, Codes, and Cryptography*, 3, 379-381, 1993.

Almost difference sets and reversible divisible difference sets, *Archiv Mathematik*, Vol. 59, 595-602, 1992.

Reply to `Comment on ``Nonexistence of certain perfect binary arrays" and ``Nonexistence of perfect binary arrays," ', with J.Jedwab, *Electronics Letters*, vol. 29 p. 1002, 1993.

A summary of Menon difference sets, with J. Jedwab, *Congressus Numerantium*, vol. 93, 203-207, 1993.

Nonexistence of certain perfect binary arrays, with J.Jedwab, *Electronics letters*, 29 No.1, 99-100, Jan.1992.

New constructions on Menon difference sets, with K.T. Arasu, J. Jedwab, and S.K. Sehgal, *Journal of Combinatorial Theory Series A*, Vol. 64 No.2, 329-336, Nov. 1993.

New constructions of divisible designs, *Discrete Mathematics*, Vol. 120, 261-268, 1993.

An exponent bound for relative difference sets in p-groups, *Ars Combinatoria*, 34, 318-320, 1992.

Construction of relative difference sets in p-groups, *Discrete Mathematics*, 103, 7-15, 1992.

A note on products of relative difference sets, *Designs, Codes, and Cryptography*, 1, 117-119, 1991.

A note on intersection numbers of difference sets, with K.T. Arasu, D.Jungnickel, and A.Pott, *European Journal of Combinatorics*, 11, 95-98,1990.

Some nonexistence results on divisible difference sets, with K.T.Arasu, D.Jungnickel, and A.Pott, *Combinatorica*, 11, 1-8, 1991.

A generalization of Kraemer's result on difference sets, *Journal of Combinatorial Theory Series A*, March, 1992, 187-192.

A result on Dillon's conjecture, *Journal of Combinatorial Theory Series A*, July 1991, 238-242.

A note on nonabelian (64,28,12) difference sets, *Ars Combinatoria*, 32, 311-314, 1991.

Difference sets in nonabelian 2-groups, *Coding Theory and Design Theory*, Part II, IMA, v.21, 65-69, Springer-Verlag.

Difference sets in abelian 2-groups, *Journal of Combinatorial Theory Series A*, 57, Issue 2, July, 1991, 262-286.

SUBMISSIONS TO INTERNATIONAL NETWORKING STANDARDS: Low complexity decoding for power controlled OFDM codes, with J. Jedwab and K.G. Paterson, ETSI BRAN

Working Group 3 (Hiperlan Type 2) Temporary Document 3hpl081a, March-April 1998.

Options for variable rate coding in OFDM using binary, quaternary and octary modulation, with J. Jedwab, A. Jones, K. Paterson, and T. Wilkinson, ETSI BRAN Working Group 3 (Hiperlan Type 2) Temporary Document 50, Sept 1997.

EXPOSITORY ARTICLES:

The most exciting thing ever in mathematics, New Book of Popular Science, Grolier, 1994.

How do we know that anything is true?, New Book of Popular Science, Grolier, 1993.

From schoolgirls to CDs to Outer Space, New Book of Popular Science, Grolier, 1992.

PRESENTATIONS:

Apple vs. Samsung: a mathematical battle, Public Lecture at the Institute for Mathematical Sciences, Singapore, May 2016.

Construction of bent functions via covering EBSs, Workshop on Combinatorics, Institute for Mathematical Sciences, Singapore, May 2016.

Apple vs. Samsung, a mathematical battle, VCU Discrete Mathematics Seminar, 27 October 2015.

Near-complete external difference families, Finite Fields conference, Saratoga Springs, NY, 14 July 2015.

Near-complete external difference families, seminar, University of Delaware, May, 2015.

Apple-Samsung and Coding Theory, Sequences, Codes, and Designs, Banff, Canada, January 2015.

The final four: difference sets in groups of order 256, Finite Geometries, Irsee, Germany, September 2014.

Linking systems in non elementary abelian groups, Combinatorics 2014 conference, Gaeta, Italy, June 9, 2014.

Apple-Samsung: the codes in 3G communication, Finite Fields conference, Magdeburg, Germany, July 25 2013.

Epic failures in math, University of Richmond, April 2013.

Apple-Samsung: a personal story, University of Richmond, August 2012.

What do those engineers do? Penn State Harrisburg, April 2012.

Difference sets, Virginia Commonwealth University, November 2011.

Difference sets and Association Schemes, AMS sectional in Lincoln Nebraska, October 2011.

Difference sets and Association Schemes, Finite Geometries in IRSEE, Germany, June 2011.

McFarland difference sets and Association schemes, Vancouver, Canada, June 2010.

Difference sets and DRADs, Finite Fields and their Applications, Dublin, Ireland, June 2009.

Linear algebra is the answer to everything: an industrial research mathematics problem, Bloomsburg University colloquim, April 2005.

Relative difference sets and cohomology, a preliminary study, University of Delaware discrete mathematics seminar, October 2004.

Bluffhead and other games, Cool Math seminar, University of Richmond, September 2004.

The mathematics of 100 base vg, Central Michigan University Colloquium, February 2004.

The mathematics of 100 base vg, Simon Fraser University Colloquium, January 2004.

The mathematics of 100 base vg, National meeting of the American Mathematical Society, Phoenix, January 2004.

Negative Latin Square type partial difference sets in non-elementary abelian groups, The British Combinatorial Conference, July, 2003.

Finite Geometry meets Galois Rings: the search for Denniston Partial Difference Sets, University of Richmond Colloquium, March, 2002.

Inequivalent difference sets, University of Delaware Combinatorics Seminar, October, 2001.

Finite Geometry meets Galois Rings: Construction of Denniston Partial Difference Sets, The National University of Ireland, Maynooth, October, 2000.

The XTR Cryptosystem, University of Bristol/Hewlett-Packard Cryptography Seminar, February, 2001.

Denniston partial difference sets, The National University of Singapore, June, 2000.

Denniston partial difference sets, The Ohio State-Denison conference, May, 2000.

The game of SETS and coding theory, Greater Richmond Council of Teachers of Mathematics, March, 2000.

Relative difference sets and Galois Rings, conference in Augsburg, Germany, August, 1999.

Relative difference sets in groups whose order is not a prime power, seminar at the University of

Delaware, March, 1999.

Golay complementary sequences and Reed-Muller codes, conference in Bad Windsheim, Germany, August 6, 1998.

Unifying construction for difference sets, Part II, conference in Bad Windsheim, Germany, August 5, 1998.

Unifying construction for difference sets, Part I, conference in Bad Windsheim, Germany, August 4, 1998.

Beauty and the mathematician, Lunchtime forum, University of Richmond, December, 1997.

Construction of relative difference sets in groups of non-prime power order, British Combinatorial Conference, July, 1997.

The Kirkman schoolgirl problem to error correcting codes, Randolph-Macon College colloquium, November, 1996.

The Kirkman schoolgirl problem to error correcting codes, University of Richmond colloquium, September, 1996.

From the Kirkman schoolgirl problem to error correcting codes: different views of difference sets, The University of Wales at Aberystwyth colloquium, May, 1996.

A unifying construction of difference sets, The Basic Research in Mathematical Sciences (BRIMS) colloquium, Bristol, England, April, 1996.

A unifying construction of difference sets, Institut fur Mathematik der Universitat Augsburg, Augsburg, Germany, March, 1996.

A recursive construction of Relative difference sets, Royal Holloway University, London, England, February, 1996.

A recursive construction of Relative difference sets, Bose Memorial Conference, Ft. Collins CO, June 1995.

A new family of difference sets, National Security Agency, May 1995.

K-matrices revisited, Special Session on Codes and Designs, University of Richmond, Nov, 1994.

Partial difference sets in p-groups, The Ohio State University/Denison University conference, March 1994.

If at first you don't succeed... a search for difference sets, Virginia Commonwealth University colloquium, November, 1993.

Partial results on partial difference sets, the British Combinatorial Conference, July, 1993.

Nonexistence results on difference sets, the conference on Difference Sets at Ohio State University, May, 1993 (one hour talk).

A survey of Menon difference sets-- the nonabelian case, Southeastern Combinatorics conference, Feb. 1993.

Difference sets and perfect binary arrays, William and Mary colloquium, Jan. 1993.

If at first you don't succeed... a search for difference sets, Central Michigan University colloquium, May, 1992.

If at first you don't succeed... a search for difference sets, talk to Ohio State combinatorics seminar, May, 1992.

New constructions of divisible designs, the conference at Denison University, May, 1992.

Almost difference sets and divisible difference sets with multiplier -1, special session on Designs and Codes, Joint meeting of the MAA and AMS, Baltimore, January 1992.

Error Correcting Codes, Sigma Xi lecture, University of Richmond, December, 1990.

Relative Difference Sets in p-Groups, Oberwolfach, April, 1990.

Coding Theory, Hampden-Sydney colloquium, October, l988.

Character Theory Applied to Difference Sets, Design Theory Workshop, IMA, June, l988.

Difference Sets in Abelian 2-Groups, Joint National Meeting of the AMS-MAA, January, l988.

Difference Sets in Abelian 2-Groups, Wright State University colloquium, January, l988.

Difference Sets in Abelian 2-Groups, National Security Agency seminar, January, l987.

CLASSES TAUGHT: 100 level: Introduction to Statistics, Statistics for the Social Sciences, Calculus with precalculus.

200 level: Discrete math, Calculus 1, Calculus 2, Multivariate Calculus, Linear Algebra.

300 level: Introduction to Abstract Reasoning, Abstract Algebra 1, Abstract Algebra 2, Coding Theory, Cryptography.

SERVICE: Board of Trustees Academic and Enrollment Management Committee, 2013-2016.

Presidential Hiring Committee, 2014-2015.

Statistics Hiring Committee, 2014-2015.

Chair, hiring committee, 2001-2002.

Chair, General Education Committee, 2001-2002.

Chair of Math and Computer Science department, 1997-2000, spring 2005.

Science Scholars Committee, 1994-present.

Chair, hiring committee, 1994-95.

Curriculum committee, 1993-95.

Colloquium chair, 1991-94.

Freshman advisor, 1991-95, 2001-Present.

Calculus committee, 1990-94.

PROFESSIONAL AFFILIATIONS: Mathematical Association of America

American Mathematical Society.

Fellow, Institute of Combinatorics and its Applications