SOFTWARE TECHNIQUES FOR RECOVERING NOISY TELEMETRY

John E. Sweet
Harlan H. Holmes
Rockwell International Corporation
Autonetics Electronic Systems Division
3370 Miraloma Avenue
Anaheim, CA 92807

INTRODUCTION

Software techniques for data quality and useability enhancement are used at two steps in the processing of PCM (Pulse Code Modulation) radio telemetry. The first is a software group synchronization which is used where traditional method has failed. The other is a tool for producing a single best quality data file from diverse receivers.

Recovering even small segments of valid information from noisy signals may be of major concert. The importance in many applications is because poor signal power is induced by events of great interest such as failure, detonation or exhaust gas dynamics. The radio receiver and bit synchronizer perform nearly optimally in processing of low signal to noise transmissions. It is found that the group synchronization process can be improved with software algorithm.

It is convenient to merge available data from a single test into a single file of best available data. Detected signals are recorded at dispersed tracking stations with varying signal quality over time. Upon achieving the best data from each tracking source the reconstructed data from a collection of all sources is further merged. By using known content to detect bit errors a single file of best quality data is available for analysis.

Comparative performance data from use on ICBM telemetry is included. A missile is an example of application where the data recovery is particularly critical at events such as staging and launch.

GENERAL

The overall objective is to produce a single file of the highest quality data possible from an ICBM Flight Test, typically about 15 minutes duration.

Figure 1a shows the hardware configuration. The software, written in Fortran and C, runs on an HP 9000/850 under UNIX. HP furnished a custom designed interface and driver for the Loral Data model 8330 Decom.

It is noted that this is not a "real-time" type processing system and thus takes advantage of this fact with the data manipulations by software algorithms discussed later.

Processing begins with an operation known as data formatting. Formatting consists of two "passes" of tape playback, one with the Decom set for "normal mode" and subsequently set for "bitblock mode". The normal Decom setup uses the usual SEARCH/LOCK strategies blocking data on frame boundries. Output data are of course both word and frame aligned and are time tagged. Transfer to computer file occurs only when in the LOCK mode. Therefore when the Decom drops to SEARCH mode, due to loss of Frame sync, discontinuities occur in the output data known as "gaps". It is the purpose of the second playback pass with Decom set for bitblock to provide data for the software to fill these gaps. The bitblock Decom setup bypasses the SEARCH/LOCK strategies by setting the entire FSP to "don' t care" and therefore keeping the Decom in LOCK and passing 100% of the bits from the Bit Sync. These data are output in time tagged blocks of frame length but of course are not blocked on either word or frame boundries.

The end result of formatting is two or more data files. One is the normal data file which may or may not contain gaps. If there are gaps one or more bitblock files are necessary.

The next step in the processing operations is "filling" the gaps in the normal data file with bit strings from the bitblock files producing an optimized data file which is used for subsequent data reduction and analysis. This operation starts by scanning the normal file for gaps. When a gap is detected the Group Software Synchronization Algorithm, discussed later in detail, kicks in. The gaps filling operation continues until the end of the normal file. Figure 1b shows a typical gap in the data file and the corresponding bitblock file from which filler bits are selected.

Figure 2 plots a parameter with and without software sync processing. This parameter was chosen to best illustrate the recovery of "lost" data but is not really a good

example of the value of software sync. Here the bad data points are obvious. With many other parameters they are not.

SOFTWARE GROUP SYNCHRONIZATION ALGORITHM

A software algorithm provides PCM group synchronization where standard techniques (hardware synchronizer/decom) have not succeeded. The portions of the data stream where software synchronization is needed are identified as gaps by the hardware synchronization processing phase. The software synchronizer reprocesses and reconstructs the data in these gaps then merges the hardware and software processed data to form a single dataset. The more deliberative non real time software acts on a PCM stream which has been bit blocked into a file, (ie only bit decisions have been made, no group synchronization) therefore no data has been discarded.

The software synchronization process adapts to the lack of data quality by applying more rigorous algorithms as degradation occurs. The simplest is a hardware like pattern match at predicted locations in forward and reverse direction. Signifcantly, the reverse search is for sync patterns by "backing into the gap", ie, from good data where group synchronization has been reestablished. This picks up data frequently discarded by the hardware synchronizer. The most rigorous involves full searches with weighted selection based on candidate sync pattern errors and bit count spacing from other candidate sync patterns. The data frames are reconstructed to correct length by adjusting (throwing out or adding dummy data), this is also done for bit slips which are handled by the conventional hardware front end.

A data section for software sync (a gap) must begin and end with bits where a good hardware sync was achieved, this is to assure that the data can be correctly matched up when it is merged back into the hardware data source file. The data is not aligned in any way to data patterns or boundaries, it is a pure string of bits across a buffer in memory. The size is constrained to what will fit in main memory, this simplifies the coding and speeds execution, the system used here can handle 20 seconds of 1.5 Mbit/sec PCM. Time of day values are associated with certain bits and fine time correlation is maintained, the details of this are spared the reader.

The following parameters used: (With FORTRAN style notation.)

FRLNG	—	Primary telemetry frame length.
RANGB	—	A range either side of anticipated sync pattern location which is possibly a valid location for the pattern (set at 6).
SPERR	—	Tolerance for allowed bit slip (set at 1)

BERR2 — t errors in sync pattern (set at 1).
PATSZ — The length of frame synchronization pattern, bit count.

The algorithm starts functioning from the beginning of data just like a hardware device in search mode. It slides though all the data bits searching for two word sync patterns that are spaced exactly by FRLNG, because of the above rule, they must be found or an abort occurs. From there it continues similar to hardware device in lock mode. To conserve computation it searches only +/-RANGB space for sync bit pattern. It allows a BERR2 bit error count and +/-SPERR bit slip in this simple software process called Hop Mode. This of course may fail eventually, up to that point the data is synchronized. The bit count to each frame sync word which has been located is held in a pointer array. An identical process to this is done backwards from the end of the buffer, this should also fail to find synch prior to "butting up" to the forward search. This is not surprisingly called Reverse Hop Mode. Note that depending on how the hardware processing was set up the hop mode could fully sync on the gap data. In general the hop modes are a computationaly expedient method to get through clean data and find the very noisy data.

A single routine is used for several purposes and it is helpful to conceive its function. The Slider is given a set of bit patterns to search for over a specified range in the bit blocked array (eg. one pattern the group sync pattern). The patterns are compared to the data at every bit offset in the specified range. The routine returns the bit offset of the three best matches for each pattern and a count of the bit mismatches at each of these.

As background observe that scanning through a large number of bits (which telemetry always has plenty of) to find a typicaly short synchronization pattern will result in too many successes, particularly if even a few bit non match counts are permitted. It is essential to consider the spacing between candidate sync patterns to assure that a correct sync location is found. This can be thought of as lengthening the sync pattern. Software Syncronization algorithm makes more extensive use of these relationships than would be practical in hardware. (see Reference)

Between the Forward Hop and Reverse Hop success endings there remains a jumble of bits which we call the Unknown Space. This could represent as little as a pair of frames with one obliterated syncpattern to as much as many frames of data with intervening sections of few discernable patterns etc etc. The entire unknown space is searched for probable sync patterns. Due to implementation this is done in segments which overlap by PATSZ - 1 bits, this assures no pattern will be undetected because it spreads across two search regions. Thus each search range procedes from the last forward hop success, forward in segments of size FRLNG + PATSZ - 1. At the end of

the unknown space if the bits remaining are less than FRLNG / 2, they are not searched. The picture at the completion of this search is for each range segment there are three candidate sync locations (in terms of bit offset from file start) and each is accompanied by its bit error count, when compared to the sought synchronization pattern.

In addition to bit errors and bit slip (where bit synchronizer throws away or adds a bit clock), in the unknown space we may have to contend with bit sync loss. The phase lock device loses its track of the transmitted bit phase. This may appear as many extra or missing bit clocks (actual bits) in the Unknown Space. The algorithm assigns a value to the mean frequency error;

FANGS — mean of possible bit sync frequency error rate (use .012).

The trivial case for unknown space is where its length is in the range of;

2 * FRLNG +/- (2 * FRLNG * FANGS)

(ie. within the drift range, two frames long). In this case the best pattern location in the middle section of that range (half the above range is chosen), if none were found the middle of the range odd bit adjusted is used. The pattern evaluation criteria is as given in the next section.

The nontrivial case is where the unknown space contains 3 to N frames of data. The sync pattern information is processed forward from the beginning of the Unknown Space, in fact the software sorts the information to accomplish this. The algorithm is now searching for the most probable frame boundary locations. A quality value is assigned to each candidate from the pattern search;

SYNCPNTS	—	An estimate of quality of the selection.
BPOINTS	—	Portion of SYNCPNTS awarded for low bit errors.
SPOINTS	—	Portion of SYNCPNTS awarded for spacing. SPOINTS.F is to a forward pattern and *.B is backwards.
TLEVELS	—	Points required to stay "synced" (set at 50).
TLEVELR	—	Points required for return to "synced" (set at 70).
BERR1	—	Maximum admissable bit errors (set at 10 for 32 bit pattern).
ERR	—	Bit error count for particular candidate.
BRW	—	Weighting factor for bit error term (set at 5).
SPERR	—	Difference from correct frame length spacing between two candidate sync patterns found by pattern search.

DRIFT	—	Frame spacing normalizer (set to .01).
BPOINTS	=	(BERR1 - ERR) * BRW IF ERR < BERR1 ; ELSE BPOINTS = 0
SPOINTS	=	((((FRLNG * DRIFT) - SPERR)) / (FRLNG * DRIFT)) * 25
SYNCPNTS	=	BPOINTS + SPOINTS.F + SPOINTS.B

Return again to the last forward hop look ahead for sync candidates that have a spacing from current offset of;

FRANGE = FRLNG +/- FRLNG * FANGS

Find the best candidate, if any, in the range it is the next candidate,

if candidate's

SYNCPNTS = > TLEVELS

set USNCH true, procede on through the file in this manner, unless SYNCPNTS falls below criteria (TLEVELS) and thus USNCH is set false. Now the bit synchronizer drift is assumed cumulative and the algorithm goes to an expanding search range for the sync pattern. The search range is function of frame count K, in non USNCH;

FRANGE = K(FRLNG +/- FRLNG * FANGS)

In this Search Mode if:

SYNCPNTS = > TLEVELR

set USNCH true and return to above Synch Mode. While in Search Mode continue to expand the search range. Note that if too high a bit rate drift is allowed that the algorithm breaks down for long drop out periods, in particular if;

K * FANGS = > .5

trouble ensues. Other anomalies may also occur, the system checks for those which have been identified, for brevity they are ignored here.

Now after this run through in trying to find sync the task is to estimate the transmited data positions. The procedure has yielded a set of best estimates of sync pattern

location in terms of bit offset from the start of the file, except in regions of Search Mode (as defined above) where a special flag is set. In cases where the estimates are spaced by FRLNG it is merely a tedious matter of shifting all the data to the desired word and output buffer boundaries. Where this ideal is unrealized bits must be slipped in or thrown out in order to achieve a nice output. At this point it is application dependent, the application we have used fixed data patterns such as counters or status flags in the data stream to decide which bits to throw out or where to insert a fixed pattern filler stream between the primary frame synchronization patterns. Where the Search Mode exists the data is blocked at size FRLNG until resynchronization is achieved, the last frame is then reconstructed based on the known patterns (working in reverse).

The software synchronized data is merged into the existing gaps with various sanity checks to assure no data duplication, or time gaps with no data have occured.

MERGING MULTIPLE DATA SOURCES

The premise to this task is that most data analysts care or know little about how the data was delivered to them. It is least traumatic if the ivory tower is delivered a single data source or file in computer parlance. A typical test may have many data sources, in particular in this case of an ICBM test, recordings may be spaced from California to Kwajalien, with each site using multiple radio reception and detection strategies.

The vagarities of signal propagation, antennae patterns, tracking error, test vehicle events, tape recording, operators (as in human), are contributors to the complexetity of a good solution.

Each viable source is processed by the formatting and synchronization strategy described above, whereupon there is a set of files representing the data of interest. The selection of the best data can only be poorly estimated by monotonically switching to each time slice source. It is necessary to look at all sources at each definable data segment and select the best at each segment. The definable segment is at worst a major data frame and for application dependent usage may be subdivided by known data patterns as mentioned previously.

The problems of merging data are solved by assuring that correct alignment is achieved and maintained. Two anomalies create the problems. One is erroneous time value associated with data, the other is missing or duplicated portions of data. It is extremely important to assure that data is correctly merged, errors may be very difficult to detect and have disastrous consequences.

The above problems are solved by using time only for gross file alignment and checking file alignment over entire overlap range. Each source file to be merged is to be assigned a record count offset which when added to each files record count will provide a value to achieve alignment. A starting point for the offset value is;

$$OFFSET = (REFETIME - FSTARTTM) / FRMPERIOD$$

REFETIME - Reference time for all files
FSTARTTM - A particular file start time
FRMPERIOD - Nominal major frame or record duration.

By using these offsets data from any pair of files can be approximately aligned. The final alignment is achieved by correlation of telemetered data values between all sets of files that overlap. The bytes to use for correlation should have dynamic content, and have low autocorrelation over the telemetry frames. The earlier file is called the base file, the other is the correlate file. The selected bytes from thee frames of the base file are compared to three records in the correlate file at eleven possible offsets around the range established by time. The quality of each offset is judged by the bit comparisons. This comparison process is repeated at selected intervals through the entire overlap period. The offset adjustment is established by the least bit comparison errors. With allowances for possible noisy data periods the exact same offset must be found at each of the check points. (see fig 3)

The same alignment process is done exhastively for all file overlaps. A further check occurs when a source overlaps two others , they both must achieve relation to the same base or an error is flagged. This process assures proper alignment and will find any missing or extra data because offset will change. Time is used only to get started and if this is wrong the correlation process will not find its minimum threshhold.

The data merging is now a trivial process of selecting the best data segments based on comparison to known data patterns of which we at least have a group synchronization pattern.

CONCLUSION

The software synchronization has been in use for over two years, the data merge system for many more years. They have been used for three deferent telemetry formats. The systems provide clear advantages in the analysis of ICBM telemetry and would be an advantage in any similar application. However if the data has a very low or very high signal to noise nothing will make any difference, the improvement only

occurs in a bit error probability range of 10e-2 to 10e-4 or arguably something around there.

The software synchronization process if implemented in a shared memory multiprocessor system could process real time data, but a delay of several frame times would be suffered. At this point one cannot speculate on the bit rate at which this might run. There has been no opportunity to obtain quantitative run speed measurements on the existing system and it has not been coded for speed. The hardware synchronizer is not dead yet!

There has been little analysis or experimentation on the weighting factors used for software sync. The algorithm was designed to be programmable and robust. It is most certainly not optimum but its robustness is demonstrated by extensive use.

ACKNOWLEDGEMENT

The programming for this Software Sync Algorithm was done by Mr. Tony Thoprakane. The initial algorithm was developed on Air Force contract F04704-84-C-0061. The software is in current use on Air Force contract F04704-88-C-0096.

REFERENCE

Cox, Timothy F, "Analysis Of An Adaptive Sequential Probability Ratio Test Type Of PCM Frame Synchronizer" ITC Proceedings 1986 Volume XXII.

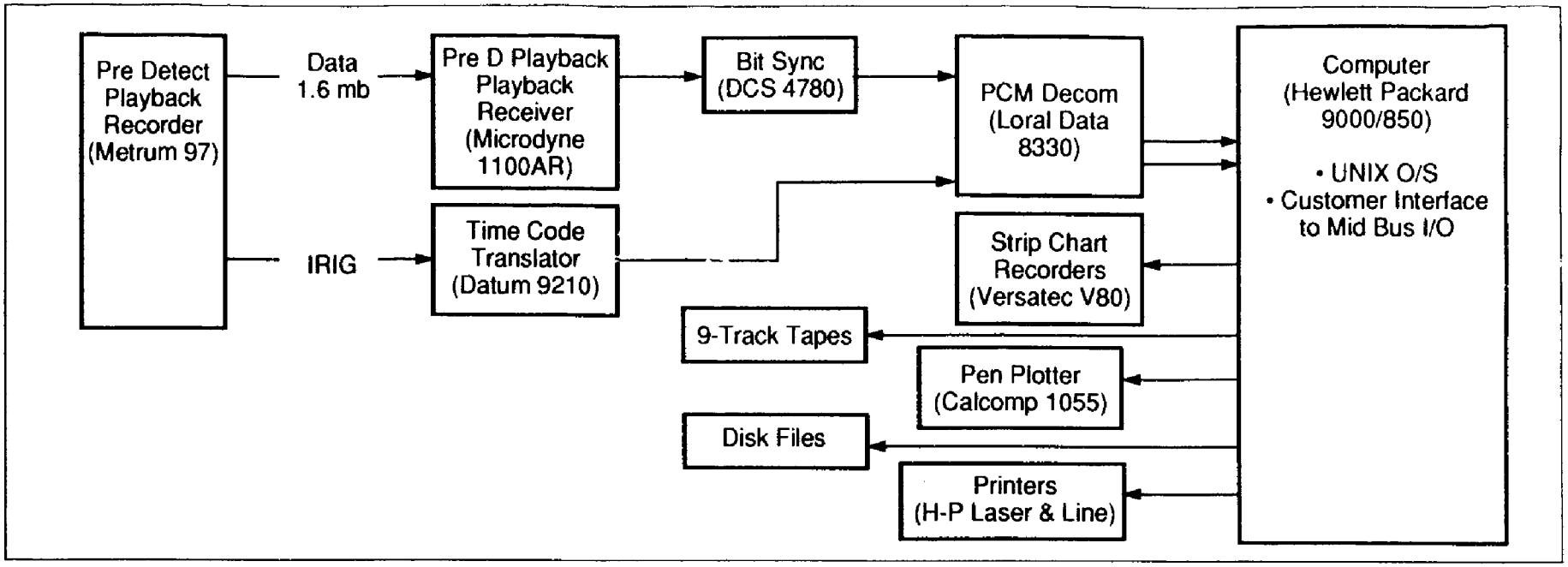
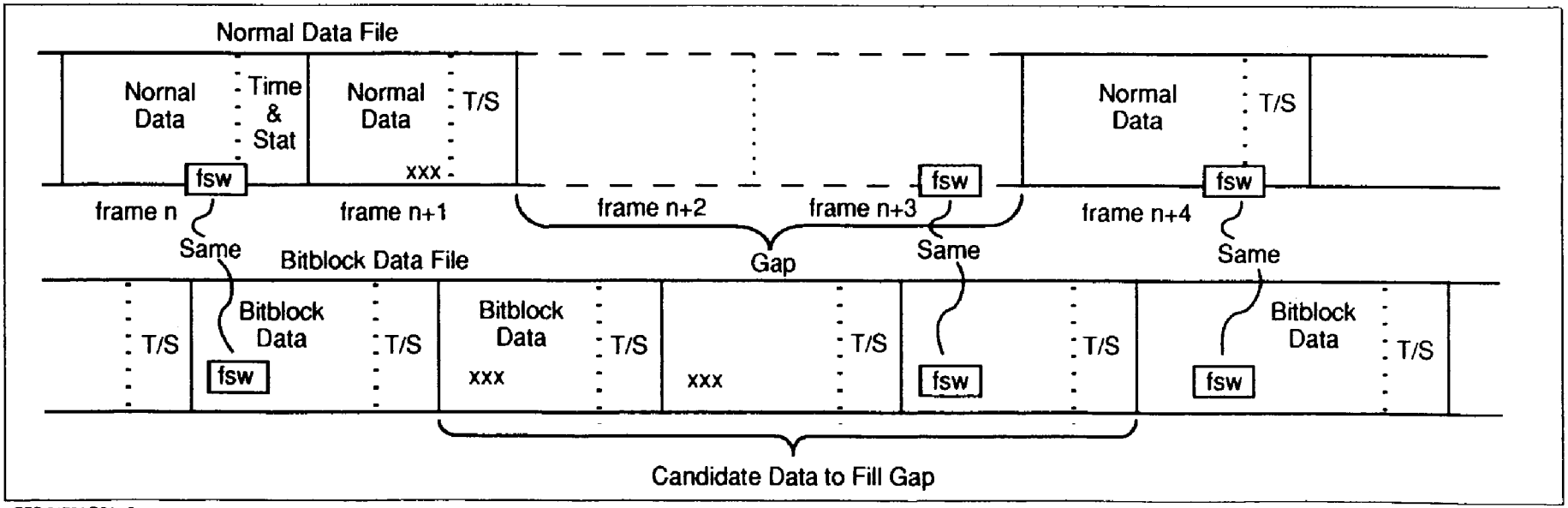


Figure 1a. Rockwell International DMS Lab Telemetry Data Processing System



EPS.24704.D01.v0

Figure 1b. Example of a Two Frame Data Gap

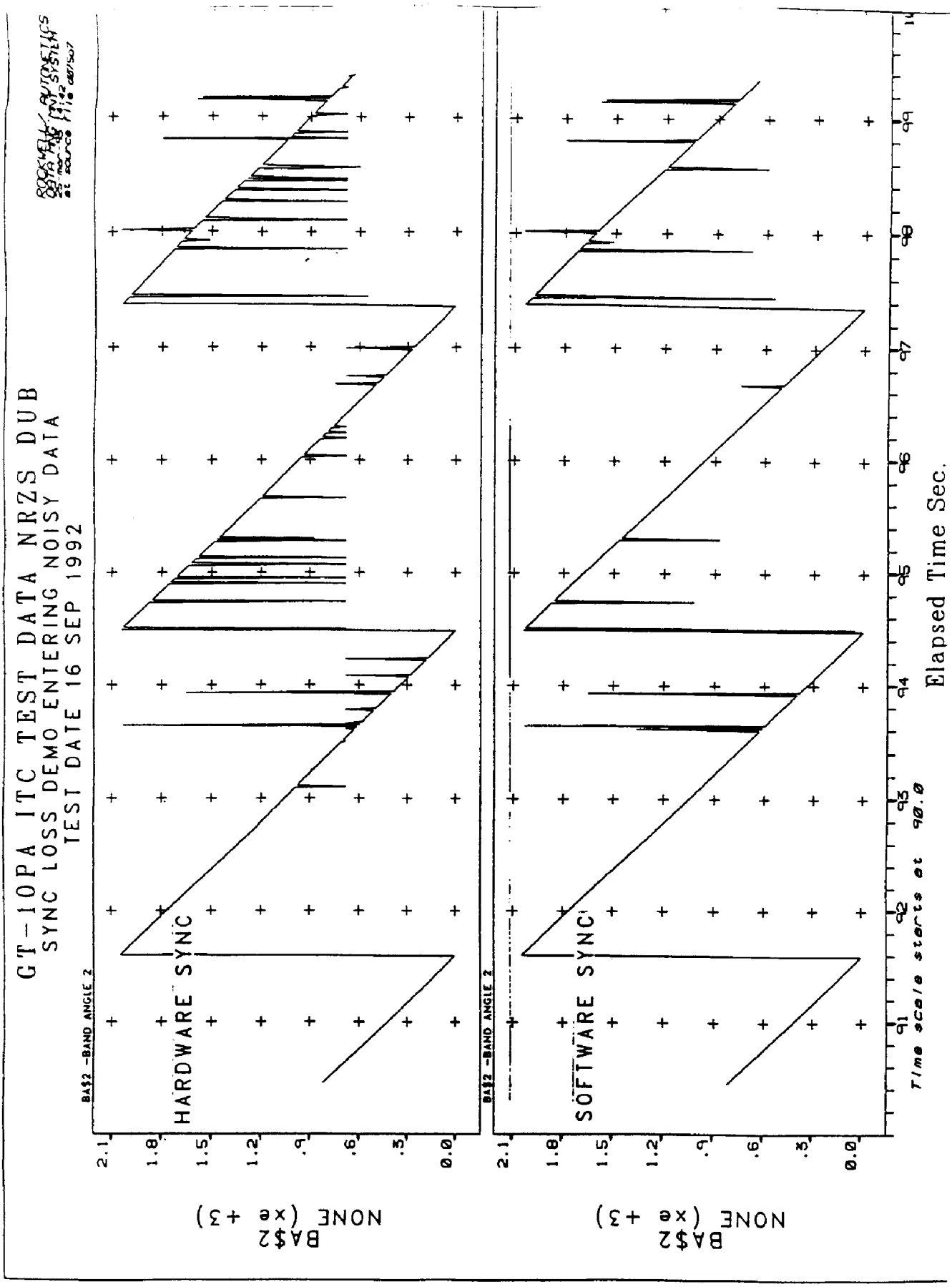

GT-10PA ITC TEST DATA NRZS DUB
SYNC LOSS DEMO ENTERING NOISY DATA
TEST DATE 16 SEP 1992
BA$2 -BAND ANGLE 2
HARDWARE SYNC
SOFTWARE SYNC
BA$2 NONE (xe +3)
Elapsed Time Sec.

Figure 2.

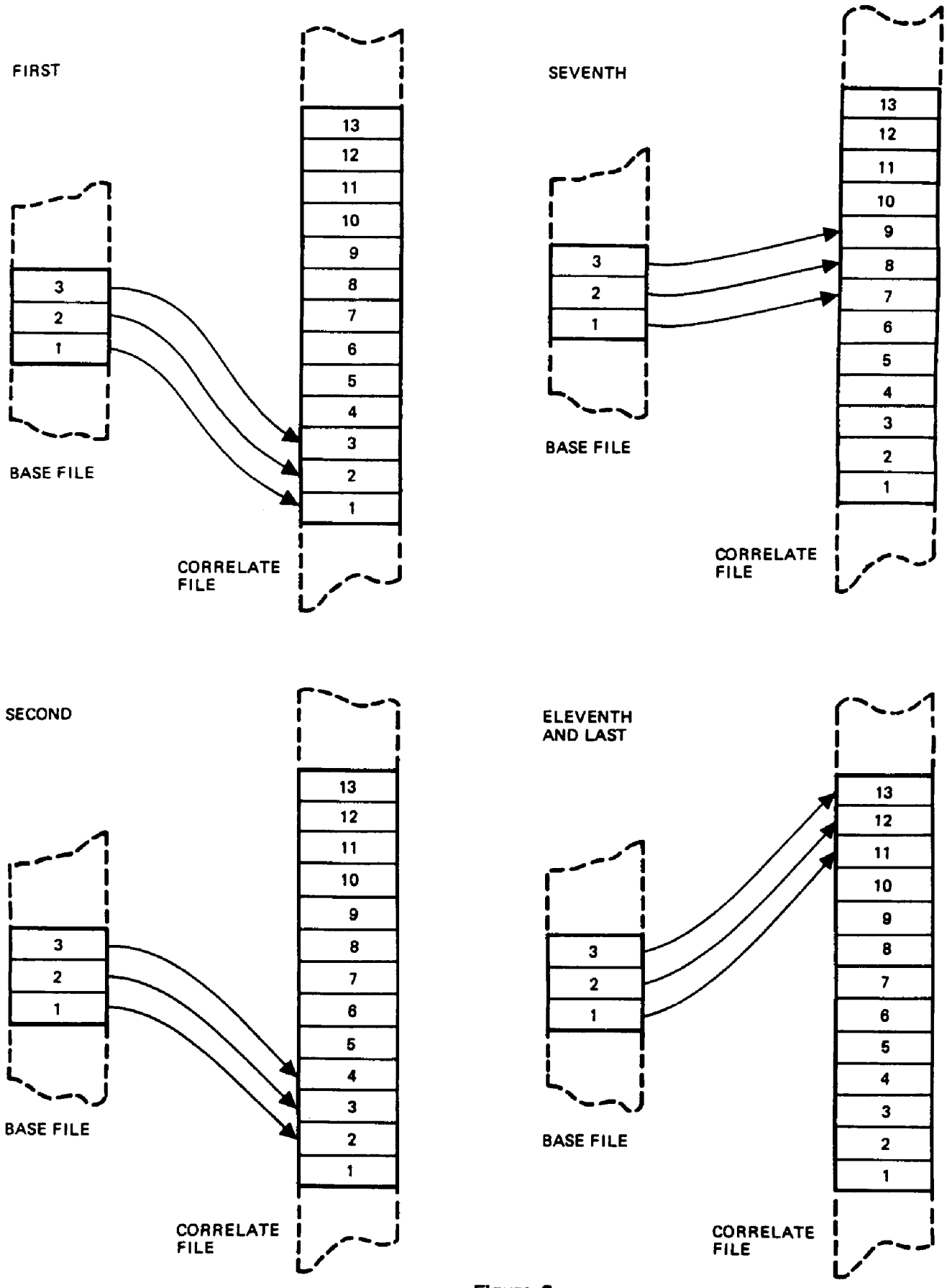

FIRST
3
2
1
BASE FILE
13
12
11
10
9
8
7
6
5
4
3
2
1
CORRELATE FILE
SEVENTH
3
2
1
BASE FILE
13
12
11
10
9
8
7
6
5
4
3
2
1
CORRELATE FILE
SECOND
3
2
1
BASE FILE
13
12
11
10
9
8
7
6
5
4
3
2
1
CORRELATE FILE
ELEVENTH AND LAST
3
2
1
BASE FILE
13
12
11
10
9
8
7
6
5
4
3
2
1
CORRELATE FILE

Figure 3.