LTO: A better format for mid-range tape

G. A. Jaquette

In the last two years, Linear Tape-Open® (LTO®) tape drives have become clear leaders in the mid-range tape marketplace. Tape drives designed to the LTO Ultrium® format were the first of the ***super drives*** *to be shipped to mid-range tape customers. This paper describes some of the eclectic features designed into the LTO format. The technical emphasis is on aspects of the logical format that are new or different from preceding drives, though some aspects of the LTO roadmap and physical format are also discussed. The logical format comprises all of the data manipulations and organization involved in writing customer data to tape. This includes data compression to compact the data, appending of error-correction codes (ECCs) to protect the data, run-length-limited encoding of the ECC-encoded data, prepending headers to the encoded data to make it self-identifying on read-back, and storing of information about the data and the way it is stored in a cartridge memory module. Physical format aspects that are discussed include encoding data into the servo pattern and write shingling. Also discussed are the format-enabled aspects of drive functionality that have been improved over previous tape drive products, including enabling backward writing, elimination of problematic failure mechanisms, dynamic rewrite of defective data, handling servo errors without stopping tape, and enabling robust reading. Contrasts are made with previous products and competing products based on other format choices. Also discussed throughout is the way in which an eclectic format can be created by cooperation among three format-development companies.*

Introduction

In 1997, Hewlett-Packard**, IBM*, and Seagate** joined together to create a new, open, best-of-breed format for linear tape. This new tape format was to be made available via open licensing to any company wishing to make a tape drive or storage cartridge that conformed to the new format specifications. Hence, the new format was named the Linear Tape-Open[†] (LTO[†]) format. The LTO format is very robust, with features supporting reliability, high data integrity, scalability, and interchangeability. The three companies that created it are known generically as the *technology provider companies* (TPCs). The LTO format was designed to enable high-data-rate tape drives to read and write high-capacity cartridges, and was envisioned as a forward-looking format that could be extended as needed to support at least four generations (**Table 1**).

The cartridge capacity of LTO Generation 1 (Gen 1) was to be 100 GB native (i.e., without data compression), and each subsequent generation would double the native cartridge capacity of the previous generation, resulting in an 800-GB capacity in Gen 4. The native data rate of the LTO tape drives is specified only within a range. In Gen 1, for example, it is specified as 10 to 20 MB/s. The intent was to enable the native data-rate range to double in each subsequent generation, resulting in a data-transfer-rate range of 80 to 160 MB/s in Gen 4. Thus, the format was designed from the outset to be extensible in both capacity and data rate, and to allow data migration.

0018-8646/03/$5.00 © 2003 IBM

Table 1 The LTO roadmap.

	Gen 1 drives	*Gen 2 drives*	*Gen 3 drives*	*Gen 4 drives*
Capacity, native (compressed)	100 GB (200 GB)	200 GB (400 GB)	400 GB (800 GB)	800 GB (1.6 TB)
Transfer rate, native (MB/s)	10–20	20–40	40–80	80–160
Tracks at a time	8	8	16	16
Cartridges that can be read (at a minimum)	Gen 1	Gen 1 Gen 2	Gen 1 Gen 2 Gen 3	Gen 2 Gen 3 Gen 4
Cartridges that can be written (at a minimum)	Gen 1	Gen 1 Gen 2	Gen 2 Gen 3	Gen 3 Gen 4

In the years from 1998 to 2000, IBM made a significant investment in developing the technology building blocks—tape heads, tape deck, data-flow processing, and servo technology—that were needed to build LTO tape drives. This investment was required to enable IBM to compete effectively in the growing and lucrative mid-range tape marketplace in which a competitor's tape-drive family held the greatest portion of market share. The first product to result from this investment was the IBM TotalStorage* Ultrium† tape drive, which conformed to the LTO Gen 1 tape standard. It was shipped in September 2000 and was the first LTO Gen 1 tape drive to become generally available. The TPCs then extended the LTO format to LTO Gen 2, and the first Gen 2 tape drives were shipped in the fourth quarter of 2002.

This paper describes some of the aspects of the first two generations of the LTO format, which incorporated several format features designed to make data-flow processing in linear tape drives more robust. Some LTO features had never been seen in a tape drive, some only in isolated instances of tape drives that are no longer existent, and some only in helical tape drives. The LTO format allows for significant variation in implementation—so long as each implementation conforms to the format in a manner that enables data interchange. This paper also describes some IBM implementation decisions made in its LTO drives.

The LTO format is the best available implementation of a linear parallel serpentine technology principally because of its strong format, which enables more concurrent channels. LTO Gen 1 enables up to eight channels, and future generations are slated to enable up to 16 channels. By comparison, a maximum of only four channels were available in the most prevalent linear serpentine implementations available in mid-range linear tape drives before the general availability of LTO Gen 1 in the third quarter of 2000. (However, enterprise-class tape drives had as many as 16 channels being read/written at a time, but they typically cost much more than mid-range tape drives.)

In addition, when LTO Gen 1 was first shipped, it offered the highest cartridge capacity and areal density of any linear tape technology in the industry. The high areal density is supported by a robust logical format that includes interleaved recording across eight data tracks, with user data protected by a true cross-product error-correction code (ECC) designed for robust multiple-track operation in the midst of a relatively high random error rate. Other aspects of the robust logical format include a new data-compression algorithm, a new run-length-limited (RLL) code, dynamic rewrite of data errors, dynamic rewrite in response to servo anomalies, use of cartridge memory (CM) in each cartridge, and longitudinal position (LPOS) information embedded into the timing-based servo (TBS) system, which enables simple low-cost and very precise longitudinal positioning.

An eclectic mid-range linear tape format

The LTO format was designed to enable superior backup-and-restore operation. LTO drives were the first of the so-called *super drives* to be shipped, delivering 100 GB of native capacity in an LTO Gen 1 cartridge (a single-reel tape cartridge that holds about 600 meters of half-inch tape).

When the first LTO Gen 1 drives were shipped in the third quarter of 2000, the 100-GB native cartridge capacity was more than double that of any competing mid-range half-inch linear tape format, and two of the three LTO Gen 1 tape drives brought to market, including IBM's, had a native data rate of 15 MB/s.[1] This was exceptional native data-rate performance for a mid-range tape drive. This

[1] The LTO format specifies a recording density for data interchange, but not the tape speed at which it is written or read. Therefore, each drive maker may select the tape speed for its product. Because data rate is a direct function of tape speed for a given recording density, data rates may differ among LTO drive makers.

type of performance was previously available only in far more expensive high-end enterprise-class tape drives.

That this type of capacity and performance was obtainable in a relatively low-cost half-inch tape drive was not as surprising as the fact that it could be obtained repeatably and with high data integrity over a spectrum of cartridges of varying quality. The LTO format itself makes this possible. It is the eclectic collection of the best ideas brought forward by HP, IBM, and Seagate for a best-of-breed mid-range tape format. In the beginning, each of the three companies came to the table with a fairly complete tape-format proposal, and each proposal was significantly different from that of the other two companies. Each TPC proposed a different RLL code, a different ECC structure, and so on. It was decided that, where separable, the LTO format would consist of the best features of each of the three proposed formats. As a result, the RLL code chosen for LTO was developed by one of the companies, the ECC scheme was largely based on a design by the second of the TPCs, and the data randomizer adopted was that proposed by the third. In some cases, the format feature incorporated into the LTO format was brought to the table in nearly its final form. In other cases, the best ideas of the different companies were melded together to yield a result that was superior to any of the initial proposals. The data-compression algorithm—streaming lossless data compression (SLDC), developed and standardized as ECMA-321—is a good example. It took the best aspects of the proposals from all three companies and combined them in a data-compression scheme superior to any of the initial proposals. The complete LTO Gen 1 format was standardized by the ECMA International in June 2001 as ECMA-319.[2]

Each of these technological advances is described in detail in the following sections.

LTO tape layout and servo pattern

When the LTO roadmap was laid out, the goal was to achieve a native cartridge capacity of 800 GB in Gen 4 LTO drives and the products were to have backward write and read capability (Table 1). The backward capability meant that advances from one generation to the next had to be achieved via small evolutionary steps, and thus it was very desirable that the lateral span of the tape head remain unchanged from one generation to the next. The projection made at the time was that to achieve an 800-GB capacity required at least 1024 data tracks across the width of the half-inch tape, a significantly higher track density than had yet been achieved in linear tape. If the lateral span of the head were to remain the same over the four generations and support 1024 data tracks in the fourth generation, the problem to be solved was this: *What was the maximum projected lateral tape-head span that could support 1024 tracks in the fourth generation?*

The answer to this question gives, in turn, the maximum lateral tape-head span that should be used in the first generation. The determination made was that if the tape-head span were restricted to only one quarter of the half-inch tape and the tape head were controlled with respect to lateral position using a closed-loop feedback system, 1024 data tracks could ultimately be achieved.

The innovative LTO design extended the concepts of the bidirectional multichannel parallel serpentine linear tape-recording technology that preceded it. The LTO format specification divides the full tape width laterally into four data bands to minimize susceptibility to changes in the lateral span of a tape relative to the span of the tape head as a result of stretching (for example, from tape tension), shrinkage, or hygroscopic expansion over time. The division into four data bands enables higher numbers of tracks than can be achieved by linear tape formats that have heads spanning the entire width of half-inch tape, or half of it, as did all previous linear tape formats with eight simultaneous channels. As a result, LTO Gen 1 was able to offer 384 data tracks at a time when the most prevalent mid-range linear tape format offered only 208 data tracks (according to ECMA-208). LTO Gen 2 extended the data track count to 512.

Each data band is straddled by two servo bands pre-formatted with an extension to the TBS pattern developed by the IBM Almaden Research Group and first shipped in the IBM 3570 tape drive [1]. The servo bands provide location information to control the positioning of the head as it writes and reads data within the data band. The head is positioned laterally on the tape by a closed-loop servo that controls the two servo heads so that the average position error signal (PES) calculation is equal to a given reference value for a given wrap.

In addition to the PES signal used to control lateral position, the TBS servo pattern is encoded with LPOS information. LPOS is an absolute longitudinal address that appears at set intervals down the tape. It was developed and patented by IBM as an extension to TBS and has become a cornerstone of the LTO format. A unique LPOS word occurs every 7.2 mm down the tape, which is once every 36 TBS servo patterns, each of which is 200 μm long. Thus, the drive can position itself longitudinally down a tape to a given LPOS to obtain a resolution of 7.2 mm. Longitudinal resolution can be improved to 200 μm by using fractional components of the LPOS. This is in sharp contrast to linear tape formats that were developed before the advent of LPOS and in which the absolute longitudinal position might not be known within a meter after a long seek of, for instance, 500 meters. This meant that determining tape position had to rely on the reading

[2] Before 1994, ECMA International was known as ECMA, the European Computer Manufacturers Association. The Generation 1 LTO format document is accessible at the ECMA Website at *www.ecma-international.org* as ECMA-319.

of previously written data blocks. With an accuracy of only one meter, the only way to precisely determine location relative to previously written data blocks was to read those data blocks and reference them to some relative longitudinal position measure, such as the position of precise fine-line tachometers on the reel motors.

For example, a write-append operation might have required that a reference tachometer position be noted on a read operation, then a reposition to locate the tape head upstream of the append target, and then finally acceleration up to speed for a write operation based on a tachometer position. If, however, any tape slippage had occurred relative to the reel motors (for example, because the tape was loosely wrapped for a revolution or two), the write append might overwrite the end of the previously written data—an error known as a *chopped block*. This failure mechanism does not occur in a tape format that reads and writes on the basis of LPOS counts, because data-flow processes begin or end at given absolute positions mastered onto the tape itself as part of the servo pattern. The chopped-block problem can be solved in tape drives without any absolute position embossed on tape, but it typically requires increased drive complexity and hence cost. Although this solution allows drives to robustly prevent inadvertent data overwrite, it adds hardware and microcode complexity and results in some amount of capacity loss, because append gaps typically must be significantly longer.

In the case of read and write error-recovery procedures, still more microcode complexity is added because the data written to tape must first be precisely indexed relative to the tachometer before the error-recovery procedures can even be executed. If any tape slippage occurs after indexing and before a given pass over that data is executed, that step might fail for positional reasons alone, and then the indexing would have to be redone and the pass attempted again. Note that even in the absence of LPOS, if one has the TBS pattern, as was the case in the IBM 3570, one can measure the time between TBS pattern groups to create a clock every 200 μm, which could be used to eliminate fine-line tachometers on the reel motors, but this is relative addressing that does not, by itself, solve the chopped-block problem because, without absolute addressing from tape itself, a system can be deceived by tape slippage (unless the complexity referred to above is added to address this problem).

LPOS gives the absolute addressing from tape itself, and the use of LPOS encoding on top of TBS has helped to increase data integrity, simplify the drive microcode, and reduce drive cost by, for instance, eliminating the need for fine-line tachometers. These advantages are especially apparent when compared with analog servo patterns. TBS and LPOS have enabled LTO drives to achieve a high level of data integrity without adding cost to the drive.

Enabling write backward

To enable data migration, the LTO format set out to make it possible for each drive generation after the first to be able to write one generation back and read two generations back, as shown in Table 1. In the case of Gen 2 tape drives, which by definition can write Gen 2 cartridges (512 data tracks), this means that they must also be able to read and write the Gen 1 format (384 data tracks) to Gen 1 cartridges. To make this possible, the problem that had to be solved was this: *How could the format enable a tape drive to write the narrower tracks of the next generation and also be capable of writing the wider tracks of the previous generation?*

One possible solution to this problem is to have a completely separate write head for the previous-generation tape format, but this adds needless cost to the tape drive. LTO uses a much more practical solution. When writing the Gen 2 format, drives use a technique called *shingling*, shown schematically in **Figures 1(a)–1(c)**. When using shingling, a write track may overlap the bottom of a previously written track. In the first pass, the tape comes out of the cartridge and no previously written data tracks exist. The first eight data tracks are written [Figure 1(a)]. For the second pass, the tape is moving in the opposite direction, back into the cartridge, and the head is now in a new servo position within the same servo band, following the linear serpentine pattern. No previously written data tracks exist in this direction (inbound to cartridge), and the next eight data tracks are written [Figure 1(b)]. For the third pass, the tape again moves out of the cartridge and the head is again in a new servo position within the same servo band, but now previously written data tracks exist in the forward direction from the first pass, directly above where the third-pass data tracks will go. The third-pass data tracks partially overwrite the bottom of the first-pass data tracks [Figure 1(c)], just as the second row of shingles overlays a first row. Similarly, the fourth-pass data tracks (not shown in the figure) partially overwrite the top of the second-pass data tracks.

In the case of the IBM LTO Gen 2 write head, the actual write head width is that of the LTO Gen 1 data tracks, 27.5 μm. But, when writing the Gen 2 format, in the next pass in the same direction the data tracks are partially overwritten to leave a narrower residual width of 20.2 μm. To read these narrower Gen 2 tracks, the new read heads are, of course, narrower than the Gen 1 read heads (at least in the IBM implementation).

To write shingled tracks at a 20.2-μm spacing, the servo heads must be positioned differently over the servo band than was the case in Gen 1. The TBS pattern written in the servo band is continuously variable and enables

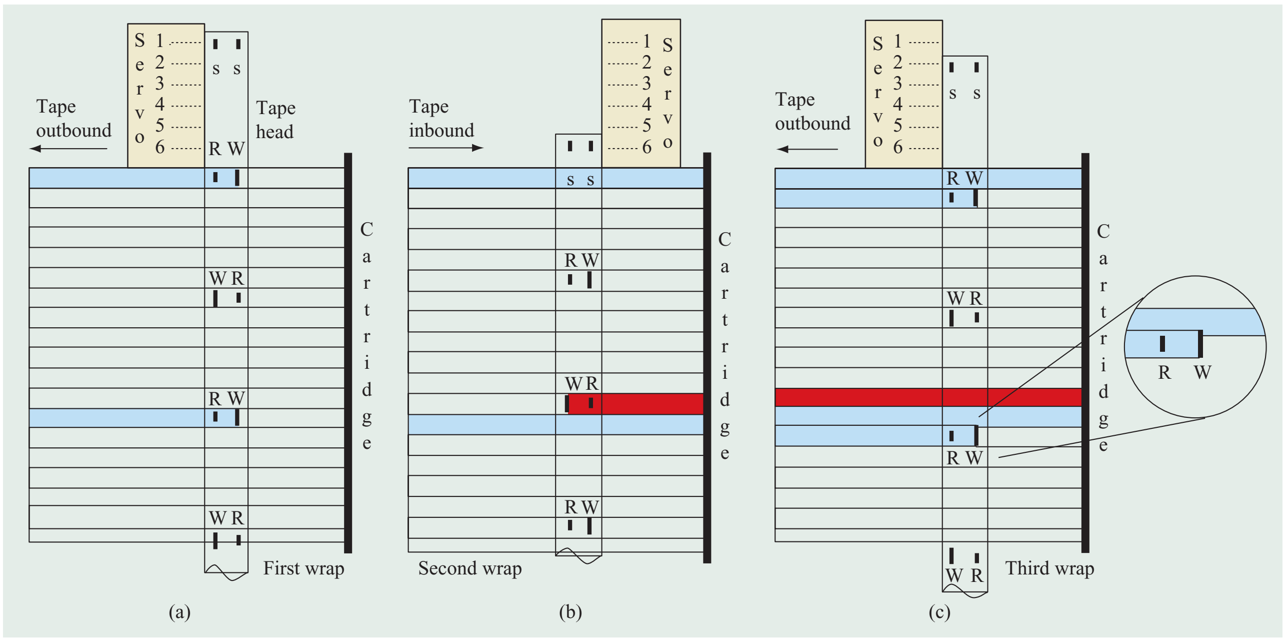


Figure 1

Schematic diagram illustrating shingling: (a) The first wrap, in which the data are written with the servo in the first position (blue stripe). (b) The second wrap, in which the data are written with the servo in the sixth position (red stripe). (c) The third wrap, in which the data are written with the servo in the second position (new blue stripe), but these data overlap those written in the first wrap.

selection of any lateral offset. Thus, it enables indexing at six different lateral offsets, as was done in Gen 1, or eight different offsets, as required for Gen 2. As we look forward to Gen 3, we may have to servo to other, different TBS offsets, likely at a finer track pitch than Gen 2. One of the advantages of the TBS servo pattern is that it inherently enables higher numbers of tracks. In fact, the IBM demonstration of the 1-TB operating point used the LTO Gen 1 servo pattern, unaltered. This required offsetting the head laterally at a minimum of 18 different offsets relative to the TBS pattern. LTO offers the best of both worlds—the extensibility built into TBS and the backward-write capability enabled by a serpentine pattern that allows shingling. Thus, we can index to six servo positions and write the Gen 1 format, or index to eight other servo positions and do shingled writing to achieve the Gen 2 format.

Cartridge memory

Tape drives require information about tape cartridges, such as which LTO generation(s) a given cartridge was designed to support, whether it is a data-storage or cleaning cartridge, etc. In previous generations, this information was gathered by means such as sensing notches or switches in the cartridge or reading special sections of tape. In LTO, this information can be gathered from the cartridge memory (CM), a silicon interface module embedded in each LTO cartridge. The module contains nonvolatile electronic memory for storage and retrieval of information about the cartridge and the data on it. While LTO was not the first tape format to adopt an electronic memory in each cartridge (e.g., ECMA-291), its implementation is somewhat different from, and perhaps better than, previous implementations.

The CM used by LTO has a noncontact passive radio frequency interface that eliminates the need for physical connections to the cartridge for power or signals, unlike a previous format that required electrical connection with the cartridge memory when the cartridge was loaded (a potential failure mechanism). Many types of information are stored in the CM that are integral to the LTO format and affect how data is written to tape. The CM can be read, enabling a system to be fed information about the condition of the cartridge and what is stored on tape, even without loading or threading the tape. As an example, the CM could be read by a CM reader attached to a picker in a tape library, providing the tape library with information about the data stored on a tape without even having to transport it to a tape drive.

There are many pages of data stored in the CM. For example, when a tape is formatted, the exact LPOS locations of various points on the tape, such as the beginning and end of the user area (BOT and EOT), are stored in the *initialization data page*. Thus, there is no need for holes in tape to identify BOT and EOT, as are used by some other tape formats—holes that weaken the already thin tape and represent a potential failure mechanism. Information about the type of cartridge is also stored in the CM. This eliminates cartridge notches or tabs that would otherwise be required to identify cartridges of different generations and to distinguish between cleaning cartridges and tape-storage cartridges.

A *write-pass number* is kept in the CM and copied to every header written to tape, enabling old or abandoned data to be simply disregarded by the hardware. The write-pass number is monotonically increased as certain events such as cartridge loads, wrap turns, and appends occur. When this data is embedded in written data, it effectively gives a precedence for the data. For example, if two datasets are found with the same dataset number, the one with the higher write pass number is the valid one because it was written after the one with the lower write pass number. Tracking the write-pass number in the CM is much more robust than in previous formats because it can be done on-the-fly as the tape is being written. Thus, even if power is subsequently lost, an update to the write-pass number in the CM is remembered.

The *tape directory page* stored in the CM contains the *record* and *file-mark* counts at the middle and end of each tape wrap, enabling the drive to determine which half of which wrap contains the target of a *space* or *locate* command. Additionally, information necessary to present *tape alert* status is stored in the CM, as is usage information on the last four loads of that cartridge. There is also a unique CM serial number that can be used to identify a given cartridge. (This number can be used as a tape label or can be useful when a physical label is used, because it enables tracking when a cartridge has been relabeled or if its external label has become unreadable.) The CM also provides a scratch-pad area that can be written by a user application. For example, a customer may need to store a manufacturer's data for warranty purposes or data concerning the age of the cartridge, the number of loads it has endured, etc. Some CM pages are copied to tape as a backup should the CM fail, be corrupted, or be removed from a cartridge. However, it should be noted that the LTO format was specifically constructed to allow reading of any user data written to tape without use of the CM module. Thus, data stored in the CM is useful and helps performance, but is not essential for data to be read from the tape.

Error-correction codes

The power of error correction designed into LTO is unprecedented in mid-range linear tape products. As an example, ECMA-286 describes a prevalent competing mid-range half-inch linear tape format that has a Reed–Solomon outer code, RS(20, 16), calculated over 8-KB blocks, but the 8-KB blocks themselves are protected by only a two-byte error-detecting code (EDC). In theory, this allows correction of any four erroneous blocks, so long as the EDC correctly detects those blocks as being in error. However, because this is a four-track format, each track nominally holds five of the twenty 8-KB blocks and thus, this format cannot nominally tolerate the loss of a whole data track, which might result from, for instance, a longitudinal scratch after writing has occurred, even if the other three tracks are pristine. If the burst error affects only four blocks, this consumes the full power of the outer code, and there is no residual power to handle random errors. It would appear that a single byte in error at an unknown location in any of the remaining 16 blocks would be uncorrectable. Finally, even without any burst errors, this form of ECC is not well suited to a high background error rate. As an example, if there are one to six bytes in error at unknown locations in half of the 8-KB blocks (which is less than one tenth of the background error rate that the LTO ECC can handle, in addition to losing all of the data on a track), this other one-dimensional ECC would not generally be able to correct them.

Viewed from a high level, the LTO ECC was designed largely along the lines originally proposed by one of the TPCs, but it was enhanced in numerous ways by the contributions of the other two TPCs. Most notably, the interleave was improved, the filling order was changed, and the dynamic rewrite of defective elements (discussed below) was added. With these improvements, the net result was an ECC design far superior to any of the initial proposals.

Each of the two existent generations of LTO products, when shipped, had the highest areal density then available in a half-inch linear tape drive: approximately 114 Mb/in.^{2} in LTO Gen 1 and slightly more than twice that in LTO Gen 2. The robust logical LTO format permits this through the use of a very strong true cross-product ECC design. Data is written to tape in a minimum recording unit which is protected with ECC. Referring to Figure 11 in the paper by Childers et al. [2, this issue], ECC-protected data is written to tape in minimum-size chunks known in LTO as *datasets*. Each dataset is interleaved both across the eight data tracks and along the tape longitudinally. Each customer record is first protected with a cyclic redundancy check (CRC) before data compression. The output of the data compressor is the compressed data stream, which, in turn, is broken into 403 884-byte chunks and then protected by ECC, with the ECC-protected entity being known as a dataset. The dataset is broken into 16 two-dimensional arrays and then protected by two different orthogonal ECC encodings. The data in each row is protected by an inner code, C1, an even–odd interleave of two Reed–Solomon codewords, each RS(240, 234), designed to allow correction of a relatively high background error rate. The data in each column is protected by a strong Reed–Solomon outer code C2, RS(64, 54), designed to allow correction of long error bursts.

This gives a true cross-product ECC design capable of simultaneously correcting both a high background error rate and long-bursted errors. As an example, this ECC is powerful enough to support reliable recovery of data even with the loss of one of eight tracks on a read operation and up to two bytes in error at unknown locations in each of the remaining inner codewords (240 bytes each) on the remaining tracks. It is this ECC power that allows the LTO format to reach the high areal densities and capacities it has achieved with high data integrity.

Because these two encodings are orthogonal, they can be done in either order. Thus, the terms *inner* and *outer* are applied in the order in which it typically makes sense to use them during correction, rather than implying an order in which they must be encoded. For example, the *inner-code* encoding can be performed first, as the data arrives, to protect the incoming data with ECC before it is placed in any form of volatile silicon memory, such as DRAM, where it might be subject to corruption, on its way to being written to tape. But on correction, during reading, it typically makes sense to perform the inner-code correction first and the outer-code correction next in an initial attempt to correct a dataset, though iteration can continue thereafter.

Each dataset is broken into 64 recording units known as *codeword quad sets*, or *CQ sets*, with each set comprising eight CQs, one for each track. Each dataset is written to tape in 64 CQ sets, with each CQ having format features inserted as it is written (see Figure 12 in [2, this issue]). After each CQ of data is written by the write head, a read head reads it and checks whether it was written properly or is obstructed by some media defect or scratch. If an error of sufficient length is found by the ECC, the entire CQ set is dynamically rewritten farther down tape, but rotated laterally so that each CQ is now on a different track. This enables the drive to establish that a correctable version of every CQ is written to tape; i.e., none of the power of the outer code has been consumed by errors left in by the drive at write time.

Read-detection techniques and RLL codes

The LTO format has a basic structure that was already in existence in tape drives such as the IBM Magstar*. Specifically, in LTO, the dataset is a minimum recording unit. An LTO dataset contains about 400 KB of user data. Inserted between datasets is a tone that is outside the run-length-limited (RLL) code used to encode the datasets. In LTO, it is known as the dataset separator (DSS) tone. Detection of the DSS tone enables the data-flow logic to know that it is outside any dataset. The formatted dataset itself has a structure common to many types of recording devices. It begins with a high-frequency tone designed to enable a circuit, known as a *phase-locked loop* (PLL), to acquire phase lock and read back data. Analog PLLs have a circuit known as a *variable frequency oscillator* (VFO), and the high-frequency tone designed for locking the PLL is known in LTO as the *VFO pattern*.

Following the VFO is a *synchronization mark* (sync mark), the detection of which is used to establish the byte count as well as the bit-level symbol boundaries for RLL decoding. Following the sync mark is a 10-byte header used to identify a given codeword pair. This is followed by a *C1 codeword pair*, which contains user data protected by C1 ECC. The first C1 codeword pair is followed by a *resynchronization mark* (resync), which enables the RLL decoder to reestablish bit-level symbol boundaries if that framing was lost in the first codeword pair (which can happen if the PLL slips one or more clocks in either direction). Following the resync is another 10-byte header, which in turn is followed by another C1 codeword pair (the second). Following the second codeword pair is a *reverse synchronization mark*, which can serve the purpose of a sync mark if the data is being read physically in reverse. Finally, there is another VFO pattern, which enables the PLL to reestablish phase lock if it has been lost.

The RLL encoding of the headers and C1 codeword pairs is historically performed to enable the PLL to stay phase-locked to the encoded bitstream. Specifically, a PLL typically requires some minimum phase update rate, regardless of the user data being recorded. Phase updates, at least in peak detection channels, occur only when magnetic transitions are detected. Thus, the RLL code must guarantee that magnetic transitions occur at some minimum rate or, conversely, that there is a maximum number of clock cells that can go by without a magnetic transition occurring (known as the k constraint of an RLL code). There can also be a minimum number of clock cells that must go by after one magnetic transition and before the next (known as the d constraint of the RLL code). Peak detection channels can typically support higher linear densities when RLL codes with d equal to 1 or 2 are used. Partial-response maximum-likelihood (PRML) channels are not advantaged by RLL codes with nonzero d constraints. In peak detection channels, the bitstream to be recorded to tape is typically represented in non-return to zero inverted (NRZI) format, in which each 1 represents a magnetic transition on tape, and each 0 represents the lack of a magnetic transition. The d constraint of an RLL code is then the minimum number of 0s between sequential 1s in the resultant encoded bitstream, and the k constraint is the maximum number of 0s between sequential 1s.

A third parameter used to characterize an RLL code is its rate. If three RLL-encoded bits are output for every two unencoded bits in, the RLL code has a rate of 2/3. The rate is simply the ratio of unencoded input bits to encoded output bits. Note that the higher the d constraint that is being enforced, the lower the rate. Similarly, the selection of k also affects the rate, and the maximum achievable rate (also known as the *Shannon limit*) becomes lower as k is made lower. Note that a 2/3 rate is very near the Shannon limit for RLL codes with the (d, k) constraints of (1, 7). RLL codes with the (d, k) constraints of (2, 7) typically have a rate of, at most, 1/2.

Generation 1

In an effort to bring the first generation of LTO devices to market as quickly as possible, it was decided that the data would be written in such a way as to allow the simplest form of read detection—*qualified peak detection*. To enable high linear densities with qualified peak detection, an RLL code can be used, as is well known in the industry. For peak detection channels, a k of 7 is typical, and d is typically 1 or 2. For LTO, a d of 1 was chosen. When the LTO Gen 1 format was being created, there were at least five 2/3 rate (1, 7) RLL codes available to choose from—for instance, the IBM Magstar tape drive used the Adler/Hassner/Moussouris code with these constraints (see ECMA-278). But the use of one of these existent 2/3 (1, 7) RLL codes would have caused needless complexity in the data-flow logic because a tape drive must be able to orient itself quickly, easily, and robustly to data being read from tape. To do so, it is best if three different types of patterns are excluded from the RLL code: the DSS pattern, the VFO pattern, and the RLL resynchronization pattern.

The LTO format specifies that a low-frequency DSS tone pattern is to be written between datasets. The DSS pattern chosen violates the k constraint of the RLL code, and thus is outside the RLL code used to encode data. Because the DSS pattern is outside the RLL code, data-flow logic can use detection of an extended length of this low-frequency tone as a clear sign that it is outside any dataset, and it can appropriately reset counters or logic in preparation for the start of the next dataset. In fact, the DSS pattern can be detected reliably without even phase-locking to the read-back waveform. Note that by definition, a DSS pattern violating the k constraint would be outside any RLL code that had that k constraint.

Additionally, each dataset begins and ends with a VFO field appropriate for locking a PLL to the frequency of the data stream being read, but VFO fields also occur at the beginning of each CQ set to enable reacquisition of phase lock in the middle of a dataset. Loss of phase lock can be caused by a section of tape with a low signal-to-noise ratio (SNR), a media defect, or a longitudinal shock wave down the tape, which causes the tape underneath the head to accelerate and then decelerate beyond the ability of a PLL to track it. Of course, acquiring phase lock in one of these intermediate VFO fields can be done robustly only if the circuits controlling the PLL can reliably detect where the VFO field is, and this was one of the problems: The best VFO pattern to use in a format utilizing a (1, 7) RLL code is 101010 . . . , but this pattern can be produced at the encoded output of any of the (1, 7) RLL codes we studied. Thus, use of one of these codes would have made detecting the VFO pattern of little value, because what appears to be a VFO field could actually be simply an RLL-encoded piece of customer data. In that case, detection of the VFO pattern, even simultaneously on multiple tracks, could not be used as a robust trigger to indicate that a VFO pattern had been entered.

Instead, precise timing circuits would have to gate detection of intermediate VFO fields so that it would occur only at precise intervals after the initial VFO field. But this strategy is unworkable when the initial VFO field is obscured, so alternate strategies would have to be adopted to handle this case. It was realized that some of this complexity occurs for the simple reason that random data could be encoded by the existent (1, 7) RLL codes to be a VFO pattern. One way of reducing the complexity of the data-flow circuits was to design a new (1, 7) RLL code for the LTO format that would exclude any extended encoding of the VFO pattern (101010 . . .). This enabled the LTO data-flow logic to definitively identify and orient itself to intermediate VFO fields, and phase-lock to them as necessary, simply by looking for extended VFO patterns. As an example, even if some defect has caused the tape to separate from the head far enough that the underlying data (e.g., DSS) is not overwritten, the data-flow circuits can reorient themselves to the CQs written after this point using simple VFO tone detectors sufficiently qualified to find only strings of adequate length.

Another consideration is how one orients the RLL decoder to the data if the initial synchronization field is missed or the clock has slipped by one cycle or more. Each CQ is composed of two inner-ECC-codeword pairs, a first codeword pair followed by a second codeword pair. In the middle of each CQ between the first codeword pair and the second there is an RLL resync field. This field is designed to allow framing of the RLL code symbol boundaries to enable RLL decode of the second codeword pair. While this is not necessary if the phase lock was maintained flawlessly through the first codeword pair, it is necessary if a phenomenon known as a *sync slip* has occurred, that is, when a PLL drifts off and then reacquires phase lock, but at an offset of one or more bits relative to the initial phase lock. A counting circuit, for example, would gain or lose one or more bits, and thus the RLL decode becomes improperly framed, with the result being that the rest of that codeword pair is erroneously RLL-decoded. The resync field is, by design, one that the RLL decoder can use to reorient itself to the bitstream.

A resync pattern is of reduced value if it can be found in RLL-encoded user data, because this creates the case in which, if the resync detection window is opened too wide, false resync detection can occur and, thus, resync detection cannot be used unequivocally. Circuit complexity can be reduced somewhat if the RLL code is designed to exclude the selected resync pattern, and this was done.

Finally, to minimize error propagation, it was desirable that the RLL decoder be implemented using a sliding-block decoder of minimal complexity. A sliding-block decoder was desirable because it does not depend on RLL-encoded data before or after it, thus tightly bounding error propagation. IBM designed and patented a new 2/3 rate (1, 7) RLL code for LTO that excluded both extended lengths of the VFO pattern and the resync pattern while still enabling use of a simple sliding-block decoder [3].

Generation 2

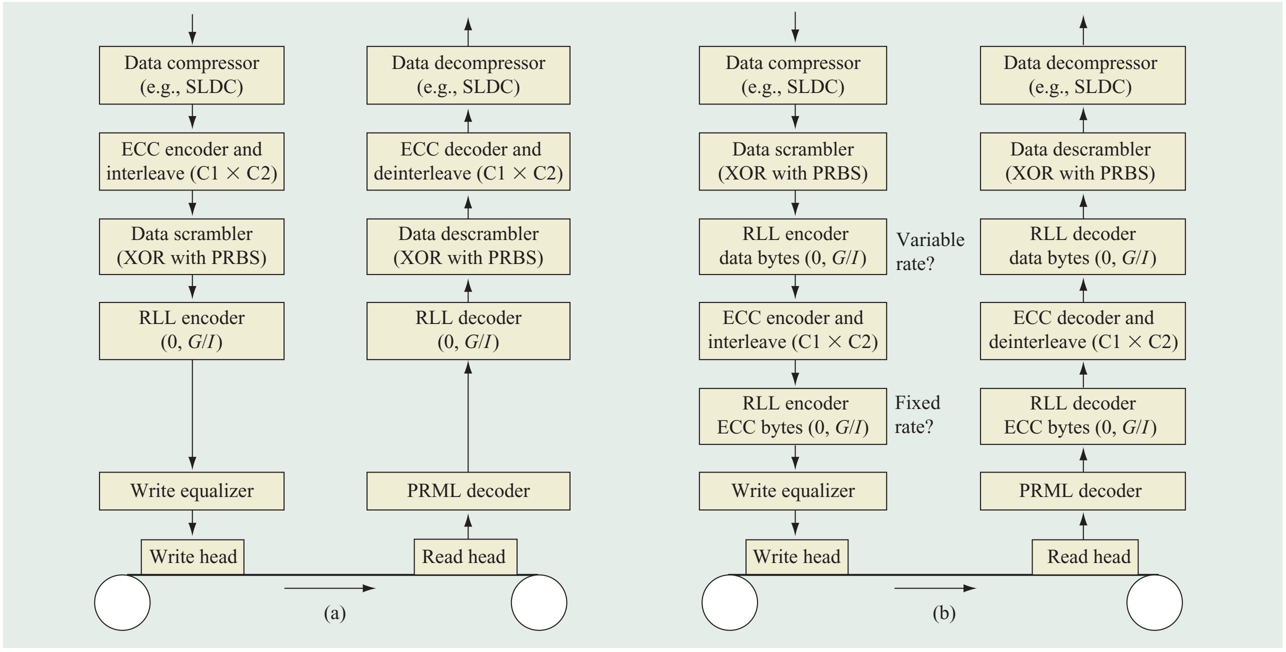
As part of LTO Gen 2, we had to double the native cartridge capacity to 200 GB without increasing tape length. This meant that the areal density had to be doubled. It was desirable to increase the linear density as much as possible to obtain the consequent increase in native data rate for a given tape speed.

A PRML read-detection channel—a technique to increase linear density that had already been proven in hard disk drives and helical tape drives—was chosen. A PRML channel enables the user bit density to be increased by as much as 50% without increasing the maximum resolvable flux transition density, thus minimizing the burden placed on the media. The resolvable flux transition density is the density of the flux transitions that must be read back to nonzero samples (e.g., the pattern 111 . . . gives all zero samples, which can be thought of as not being resolved by the read-back system). The 50% increase in linear density is achieved only if a rate-1 (0, k) RLL code is used in place of a 2/3 rate (1, 7) code. Code rates of nearly 1 (which is the limit, of course) were being achieved in some RLL codes designed for hard disk drives (HDDs), but in order to achieve these very high rates, few or no constraints were being placed on the encoded output. In RLL codes designed for partial response class 4 (PR-4) detection, the k constraint is typically called the G constraint and is the maximum number of 0s that can occur between sequential 1s in the RLL-encoded bitstream. Another constraint of interest in PR-4 channels is the maximum number of 0s that can occur on each interleave (a PR-4 detector can detect the bitstream as if it were two independent even/odd interleaved bitstreams). This constraint is called the I constraint. Thus, the G and I constraints are typically quoted as G/I in place of k. Thus, a 16/17 rate (0, 13/11) RLL code would encode each 16 input bits to 17 RLL-encoded output bits while enforcing a G constraint of 13 and an I constraint of 11. (Note that a d of 0 is not really a constraint, but rather a lack of one.)

As was the case in LTO Gen 1, each of the three TPCs brought forward a different proposal for a (0, k) RLL code. However, before describing the RLL code selected by the TPCs for use in LTO Gen 2, I discuss a different type of code, a variable-rate RLL code. This is of interest because a number of other linear tape format developers adopted variable-rate RLL coding when they began using PRML recording. Perhaps because of this, one of the TPCs proposed that a variable-rate (0, k) RLL code be used by the LTO format, and this was studied at length.

A variable-rate RLL code typically calls for randomizing the compressed and ECC-protected user data so that it is a pseudorandom sequence of 0s and 1s and then bit-stuffing 1s into the resultant bitstream when necessary to meet the desired G and I constraints. The advantage of this technique is that it can typically yield a higher RLL rate for a given set of G and I constraints than can a fixed-rate RLL code of limited block size—and limiting the block size is typically necessary to reduce complexity of a fixed-rate RLL decoder to an acceptable level. As an example, a variable-rate (0, k) code might achieve an average rate of 0.997, which can be contrasted to the 16/17 (which is 0.941) rate of a relatively simple fixed-rate (0, k) code or the 32/33 (which is 0.970) rate of a somewhat more complex (or significantly weaker) one. The RLL rate difference of a variable-rate RLL code might yield an increase in capacity of 5.9% (versus a 16/17 rate code, only 2.8% relative to a 32/33 rate code), if all other things are held equal (ECC, flux transition density, etc.), which would be desirable. But this rate difference is not free, which is perhaps why HDD developers have not flocked to variable-rate (0, k) RLL codes.

If RLL encoding is performed after ECC encoding, which is standard, the RLL encoding and decoding process is not protected by the ECC; rather, the ECC must be used to correct any RLL decoding error caused by a misdetected channel bit. Thus, it is desirable to minimize the RLL decoding errors that can result from a given number of misdetected channel bits (which is why sliding-block RLL decoders are preferable where possible). However, when attempting to reverse the bit-stuffing process proposed for variable-rate encoding (destuffing), there is a susceptibility to a single misdetected channel bit (or misdetected pair of bits, as is more likely in the case of a PRML detector) that can cause a bit to be destuffed



Data compressor (e.g., SLDC)
ECC encoder and interleave (C1 × C2)
Data scrambler (XOR with PRBS)
RLL encoder (0, G/I)
Write equalizer
Write head
Data decompressor (e.g., SLDC)
ECC decoder and deinterleave (C1 × C2)
Data descrambler (XOR with PRBS)
RLL decoder (0, G/I)
PRML decoder
Read head
(a)
Data compressor (e.g., SLDC)
Data scrambler (XOR with PRBS)
RLL encoder data bytes (0, G/I)
Variable rate?
ECC encoder and interleave (C1 × C2)
RLL encoder ECC bytes (0, G/I)
Fixed rate?
Write equalizer
Write head
Data decompressor (e.g., SLDC)
Data descrambler (XOR with PRBS)
RLL decoder data bytes (0, G/I)
ECC decoder and deinterleave (C1 × C2)
RLL decoder ECC bytes (0, G/I)
PRML decoder
Read head
(b)

Figure 2

(a) Standard-order and (b) reverse-order concatenation of variable-rate RLL codes.

that should not be, or vice versa. The result is a phenomenon known as *catastrophic* (or *infinite*) *error propagation*, which is to say that the output of the destuffer might be in error to the end of that block because of a bit shift, virtually guaranteeing that the inner ECC code, if there is any, will not be able to correct it.

To be clear, not all misdetected bits will cause destuffing errors (with the ratio of those that do being dependent on the constraints being enforced by the variable-rate code); however, even if only a small percentage of the simple errors that could otherwise be corrected by an inner ECC code instead cause catastrophic error propagation, it is more than is desirable because the errors consume some of the power of the outer ECC code, reducing the power available to address the other types of burst errors for which it was designed. Strengthening the outer ECC code to counteract this can consume some, or all, of the capacity gain achieved by the higher RLL code rate of the variable-rate code.

An alternative that addresses the catastrophic error propagation problem is *reverse concatenation* (RLL encoding before ECC encoding) [4]. Reverse concatenation would RLL-encode only the data part of the ECC codewords, but if the RLL constraints are needed for reading, the ECC bytes themselves would have to be separately RLL-encoded, as shown in **Figure 2**. And if the ECC bytes are RLL-encoded, it would seem that these should be encoded with a fixed-rate RLL code to avoid the catastrophic error propagation problem and, if this is done, the average RLL encoding rate will have been decreased.

In any case, if reverse concatenation is used to avoid catastrophic error propagation, two different RLL encoding stages are necessary; the susceptible variable-rate RLL encoding stage must first be placed before the ECC is appended, then a separate form of RLL encoding must be used for the appended ECC bytes. All of this complexity for less than 6% in capacity (versus a 16/17 rate code) is not a wise tradeoff. The other choice, the one which is apparently typically made, is standard concatenation and accepting some amount of catastrophic error propagation. When these issues were understood, the variable-rate RLL proposal was abandoned by the TPC that had brought it forward, leaving the group to look for a standard fixed-rate (0, G/I) RLL code for LTO Gen 2.

Two of the three TPCs proposed different existent 16/17 rate (0, 6/6) RLL codes but, as in Gen 1, it was desirable that the DSS, VFO, and resync fields be outside the RLL code chosen, something not found in the available RLL codes. Most, if not all, of the existent fixed-rate (0, G/I) RLL codes were designed for an HDD environment which, from the point of view of timing, is a much more stable recording environment than tape. In HDD, the data are essentially read off a flywheel, which creates a detected bitstream with very little frequency variation. In helical tape, the high tape-to-head speed is achieved predominantly because of the rate at which the rotor holding the head is turned. The tape itself is actually moved quite slowly, so the timing is again dominated by a flywheel.

In linear tape, the data are read off an elastic ribbon of thin plastic as it flies by a read head, an environment that creates much more frequency variation in the read-back signal because it is dominated by movement of the recording tape, whose properties are more akin to those of a wet noodle than a flywheel. For this reason, it is desirable to design the RLL code to generate encoded data that gives the best phase updates to the PLL so that it can stay phase-locked to the read-back signal even as it varies in frequency. The best phase updates give a high SNR linear phase error signal when the PLL clock is not perfectly aligned with the phase of the read-back signal. In a PR-4 channel, the best data features for phase updates are isolated peaks. For example, if an encoded 1 represents a flux transition, an isolated transition is a 1 that has a 0 on either side of it, resulting in the binary sequence 010. These features also give a full amplitude peak that can be detected by an analog automatic gain control (AGC) circuit to enable the gain to be controlled before analog-to-digital conversion (ADC). In the absence of full amplitude peaks, one must lower the amplitude into the ADC to allow for gain error. A final subject that interested the TPC members was the error propagation properties of each proposed $(0, G/I)$ RLL code. Three new parameters of interest were defined in the comparison of $(0, G/I)$ RLL codes:

- MD = maximum distance between isolated peaks (i.e., between isolated NRZI ones).
- AMD = average maximum distance (in time) between isolated peaks.
- EP3 = error propagation (minimum length of channel error burst, measured in NRZ bits, that can corrupt three user bytes).

The MD and AMD parameters were found to be infinity for the two existent $(0, k)$ codes we considered. EP3, the minimum number of channel bits in error that could cause three decoded bytes to be in error, was determined to be one bit for one of these codes and five bits for the other. It was decided that MD and AMD were undesirably high for a linear-tape application, and that EP3 was undesirably low for the two existent RLL codes. Therefore, a new RLL code with a 16/17 rate (0, 13/11) was designed which guaranteed at least one isolated peak per 17 encoded bits out (and thus an AMD parameter of 17), a maximum distance between isolated peaks (MD) of 23, and which required nine channel bits to be in error to cause three decoded bytes to be in error. It was determined that the MD, AMD, and EP3 parameters of the newly designed RLL code were superior in a linear-tape application to those of any other $(0, k)$ RLL code considered, and this new RLL code was the third proposal, which ultimately won out and was selected for use by the LTO Gen 2 format.

Read-while-write and dynamic rewrite of defective data

The LTO format dictates that while writing to tape, each write head must be followed by a read head to allow immediate verification of the data written. This means that if there is an error in written data, it can be immediately detected and rewritten. This capability addresses problems that arise from tape defects and scratches, which, unfortunately, are not uncommon. This capability, known as *read-while-write* (RWW), has been available in linear-tape drives long before LTO.

What effect does this rewriting have on write-throughput performance? The effect will always be negative because the process involves rewriting previously written data instead of new data, but the negative effect can be more pronounced in some formats than others. For example, the IBM 3590 format also dictated RWW recording, but when data errors were found, it required that the whole dataset be overwritten, or erased and rewritten down tape. In that case, the first rewrite attempt may occur after a reposition in the same location, but if RWW indicates that the rewritten dataset block is still in error (as one might expect if it were due to a tape defect at that location), it is then erased and rewritten farther down the tape. With this approach, one and only one copy of each dataset—written correctly—occurs on tape, after another reposition. This can be thought of as a *write-pristine* format. But if the tape being recorded has a significant background error rate due, for instance, to small defects, and the tape format has little or no inner-ECC-code power, the write throughput of a write-pristine format can be dominated by the amount of time required to reposition the tape, and one or more seconds can be added to the amount of time required to write each dataset that must be rewritten. This can result in a significant loss in write throughput to the tape drive in a tape environment with a high dataset rewrite rate, as can happen with marginal tapes.

One solution to reduce write throughput time is to add an inner ECC code capable of correcting the smaller of the errors resulting from tape defects. This helps to eliminate the need for some of the rewriting that would otherwise occur. IBM applied this solution in the second and third generations of its 3590 product. This approach improved the write performance and enhanced the power of its ECC in the presence of a background error rate that climbed as areal density was increased by a factor of 3 (third generation versus first) without changing media.

While writing to tape, media defects will be encountered that will defeat even the strong inner ECC code of the LTO. There are various strategies to handle errors that are beyond the capability of an inner ECC code. As mentioned above, one option is to erase the dataset and rewrite it down the tape, but this can severely limit write-throughput performance. Another option, if one had ample power in the outer ECC code, would be to simply leave the error and continue on, knowing that while the inner ECC code has been defeated, the outer ECC code will be able to correct it. However, each error left in at write for the outer ECC code to handle reduces the amount of outer ECC power left to correct burst errors which might occur on a read, but which were not present when the RWW occurred. As an example, if the tape were scratched longitudinally, for example, by passing at high speed over a piece of debris on some fixed element (rare, but not unknown), a whole data track could be destroyed. Most, if not all, of the power of the outer ECC code must be left to handle these types of burst-error events. A better method of handling errors discovered during read-while-write is to dynamically rewrite the affected data down the tape without stopping. This enables continuous writing through a region of tape that is bad in some tracks. It is also powerful enough to permit recording a good copy of all user data, even if one write head is completely nonfunctional for some period of time.

Dynamic rewrite of data had been implemented in other tape drives, but the LTO implementation is perhaps better. In LTO, a dataset is broken into 64 CQ sets. (A CQ set is essentially 1000 bytes of formatted data on each of the eight tracks.) If a CQ set has to be rewritten (perhaps because of a tape defect seen by only one of the data tracks), the whole CQ set is rewritten down tape, though it is rotated so that the data originally written on any given physical track is rewritten to a different physical track. Each CQ set is self-identifying because of the two headers written with the 1000 bytes of formatted data on each of the eight tracks, and each header is protected by CRC. Thus, there are nominally 16 CRC-protected headers associated with each CQ set, and if any of the headers are corrupted, the CRC is designed to point them out. Voting logic can be applied to the header information gathered from the 16 headers, and the CQ set can typically be read successfully from all eight tracks, even if several of the headers appear to have been corrupted because the CRC check fails. This redundant header information gives the rewritten data (which, by definition, is not at its nominal location within a dataset) a label that can be read very reliably, even in the presence of significant noise, so that the dataflow logic can put the rewritten data in its proper place within the dataset before outer-ECC correction is attempted.

Handling servo errors without stopping the tape

An LTO tape head spans a data band and the two servo bands that straddle the data band. Thus, the LTO head can use information from two separate servo bands to laterally position the data heads as accurately as possible over the data tracks in the data band. Even though an LTO head has simultaneous access to two servo bands, it is possible that neither can be read reliably. For instance, both servo regions may have suffered damage for a short stretch of tape, or both may be simultaneously obscured by a media defect. Alternately, the drive could experience some mechanical transient such as shock, which causes the read/write head to be displaced laterally with respect to the tape, and the servo pattern indicates that the head is beyond a predetermined *stop-write* threshold. These conditions can be referred to generically as *servo anomalies*.

In any of these cases, the write operation must be discontinued to guard against inadvertent overwrite of adjacent tracks. In a write-pristine format, any stop-write would have to be followed by repositioning and error-recovery procedures. With a write-pristine format, when the servo error is permanent it must be marked with a *servo demark* to prevent read error-recovery procedures from trying to recover data from a section of tape that has servo problems. Then, to maintain optimal write performance, the defective servo area should be "remembered" so the system can avoid encountering it all over again on a subsequent write. Here again the cost of a write-pristine format is poor performance and complexity.

In the LTO format, writing is stopped when a servo anomaly is encountered until both of the following conditions are met: At least one of the servo patterns can be read reliably, and the lateral position indicated by the servo patterns shows the read/write head to be within the stop-write threshold. At this point, the write operation can continue, and it can do so without repositioning of the tape. It may be necessary to rewrite some of the data written before the stop-write, but the key element is that the LTO format enables the writing to continue without stopping or repositioning the tape. Without this feature, a drive operated in a high-vibration environment might yield abysmal write performance.

Enabling robust reading

The reliable detection of format features such as the DSS, VFO, sync, resync, and reverse sync (as discussed in the RLL section), when used in conjunction with the precise LPOS data derived from the servo bands, allows the drive to position itself precisely with respect to datasets—much more precisely than any previous linear-tape product. This ability speeds search operations such as *space* and *locate*, and dramatically simplifies and enables rapid execution of any error-recovery operations required. This makes for very robust data processing.

As discussed above, the LTO format allows dynamic CQ rewrites to enable writing to continue even in the midst of data errors that exceed the power of the inner ECC code. The dynamic rewrite of a CQ may occur within a dataset (before the last CQ is written), but note that there is some latency involved in the RWW process, since the read head typically trails the write head by a distance that is more than one CQ and may be, conceivably, several CQs in length. Because of this, the CQ rewrite may not occur until after the write head has nominally finished writing the last CQ of the dataset; indeed, the write head may even have begun writing the first CQs of the next dataset. If so, the write of the new dataset is abandoned (the CQs are logically invalidated) and the defective CQ of the first dataset is rewritten, and this again nominally ends that first dataset.

This results in a dataset fragment that is effectively a logical continuation of the original dataset, but from a high level appears to be a completely different dataset. The LTO format also enables writing to continue after a servo error without stopping the tape. Discontinued writing in the midst of a servo error can cause old data to be left (not overwritten) in the middle of new data. Thus, to enable high write-throughput performance, the LTO format allows for rewritten CQ sets, dataset fragments, and old data to be left in stop-write gaps.

But the LTO format also has features that enable data-flow logic circuits to handle these cases reliably and on-the-fly in hardware, even on interchange. This is possible because of features such as the *dataset-number* and *write-pass number* fields that are written into every CQ header written to tape. These fields enable old data, or data that has been abandoned, to be read and simply discarded without ever even being transferred to buffer. The write-pass number is tracked in the CM, a much more robust way to track such a field than was done in previous formats because it can be done on-the-fly as the tape is being written. Thus, even if power is subsequently lost during a write operation, an update to the write-pass number in CM is remembered. Additionally, each codeword pair has a unique identifier contained in its associated header. The format is very robust because it has headers that are both written redundantly (16 to a CQ set) and contain all of the information needed to determine where the data belongs and whether it is valid. The strength of the LTO format is shown by its ability in three LTO drives from three different manufacturers to give excellent performance and high data integrity and interchangeability—even on a marginal piece of tape.

Streaming lossless data compression

The LTO format was designed to enable the highest performance possible, even for cases in which data was being read 512 bytes at a time from an HDD and was being written to tape transparently (without autoblocking). Many of the existent tape formats would display very poor capacity and performance in this scenario. As an example, some formats call for a 32-byte header to precede each host block and a 32-byte trailer to follow it. Others call for an eight-byte memory address pointer for every host block. Not only are these specifications inefficient from the point of view of capacity, but they can also reduce the write throughput because the embedded controller in the tape drive is consumed writing headers and trailers. In fact, this overhead can dominate the write throughput in these cases.

What was desired was a processing algorithm that eliminated the need for headers and trailers. For example, if reserved codewords were used by the compression algorithm, these could be put to good use as record boundary markers. Similarly, a *file mark*—which most, if not all, previous formats had required to be written as a separate dataset—could be a single reserved codeword. It was also desirable to solve another problem suffered by prevalent data-compression algorithms such as ECMA-151 and ECMA-222: the expansion of incompressible data. That is, compression algorithms used by tape drives typically reduce the number of bytes, but only if the data has redundancy that enables compression. If the data has little redundancy because it was previously compressed or encrypted, or was simply random, it does not compress; in fact, it expands.

For example, when fed completely incompressible data, one of these algorithms expands it by 25%, the other by 12.5%. One tape-drive format offered a solution to this, calling for monitoring the size of each record after compression. If the compressed record was larger than the original record, the original (uncompressed) record was recorded, and a flag to indicate this was set in the header of the record. Presumably, one could either save a copy of the original uncompressed record or decompress the compressed output to get it. However, depending on how it is implemented, this is inefficient from the point of view of either the processing or the buffer. It also does not allow for the case in which the data within a record may have both compressible and incompressible sections because it requires that the entire record be output compressed or uncompressed.

For LTO, one of the TPCs proposed a dual-scheme data-compression algorithm whereby the data could be output compressed either through a primary compression scheme or through a second scheme that is not capable of data compression but passes incompressible data through without expansion. The transitions between one data-compression scheme and another would be marked by reserved codewords in the compression algorithm, enabling a decompressor to always know in which scheme data was encoded. Thus, reserved codewords are used to mark record boundaries, file marks, compression-scheme swaps, compression-history resets, and for several other cases.

In all, eight reserved codewords were used to denote various transitions or special cases. These reserved codewords were then embedded into the compressed data stream as appropriate by the data-compression algorithm, and they are thus referred to as *embedded codewords*. The original proposal was made with DCLZ from ECMA-151 as the primary compression algorithm. However, a better data-compression algorithm was readily available in ECMA-222 (ALDC), and this was, at the insistence of another TPC, adopted in place of DCLZ as the primary data-compression algorithm. Casting the concepts of dual-scheme data compression and embedded codewords onto ALDC was not a trivial extension of the original proposal, in that DCLZ and ALDC are quite different data-compression algorithms, but a new, truly eclectic data-compression algorithm (ECMA-321) was born. It marked record, file mark, and compression-scheme boundaries with embedded codewords that supported ALDC as its primary data-compression algorithm, and it provided a secondary pass-through scheme that enabled incompressible data to be output without expansion. This new algorithm was adopted as the LTO data-compression algorithm and standardized under the name of *streaming lossless data compression* (SLDC).

It should be noted that because SLDC supports dual data-compression schemes and embedded codewords, a great deal of flexibility has been added, but it also creates a case in which there is no single right way to compress a set of data with SLDC. One company's implementation may choose to compress each record in its entirety in one scheme or the other (i.e., it will not perform a scheme swap inside a record). Another implementation may choose to let its data compressor adapt to the data by looking at how data compression has been going, and then swapping to the pass-through scheme if the most-recently-compressed data actually expanded, or vice versa. This is a reactive form of scheme adaptation. And finally, a third company may choose to look ahead at the compressibility of data and, if it sees incompressible data coming, swap to the pass-through scheme at exactly the right time to avoid data expansion. This might be called *optimal* (or *feed-forward*) *scheme swapping*, and it should systematically yield the best data-compression results of the three. IBM chose to implement its SLDC compressor using optimal scheme swapping.

In LTO Gen 1, each TPC implemented a different one of the three scheme-swapping methods just discussed. Thus, each company's drive potentially created a different compressed data stream out for a given set of data in. This is not a problem for data interchange, however, because each compressed data stream is decompressible by the rules of SLDC, and thus each company's decompressor can decompress the compressed data stream created by the other two compressor implementations, as was shown in data interchange among LTO Gen 1 devices of different manufacture.

Challenges for the LTO generation 3 and 4 formats

A number of format extensions are needed to address the next two generations of the LTO format. For one, the LTO roadmap calls for the number of simultaneous tracks to be doubled from eight to 16 tracks as we move forward to Gen 3. This will double the native transfer rate without increasing the tape speed. Additionally, it may be necessary to increase the minimum ECC interleave unit (dataset) size recorded to tape, because it becomes smaller with each areal density increase, making it potentially more susceptible to being unreadable due to a media defect or scratch of a given size. Additionally, as the capacity is doubled with each generation, the areal density will be pushed higher, likely at a faster rate than improvements can be made to the media coating. If so, the SNR will decrease with each new generation. This will likely require the enablement of more advanced read-channel techniques beyond simple PR-4, such as those used in HDD. It also seems likely that the power of the ECC dedicated to correction of the background (non-burst) errors will have to be increased to provide the same levels of data protection.

Open tape format

The LTO Gen 1 format was standardized in June 2001 (ECMA-319), though the format was made available much earlier than this through open licenses to any drive maker wishing to make an LTO tape drive that would allow data interchange with the LTO drives made by the three TPCs. Similarly, the LTO Gen 2 format is available through open licensing. The LTO trademark, however, is granted only after the developed tape drive has passed a stringent LTO-format compliance-verification process designed to preserve data interchangeability among LTO tape drives [5].

Summary and conclusion

Some of the more salient features of the LTO format have been described, especially those features that have helped make LTO the preeminent format in the mid-range tape-drive marketplace over the last two years. The LTO format enabled the super-drive aspects of LTO tape drives, which in turn enabled that success. It made possible tape drives with previously unheard of performance and cartridge capacity. It is an eclectic collection of the best ideas brought forward by HP, IBM, and Seagate for a best-of-breed mid-range tape format. Encoding absolute position into the servo pattern is one of the most fundamental of these format features, and many aspects of the LTO format were built on this cornerstone. Having a silicon cartridge memory in each LTO cartridge is similarly fundamental and enabling. The use of other features, such as on-the-fly write skipping, improved write performance. Features such as shingling support a roadmap that enables write-backward capability. The use of a true cross-product ECC enabled areal densities previously unknown in mid-range linear tape drives. These format features have also been contrasted with those of previous and competing products based on other formats. The three TPCs worked together with the mind-set that we would develop a best-of-breed format, and our efforts led to just that—a truly eclectic combination of the best ideas the three companies had to offer.

Acknowledgment

I acknowledge my LTO logical format technical working group counterparts who worked with me to create the LTO logical format: HP's Paul Bartlett of Bristol, England, and Seagate's Greg Unruh of Costa Mesa, California. It was a rare privilege to work so cooperatively with them in search of the best possible format. I also acknowledge my many colleagues in IBM Tucson who helped to create the logical and physical LTO tape formats, including especially Eric Christensen, Jud McDowell, and Paul Seger. Significant contributions were also made to the LTO format by IBM Almaden, including especially those by Tom Albrecht, Mario Blaum, Jim Eaton, and Brian Marcus.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Hewlett-Packard Corporation or Seagate Corporation.

[†]Ultrium, Linear Tape-Open, and LTO are registered trademarks of International Business Machines Corporation, Hewlett-Packard Corporation, and Seagate Corporation in the United States, other countries, or both.

References

1. R. C. Barrett, E. H. Klaassen, T. R. Albrecht, G. A. Jaquette, and J. H. Eaton, "Timing-Based Track-Following Servo for Linear Tape Systems," *IEEE Trans. Magn.* **34,** No. 4, 1872–1877 (July 1998).
2. E. R. Childers, W. Imaino, J. H. Eaton, G. A. Jaquette, P. V. Koeppe, and D. J. Hellman, "Six Orders of Magnitude in Linear Tape Technology: The One-Terabyte Project," *IBM J. Res. & Dev.* **47,** No. 4, 471–482 (2003, this issue).
3. B. H. Marcus, G. A. Jaquette, J. J. Ashley, and P. J. Seger, "Run Length Limited Encoding/Decoding with Robust Resync," U.S. Patent 5,969,649, October 1999.
4. W. G. Bliss, G. L. Dix, N. N. Heise, D. N. Moen, and G. C. Thomas, "Error Correction Code," *IBM Tech. Disclosure Bull.* **23,** No. 10, 4629–4632 (March 1981).
5. For additional information on LTO technology, see *http://www.ultrium.com/*.

Received February 11, 2003; accepted for publication March 18, 2003

Glen A. Jaquette *IBM Systems Group, 9000 South Rita Road, Tucson, Arizona 85744 (jaquette@us.ibm.com).* Mr. Jaquette is a Senior Technical Staff Member working in the Tape Technology and Drives area. He graduated from the University of California at Davis in 1983 with a B.S.E.E. degree and received his M.S.E.E. degree in 1988 from the University of Arizona, with major emphasis in signal processing. He joined IBM in 1983, working on 3380 hard disk drives at IBM San Jose. He transferred to the IBM Tucson facility in 1986 to work on optical disk drives in the read/write channel group. Since 1996 he has been working on tape drives. Mr. Jaquette holds 39 U.S. patents and has eight patents pending. He received an IBM Outstanding Technical Achievement Award for his contributions to the creation of the Linear Tape-Open (LTO) logical format and the electronic architecture of the IBM first-generation LTO tape drive, Ultrium-1. He is currently responsible for the architecture of future IBM tape drives.