Application Note 358-3

HP 5371A Frequency and Time Interval Analyzer

Time Domain Characterization of Magnetic Disk Drives

HEWLETT PACKARD

Table of Contents

Introduction 1

Key Contributions of the HP 5371A Frequency and Time Interval Analyzer 2

Magnetic Disk Drive Measurements 4

- Read Noise Characterization 5
- Write Noise (Transition Noise) Characterization 10
- Timing Asymmetry (Pulse Pairing) Characterization 11
- Peak Shift (Pulse Crowding) Characterization 17

Time Interval Results and Window Margin Analysis 25

Appendices 27

- A. Glossary 27
- B. Statistical Reminders 28
- C. Related Literature 30
- D. Program Listings 31
  - Read and Write Noise Program Listing 31
  - Timing Asymmetry Program Listing 35
  - Peak Shift Program Listing 42

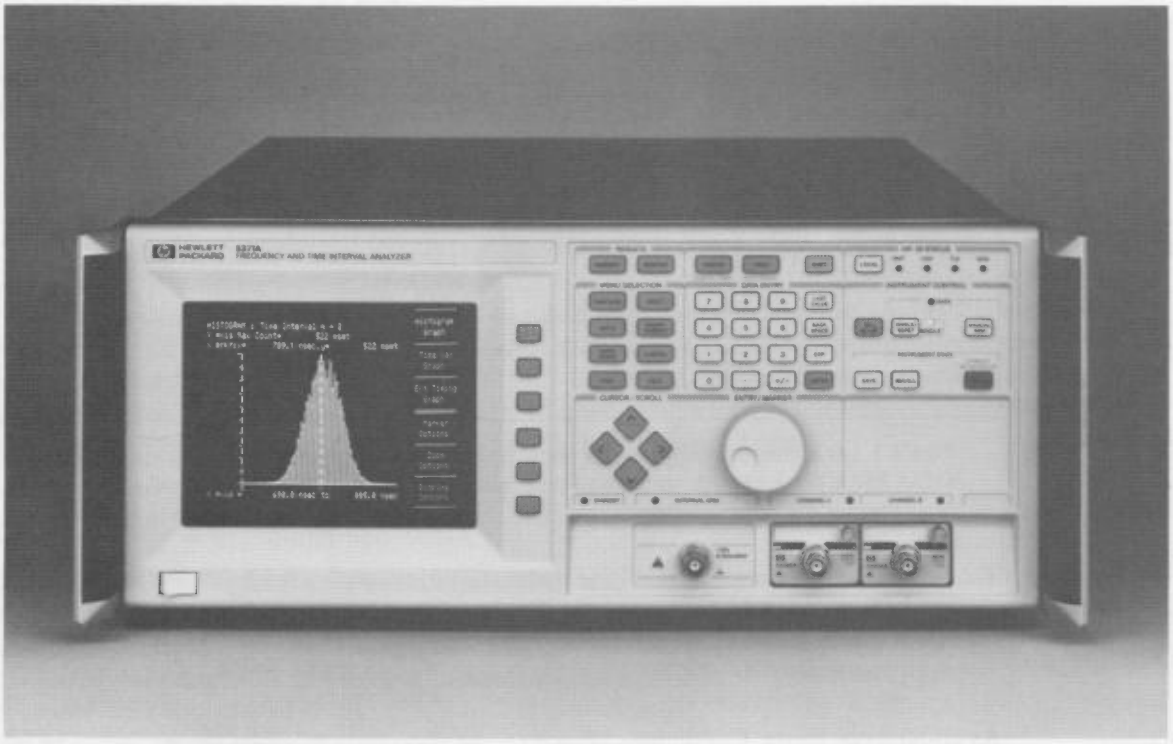
W1BG

Introduction

Characterization of timing in the disk drive read channel is a critical task for the drive designer. While a variety of parametric and time domain measurement techniques exist, timing characterization gives the designer the most relevant indication of the drive's ability to reliably store and retrieve data. The predominant time domain technique, popularly called "phase margin analysis", extrapolates the available timing window margin for a given error rate using error acceleration methods.

The HP 5371A Frequency and Time Interval Analyzer improves the information that the phase margin analyzer (PMA) provides. While the PMA excels at a rapid overview of the drive's aggregate timing performance, the HP 5371A can be used to break this aggregate timing information into distinct components: aggregate PMA noise into read noise and write noise; and aggregate PMA offsets into peak shift and asymmetry. With this information, the designer can now determine which sources of timing error limit the drive's error rate performance and focus on those with the greatest potential for improvement. For example, understanding that write noise effects are limiting phase margin noise performance can help the drive designer work more effectively with the media supplier to gain the greatest timing margin improvements.

This application note describes procedures to characterize read noise, write noise, timing asymmetry and peak shift of magnetic disk drives using the HP 5371A and a desktop computer. This note will concentrate primarily on the configuration of the HP 5371A, the computer algorithms, and the related "theory" of measurement implementation. For the reader who is new to the above timing effects or prefers a clarification of the definition of these, Appendix A contains a list of measurement definitions. Appendix B offers a brief summary of key statistical concerns and an example of the HP 5371A's measurement precision.



The HP 5371A allows the drive designer to break apart aggregate timing noise and offset effects to optimize error rate performance.

Key Contributions of the HP 5371A Frequency and Time Interval Analyzer

Continuous Measurement

The HP 5371A features an important breakthrough in the measurement of frequency or time interval. This new capability, termed "Continuous Measurement", offers unique performance benefits over other time interval measurement techniques. These benefits include not only dramatic increases in throughput, but measurements can be related to each other by time or events. It will be shown in this application note that the exploitation of the latter is key to several disk drive measurements.

A simple analogy will serve to illustrate the "relative" sense of continuous data. Consider the ruler in Figure 1a. It is easy to see that the distance between the 4-inch mark and the 3-inch mark is 1 inch (4 inches - 3 inches). In a similar fashion, the distance from the 6-inch mark to the 2-inch mark is 4 inches 6 inches - 2 inches). Clearly, the separation between any two points can be determined with the same measurement precision as between two adjacent points by finding the difference between appropriate inch marks on the ruler.

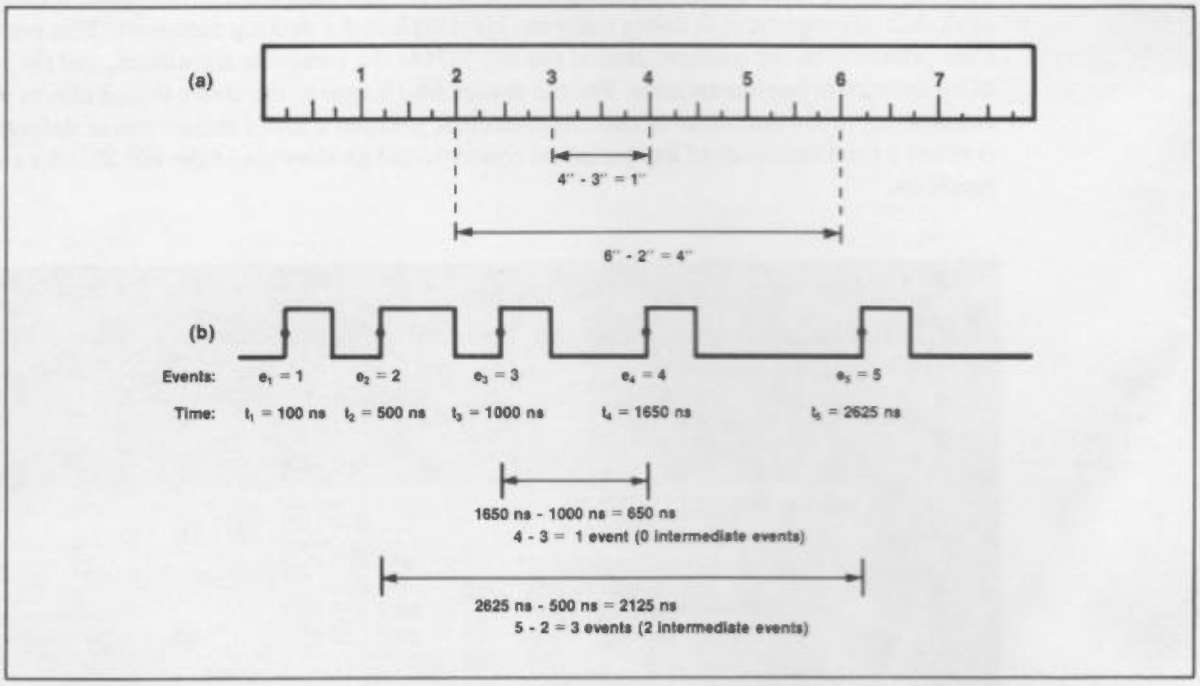


Figure 1. (a) A ruler analogy depicts the concept of the continuous measurement data format. (b) The HP 5371A keeps track of both events and time. Note that the time interval measurement precision is constant, regardless of the pulse spacing.

The HP 5371A uses several specialized count registers to keep track of both time and event data. These registers are never reset during a series of measurements and so the memory is filled with time and event "samples" analogous to the inch marks of the ruler. Figure 1b illustrates how the time and number of events between any pair of these time samples can be determined without a loss in precision.

As pointed out above, Continuous Measurement also offers excellent throughput. The minimum time to store time and event sample information is 100 ns. Therefore, the HP 5371A measures EVERY specified data event up to a 10 MHz rate. It is important to note, however, that even if events occur faster than this 10 MHz rate, the HP 5371A continues to count events and time. The data reflects precisely which events are measured (or "time sampled") and which are not. This capability will be shown later to be especially important for the characterization of timing asymmetry and peak shift.

Flexible Arming

The HP 5371A offers a wide selection of arming and triggering features to control when time samples occur. This built-in arming capability reduces or eliminates the need for specialized outboard hardware circuitry to arm the HP 5371A.

Measurements may be acquired in groups (or blocks) of up to 1000 measurements (up to 4095 measurements per block are available when using the HP-IB binary output mode). The arming configuration is specified in a two-stage fashion as "block holdoff/measurement sample." The first term describes the sequence which is required to begin the group of contiguous measurements. The second term describes the condition which will cause each measurement sample to occur within a block.

In general, block holdoff arming may be defined as follows:

- Automatic - begin the measurement block as soon as possible.
- Edge Holdoff - begin the measurement block after the occurrence of a signal edge.
- Event Holdoff - given a reference signal edge (such as an index pulse), delay by some number of events and then begin the measurement block.
- Time Holdoff - given a reference signal edge (such as index pulse), delay by some amount of time and then begin the measurement block.

Sample arming may be defined as follows:

- Automatic - sample time as quickly as possible.
- Edge Sampling - sample time after the occurrence of a signal edge.
- Cycle Sampling - sample time after a specified number of cycles of the input signal.
- Interval Sampling - sample time after a specified time (continuous).
- Time Sampling - sample time after a specified time (non-continuous).
- Event Sampling - sample time after a specified number of events (non-continuous).
- Parity Sampling - sample time after the occurrence of a pair of start and stop events.[1]

Any of the three input channels (channel A, channel B, or External Arm) may be used for arming configurations.

Fast HP-IB

The HP 5371A features excellent HP-IB performance with binary output rates to 20,000 measurements per second.[2] In addition, a choice of three output formats is available: ASCII, floating point, and binary. The binary format offers the time and event sample information which will be used in this application note for timing asymmetry and peak shift characterization. The IEEE double-precision floating point format matches the numeric format of the HP 9000 Series 200/300 desktop computers. This latter output format simplifies I/O operations for computers with the same internal data format. The floating point format will be demonstrated in the read and write noise program example.

Measurement Precision

Measurement precision is fundamental to the integrity of time interval data. The HP 5371A features 150 ps rms resolution for single-shot measurements, 2 mV trigger level resolution, and a 500 MHz input bandwidth. Input bandwidth and voltage triggering precision, as well as measurement resolution, are critical for quantification of nanosecond and sub-nanosecond noise components found in high performance disk drives.

[1]The arming capabilities are also related to the measurement mode. A comprehensive description of the HP 5371A's arming capability can be found in the HP 5371A Product Note/Specification Guide and the HP 5371A Operating and Programming Manual. See Appendix C.

[2]The HP 5371A Product Note/Specification Guide lists HP-IB benchmarks for the various output formats, and a binary data processing algorithm for this benchmark.

Magnetic Disk Drive Measurements

This section will describe the configuration of the HP 5371A for four types of measurements:

- **read noise**
- **write noise**
- **timing asymmetry**
- **peak shift**

A discussion of measurement theory, including key assumptions, is included. Program examples to process the HP 5371A data are also presented. For illustration purposes, these programs are coded in "Rocky Mountain Basic", available on HP 9000 Series 200/300 desktop computers. Copies of these programs can be obtained on floppy disk by returning the reply card in this application note.[3]

Measuring transition-to-transition and compensating for correlated noise

The measurements discussed will be configured around measuring time intervals from transition-edge to transition-edge, rather than transition-edge to PLL clock-edge. This technique is proposed to eliminate uncertainty caused by PLL clock jitter when measuring transition-to-clock. However, the HP 5371A may be used to measure timing jitter in a transition-to-clock fashion if desired, and the example routines may be modified to suit this purpose. When using this latter technique, it may be desirable to use the HP 5371A to characterize the jitter on the PLL clock in order to remove that component from measurement results.

Certain precautions must be exercised when using transition-to-transition time interval measurement techniques. Isolated pulses found in magnetic disk drives in general exhibit "tails" that interact with adjacent pulses, especially when data pulses are spaced close together. The interaction of these tails with neighboring pulses causes timing jitter or noise to be correlated between adjacent pulses. In other words, the noise on the current pulse is, in part, influenced by the preceding pulse in the read channel. For this reason, the algorithms described will take precautions to avoid measuring time intervals between adjacent pulses, and in fact will measure between relatively "distant" pulses to avoid this noise correlation. This precaution is easy to implement given the arming capabilities of the HP 5371A and the continuous measurement data format.

For all of the measurements discussed in this application note, it is assumed that the HP 5371A is connected to the read channel at a point after the zero-crossing circuitry. In this configuration the HP 5371A measures digital logic signals with an RZ (return-to-zero) signal, rather than the bipolar read signal, or an NRZ (non-return-to-zero) signal. See Figure 2.

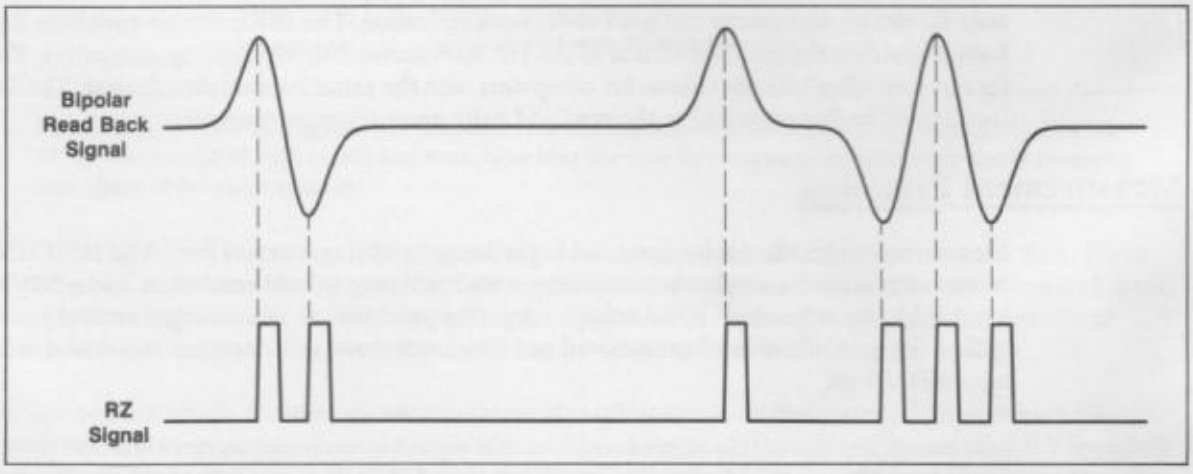


Figure 2. The HP 5371A is connected to the read channel electronics at a point after the zero-crossing detector to measure an RZ logic data signal.

[3]*This software is offered at no charge as an example of the techniques described in this application note. Software performance is not warranted by Hewlett-Packard.*

The configuration of the HP 5371A and external computer will allow read noise and write noise to be characterized from the same set of raw data. The following will describe each measurement separately, while referencing a single configuration and computer program.

Read Noise Characterization

Measurement Theory

The approach to read noise characterization is to measure a specific time interval on the disk repeatedly. The variation of these repeated measurements describes the effect of noise in the system on readback timing. Since only a particular interval is measured, write noise effects are removed. In other words, perturbations due to the write process are fixed for a specific interval; the variation of successive measurements of a particular time interval is due to read noise.

The standard deviation is a means of quantifying these variations. Note that a time interval measurement includes the variation of both the starting transition edge of the time interval measurement and the stopping edge. Assuming that noise affects both edges equally, the variation should be divided equally between starting and stopping transition edges. This is accomplished by dividing the statistical variance by 2, or the standard deviation by $\sqrt{2}$. Appendix B offers a reference for key statistical equations and tables.

Read noise measurements are dependent on the particular read channel electronics and the measurement instrumentation. The measurement resolution of the HP 5371A must be accounted for to obtain the true measure of read noise. Adding the HP 5371A resolution to the read noise term in a "sum-of-the-squares" fashion results in the following equation:

$$(\text{Measured Result})^2 = \frac{(\text{HP 5371A Measurement Resolution})^2 + (\text{Actual Read Noise})^2}{2}$$

This equation can then be solved for the actual read noise. Appendix B provides an example calculation of HP 5371A measurement resolution.

Read noise will generally be the greatest for patterns which smooth the signal peaks, resulting in a more gradual, or flatter, slope of the differentiated signal. Figure 3 depicts how this slower slewing signal will be more susceptible to noise at the zero crossing detector. Parametric measurement techniques (signal-to-noise ratio) can fail to identify this effect for signals with equal amplitude.

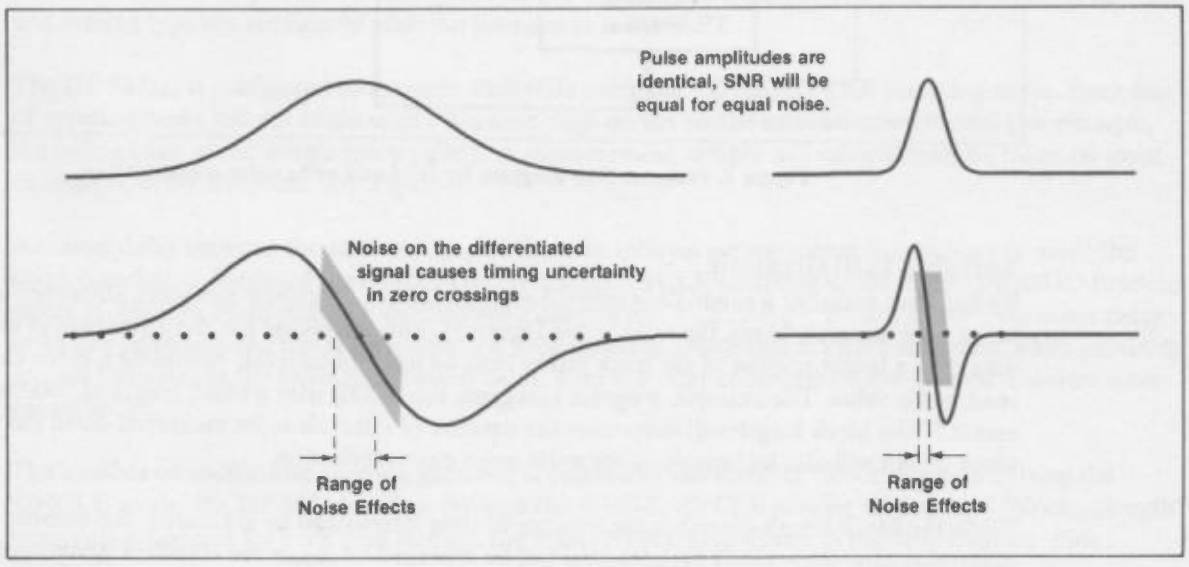


Figure 3. "Flatter" readback signal peaks are more susceptible to system noise than sharper peaks. Parametric SNR techniques can fail to predict this effect for signals of equal amplitude.

A constant frequency pattern is assumed in the example algorithm. In general, lower frequency patterns demonstrate greater read noise: pulse interaction causes lower amplitudes (resulting in degraded signal-to-noise ratio), but superposition effects actually serve to sharpen the pulse peaks. These sharper peaks result in a steeper differentiated signal and subsequently less noise susceptibility at the zero-crossing detector. These superposition effects are greatest when pulses are closely spaced. Low frequency patterns have smaller superposition effects and therefore demonstrate greater read noise.

Non-symmetrical isolated pulses such as those found in thin-film systems can cause complex inter-pulse relationships that create unexpected noise improvement or degradation for various transition spacings. The designer may wish to experiment with various frequency patterns to determine the worst-case read noise pattern for a particular system.

Measurement Configuration and Computer Algorithm

Figure 4 is a flow diagram of the program to characterize both read and write noise using the HP 5371A. Major blocks and key functions for read noise are discussed below.

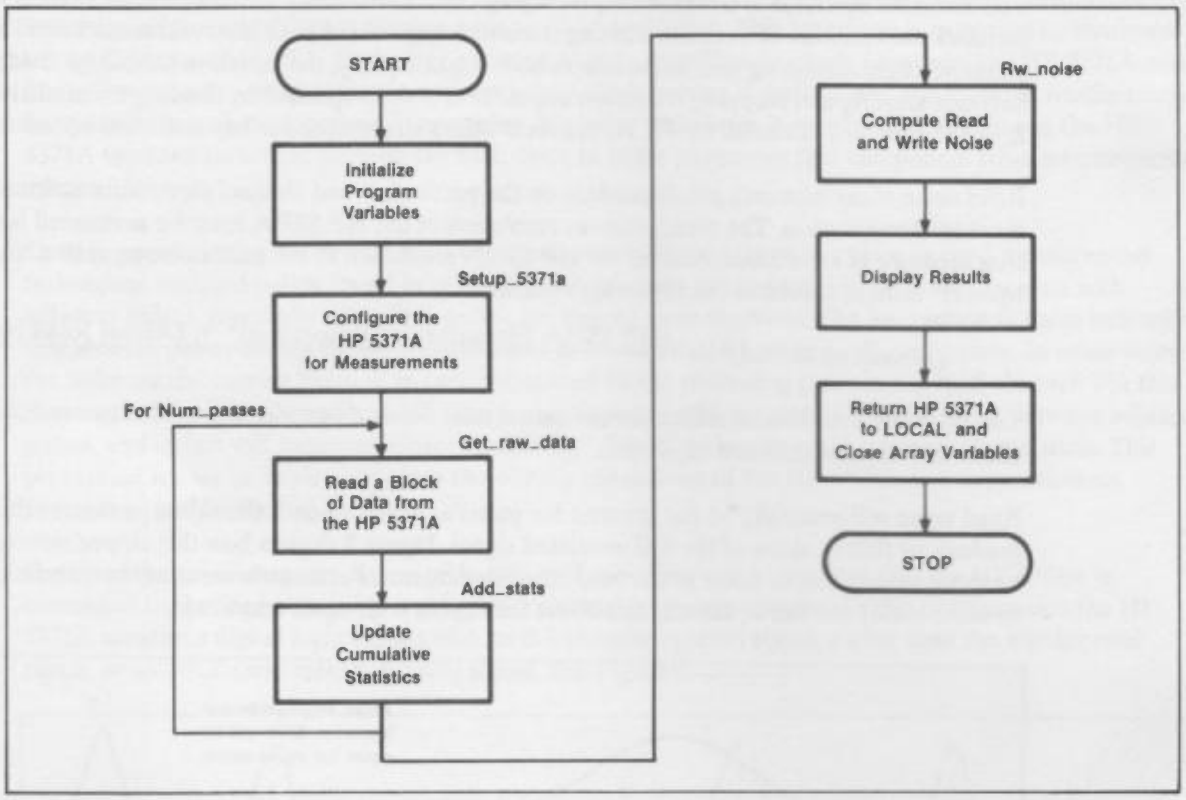


Figure 4. Program flow diagram for read and write noise computations.

Variable Initialization

Rather than measure a single time interval repetitively, the described computer algorithm will measure many time intervals "down the track" (See Figure 5). This technique has the advantage of characterizing data over a larger portion of the track rather than an isolated interval, providing a more characteristic read noise value. The example program (program line #330) uses a block length of 1000 measurements.[4] The block length will determine the number of intervals to be measured down the track. (The block length will also be important for write noise characterization.

While the block length describes the number of time intervals to be measured, the number of passes will determine how many times each interval will be averaged to obtain the standard deviation (sigma) of that particular interval. As few as 100 passes may demonstrate reasonable validity (variable Num_passes, program line #350).

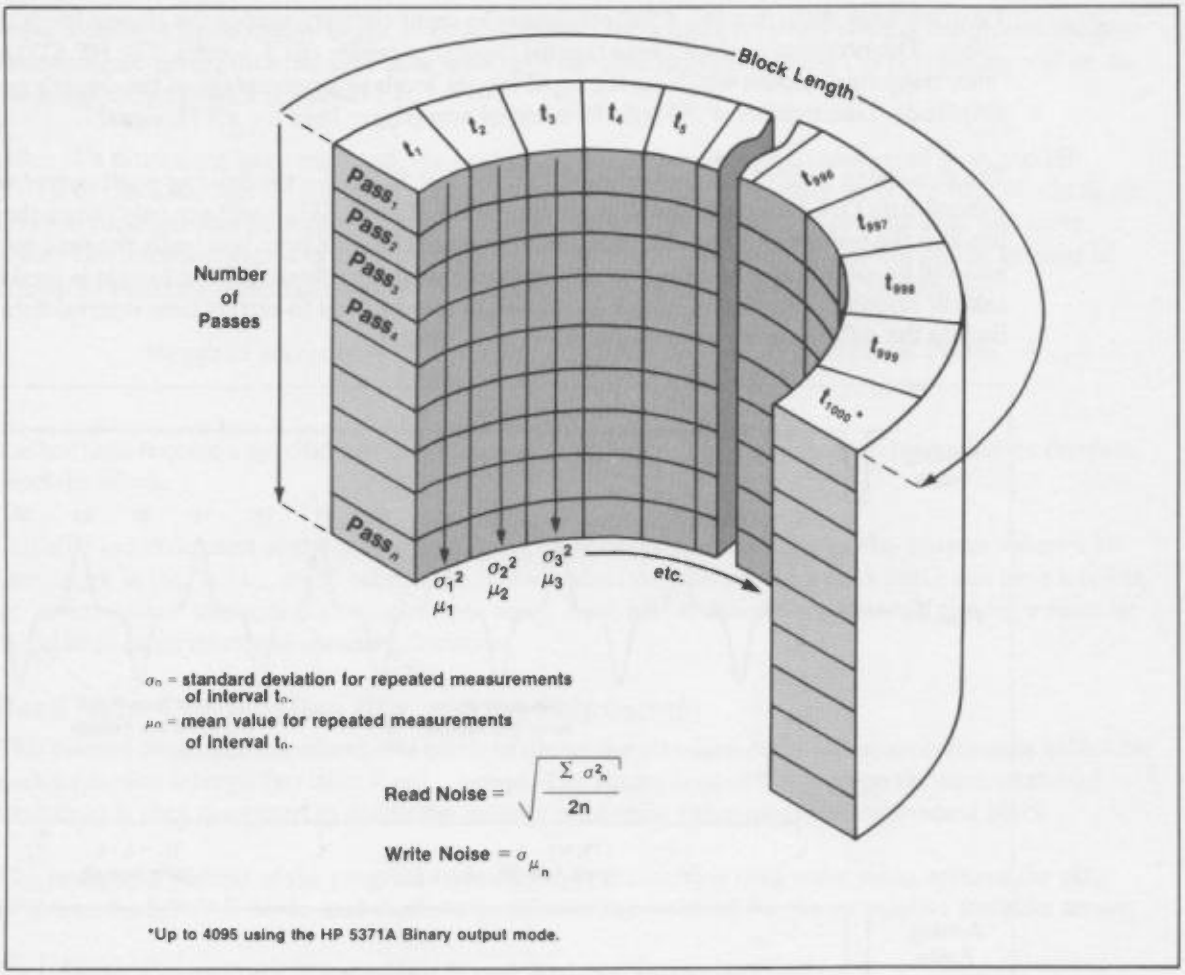


Figure 5. Read and write noise are characterized over many measurements "down the track"*. Multiple passes over this same portion of the track are also used.

HP 5371A Configuration (Setup__5371a Subroutine)

The subroutine "Setup__5371a" configures the HP 5371A for data collection. The measurement function and arming type are configured after the instrument is PRESET.

The HP 5371A is configured to measure PERIOD using the EDGE/CYCLE sampling mode. Each block of measurements will not begin until a negative edge occurs on the external arm channel (for example, the falling edge of the drive's index pulse). A measurement sample will subsequently be taken on input channel A, every 16 cycles. See Figure 6.

An event delay between the start and stop of the time interval measurement is necessary to avoid the noise correlation discussed previously. The HP 5371A CYCLE sampling mode for the PERIOD function offers continuous event delays in the following increments: 2^4, 2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}, and 2^{28}. An event delay of 16 (2^4) addresses the correlation criterion appropriately. (Note that EVENT sampling, while providing greater flexibility in the value of the event delay, does not offer consecutive time interval measurement capability.)

The number of consecutive samples gathered is defined by the variable "Block__length". Using the SINGLE mode, the HP 5371A will go through this EDGE/CYCLE arming sequence of "Block__length" consecutive samples for each pass through the FOR/NEXT loop shown in the flow diagram (See Figure 4).

[4]*The maximum block size is 1000 with the floating point mode output format. However, up to 4095 consecutive measurements can be obtained using the binary output mode.*

Program lines #950 through #980 configure the input circuitry, setting the trigger levels to appropriate values. The program as shown uses manual triggering modes (ECL levels). The HP 5371A also features "auto triggering" modes which set the input trigger levels to a percentage of the signal's peak-to-peak amplitude. Line number #980 sets the external arm trigger level for a TTL signal.

The remaining lines of this routine configure the HP 5371A for the floating point output mode with expanded data ON. In the expanded mode for period, the HP 5371A will not only return the period value, but also the precise gate time of each measurement. This gate time is actually the time interval of interest and will be used in lieu of a time interval measurement. The floating point format is used here for the sake of example. The binary output format could also be used to extract time interval data directly by finding the difference between consecutive time samples.

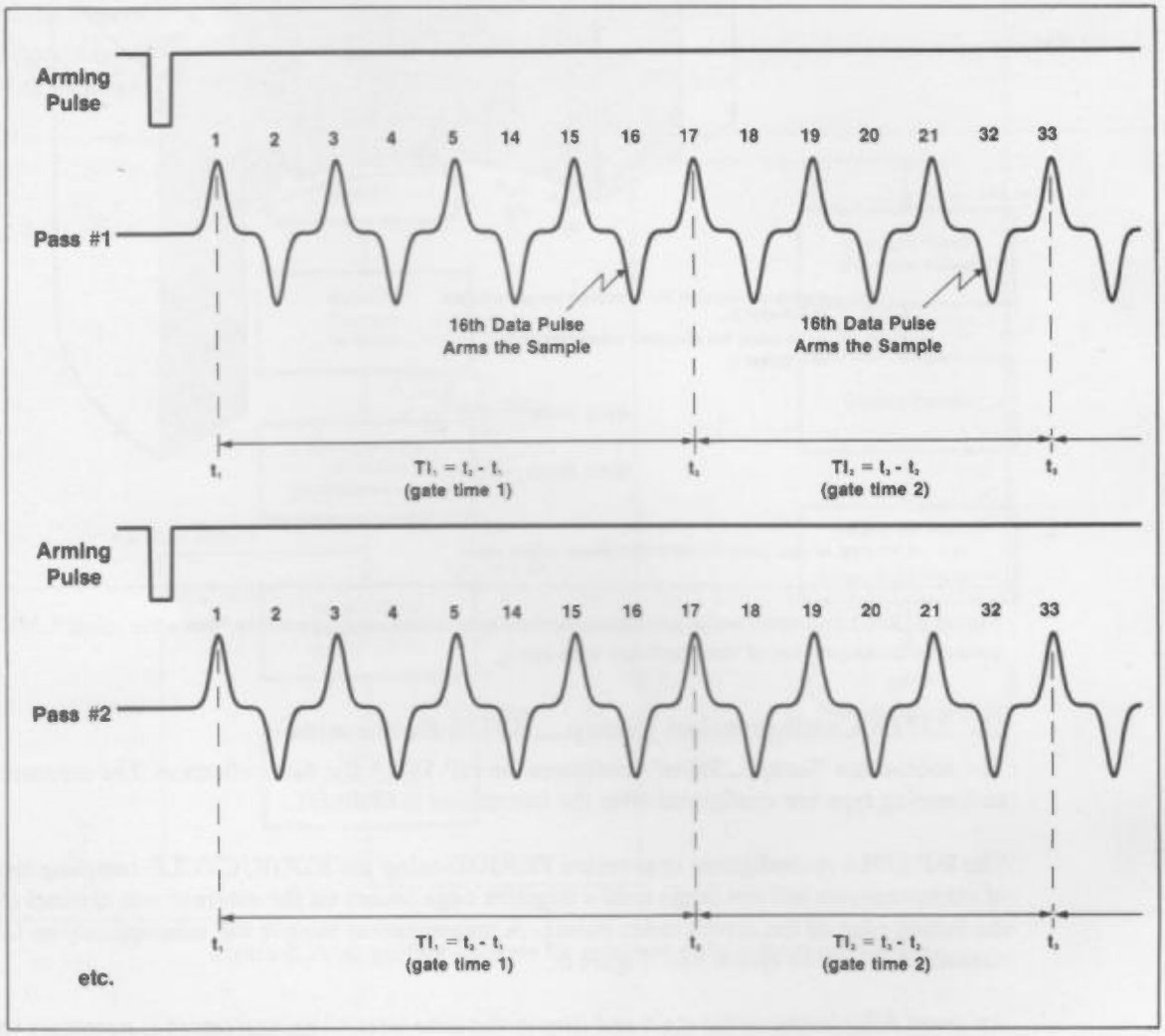


Figure 6. The EDGE/CYCLE arming mode is used to acquire data for read and write noise characterization. A delay of 16 cycles (2^4) insures adequate spacing to avoid noise correlation effects.

For each transition down the track, the $\left(\frac{\text{variance}}{2}\right)$ over repeated passes will describe a read noise value. These variance values will then be averaged to determine the overall read noise for this block of measurements, described as the square root of the average variance (standard deviation).

Acquiring the Data from the HP 5371A (Get_raw_data Subroutine)

This portion of the program will retrieve the floating point data from the HP 5371A block-by-block. As discussed earlier, the HP 5371A is configured in the floating point output mode with expanded data ON. The data from the HP 5371A is sent in pairs to the controller in the expanded mode. The first returned

value is the average period over the 16 cycle delay and the second returned value is the precise measurement gate time. Since this gate time is actually the time interval desired, the period data will be discarded as shown in lines 1280 and 1290.

After this processing has completed, the next block will be acquired and transferred from the HP 5371A to the controller. A potentially faster technique would be to transfer all of the data blocks to an array in the controller at once. The entire array can then be processed to compute read and write noise. The implementation of this technique has the drawback of being dependent on the amount of computer memory available, based on the number of passes and block length chosen.

$$\text{Required Memory} = 16 \text{ bytes} \times (\text{Block length} + 1) \times \text{Number of Passes}$$

Rather than require a specific amount of computer memory, this program example processes the data block-by-block.

Statistics are computed cumulatively by keeping track of the sum and sum-of-the-squares values with each block in the "Add__stats" subroutine. These values refer to sums for each particular time interval, or "column-wise" statistics of the entire data array. Appendix B describes a general form for a running calculation of variance and standard deviation.

Read Noise Computation (Rw__noise Subroutine)

This routine processes the cumulative totals to derive the standard deviation of measurement values for each particular interval (variable Read__temp). The square root of the average variance (standard deviation) is then computed to derive the average read noise value for the measurement block.

The remaining portion of the program code displays the resulting read noise value, returns the HP 5371A to the LOCAL mode, and deallocates the memory reserved for the cumulative statistics arrays.

Summary Comments Regarding Read Noise Characterization

In summary, these key points are worthy of special consideration:

1) This technique is a characterization of the effect of noise on the differentiated read signal. The read noise value will be dependent on the particular drive electronics as well as the measurement instrumentation. Note that the precision of the HP 5371A may be accounted for as demonstrated earlier, as can any other independent (and normally distributed) noise sources (e.g. PLL jitter for transition-to-clock measurements, etc.).

2) The shape of the read signal peaks are of particular importance, since they determine the slope of the differentiated signal and hence the susceptibility to noise at the zero-crossing detector. Signal amplitude is not necessarily an issue as might be predicted with parametric SNR techniques (except as required by the channel for gain and amplitude qualification considerations). A wide pulse with a relatively flat peak will be more susceptible to read noise than a narrow pulse with the same amplitude.

3) When measuring from transition-edge to transition-edge it is important to consider the effects of correlated noise from one pulse to the next. For this reason, the example algorithm uses relatively "distant" data-edges for time interval measurements (16 data pulses between the start and stop of the interval). Note that measuring the time of every transition is not critical to this technique. Indeed, a 16 cycle delay has been imposed between every sample to avoid noise correlation. Therefore, the HP 5371A and this application software may be used to characterize read noise on systems with transfer rates up to 160 MHz (using the 16 cycle sampling mode).

4) Careful preconditioning of the media is critical for obtaining repeatable measurement results. For comparable results, always use the same erasure, whether it is AC erasure or DC erasure with a particular bias.

Write Noise (Transition Noise) Characterization

Measurement Theory

To characterize write noise or transition noise, the actual transition-edge timing must be determined in the absence of read noise. Where the variance of successive reads of a specific time interval describes the read noise, the mean value of these successive reads is the actual spacing of the transitions on the media (assuming the read noise is normally distributed). In other words, read noise is "averaged out" to obtain the actual spacing on the media for a specific pair of data edges. The variability of this transition spacing "down the track" for a constant frequency pattern describes the write noise of the head/media system.

For read noise measurements, the standard deviation of measurements for a particular time interval (a column in the data array) is of interest. For write noise, the mean value for that same column of measurements becomes of interest, allowing write noise to be determined from the same set of data as read noise. See Figure 5.

The assumption is made that write noise affects both the start and stop transitions of the time interval measurement equally. As with the read noise measurements, the standard deviation of the mean intervals down the track should be normalized by $\sqrt{2}$ to account for the measurement configuration.

The use of transition-to-transition intervals rather than transition-to-clock intervals gains further significance for write noise characterization, as low frequency PLL tracking errors may tend to bias the write noise results.Precautions regarding correlated noise between adjacent pulses should be taken: using the same algorithm for read noise, the minimum spacing requirements are already in place to avoid noise correlation problems in the write noise measurements (16 data edges between the start and stop of the interval measurement).

Preconditioning of the media may affect measurement results. It is critical to consistently set the same conditions for the head/media system by using the same erasure for comparable results.

Unlike read noise, write noise characterization is independent of read channel electronics and measurement instrumentation. Limited repeatability due to special electronics or various testers can be eliminated using this HP 5371A technique. The HP 5371A can serve as an excellent verification tool between the drive designer and the component supplier.

For a constant frequency pattern, the head system will attempt to space transitions on the media equally apart. Any variation in what should be equal intervals for the constant frequency pattern is write noise (assuming read noise is averaged out). The example algorithm assumes that a constant frequency pattern has been recorded (the same assumption applies to the read noise measurement). Again, the designer may wish to experiment with several different code frequencies for write noise characterization.

Measurement Configuration and Computer Algorithm

The majority of the example program is identical in purpose and operation to that of read noise. Only the portions of the program of particular interest to write noise characterization are discussed here. Refer to Figure 4.

Variable Initialization

The block length and number of passes are important for write noise and read noise, but the importance of each number is essentially reversed. For the write noise calculation, the number of passes over the data determines how much the read noise is "averaged out" of the calculations (Num_passes). The number of consecutive intervals measured (Block_length) determines the sample size over which the variability, or the write noise, is computed. The designer may wish to experiment with these values to determine significance. For purposes of illustration, the program uses 100 passes (Num_passes) over a 1000 measurement block (Block_length).

Write Noise Computation (Rw__noise Subroutine)

This routine determines the average value of a particular interval (a column in Figure 5). The mean and mean-squared values for each of these column-wise mean intervals is maintained cumulatively. The final line of the routine (line #1990) determines the standard deviation of the column-wise means (normalized by $\sqrt{2}$). This standard deviation value describes the write noise of the system.

Summary Comments Regarding Write Noise Characterization

Several comments regarding characterizing write noise are worthy of consideration:

1) A system-dependent effect is write-clock jitter. The assumption has been made that this jitter is far less than the write noise itself. Of course, this assumption may be characterized using the HP 5371A to measure the stability of the write circuitry oscillator. As mentioned earlier, a major advantage of this HP 5371A technique is its independence of read channel electronics and measurement instrumentation.

2) Precision spindle speed control is necessary for accurate results (variability in motor speed between passes can cause a smearing of the distributions); however, reductions in data acquisition times made possible by the HP 5371A's consecutive measurement capability reduces the impact of spindle variations relative to slower, single shot measurement systems.

3) The algorithm assumes that a constant frequency pattern has been written on the media. This serves to simplify the calculations for write noise. The designer may wish to characterize write noise over several different frequencies.

4) Circumferential location on the media may influence write noise results, especially if the media exhibits "spoking" or coercivity gradients. It may be of interest to measure write noise at several locations around a data track to determine any variability with track location.

5) Measuring transition-to-transition intervals instead of transition-to-clock intervals eliminates concerns about the PLL tracking error (this tracking error can bias write noise results). For transition-to-transition measurements, minimum spacing requirements should be maintained to avoid noise correlation problems. As with the read noise measurement, measuring the time of every transition is not critical to this technique. Therefore, the HP 5371A and this application software may be used to characterize write noise on systems with transfer rates up to 160 MHz (using the 16 cycle sampling mode).

6) The algorithm always measures intervals across an even number of transitions (every 16th event), so the effects of timing asymmetry do not impact the results. By measuring time interval in this fashion, it is insured that the start and stop events are always of the same polarity (of the bipolar read signal), and therefore asymmetry effects are avoided. If an odd number of transitions are within the interval, then separate accounting of alternate measurements is necessary so that write noise can be determined for either (or both) groups.

Timing Asymmetry (Pulse-Pairing) Characterization

Measurement Theory

The effect of timing asymmetry is to shift positive bipolar data pulses in one direction (that is, advance or delay in time), while shifting negative bipolar data pulses in the opposite direction. For a constant frequency pattern, this will result in alternating short and long intervals rather than a constant spacing between transitions. This effect, also called "pulse-pairing", can be seen graphically as a bimodal histogram. (See Figure 7)

While the histogram technique provides easy identification of the bimodality of the distribution, some guesswork about the tails of each distribution is required to determine the mean separation of the two distributions. The numerical techniques to do this are quite complex. A simpler method is to group positive-pulse to negative-pulse transitions and negative-pulse to positive-pulse transitions separately, and then determine the mean separation.

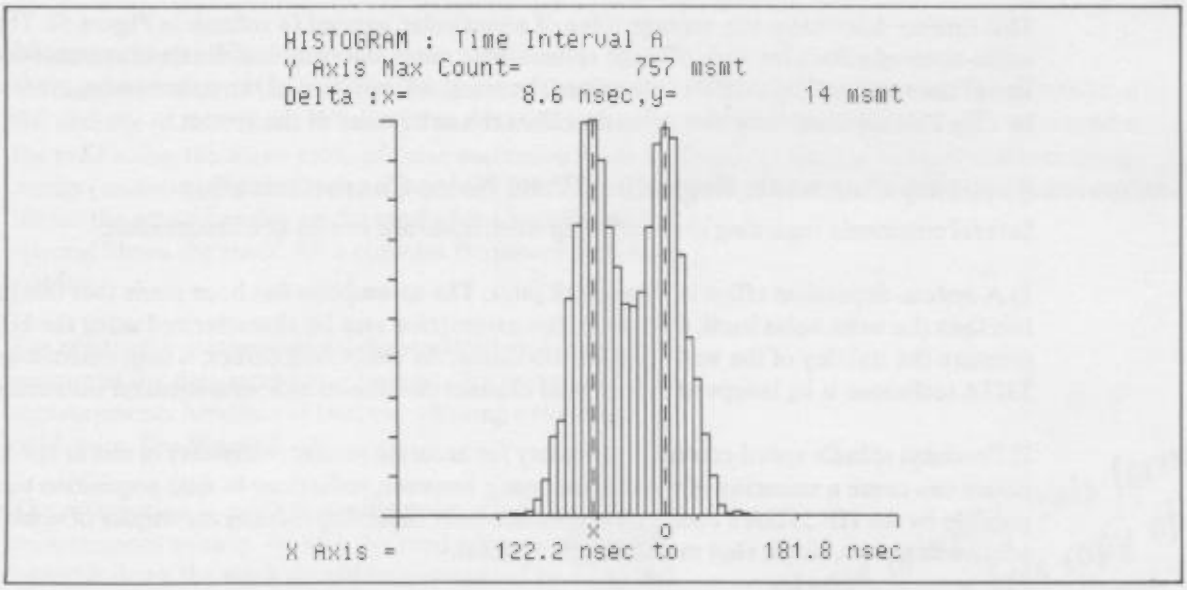


Figure 7. This HP 5371A histogram display clearly shows the effects of timing asymmetry. Complete quantification of asymmetry can be done with the timing asymmetry program example.

The binary format of the HP 5371A offers the capability to distinguish alternating positive and negative pulses by virtue of continuous event information. This is critical in light of the fact that the HP 5371A is actually measuring an RZ (return-to-zero) signal, not the bipolar read signal itself. In the example shown in Figure 8, odd numbered events correspond to positive bipolar data pulses, while even numbered events correspond to negative bipolar data pulses.

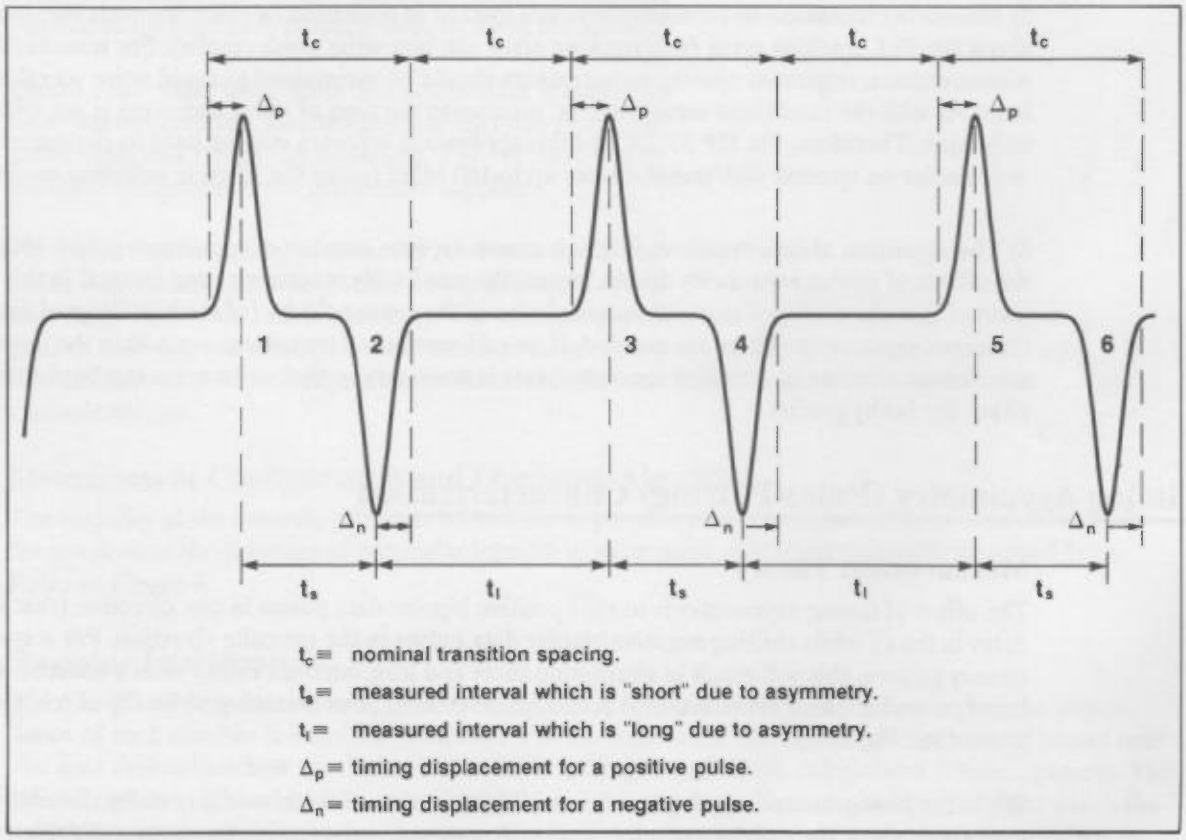


Figure 8. Timing asymmetry in a constant frequency data pattern causes a series of equal transition-to-transition intervals to alternate short-long, short-long, etc. In this example, the short interval occurs between a positive-to-negative (odd-to-even) pulse pair, and the long interval occurs between a negative-to-positive (even-to-odd) pulse pair.

In general, the particular polarity versus odd or even event correlation is unknown. However, the key point is that the continuous event information offers a method to group measurement values by data polarity (assuming no amplitude dropouts). Statistics can be calculated on each group, and consequently the true mean separation of the distributions can be determined and the amount of timing asymmetry calculated.

The following equations show the derivation of the asymmetry calculation:

Assuming $\Delta_p = \Delta_n = \Delta$

$$t_c = t_s + 2\Delta$$

$$t_c = t_l - 2\Delta$$

In general

$$t_{oe} = t_c \pm 2\Delta$$

$$t_{eo} = t_c \mp 2\Delta$$

Where the terms are defined as follows:

t_{oe} = measured interval from an odd-numbered pulse to an even-numbered pulse.

t_{eo} = measured interval from an even-numbered pulse to an odd-numbered pulse.

Averaging over n intervals to eliminate read/write noise:

$$\sum_{i=1}^{n} t_{oe} = nt_c \pm n2\Delta$$

$$\sum_{i=1}^{n} t_{eo} = nt_c \mp n2\Delta$$

$$\frac{\sum_{i=1}^{n} t_{oe} - \sum_{i=1}^{n} t_{eo}}{n} = \pm 4\Delta$$

Results should reflect the effect of asymmetry in the decoding window ($\pm \Delta$):

$$\pm \Delta = \frac{\sum_{i=1}^{n} t_{oe} - \sum_{i=1}^{n} t_{eo}}{4n}$$

Disk preconditioning plays a critical role for asymmetry measurements. The designer or test engineer may wish to characterize timing asymmetry for both biases of DC erasure as well as AC erasure. The results of this measurement are closely related to the particular method of disk preconditioning. It should be noted that DC erasure will cause the greatest asymmetry; high frequency AC erasure will cause the least asymmetry. The difference between these results describes the time domain effects of overwrite performance.

As with read noise and write noise characterization, a constant frequency pattern is used for this measurement algorithm. In general, the designer may wish to characterize a variety of constant frequency patterns to determine the worst-case data pattern for timing asymmetry.

Measurement Configuration and Computer Algorithm

The measurement algorithm primarily involves sorting the HP 5371A raw data to compute odd-to-even and even-to-odd measurement intervals, corresponding to the positive-to-negative and negative-to-positive bipolar transition-to-transition intervals (or vice-versa). While sorting the data, the algorithm must also check for minimum spacing requirements to avoid noise correlation. Once the data has been sorted, an average of each group of intervals is determined to compute timing asymmetry. Figure 9 is a simple flow diagram of the program.

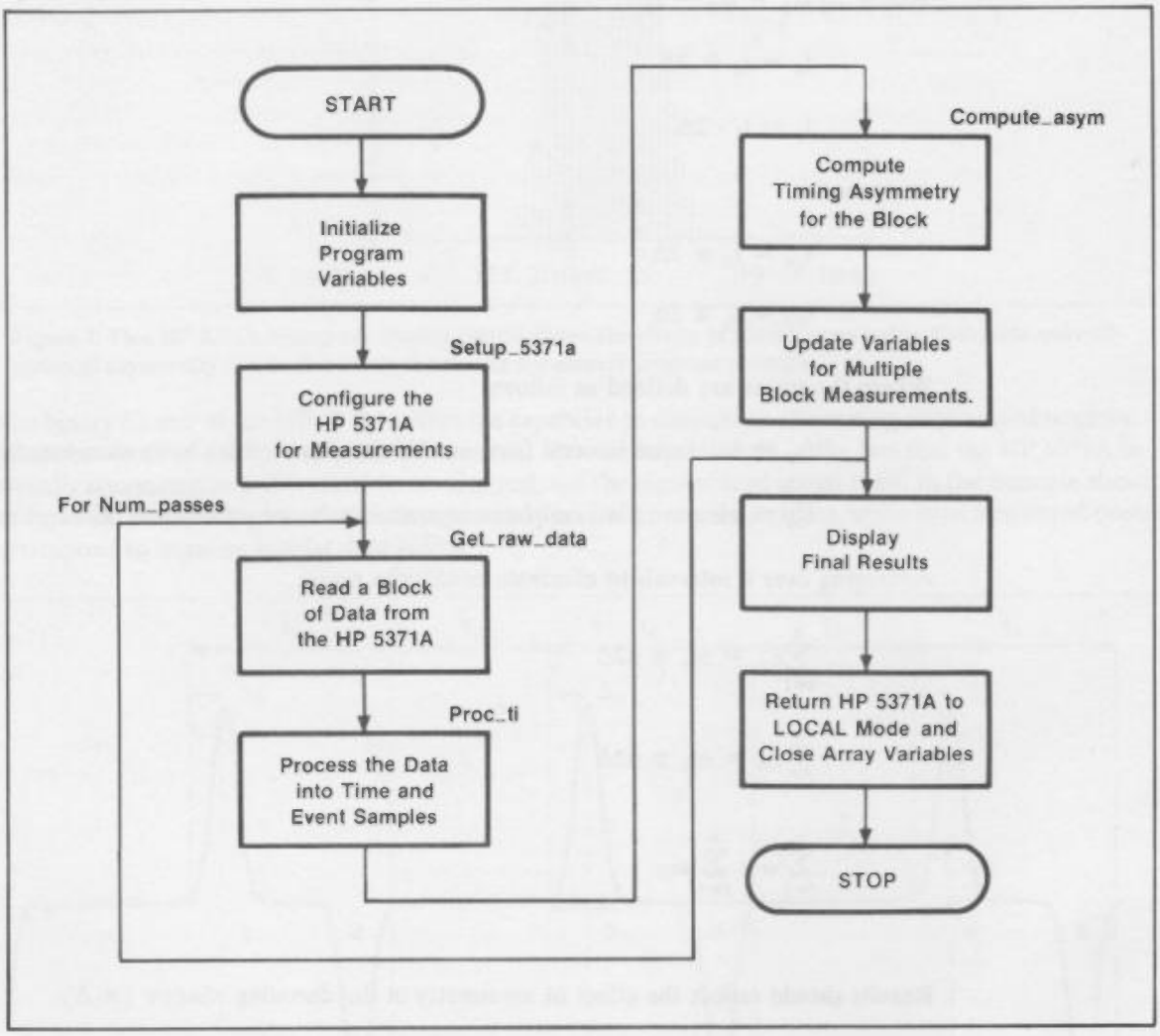


Figure 9. Program flow diagram for timing asymmetry computations.

Variable Initialization

The algorithm will measure many time intervals "down the track" to determine timing asymmetry. The number of intervals measured is set with the variable "Block_length" (line #240). Block length can take on values up to 4095 when using the HP 5371A binary output mode. The value of Block_length determines the number of intervals averaged to remove read and write noise effects.

If more intervals need to be averaged, the variable "Num_passes" (line #250) may be modified. This will cause multiple passes over the same portion of the track to average read and write noise further.

The variable "Min_spacing" (line #390) is used to control the spacing between time interval measurements. This value is important to avoid noise correlation when measuring from transition to transition. The INTERVAL sampling mode will be used with the HP 5371A and the sample interval value is set to the value of "Min_spacing".

HP 5371A Configuration (Setup__5371a Subroutine)

The subroutine "Setup__5371a" configures the HP 5371A for data collection. The measurement function and arming type is configured after the instrument is PRESET.

The HP 5371A is configured to the TIME INTERVAL mode using the EDGE/INTERVAL arming mode. Each block of measurements will not begin until a negative EDGE occurs on the external arm channel (for example, the falling edge of the drive's index pulse). Once the block has begun, a TIME INTERVAL measurement between two consecutive rising transition edges on input channel A will be taken. The following TIME INTERVAL measurement will not begin until an INTERVAL delay has expired. The minimum value that INTERVAL sampling can take is 600 ns. The sorting routine will also double-check to insure that minimum spacing requirements are observed.

Using the SINGLE mode, the HP 5371A will go through this EDGE/INTERVAL arming sequence of "Block ____ length" consecutive samples for each pass through the FOR/NEXT loop shown in the flow diagram (as determined by the variable "Num__passes").

Program lines 1120 to 1150 configure the input circuitry, setting the trigger levels to appropriate values. The program uses manual trigger modes at ECL levels. The external arm channel is configured to trigger on TTL levels.

The remaining lines of code in this subroutine configure the HP 5371A for the binary output mode with expanded data ON. In this mode, the HP 5371A will return both time samples and the event samples to be used for the asymmetry calculations.

Acquiring and Processing Data from the HP 5371A

(Get__raw__data and Proc__ti Subroutines)

The Get__raw__data subroutine retrieves a block of binary data from the HP 5371A and extracts the initial header information. The unprocessed data is stored in a buffer and passed to the processing routine through a COMMON variable construct.

The Proc__ti subroutine converts the binary data to 32-bit real values for the event and time samples. These individual samples are stored in arrays for later computation to determine the appropriate odd-to-even or even-to-odd intervals.

The TIME INTERVAL configuration uses two separate hardware time interpolators. The configuration has a 600 ps differential channel delay, internal to the HP 5371A. This 600 ps must be added to each "stop" time sample to correct for the resulting skew.

Asymmetry Computations (Compute__asym Subroutine)

The "Compute__asym" subroutine uses the event and time samples to determine and group appropriate odd-even or even-odd pairs, and subsequently calculate timing asymmetry.

As shown earlier in the derivation of the asymmetry computations, the intended spacing for the constant frequency pattern, must be determined. This calculation is done in lines #2650 to #2710 by taking the average period of the entire measurement block, $\left(\frac{\text{total time}}{\text{total events}}\right)$. To eliminate asymmetry affects from the end points in this calculation, the algorithm uses transition edges of the same polarity.

Assuming a constant frequency pattern, the next portion of code checks for a possible amplitude dropout. This is accomplished by determining the interval (or average interval, if events were not time stamped) between all adjacent time samples in the block of measurements. If the average period between adjacent samples exceeds 1.3 times the average transition time (program line #2780), an amplitude dropout is detected.[5] The asymmetry computations for this block of data are skipped and the data discarded.

[5]A value of 1.3 reflects a 30% margin. It may be desirable to modify this value for systems exhibiting greater write noise.

The algorithm next finds the first time interval starting with an odd-numbered event and ending with an even-numbered event. Beginning with an odd-numbered event, the algorithm searches for the first even-numbered event which satisfies the minimum spacing requirement.

Once the interval has been determined, the value is normalized to a single transition-to-transition interval. This satisfies the general case where all of the odd-to-even and even-to-odd intervals may not represent the same number of transition intervals (this case may arise due to the minimum spacing requirements between the start and stop data pulses necessary to avoid noise correlation). The normalization is determined as follows:

Measured Interval = (Number of events over the interval × Minimum Transition Interval) + 2Δ

Rearranging:

2Δ = Measured Interval - (Number of events over the interval × Minimum Transition Interval)

Normalizing the interval to one clock period (Minimum Transition Interval + 2Δ):

Normalized Interval = Minimum Transition Interval +

(measured interval - (number of events over the interval) × Minimum Transition Interval)

Finally, the normalized interval is added to a running sum of odd-to-even intervals.

The algorithm then increments the starting pointer to an even-numbered event and determines an even-to-odd interval in a similar fashion as that for the odd-to-even interval. The interval is normalized and added to a running sum of even-to-odd intervals.

The program continues to determine intervals, alternating odd-to-even and even-to-odd until the last time sample is encountered. If this is found on an even-to-odd pass, the last odd-to-even interval is removed from its running sum so mean calculations are performed on equivalent sample sizes.

Finally, the resulting asymmetry computations are displayed for the current measurement block.

The remaining program code compiles an average asymmetry term over multiple blocks of data and displays the cumulative results.

Summary Comments Regarding Timing Asymmetry Characterization:

In summary, these points are worthy of consideration:

1) The algorithm assumes that a constant frequency pattern has been written on the media.

2) In order to isolate asymmetry effects, time interval measurements must be taken between positive and negative bipolar data pulses, and vice-versa. The event information which is available with the HP 5371A binary data format is used to keep track of the sense of the pulse polarity. This method can be invalid if an amplitude dropout occurs. The algorithm includes a check for dropouts before the asymmetry calculations are performed.

3) Intervals used in asymmetry calculations must be long enough to avoid noise correlation between the start and stop pulses of the time interval measurement (transition-to-transition type measurements). This minimum spacing can be controlled by the user in the example program. Measuring the time of every transition is not critical to this technique. Therefore, the HP 5371A and this application software may be used to characterize timing asymmetry on systems with transfer rates well in excess of the specified sampling rate (up to the 500 MHz bandwidth limit of the HP 5371A).

4) Media preconditioning is important for effective characterization of timing asymmetry; DC erasure will create worst-case asymmetry effects. The designer may wish to characterize timing asymmetry for both directions of DC erasure.

5) Errors in the channel electronics (i.e. offsets at the zero-crossing detector comparator, etc.) can cause asymmetry effects that cannot be separated from the head/media asymmetry under characterization. However, the HP 5371A can be used to characterize channel errors by inputting a sine wave to the channel electronics and using these routines to characterize any subsequent pulse-pairing.

6) The algorithm averages data over many intervals to remove the effects of read and write noise.

7) In creating the asymmetry algorithm, the write clock has been assumed to be relatively constant (e.g. the write clock was stable, as was the spindle speed during the write and subsequent measurement processes). The average transition spacing value is displayed with the asymmetry results in the example program. In addition, asymmetry is assumed to be constant over the block of data.

8) In general, the timing offset effects of asymmetry and peak shift may reinforce each other, or may reduce each other, depending on the media preconditioning and the data pattern overwritten. A constant frequency pattern is used in this example for asymmetry characterization to minimize peak shift effects. Non-constant frequency patterns such as "tripole" patterns tend to enhance peak shift effects. The peak shift measurement to be described next will also detect timing asymmetry in a tripole data pattern.

Peak Shift or "Pulse Crowding" Characterization

Measurement Theory

Like timing asymmetry, the effect of peak shift is to systematically displace transition edges from their intended position. While the effects are similar, the causes are different. Peak shift is the result of the superposition of pulse shapes, so worst case data patterns are not constant frequency patterns, but patterns which cause unequal superposition on either "side" of a pulse.

For conventional head/media systems, where the isolated pulse shapes tend to be symmetrical, the dipole pattern (such as MFM(1,3) "DB6") exhibits worst-case peak shift. However, for thin-film technologies, the isolated pulse shapes tend to be asymmetrical. For these types of head/media systems, other data patterns exhibit worst-case peak shift results. The example program demonstrates peak shift characterization for a tripole pattern (conventional system tripole peak shift is generally equivalent to its dipole peak shift).

The measurement technique will be to "time-sample" each edge in the tripole set. Like timing asymmetry characterization, the polarity of the bipolar pulses will be followed by the odd/even event count values. Keeping track of the pulse polarity offers the capability to also compute timing asymmetry for the tripole data pattern.

To further demonstrate the capability of the HP 5371A's continuous measurement format, the peak shift algorithm is designed to measure minimum transition-to-transition spacings as small as 50 ns. This will require two passes over the measurement area. A minor change in the HP 5371A's measurement configuration will allow every transition of the tripole set to be time-sampled in these two passes. (The "Continuous Time Interval" and "Time Interval" measurement modes are both used.)

The first pass over the data will provide time samples for the "outer" data pulses of the tripole set (first and third). The second pass will provide time samples for the first and second (middle) data pulses. These two sets of time samples can then be used to compute peak shift as well as timing asymmetry for the tripole data pattern.

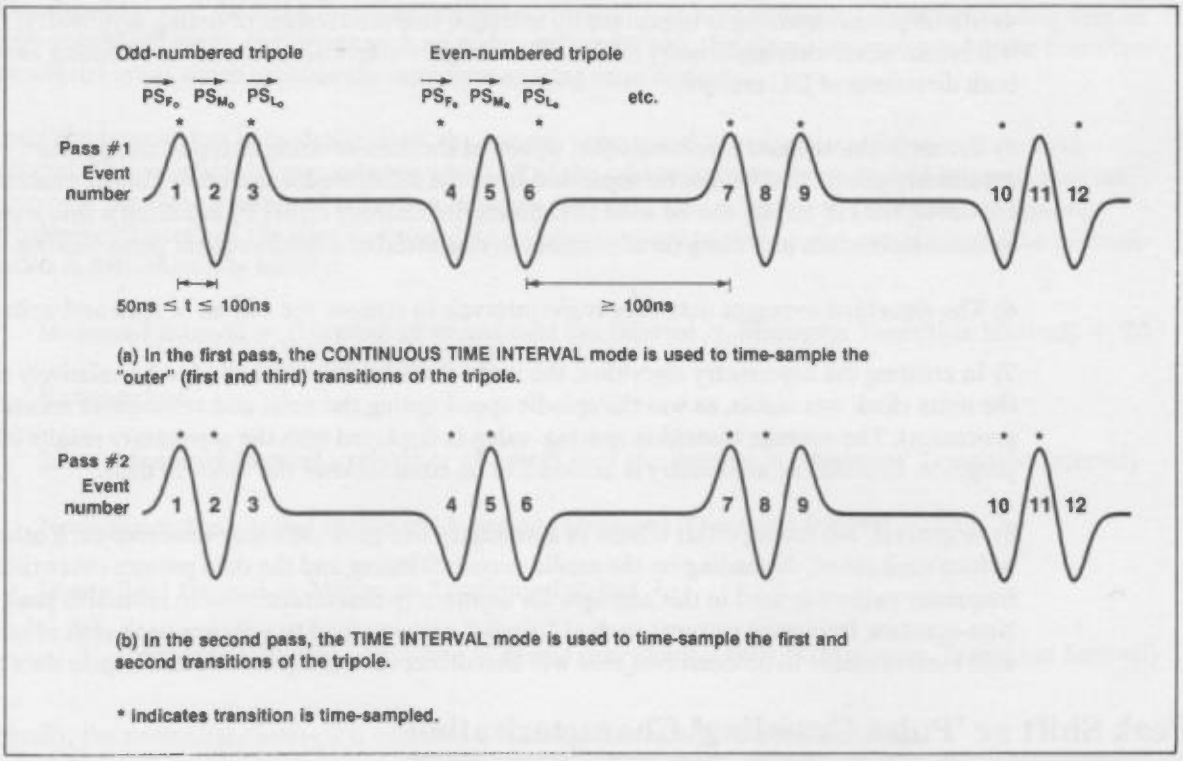


Figure 10. Two-pass technique to time-sample every pulse in the tripole set. This technique can be used for minimum transition-to-transition spacings between 50 ns and 100 ns.

Figure 10 illustrates how two passes over the measurement area retrieves the necessary timing information. In order to calculate asymmetry, as well as to prevent it from biasing peak shift calculations, the data is sorted into two groups: a tripole set with a "leading" odd-numbered pulse (for example, the tripole with data pulses 1, 2 and 3 - in this example, odd-numbered pulses are positive polarity) and a tripole set with a leading even numbered pulse (for example, the tripole with data pulses 4, 5 and 6 - the even-numbered pulses are negative polarity). The following equations illustrate the calculation of peak shift for each set of tripoles.

$$\text{Odd\_odd\_first\_last} = ((6d + 2k + 8) \times \text{Clock}) - PS_{F_o} + PS_{L_o}$$

$$\text{Odd\_odd\_first\_middle} = ((5d + 2k + 7) \times \text{Clock}) - PS_{F_o} + PS_{M_o}$$

$$\text{Odd\_even\_first\_first} = ((2d + k + 3) \times \text{Clock}) - PS_{F_o} + PS_{F_e}$$

$$\text{Odd\_even\_first\_middle} = ((3d + k + 4) \times \text{Clock}) - PS_{F_o} + PS_{M_e}$$

$$\text{Odd\_even\_first\_last} = ((4d + k + 5) \times \text{Clock}) - PS_{F_o} + PS_{L_e}$$

Where the terms are defined as follows:

PS_{F_o} = peak shift of the first pulse position in a tripole with an odd-numbered first pulse.

PS_{F_e} = peak shift of the first pulse position in a tripole with an even-numbered first pulse.

PS_{M_o} = peak shift of the middle pulse position in a tripole with an odd-numbered first pulse.

PS_{M_e} = peak shift of the middle pulse position in a tripole with an even-numbered first pulse.

PS_{L_o} = peak shift of the last pulse position in a tripole with an odd-numbered first pulse.

PS_{L_e} = peak shift of the last pulse position in a tripole with an even-numbered first pulse.

d = minimum number of clock periods between transitions.

k = maximum number of clock periods between transitions.

For each tripole set, three peak shift values are to be determined, for a total of six distinct peak shift values. Since peak shift must be described as a shift from the reference clock, an extra equation must be included to describe this reference. This equation is formed by assuming that the phase lock loop acts appropriately to compensate for peak shift in the system. The PLL should cause the total peak shift across two adjacent tripole sets to add to zero (See Figure 11):

$$PS_{F_o} + PS_{M_o} + PS_{L_o} + PS_{F_e} + PS_{M_e} + PS_{L_e} = 0$$

These six equations can be used to determine the distinct peak shift values for each transition (and pulse polarity) in the tripole signal.

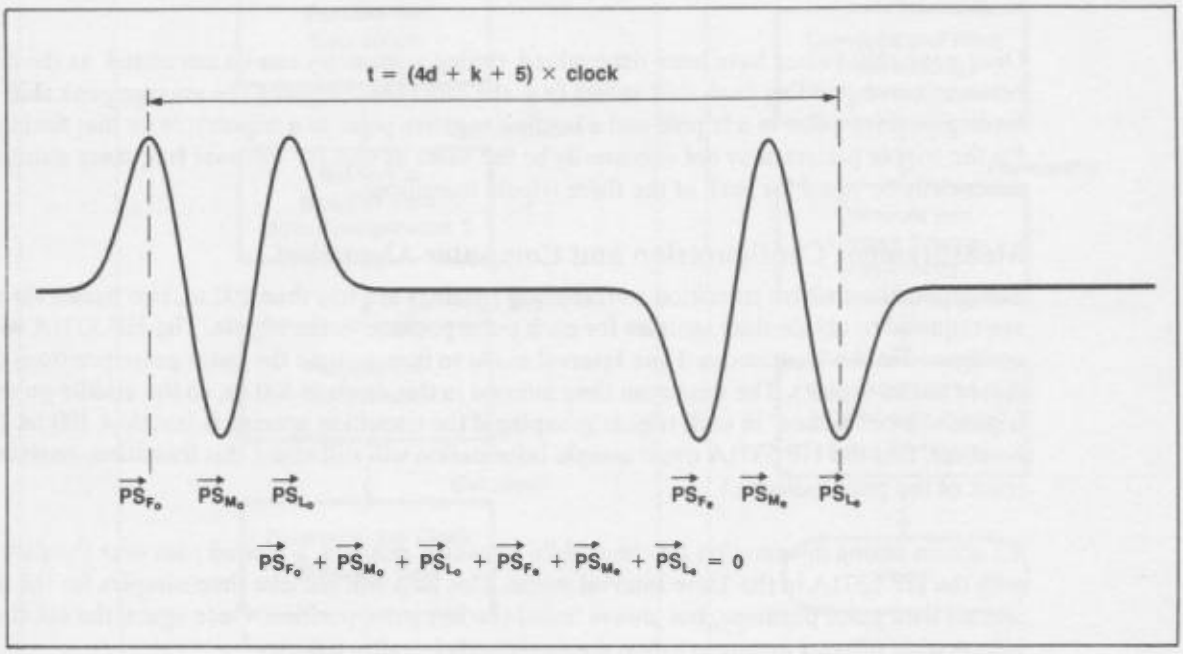


Figure 11. An extra equation is used to determine peak shift. The equation is derived by assuming the PLL forces the total peak shift offset to zero over adjacent tripole sets.

In practice the acquired data will also have read and write noise. Averages of values for the various intervals are used in the above equations to determine peak shift values. Read and write noise effects are "averaged out".

The clock period is determined by using the middle transitions of adjacent even-numbered polarity tripoles and dividing by the number of intermediate clock cycles. This is determined using the particular (d,k) coding scheme. For example: a 2,7 code should have a total of 28 clock periods between "middle" data pulses of tripoles with the same polarity. See Figure 12.

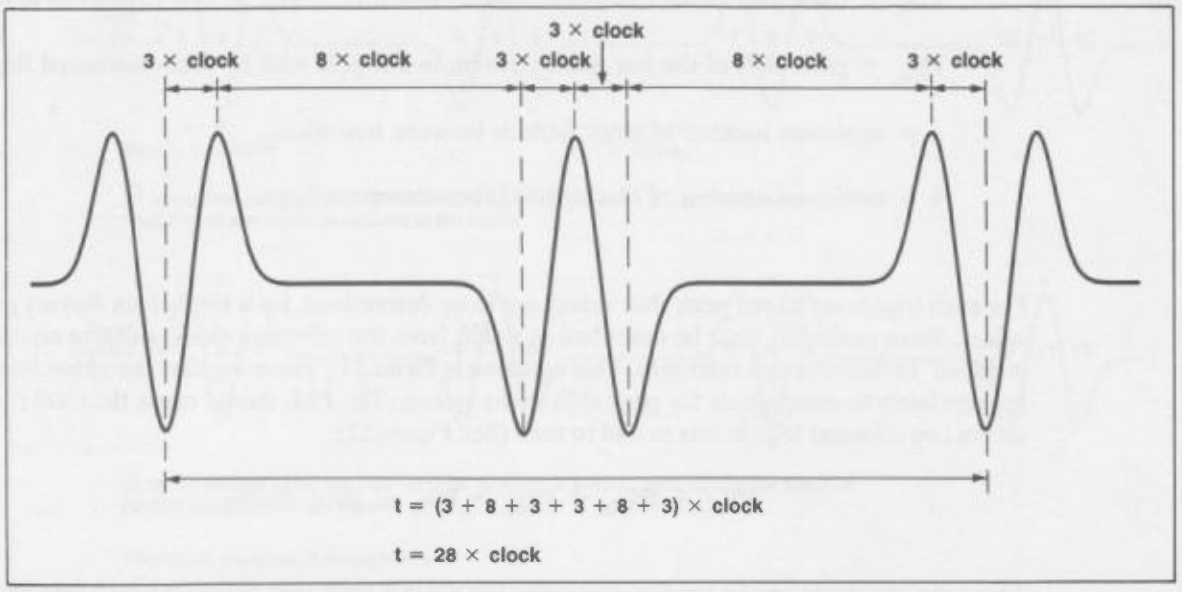


Figure 12. The "clock" is determined by using time samples from middle transitions of tripoles with the same pulse polarity. This measured interval is then divided by the required number of clock periods. For a RLL (2,7) code, there are 28 clock periods between these transition pulses.

Once peak shift values have been determined, timing asymmetry can be calculated as the difference between corresponding peak shift values (e.g. the difference between the average peak shift value for a leading positive pulse in a tripole and a leading negative pulse in a tripole). Note that timing asymmetry for the tripole pattern may not necessarily be the same as that for constant frequency patterns, nor will it necessarily be equal for each of the three tripole transitions.

Measurement Configuration and Computer Algorithm

For applications where transition-to-transition spacings are less than 100 ns, two passes through the data are required to obtain time samples for each pulse position in the tripole. The HP 5371A will first be configured in the Continuous Time Interval mode to time sample the outer pulse positions (first and last pulses of the tripole). The minimum time interval in this mode is 100 ns, so the middle pulse of the tripole will be "missed" in each tripole grouping if the transition spacing is less than 100 ns. (Note, however, that the HP 5371A event sample information will still count this transition, enabling us to keep track of the pulse polarity.)

To obtain timing information for the middle transition position, a second pass over the data will be made with the HP 5371A in the Time Interval mode. This pass will retrieve time samples for the first and second data pulse positions, but always "miss" the last pulse position. Once again, the event sample information offers a means to group measurements by pulse polarity.

These two data records offer all of the timing information necessary to compute peak shift using the equations developed above. Notice that the technique requires that measurements begin with the first pulse of a tripole. If the drive configuration does not allow this, a simple test may be added to the program to determine the first time sample in a record that corresponds to the first transition position in the tripole. This can be done by a comparison of measured time values.

Figure 13 shows a flow diagram for the peak shift program example.

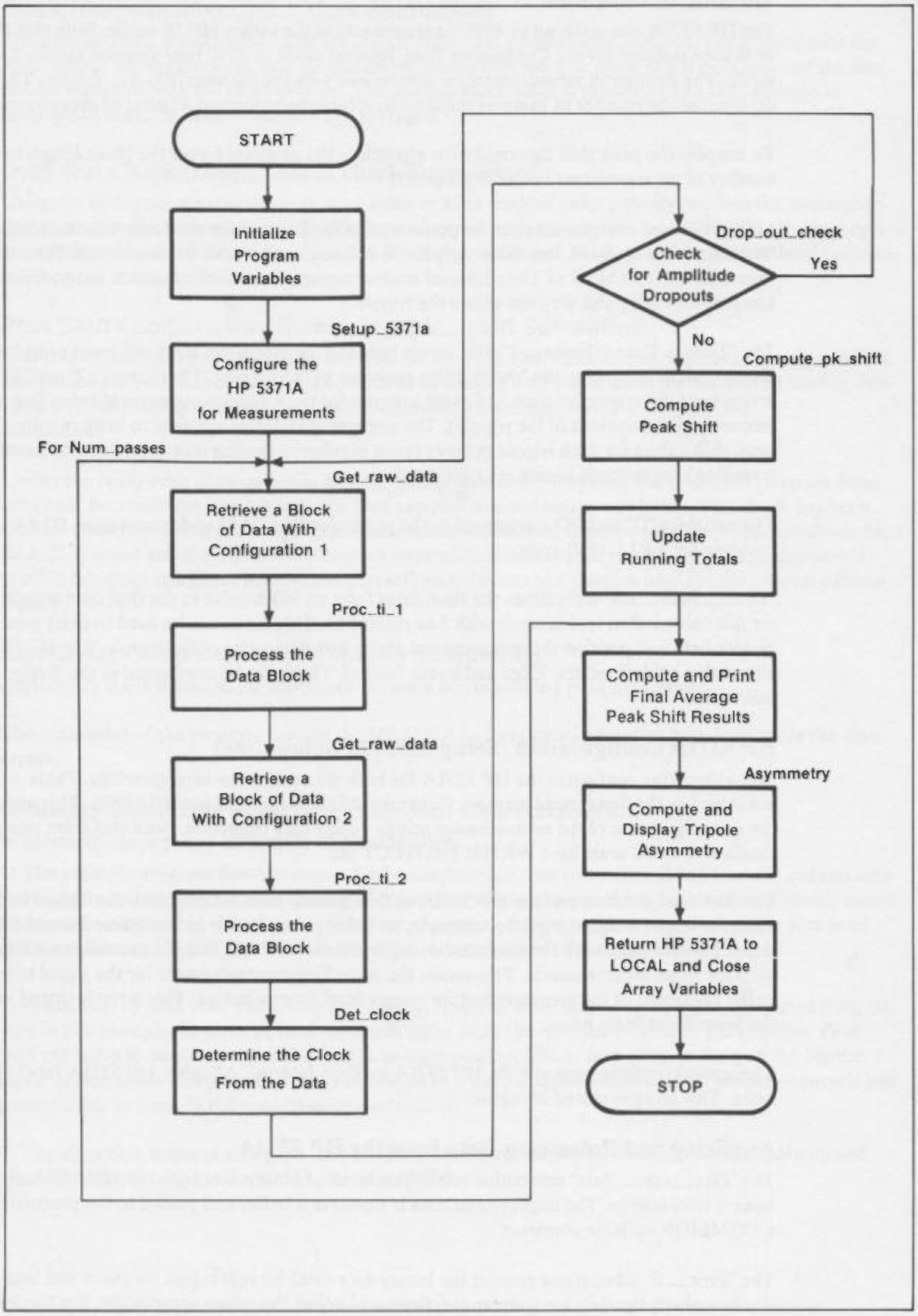


Figure 13. Program flow diagram for peak shift computations.

Variable Initialization

The HP 5371A can make up to 4095 measurements in the binary HP-IB mode. Note that this implies 4096 time samples for the Continuous Time Interval mode or 8190 time samples for the Time Interval mode. The number of measurements is determined with the variable "Block_length". The HP 5371A determines the number of samples required to return the requested number of measurements.

To simplify the peak shift determination algorithm, the program forces the block length to an odd number of measurements in line number 390.

Multiple passes over the data can be performed to further average read and write noise. The variable "Num_passes" controls this value. A "pass" is defined as one block of Continuous Time Interval measurements and one block of Time Interval measurements. One block of each is necessary to retrieve time samples for each pulse position within the tripole.

The "Time_1" and "Event_1" data arrays hold the the respective time and event samples for the first measurement setup (e.g. the "outer" pulse positions of the tripole). The "Time_2" and "Event_2" data arrays hold the respective time and event samples for the second measurement setup (e.g. the "first" and "second" pulse positions of the tripole). The remaining variables are used to keep running totals of the peak shift values for each tripole polarity (even-numbered leading transition and odd-numbered leading transition) for multiple passes over the data.

The variables "D" and "K" correspond to the particular RLL (d,k) code convention. The program example is shown for a RLL (2,7) code.

"Time_hold_off" determines the time delay from an index pulse to the first time sample. The range for this value is 2 ns to 8 seconds with 2 ns resolution. This mode can be used to delay past the sector header field and position the measurement at the first data pulse of the tripole. The HP 5371A offers two alternative holdoff modes: Edge and Event holdoff. These can be configured in the "Setup 5371a" subroutine.

HP 5371A Configuration (Setup_5371a Subroutine)

This subroutine configures the HP 5371A for both data collection configurations. These configurations are stored in the front panel memory (Instrument State Menu) of the HP 5371A. This approach speeds the reconfiguration of the measurement setups during data collection. Note that front panel memory locations 1 and 2 must have WRITE PROTECT off.

The first configuration sets the HP 5371A to Continuous Time Interval with the Time Holdoff arming mode. A negative signal edge (for example, an index pulse) on the external arm channel begins the time delay. Positive pulses are then counted on input channel A. Note that the manual input triggering mode is used for these measurements. This avoids the Auto Trigger requirement for the signal to be present prior to the beginning of the measurement for trigger level determination. This setup is stored in register 1 of the Instrument State menu.

The second configuration sets the HP 5371A to Time Interval. All other HP 5371A functions remain the same. This setup is stored in register 2.

Acquiring and Processing Data from the HP 5371A

The "Get_raw_data" subroutine retrieves a block of binary data from the HP 5371A and extracts the header information. The unprocessed data is stored in a buffer and passed to the processing routines via a COMMON variable construct.

The "Proc_ti" subroutines convert the binary data to 32-bit real values for event and time samples. The routines check the data for counter overflows and adjust the values accordingly. For the Time Interval mode, a 600 ps systematic term must be added to the "stop" sample to compensate for internal differential channel delays in the HP 5371A.

Clock Determination (Det__clock Subroutine)

The average clock is calculated using the middle transition position time samples. The routine uses the first "middle" pulse (assuming that measurements begin with the first pulse of a tripole set) and the last, middle transition with the same polarity. The average clock value is calculated and the precision is determined using the number of intervals averaged.

Drop Out Check (Drop__out__check Subroutine)

Using the odd/even numbered event convention to keep track of pulse polarity requires the assumption that no amplitude dropouts occur. This subroutine verifies this assumption by checking for the appropriate number of events between time samples. If a dropout is found, the data is discarded without performing the peak shift calculations.

Peak Shift Computations (Compute__pk__shift Subroutine)

This routine searches for appropriate intervals for the peak shift calculations. Separate running sums are maintained for both sets of tripoles: odd-numbered leading data pulses and even-numbered leading data pulses. These running sums are then used to calculate an average value for the peak shift calculations.

Unlike the read/write noise program and the timing asymmetry program, the peak shift program does not check for minimum spacing between time samples to avoid noise correlation. This check has been omitted since the coding pattern provides for a maximum interval between tripoles (8 clock intervals for a RLL (2,7) code) and the algorithm computes intervals between tripole groups. In general, this should provide adequate margin to avoid pulse interactions between time samples used for the interval calculations.

Asymmetry Computations (Asymmetry Subroutine)

Asymmetry is calculated as the difference between corresponding peak shift values.

The remainder of the program returns the HP 5371A to front panel operation and deallocates the data arrays.

Summary Comments Regarding Peak Shift Characterization

In summary, these points are worthy of consideration:

1) The example program demonstrates a fairly complex case. For conventional head/media systems with minimum transition-to-transition spacing of more than 100 ns, the algorithm becomes significantly easier. Data can be acquired in a single pass. However, many new drive designs are employing thin-film head media systems and higher data rates that warrant demonstration of the tripole case.[6]

2) In addition to peak shift and timing asymmetry, read and write noise could also be calculated from the data in this example. In general, read noise will differ from the constant frequency pattern case. Peak shift can serve to improve read noise figures as superposition effects may serve to sharpen the bipolar signal peaks. These sharper pulse peaks will result in a steeper differentiated signal and subsequently less susceptibility to noise in the read channel electronics.

3) The algorithm assumes that peak shift effects are constant for any particular transition polarity and position within the tripole for the measurement block.

[6]*The example program for peakshift calculations demonstrates a technique for gathering timing information for a minimum transition-to-transition spacing which; is greater than 50 ns and less than 100 ns. The interval between tripoles must also be greater than 100 ns. For systems with minimum transition-to-transition spacings greater than 100 ns, the Continuous Time Interval mode may be used and the data is acquired in a single pass.*

4) The results of this peak shift characterization may be used to determine the amount of precompensation required by the drive. In fact, it may be of interest to develop data reflecting the resulting peak shift versus various precompensation values. Determining precompensation is generally an empirical process since precompensation does not eliminate peak shift, but attempts to compensate for its effects.

5) Media preconditioning is important, as always, for comparable results. This is primarily due to timing asymmetry effects which may add-to or subtract-from peak shift effects, depending on the overwritten pattern and the media preconditioning.

6) Group delay problems in the read channel (or similar problems) will cause aggregate peak shift terms that do not describe the actual head/media effects. The channel electronics should be characterized first to determine whether such effects are biasing the data (an HP 8770A Arbitrary Waveform Synthesizer can be used to input a tripole pattern to the channel electronics for subsequent characterization using this HP 5371A technique).

7) This technique for determining peak shift derives a distinct term for each transition pulse position and polarity in the tripole. Peak shift effects are normally seen as offsets or "kick outs" on the phase margin plot. A large peak shift term for a particular transition pulse position may be difficult to detect with conventional phase margin analysis as the offset occurs only once every six pulses. This HP 5371A technique clearly depicts any variation in peak shift versus pulse position.

Time Interval Results and Window Margin Analysis

The techniques described in this application note serve to enhance the information provided by phase margin analysis rather than replace it. While the phase margin analyzer excels at a rapid overview of the drive's aggregate timing performance, time interval techniques give the designer the ability to isolate various timing effects: read noise and write noise can be separated from total noises described by the roll-off in a typical phase margin plot, and aggregate offsets described by displacements in that plot can be separated into individual peak shift and/or asymmetry terms. (See Figure 14).

Other time interval instrumentation is available which provide an alternate means to obtain the information already provided by the phase margin analyzer. These products use histographic techniques to present time interval information. It should be noted that while these products measure time interval directly, they do not provide the continuous measurement capability of the HP 5371A Frequency and Time Interval Analyzer. This continuous measurement capability is fundamental to the measurement techniques described in this application note.

Throughout this application note, measurements have been computed over sample sizes of several thousand. This differs from phase margin techniques which typically use sample sizes of 10^5 or 10^6. These large sample sizes are necessary to confidently extrapolate timing margin performance to error rates of 10^{-10} or less.

Recognizing that the statistical sample sizes differ significantly, it is possible to develop a model which relates the time interval results for read noise, write noise, timing asymmetry and peak shift to aggregate timing margin. The model must combine the offsets and noises of the head/media system with the offsets and noises from the drive system.

Head/media system contribution to window margin loss at a 10^{-10} error rate is:

$$\text{timing asymmetry} + \text{peak shift} + 6.36 \times \sqrt{(\text{read noise})^2 + (\text{write noise})^2}$$

For the whole system it becomes:

$$\text{Window margin} = \text{Decoder half window} - \left(\sum\text{head/media offsets} + \sum\text{channel offsets} + 6.36 \times \sqrt{\sum(\text{head/media noises})^2 + \sum(\text{channel noises})^2}\right)$$

Where:

$$\sum\text{head/media offsets} = \text{timing asymmetry} + \text{peak shift}$$

$$\sum(\text{head/media noise})^2 = (\text{read noise})^2 + (\text{write noise})^2$$

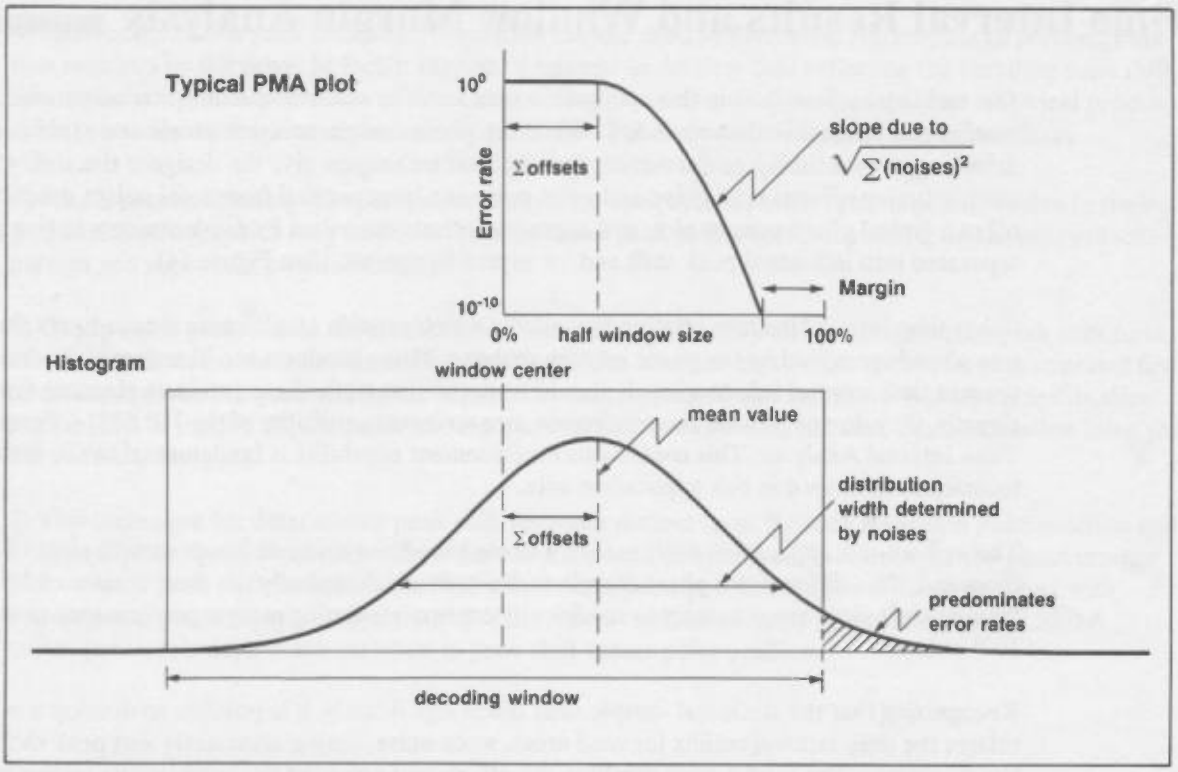


Figure 14. This figure shows the relationship of the cumulative probability density function plot (phase margin plot) and the histogram, as well as the relationship to the mathematical model described in the text.

Timing asymmetry and peak shift are used in their adjusted forms which directly account for their effects in the timing window. The read and write noise terms are combined in a "sum-of-the-squares" fashion (assuming that each noise term is independent). The aggregate noise is then multiplied by 6.36 (single-sided calculation) to describe the required number of sigmas to obtain an error probability of 10^{-10} (assuming that the noise follows a Gaussian distribution).

As the equation shows, window margin is the difference between the half decoding window and the sum of the offsets (asymmetries and peak shifts) and the sigmas required for a 10^{-10} probability of error. It should be noted that this model describes the effects of one tail of the distribution, so the model is not entirely accurate. However, unless the offset terms are zero, the tail described by this model will dominate any error rate calculation (e.g. ignoring the non-dominant tail will produce an extremely small error term).

Appendices

Appendix A: Glossary

Noise: generally expressed as a ratio to signal amplitude or by its sigma in nanoseconds. Noise can refer either to the aggregate of all noises in the system or to the individual noise components themselves (see Read Noise, Write Noise, and System Noise). Noises are assumed to be Gaussian (an assumption that has been directly confirmed by phase margin analysis on conventional head/media systems through error rates of 10^{-10}).

Offsets: a term describing aggregate displacement (in ns) of the transition distribution mean within the data decoding window (usually aggregate offset is the combination of any peak shift and asymmetry effects).

Peak Shift: displacement of a transition peak due to interaction with adjacent pulses. This is generally dominated by readback superposition, but secondarily affected by write process interactions.

Phase Margin Analysis: a time domain measurement which determines the available timing margin in the decoder half window. Margin estimates for error rates on the order of 10^{-10} are accomplished by extrapolating from sample sizes of 10^5 or 10^6 measurements. Typically error rate data is gathered by accelerating error rates by "sliding" the timing window with respect to its nominal center position or by "shrinking" the timing window about its center.

Read Noise: a term used to refer to the effects of system noise (generally preamp-dominated) on the ability of the read path circuitry to reliably locate the actual transition position.

RLL Code: [Run-Length-Limited Code] general description for disk drive data encoding schemes.

SNR: [Signal-to-Noise Ratio] measurement (traditionally expressed in decibels) that relates the amount of noise in the readback signal to the amplitude of that signal (this figure can be misleading in that it does not necessarily describe the effects of the noise at the differentiator).

System Noise: a term describing electronic noises in the system (not to be confused with read noise).

Timing Asymmetry: also referred to as pulse-pairing; this term describes the advancement of positive transition peaks relative to negative peaks, or vice-versa. This offset is generally caused by an asymmetry in the head/media system's isolated pulse; however, various read channel issues can also contribute to timing asymmetry.

Window Margin: the difference between the data decoding window edge and the point on the closest tail of the transition distribution that provides a probability of error equal to a target error rate.

Write Noise: also referred to as transition noise; this term describes the inability of the media to support transitions at the exact locations specified during write operations.

Appendix B: Statistical Reminders

Several statistical assumptions are important to the treatment of the measurement data as discussed in this application note. These assumptions are not unusual to common measurement practice, but are pointed out here for completeness.

It is assumed that the "noise" processes that are present in the disk drive environment are "random". More precisely, it is assumed that the sources of noise are independent and fit a Gaussian (normal) distribution. The mean and standard deviation values of this data, combined with the assumption of normal distribution, allow a convenient description of transition timing variability. Assuming the distribution is Gaussian and normally distributed, read noise and write noise can be combined by virtue of the Central Limit Theorem to obtain the window margin model described in this note. The normal distribution assumption may be verified using the Chi-Square test. This test gives a value indicating the "goodness of fit" of sorted data (histogram) to a theoretical distribution (in this case, the Gaussian distribution).

Computational Formulas

The following formulas provide a convenient form to compute mean, variance and standard deviation.

$$\text{Mean} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\text{Variance} = \frac{n\sum_{i=1}^{n}(x_i - x_1)^2 - \left[\sum_{i=1}^{n}(x_i - x_1)\right]^2}{n(n-1)}$$

$$\text{Standard Deviation} = \sqrt{\frac{n\sum_{i=1}^{n}(x_i - x_1)^2 - \left[\sum_{i=1}^{n}(x_i - x_1)\right]^2}{n(n-1)}}$$

Running sums of data and the data^2 values simplify the computations for large sample sizes.

For read and write noise calculations, the standard deviation result is divided by $\sqrt{2}$. This value is appropriate if the noise effects are equal and independent between the start and stop transitions of the time interval measurement. Therefore, the variance value is divided by 2, or the standard deviation by $\sqrt{2}$:

$$\text{Standard Deviation} = \sqrt{\text{Variance}}$$

So,

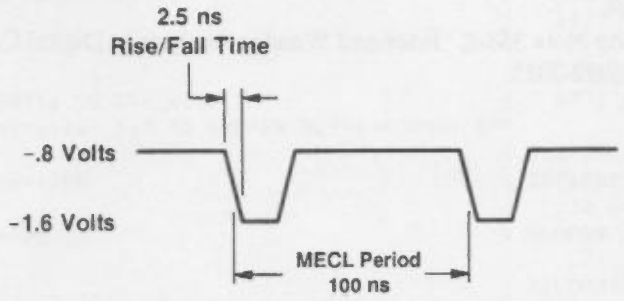
$$\frac{\text{Standard Deviation}}{\sqrt{2}} = \sqrt{\frac{\text{Variance}}{2}}$$

The following table relates sigma (standard deviation) values, confidence intervals, and "error probability".

	Confidence Interval	*Error Probability*
1σ	.84	$10^{-.8}$
2σ	.977	$10^{-1.6}$
3σ	.9985	$10^{-2.9}$
4σ	.99997	$10^{-4.5}$
5σ	.9999997	$10^{-6.5}$
6σ	~1.0	$10^{-9.0}$
6.36σ	~1.0	10^{-10}
7σ	~1.0	$10^{-11.9}$
8σ	~1.0	$10^{-15.2}$

HP 5371A Resolution Example

A time interval measurement is made from falling edge to falling edge of a MECL signal. The signal has 1 mV rms of noise with a fall time of 2.5 ns over an 800 mV swing. The HP 54002A 50 ohm input pod is used with a -2 volt termination. The measured value is 100 ns.



Measurement uncertainty example using Time Interval to measure from falling edge to falling edge of an ECL signal.

Resolution:

$$= \pm 150 \text{ ps rms} \pm \text{Start Trigger Error} \pm \text{Stop Trigger Error.}$$

$$= \pm 150 \text{ ps rms} \pm \frac{\sqrt{(200\ \mu\text{V rms})^2 + (1\ \text{mV rms})^2}}{.32\ \text{V/ns}} \pm \frac{\sqrt{(200\ \mu\text{V rms})^2 + (1\ \text{mV rms})^2}}{.32\ \text{V/ns}}$$

$$= \pm 156 \text{ ps rms.}$$

The resolution value computed above should be entered into the equation found on page 3 of this application note:

$$(\text{Measured Result})^2 = \frac{(156\ \text{ps rms})^2 + (\text{Actual Read Noise})^2}{2}$$

More examples of these kinds of calculations may be found in the Appendix of the HP 5371A Product Note/Specification Guide.

Appendix C: Related Literature

The following literature also describes the use of Hewlett-Packard products for disk drive applications. Contact your local Hewlett-Packard Sales Office for more information.

1. Using the HP 5180A Waveform Recorder to Evaluate Floppy Disc Media and Drive Electronics (AN 313-9). Literature number 5952-7701.
2. HP 8770A Arbitrary Waveform Synthesizer Demo for Rigid Disc Applications (DISCDEMO).
3. User's Guide: HP 8770 Artibrary Waveform Synthesizer Demo for Rigid Disc Applications.
4. A. Kovalic, "HP 8770A Applications in Magnetic Disc Recording," HP Journal, April 1988.
5. Synthesizing Magnetic Disc Read and Servo Signals with the HP 8770S (AN 314-2). Literature number 5954-6357.
6. HP 8770A Arbitrary Waveform Synthesizer Data Sheet for Rigid Disc Applications. Literature number 5952-6408.
7. Control System Development Using Dynamic Signal Analyzers (AN 243-2). Literature number 5952-5136.

HP 5371A Literature

1. HP 5371A Frequency and Time Interval Analyzer Data Sheet/Brochure, "Bringing A New Dimension to Measurement Analysis," Literature number 5952-7940.
2. HP 5371A Product Note/Specification Guide, Literature number 5952-7927.
3. Application Note 358-1. "Characterization of Frequency-Agile Signal Sources," Literature number 5952-7924.
4. Application Note 358-2, "Jitter and Wander Analysis in Digital Communications," Literature number 5952-7925.

The majority of the techniques and concepts discussed in this application note were developed by Steve Brittenham of Hewlett-Packard's Disc Memory Division; subsequent HP 5371A implementation was performed by Bruce Greenwood of Hewlett-Packard's Santa Clara Division.

Appendix D: Program Listings

```
10        !
20        !  Program to Compute Read and Write Noise
30        !  FROM HP 5371A DATA
40        !
50        !  PROGRAM REV. 3/21/88
60        !
70        !  COMPATIBLE WITH HP 5371A FIRMWARE  REV. 2745 OR LATER
80        !
90        !*************************************************************************
100                                                       !
110   GRAPHICS OFF                                        ! CLEAR SCREENS
120   OUTPUT 2;CHR$(255)&CHR$(75);
130                                                       !
140   OPTION BASE 1                                       ! SET OPTION BASE
150                                                       ! DECLARE VARIABLES
160   DIM Time_results(1000),Input_data(15000)            ! DIMENSION ARRAYS
170   DIM First_interval(1000)
180   INTEGER Isc,Ctr_addr                                ! SET UP COMMON BUFFERS
190   INTEGER Num_passes,Block_length                     !   FOR 5371 TRANSFERS
200   INTEGER Pass_number
210   REAL R_ns,W_ns
220   COM Buff$[16008] BUFFER,@Hp5371a,@Controller_buf
230                                                       !
240   Isc=7                                               ! USE INTERNAL HPIB Isc 7
250   Ctr_addr=3                                          ! SET UP 5371 HPIB
260   Ctr_addr=Ctr_addr+100*Isc                           !   ADDRESSES AND RESETS
270   RESET Isc                                           !   IT
280   CLEAR Isc
290                                                       ! ASSIGN BUFFER PATH TO
300   ASSIGN @Hp5371a TO Ctr_addr                         !   5371 AND CONTROLLER
310   ASSIGN @Controller_buf TO BUFFER Buff$;FORMAT OFF
320                                                       ! INITIALIZE VARIABLES
330   Block_length=1000                                   ! INTERVALS WITHIN A BLOCK
340                                                       !   >10 AND <OR= 1000
350   Num_passes=100                                      ! NUMBER OF BLOCKS OF DATA
360                                                       !
370                                                       ! ALLOCATE TEMP ARRAYS
380   ALLOCATE REAL Sum(1:Block_length),Sum_squared(1:Block_length)
390   MAT Sum= (0)                                        ! INITIALIZE TEMP ARRAYS
400   MAT Sum_squared= (0)
410                                                       !
420   Setup_5371a(@Hp5371a,Block_length)                  ! SET UP 5371 TO TAKE DATA
430                                                       ! LOOP TO ACQUIRE DATA
440                                                       !  OVER Num_passes
450                                                       !
460   FOR Pass_number=1 TO Num_passes
470     PRINT "WORKING ON PASS ";VAL$(Pass_number);" OF ";
480     PRINT VAL$(Num_passes)
490     Get_raw_data(Input_data(*))
500                                                       !
510     IF Pass_number=1 THEN
520       FOR Interval=1 TO Block_length
530         First_interval(Interval)=Input_data(Interval)
540       NEXT Interval
550     END IF
560                                                       !
570     Add_stats(Sum(*),Sum_squared(*),Input_data(*),First_interval(*),Block_le
ngth)
580   NEXT Pass_number
590                                                       !
```

```
600                                                     ! COMPUTE READ AND WRITE
610                                                     !   NOISES
620   Rw_noise(Sum(*),Sum_squared(*),R_ns,W_ns,Input_data(*),First_interval(*),B
lock_length,Num_passes)
630                                                     !
640   PRINT                                             ! PRINT RESULTS
650   PRINT
660   PRINT "READ NOISE:  ";
670   PRINT PROUND(R_ns,-3);" ns"
680   PRINT "WRITE NOISE: ";
690   PRINT PROUND(W_ns,-3);" ns"
700                                                     !
710                                                     ! RESTORE 5371A TO ASCII
720   OUTPUT @Hp5371a;"INT;OUTPUT ASCII"                !   STATE
730   LOCAL @Hp5371a                                    ! LOCALIZE 5371A
740                                                     !
750   DEALLOCATE Sum(*),Sum_squared(*)                  ! DEALLOCATE ARRAYS
760                                                     !
770   END
780                                                     !
790                                                     !
800                                                     !
810     !*************************************************************************
820   SUB Setup_5371a(@Hp5371a,INTEGER Block_length)
830     ! *************************************************************************
840 Setup_5371a:! SUBPROGRAM TO SET UP THE HP 5371A FOR PERIOD MEASUREMENT MODE
850     ! WITH "CYCLE" ARMING.  A TIME SAMPLE WILL BE TAKEN EVERY 16 CYCLES OF
860     ! THE INPUT SIGNAL.
870     !
880     DISP TAB(35);"CONFIGURING HP 5371A"
890     REMOTE @Hp5371a
900     OUTPUT @Hp5371a;"PRESET;SMODE SINGLE"
910     OUTPUT @Hp5371a;"MENU INFO"
920     OUTPUT @Hp5371a;"MEAS;FUNCTION PERIOD;SOURCE A"
930     OUTPUT @Hp5371a;"MSIZE";Block_length
940     OUTPUT @Hp5371a;"ARMING EDCYCLE;START;CHANNEL X;SLOPE NEG;SAMPLE;DCHANNE
L A;DELAY 16"
950     OUTPUT @Hp5371a;"INPUT;MODE,SEPARATE"
960     OUTPUT @Hp5371a;"INPUT;SOURCE A;TRIGGER MANUAL;LEVEL -1.3;SLOPE NEG"
970     OUTPUT @Hp5371a;"INPUT;SOURCE B;TRIGGER MANUAL"
980     OUTPUT @Hp5371a;"INPUT;SOURCE X;LEVEL 1.4"      ! SET X CHANNEL TRIG LEVEL
990     OUTPUT @Hp5371a;"INT;OUTPUT FPOINT"             ! TURN ON FLOATING POINT
1000                                                    !  OUTPUT FORMAT
1010    OUTPUT @Hp5371a;"NUM;EXPAND ON"                 ! EXPANDED DATA GIVES
1020                                                    !  GATE TIME INFO
1030    DISP ""                                         ! CLEAR PROMPT
1040  SUBEND
1050                                                    !
1060                                                    !
1070    ! *************************************************************************
1080  SUB Get_raw_data(Input_data(*))
1090    ! *************************************************************************
1100 Get_raw_data:! SUBPROGRAM TO GET RAW 5371 DATA AND STRIP THE HEADER
1110    ! INFORMATION FROM THE BEGINNING OF THE DATA BLOCK
1120                                                    !
1130                                                    ! DECLARE COMMON VARIABLES
1140    COM Buff$ BUFFER,@Hp5371a,@Controller_buf
1150                                                    !
1160    INTEGER I                                       ! DECLARE INTEGER
1170                                                    !
```

```
1180    DISP TAB(35);"GETTING DATA"                          ! PROMPT USER
1190                                                         ! PREPARE 5371 TO GET DATA
1200    RESET @Controller_buf
1210    TRIGGER @Hp5371a
1220                                                         ! TRANSFER DATA
1230    ENTER @Hp5371a USING "#,7A";Header$
1240    Number_of_bytes=VAL(Header$[3])
1250    TRANSFER @Hp5371a TO @Controller_buf;COUNT Number_of_bytes
1260                                                         !
1270    FOR I=1 TO Number_of_bytes/16                        ! PARSE DATA
1280      ENTER @Controller_buf;Input_data(I)                ! ENTER AVERAGE (NOT USED)
1290      ENTER @Controller_buf;Input_data(I)                ! ENTER WHOLE PERIOD
1300                                                         !  (MEASUREMENT GATE TIME)
1310    NEXT I
1320                                                         !
1330    DISP ""                                              ! CLEAR PROMPT
1340                                                         !
1350  SUBEND
1360                                                         !
1370     ! ************************************************************************
1380  SUB Add_stats(Sum(*),Sum_squared(*),Input_data(*),First_interval(*),INTEGE
R Block_length)
1390     ! ************************************************************************
1400 Add_stats:! SUBPROGRAM TO KEEP RUNNING STATISTIC TOTALS
1410                                                         !
1420    DISP TAB(29);"DOING INTERMEDIATE STATS"              ! PROMPT USER
1430                                                         !
1440                                                         ! CREATE RUNNING SUMS FOR
1450    FOR Pointer=1 TO Block_length                        !   EACH SAMPLE IN BLOCK
1460      Sum(Pointer)=Sum(Pointer)+(Input_data(Pointer)-First_interval(Pointer)
)
1470      Sum_squared(Pointer)=Sum_squared(Pointer)+(Input_data(Pointer)-First_i
nterval(Pointer))^2
1480    NEXT Pointer
1490                                                         !
1500    DISP ""                                              ! CLEAR PROMPT
1510                                                         !
1520  SUBEND
1530                                                         !
1540                                                         !
1550     ! ************************************************************************
1560  SUB Rw_noise(Sum(*),Sum_squared(*),R_ns,W_ns,Input_data(*),First_interval(
*),INTEGER Block_length,Num_passes)
1570     ! ************************************************************************
1580 Rw_noise:! SUBPROGRM TO COMPUTE READ AND WRITE NOISES
1590                                                         !
1600    DIM Temp$[80]                                        ! DIMENSION TEMP STRING
1610                                                         !
1620    Sqrt2=SQR(2)                                         ! INITIALIZE CONSTANT
1630                                                         !
1640    Read_temp=0                                          ! CLEAR READ AND WRITE
1650    Write_sum=0                                          !   NOISE RUNNING SUMS
1660    Write_sum2=0
1670    Read_noise=0
1680    Mean1=(Sum(1)/Num_passes)+First_interval(1)
1690                                                         !
1700                                                         ! COMPUTE INTERMEDIATE
1710    FOR Interval=1 TO Block_length                       !   READ NOISES
1720                                                         ! PROMPT USER OF PASS
1730      Temp$="COMPUTING R/W NOISE ON INTERVAL "&VAL$(Interval)
```

```
1740      DISP TAB(41-LEN(Temp$)/2);Temp$
1750                                                ! COMPUTE MEAN FOR THE
1760                                                !  CURRENT COLUMN
1770      Mean=(Sum(Interval)/Num_passes)+First_interval(Interval)
1780                                                ! COMPUTE INTERMEDIATE
1790                                                !   READ NOISE ARGUMENT
1800      Read_temp=(Sum_squared(Interval)-((Sum(Interval)^2)/Num_passes))/(Num_
passes-1)
1810                                                !
1820      IF Read_temp<=0 THEN                      ! TEST ARGUMENT FOR CASE
1830        Read_temp=0                             !   OF ZERO READ NOISE
1840      ELSE
1850        Read_temp=Read_temp/2
1860      END IF
1870                                                !
1880                                                ! SUM READ NOISES FOR
1890      Read_noise=Read_noise+Read_temp           !   COMPUTATION OF AVERAGE
1900                                                !
1910      Write_sum=Write_sum+(Mean-Mean1)          ! COMPUTE WRITE NOISE SUMS
1920      Write_sum2=Write_sum2+(Mean-Mean1)^2
1930                                                !
1940    NEXT Interval                               ! LOOP THROUGH ALL COLUMNS
1950                                                !
1960    DISP ""                                     ! CLEAR PROMPT
1970                                                ! COMPUTE FINAL READ AND
1980    R_ns=SQR(Read_noise/Block_length)*1.E+9     !   WRITE NOISE
1990    W_ns=(((((Write_sum2-(Write_sum^2/Block_length))/(Block_length-1))^.5))/
Sqrt2)*1.E+9
2000                                                !
2010  SUBEND
```

Program to Compute Timing Asymmetry

```
10        !
20        !
30        !  FROM HP 5371A DATA
40        !
50        !  PROGRAM REV. 3/21/88
60        !
70        !  COMPATIBLE WITH HP 5371A FIRMWARE REV. 2745 OR LATER
80        !
90        ! ************************************************************************
100                                                    !
110       GRAPHICS OFF                                 ! CLEAR SCREENS
120       OUTPUT 2;CHR$(255)&CHR$(75);
130                                                    !
140       OPTION BASE 1                                ! SET OPTION BASE
150                                                    !
160       COM /Data/ INTEGER Buff(1:8190,1:5) BUFFER,@Hp5371a,@Controller_buf
170       INTEGER Odd_ptr,Even_ptr,Block_length,Counter  ! DECLARE INTEGERS
180       INTEGER Ctr_addr,Rep,Pass,Num_passes,Isc
190       REAL Events_oe,Events_eo,Interval_oe,Interval_eo
200       REAL Trans_spacing,Asymmetry,Min_spacing,Ave_asym
210       REAL Tot_tran_spac,Ave_tran_spac,Tot_asym,Tot_counter
220                                                    !
230                                                    ! DETERMINE NUMBER OF
240       Block_length=1000                            !  SAMPLES: >10 AND <4096
250       Num_passes=1                                 ! DETERMINE NUMBER OF
260                                                    !    BLOCKS
270       Isc=7                                        ! USE INTERNAL HPIB Isc 7
280       Ctr_addr=3
290       Ctr_addr=Ctr_addr+Isc*100                    ! SET 5371 HP-IB ADDRESS
300       RESET Isc
310       CLEAR Isc
320       ASSIGN @Hp5371a TO Ctr_addr
330       ASSIGN @Controller_buf TO BUFFER Buff(*)     ! SET UP BUFFERS FOR
340                                                    !  5371A TRANSFERS
350                                                    !
360       ALLOCATE REAL Time(8190)                     ! DIMENSION DATA ARRAYS
370       ALLOCATE REAL Events(8190)
380                                                    ! SET MINIMUM INTERFERENCE
390       Min_spacing=6.00E-7                          !   FREE SPACING
400                                                    !
410       Events_oe=0                                  ! INITIALIZE INTERMEDIATE
420       Events_eo=0                                  !   SUMS
430       Interval_oe=0
440       Interval_eo=0
450       Tot_asym=0
460       Tot_tran_spac=0
470       Tot_counter=0
480       Pass=0
490                                                    !
500                                                    !
510       Setup_5371a(Min_spacing,@Hp5371a,Block_length)  ! SET UP 5371 TO TAKE DATA
520                                                    !
530                                                    !
540                                                    ! LOOP TO ACQUIRE DATA
550                                                    !  OVER Num_passes
560       FOR Rep=1 TO Num_passes
570         Get_raw_data(Block_length)
580         Proc_ti(Time(*),Events(*),Block_length)
590         Compute_asym(Time(*),Min_spacing,Events(*),Asymmetry,Trans_spacing,Block
_length,Rep,Counter,Pass)
```

```
600                                                     ! UPDATE CUMULATIVE TOTALS
610                                                     !  FOR EACH PASS THROUGH
620                                                     !  THE LOOP
630     Tot_asym=Tot_asym+ABS(Asymmetry)
640     Tot_tran_spac=Tot_tran_spac+Trans_spacing
650     Tot_counter=Tot_counter+Counter
660   NEXT Rep
670                                                     !
680   IF Pass=0 THEN Skip_comp                          ! SKIP COMPUTATIONS IF ALL
690                                                     !  DATA DISCARDED BECAUSE
700                                                     !  OF DROPOUTS
710                                                     !
720   Ave_asym=Tot_asym/Pass
730   Ave_tran_spac=Tot_tran_spac/Pass
740                                                     !
750                                                     !
760 Skip_comp:                                          ! PRINT THE FINAL RESULTS
770   PRINT
780   PRINT
790   PRINT
800   PRINT "          FINAL RESULTS"
810   PRINT "ASYMMETRY IS +/- ";VAL$(ABS(PROUND(Ave_asym*1.E+9,-3)));" ns"
820   PRINT "TRANSITION SPACING IS ";VAL$(PROUND(Ave_tran_spac*1.E+9,-3));
830   PRINT " ns"
840   PRINT "VALUES AVERAGED OVER";Pass;"PASS(ES)";
850   PRINT " OR";Tot_counter;"INTERVALS FOR EACH DISTRIBUTION."
860                                                     !
870                                                     ! RESTORE 5371A TO ASCII
880                                                     !  STATE
890   OUTPUT @Hp5371a;"INT;OUTPUT ASCII"
900   LOCAL @Hp5371a                                    ! LOCALIZE 5371A
910   DEALLOCATE Time(*),Events(*)                      ! DEALLOCATE ARRAYS
920   END
930                                                     !
940       !*********************************************************************
950   SUB Setup_5371a(Min_spacing,@Hp5371a,INTEGER Block_length)
960       !*********************************************************************
970 Setup_5371a:! SUBPROGRAM TO SET UP THE HP 5371A FOR TI MODE WITH INTERVAL
980     ! SAMPLING.  A TIME SAMPLE WILL BE TAKEN EVERY "Min_spacing" SECONDS.
990                                                     !
1000    DISP TAB(30);"CONFIGURING THE HP 5371A"         ! PROMPT USER
1010                                                    !
1020    REMOTE @Hp5371a
1030    OUTPUT @Hp5371a;"PRESET;SMODE SINGLE"
1040    OUTPUT @Hp5371a;"MENU INFO"
1050    OUTPUT @Hp5371a;"NUM;EXPAND ON"                 ! EXPANDED MODE RETURNS
1060                                                    !  EVENT INFORMATION ALSO
1070    OUTPUT @Hp5371a;"INT;OUTPUT BINARY"             ! TURN ON BINARY OUTPUT
1080                                                    !  FORMAT
1090    OUTPUT @Hp5371a;"MEAS;FUNCTION TINTERVAL;SOURCE A"
1100    OUTPUT @Hp5371a;"MSIZE";Block_length
1110    OUTPUT @Hp5371a;"ARMING EDINTERVAL;START;CHANNEL X;SLOPE NEG;SAMPLE;DELA
Y";Min_spacing
1120    OUTPUT @Hp5371a;"INPUT;MODE SEPARATE"
1130    OUTPUT @Hp5371a;"INPUT;SOURCE A;TRIGGER MANUAL;LEVEL -1.3;SLOPE POS"
1140    OUTPUT @Hp5371a;"INPUT;SOURCE B;TRIGGER MANUAL"
1150    OUTPUT @Hp5371a;"INPUT;SOURCE X;LEVEL 1.4"
1160                                                    !
1170    DISP ""
1180  SUBEND
```

```
1190                                                    !
1200     ! ************************************************************************
1210  SUB Get_raw_data(INTEGER Block_length)
1220     ! ************************************************************************
1230 Get_raw_data: ! SUBPROGRAM TO CAPTURE RAW BINARY DATA FROM THE HP 5371A AND
1240     ! STRIP THE HEADER INFORMATION OFF OF THE BLOCK.  DATA IS STORED IN
1250     ! 16 BIT WORDS AND TRANSFERRED TO THE PROC_TI ROUTINE VIA A "COMMON"
1260     ! CONSTRUCT.
1270                                                    !
1280                                                    ! DECLARE COMMON
1290                                                    !   VARIABLES
1300     COM /Data/ INTEGER Buff(*) BUFFER,@Hp5371a,@Controller_buf      !
1310                                                    !
1320     REAL Num_bytes
1330                                                    !
1340     DISP TAB(35);"GETTING DATA"                    ! PROMPT USER
1350                                                    ! TRANSFER DATA
1360                                                    !
1370                                                    !
1380     TRIGGER @Hp5371a
1390                                                    !
1400     ENTER @Hp5371a USING "#,A";Character$          ! CHECK FOR FIRST
1410                                                    !  CHARACTER=  #
1420     IF Character$<>"#" THEN
1430       BEEP
1440       DISP "BAD FIRST CHARACTER."
1450       PRINT CHR$(128)
1460       CONTROL 1,5;139
1470       STOP
1480     END IF
1490                                                   !
1500     ENTER @Hp5371a USING "#,A";Character$         ! CHECK FOR SECOND
1510                                                   !   CHARACTER= 6
1520     IF Character$<>"6" THEN
1530       BEEP
1540       DISP "BAD SECOND CHARACTER."
1550       PRINT CHR$(128)
1560       CONTROL 1,5;139
1570       STOP
1580     END IF
1590                                                   !
1600     ENTER @Hp5371a USING "#,6A";A$                ! GET THE NUMBER OF
1610     Num_bytes=VAL(A$)                             !   BYTES EXPECTED
1620                                                   !
1630                                                   ! TRANSFER BLOCK OF
1640                                                   !   DATA TO CONTROLLER
1650                                                   !
1660     IF Num_bytes<>2.0*Block_length*10 THEN        ! TIME INTERVAL MODE HAS
1670       BEEP                                        !   2 SAMPLES PER
1680       DISP "INVALID NUMBER OF BYTES."             !   MEASUREMENT
1690       STOP
1700     END IF
1710                                                   !
1720                                                   !
1730     RESET @Controller_buf                         ! RESET CONTROLLER BUFFER
1740     TRANSFER @Hp5371a TO @Controller_buf;COUNT Num_bytes,WAIT
1750     STATUS @Controller_buf,4;Num_bytes
1760                                                   !
1770                                                   !
1780     DISP ""
```

```
1790  SUBEND
1800                                                          !
1810        !*********************************************************************
1820  SUB Proc_ti(Time(*),Events(*),INTEGER Block_length)
1830        !*********************************************************************
1840 Proc_ti: !SUBPROGRAM TO PROCESS THE RAW TI DATA AND RETURN CUMULATIVE
1850     ! TIME AND EVENT ARRAYS.  THIS PROGRAM ALSO CHECKS FOR OVERFLOWS IN
1860     ! THE COUNT REGISTER VALUES.
1870                                                          !
1880     COM /Data/ INTEGER Buff(*) BUFFER,@Hp5371a,@Controller_buf
1890                                                          !
1900     DISP TAB(30);"PROCESSING RAW DATA"
1910     INTEGER Row,Col,Interpolator,Byte,Words_per_samp
1920     REAL Systematic_err,Unsigned_2_byte,Unsigned_4_byte
1930     REAL Ovflw_ctr_event,Ovflw_ctr_time
1940     Byte=8
1950     Unsigned_2_byte=2^16
1960     Unsigned_4_byte=2^32
1970     Ovflw_ctr_event=0
1980     Ovflw_ctr_time=0
1990     Words_per_samp=5                                     ! 5 WORDS PER SAMPLE
2000     Systematic_err=6.00E-10                              ! ADD IN 600PS SYSTEMATIC
2010                                                          !  ERROR FOR TI A
2020     Row=1
2030     Col=1
2040     Events(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff(Row
,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)
2050     Time(Row)=((Buff(Row,Col+2)+Unsigned_2_byte*(Buff(Row,Col+2)<0))+(Buff(R
ow,Col+3)<0))*Unsigned_2_byte+Buff(Row,Col+3)
2060     Interpolator=SHIFT(Buff(Row,Col+4),Byte)
2070     Time(Row)=Time(Row)*2.E-9-BINAND(31,Interpolator)*1.E-10
2080     Time(Row)=PROUND(Time(Row),-10)
2090                                                          ! LOOP TO DETERMINE
2100                                                          !  EVENT AND TIME SAMPLES
2110                                                          !
2120                                                          ! FIRST FIGURE EVENTS
2130     FOR Row=2 TO Block_length*2
2140       Events(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff(R
ow,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)+Ovflw_ctr_event
2150                                                          !
2160       IF Events(Row)<Events(Row-1) THEN                  ! CHECK FOR OVERFLOWS AND
2170         Events(Row)=Events(Row)+Unsigned_4_byte          !  ADJUST ACCORDINGLY
2180         Ovflw_ctr_event=Ovflw_ctr_event+Unsigned_4_byte
2190       END IF
2200                                                          !
2210                                                          ! FIGURE TIME SAMPLES
2220       Time(Row)=((Buff(Row,Col+2)+Unsigned_2_byte*(Buff(Row,Col+2)<0))+(Buff
(Row,Col+3)<0))*Unsigned_2_byte+Buff(Row,Col+3)
2230                                                          !
2240       IF Row MOD 2=0 THEN
2250                                                          ! PROCESS STOP
2260                                                          !  INTERPOLATOR DATA
2270         Time(Row)=Time(Row)*2.E-9-BINAND(31,Buff(Row,Col+4))*1.E-10
2280         Time(Row)=Time(Row)+Systematic_err
2290       ELSE
2300                                                          ! PROCESS START
2310                                                          !  INTERPOLATOR DATA
2320         Interpolator=SHIFT(Buff(Row,Col+4),Byte)
2330         Time(Row)=Time(Row)*2.E-9-BINAND(31,Interpolator)*1.E-10
2340       END IF
```

```
2350                                                    !
2360      IF Time(Row)<Time(Row-1) THEN                 ! CHECK FOR OVERFLOWS
2370        Time(Row)=Time(Row)+Unsigned_4_byte*2.E-9
2380        Ovflw_ctr_time=Ovflw_ctr_time+Unsigned_4_byte
2390      END IF
2400                                                    !
2410    NEXT Row
2420                                                    !
2430    DISP ""
2440  SUBEND
2450                                                    !
2460       !*********************************************************************
2470  SUB Compute_asym(Time(*),Min_spacing,Events(*),Asymmetry,Trans_spacing,INT
EGER Block_length,Rep,Counter,Pass)
2480       !*********************************************************************
2490 Compute_asym:! SUBPROGRAM TO PERFORM ASYMMETRY COMPUTATIONS.  THE PROGRAM
2500    ! ALSO DETERMINES THE WRITE CLOCK AND CHECKS FOR AMPLITUDE DROPOUTS.
2510                                                    !
2520    DIM Msg$[40]
2530    Msg$=""
2540    DISP TAB(35);"COMPUTING ASYMMETRY"              ! PROMPT USER
2550    Odd_ptr=1                                       ! INITIALIZE POINTERS INTO
2560    Even_ptr=Odd_ptr                                !   THE DATA
2570    Trans_spacing=0
2580    Counter=0
2590    Pass=Pass+1
2600                                                    !
2610                                                    ! DETERMINE AVERAGE
2620                                                    !  TRANSITION SPACING BY
2630                                                    !  MEASURING TOTAL
2640                                                    !  INTERVAL BETWEEN LIKE
2650    Ptr=Block_length*2                              !  POLARITY PULSES
2660                                                    !
2670    IF Events(1) MOD 2=Events(Ptr) MOD 2 THEN
2680      Trans_spacing=(Time(Ptr)-Time(1))/(Events(Ptr)-Events(1))
2690    ELSE
2700      Trans_spacing=(Time(Ptr-1)-Time(1))/(Events(Ptr-1)-Events(1))
2710    END IF                                          !
2720                                                    !
2730                                                    !
2740                                                    ! CHECK FOR DROPOUTS
2750    FOR I=2 TO Block_length*2
2760      Period=(Time(I)-Time(I-1))/(Events(I)-Events(I-1))
2770                                                    !
2780      IF Period>Trans_spacing*1.3 THEN
2790        BEEP
2800        PRINT "AMPLITUDE DROPOUT FOUND, NO CALCULATIONS PERFORMED"
2810        Pass=Pass-1
2820        GOTO Exit_compute
2830      END IF
2840                                                    !
2850    NEXT I
2860                                                    !
2870                                                    ! FIND FIRST ODD EVENT
2880    IF Events(1) MOD 2=0 THEN
2890      Odd_ptr=Odd_ptr+1
2900      Even_ptr=Odd_ptr
2910    END IF
2920                                                    !
2930                                                    !
```

```
3540                                                    !
3550    Temp_ptr=Odd_ptr                                ! FIND NEXT LEGAL EVEN
3560    Odd_ptr=Even_ptr                                !   STARTING POINT FOR
3570    Even_ptr=Temp_ptr                               !   THE ODD POINTER, SET
3580                                                    !   UP EVEN POINTER FOR
3590    REPEAT                                          !   SUBSEQUENT SEARCH
3600      Odd_ptr=Odd_ptr+1                             !   (ABOVE) OR EXIT IF
3610      IF Odd_ptr>Block_length*2 THEN Compute        !   TOO LARGE
3620    UNTIL Events(Odd_ptr) MOD 2>.5
3630                                                    !
3640    GOTO Odd_to_even                                ! LOOP UNTIL DONE
3650                                                    !
3660 Subtract_oe:Interval_oe=Interval_oe-Norm_int       !  SUBTRACT THE LAST ODD
3670    Msg$="LAST ODD TO EVEN INTERVAL WAS NOT USED"   !    TO EVEN INTERVAL TO
3680                                                    !    EVEN UP THE SAMPLES
3690                                                    !
3700                                                    ! COMPUTE THE DATA
3710 Compute:                                           !
3720    Asymmetry=(Interval_oe-Interval_eo)/Counter/4   ! DIVIDE BY 4 TO DETERMINE
3730                                                    !  EFFECT IN THE HALF-
3740                                                    !  WINDOW
3750                                                    !
3760                                                    ! PRINT RESULTS
3770    OUTPUT 2;CHR$(255)&CHR$(75);
3780    PRINT "CALCULATIONS ON PASS NUMBER";Rep
3790    PRINT
3800    PRINT
3810    PRINT Msg$
3820    PRINT "ASYMMETRY IS +/- ";VAL$(ABS(PROUND(Asymmetry*1.E+9,-3)));" ns"
3830    PRINT "AVERAGE TRANSITION SPACING IS ";VAL$(PROUND(Trans_spacing*1.E+9,-
3));
3840    PRINT " ns"
3850    PRINT Counter;" INTERVALS WERE USED FOR EACH DISTRIBUTION"
3860                                                    !
3870 Exit_compute:!
3880                                                    !
3890    DISP ""
3900                                                    !
3910  SUBEND
```

Program to Compute Peak-Shift

```
10       !
20       !  FROM HP 5371A DATA
30       !
40       !  PROGRAM REV. 3/21/88
50       !
60       !  COMPATIBLE WITH HP 5371A FIRMWARE REV. 2745 OR LATER
70       !
80       !  THIS PROGRAM USES THE HP 5371A FRONT PANEL SAVE/RECALL REGISTERS
90       !  1 AND 2.  FOR PROPER OPERATION, THE WRITE PROTECT ON THESE REGISTERS
100      !  MUST BE TURNED OFF.  SET-UPS STORED IN THESE LOCATIONS WILL BE
110      !  OVERWRITTEN.
120      !
130      !  THIS PROGRAM WILL FUNCTION CORRECTLY FOR MINIMUM TRANSITION-TO-
140      !  TRANSITION SPACINGS WITHIN A TRIPOLE THAT ARE GREATER THAN 50 NS AND
150      !  LESS THAN 100 NS.
160      ! ************************************************************************
170                                                    !
180      GRAPHICS OFF                                  ! CLEAR SCREENS
190      OUTPUT 2;CHR$(255)&CHR$(75);
200                                                    !
210      OPTION BASE 1                                 ! SET OPTION BASE
220                                                    !
230      COM /Data/ INTEGER Buff(1:4096,1:7) BUFFER,@Hp5371a,@Controller_buf
240                                                    !
250      INTEGER Ctr_addr,D,K,Rep_counter,Num_passes,Isc,I
260      INTEGER Block_length
270      REAL O_ps_first_sum,O_ps_mid_sum,O_ps_last_sum,Clock
280      REAL E_ps_first_sum,E_ps_mid_sum,E_ps_last_sum
290      REAL O_first,O_mid,O_last,Mode_bytes,Num_bytes
300      REAL E_first,E_mid,E_last
310      REAL Time_hold_off
320                                                    !
330      Block_length=1000                             ! DETERMINE NUMBER OF
340                                                    !  SAMPLES: >10 AND <4096
350                                                    !
360                                                    ! FORCE BLOCK LENGTH TO
370                                                    !  ODD VALUE TO SIMPLIFY
380                                                    !  PEAK SHIFT ALGORITHM
390      IF Block_length MOD 2=0 THEN Block_length=Block_length-1
400                                                    !
410                                                    !
420      Num_passes=1                                  ! DETERMINE NUMBER OF
430                                                    !  PASSES
440                                                    !
450      Isc=7                                         ! USE INTERNAL HPIB Isc 7
460      Ctr_addr=3
470      Ctr_addr=Ctr_addr+Isc*100                     ! SET HP5371 HP-IB ADDRESS
480      RESET Isc
490      CLEAR Isc
500      ASSIGN @Hp5371a TO Ctr_addr
510      ASSIGN @Controller_buf TO BUFFER Buff(*)
520                                                    !
530      ALLOCATE REAL Time_1(4096)                    ! DIMENSION DATA ARRAYS
540      ALLOCATE REAL Time_2(8192)
550      ALLOCATE REAL Events_1(4096)
560      ALLOCATE REAL Events_2(8192)
570                                                    !
580                                                    ! INITIALIZE INTERMEDIATE
590                                                    !  SUMS
600      O_ps_first_sum=0
```

```
610   O_ps_mid_sum=0
620   O_ps_last_sum=0
630   E_ps_first_sum=0
640   E_ps_mid_sum=0
650   E_ps_last_sum=0
660   Rep_counter=0
670                                                    !
680   D=1                                              ! ENTER THE (d,k) CODE
690   K=3                                              !  DESCRIPTION
700                                                    !
710   Time_hold_off=1.0E-6                             ! SET THE TIME HOLDOFF
720                                                    !  VALUE
730                                                    !
740   Setup_5371a(@Hp5371a,Time_hold_off,Block_length)! SETUP 5371A TO
750                                                    !  TAKE DATA IN BOTH TI
760                                                    !  AND CONT. TI MODES
770                                                    !
780                                                    ! LOOP TO ACQUIRE DATA
790                                                    !  OVER Num_passes
800   FOR Rep=1 TO Num_passes
810     PRINT TABXY(35,1);"PASS NUMBER ";Rep
820     OUTPUT @Hp5371a;"*RCL,1"                       ! GET CONT. TI SETUP
830     Mode_bytes=14.0*(Block_length+1)
840     Get_raw_data(Mode_bytes)
850     Proc_ti_1(Time_1(*),Events_1(*),Block_length)
860     OUTPUT @Hp5371a;"*RCL,2"                       ! GET TI SETUP
870     Mode_bytes=(14.0*2*Block_length)
880     Get_raw_data(Mode_bytes)
890     Proc_ti_2(Time_2(*),Events_2(*),Block_length)
900                                                    ! DETERMINE THE CLOCK
910                                                    !  FROM THE TI DATA
920     Det_clock(Time_2(*),Events_2(*),Clock,Block_length,D,K)
930                                                    !  CHECK FOR AMPLITUDE
940                                                    !   DROPOUTS
950     Drop_out_check(Events_1(*),Events_2(*),Result$,Block_length,D,K)
960                                                    !
970     IF Result$="DROPOUT" THEN Skip_block           !  SKIP THIS PASS IF
980                                                    !   AN AMPLITUDE DROPOUT
990                                                    !   IS FOUND
1000                                                   !
1010                                                   !  COMPUTE PEAK SHIFT
1020    Compute_pk_shft(Time_1(*),Time_2(*),Events_1(*),Events_2(*),Clock,O_ps_f
irst,O_ps_mid,O_ps_last,E_ps_first,E_ps_mid,E_ps_last,Block_length,D,K)
1030                                                   !
1040    O_ps_first_sum=O_ps_first_sum+O_ps_first
1050    O_ps_mid_sum=O_ps_mid_sum+O_ps_mid
1060    O_ps_last_sum=O_ps_last_sum+O_ps_last
1070    E_ps_first_sum=E_ps_first_sum+E_ps_first
1080    E_ps_mid_sum=E_ps_mid_sum+E_ps_mid
1090    E_ps_last_sum=E_ps_last_sum+E_ps_last
1100                                                   !
1110    Rep_counter=Rep_counter+1
1120 Skip_block:!
1130  NEXT Rep
1140                                                    !
1150                                                    !
1160  IF Rep_counter=0 THEN                             ! IF ALL DATA PASSES HAVE
1170    PRINT "NO_CALCULATIONS"                         !  DROPOUTS, THEN NO
1180    Rep_counter=1                                   !  CALCULATIONS SHOULD
1190    GOTO Local_ctr                                  !  BE PERFORMED
```

```
1200  END IF
1210                                                    !
1220                                                    !
1230  O_first=PROUND(O_ps_first_sum/Rep_counter,-11)
1240  O_mid=PROUND(O_ps_mid_sum/Rep_counter,-11)
1250  O_last=PROUND(O_ps_last_sum/Rep_counter,-11)
1260  E_first=PROUND(E_ps_first_sum/Rep_counter,-11)
1270  E_mid=PROUND(E_ps_mid_sum/Rep_counter,-11)
1280  E_last=PROUND(E_ps_last_sum/Rep_counter,-11)
1290                                                    !
1300                                                    ! PRINT THE RESULTS
1310  PRINT "CLOCK: ";Clock*1.E+9;" ns"
1320  PRINT
1330  PRINT "ODD-NUMBERED TRIPOLE RESULTS:"
1340  PRINT
1350  PRINT "   FIRST EDGE PEAK SHIFT: ";O_first*1.E+9;" ns"
1360  PRINT "   MIDDLE EDGE PEAK SHIFT: ";O_mid*1.E+9;" ns"
1370  PRINT "   LAST EDGE PEAK SHIFT: ";O_last*1.E+9;" ns"
1380  PRINT
1390  PRINT "EVEN-NUMBERED TRIPOLE RESULTS:"
1400  PRINT
1410  PRINT "   FIRST EDGE PEAK SHIFT: ";E_first*1.E+9;" ns"
1420  PRINT "   MIDDLE EDGE PEAK SHIFT: ";E_mid*1.E+9;" ns"
1430  PRINT "   LAST EDGE PEAK SHIFT: ";E_last*1.E+9;" ns"
1440  PRINT
1450                                                    !
1460                                                    ! CALCULATE AND DISPLAY
1470                                                    !  ASYMMETRY
1480                                                    !
1490  Asymmetry(O_first,E_first,O_mid,E_mid,O_last,E_last)
1500                                                    !
1510                                                    !
1520 Local_ctr:!                                        ! RESTORE 5371A TO ASCII
1530  OUTPUT @Hp5371a;"INT;OUTPUT ASCII"                !  STATE
1540  LOCAL @Hp5371a                                    ! LOCALIZE THE 5371A
1550                                                    !
1560  DEALLOCATE Time_1(*)                              ! DEALLOCATE ARRAYS
1570  DEALLOCATE Time_2(*)
1580  DEALLOCATE Events_1(*)
1590  DEALLOCATE Events_2(*)
1600  END
1610                                                    !
1620                                                    !
1630       !*********************************************************************
1640  SUB Setup_5371a(@Hp5371a,Time_hold_off,INTEGER Block_length)
1650       !*********************************************************************
1660 Setup_5371a:! SUBPROGRAM TO SET UP HP 5371A FOR TI AND CONTINUOUS TI MODES.
1670   ! SETUPS WILL BE STORED IN THE 5371A FRONT PANEL MEMORY (INSTRUMENT
1680   ! STATE) FOR FASTER RECONFIGURATION.  BOTH MEASUREMENT MODES USE TIME
1690   ! HOLDOFF ARMING TO POSITION THE BEGINNING OF THE MEASUREMENT AT THE
1700   ! BEGINNING OF A TRIPOLE SET.  THIS HOLDOFF HAS 2 NS RESOLUTION.
1710                                                    !
1720    DISP TAB(30);"CONFIGURING THE HP 5371A"         !
1730                                                    !
1740                                                    ! SETUP #1: CONTINUOUS
1750                                                    !  TIME INTERVAL MODE
1760                                                    !  TO GET OUTER DATA
1770                                                    !  EDGES OF TRIPOLE
1780    REMOTE @Hp5371a
1790    OUTPUT @Hp5371a;"PRESET;SMODE SINGLE"
```

```
1800    OUTPUT @Hp5371a;"INT;OUTPUT BINARY"
1810    OUTPUT @Hp5371a;"NUM;EXPAND ON"
1820    OUTPUT @Hp5371a;"MENU INFO"
1830    OUTPUT @Hp5371a;"MEAS;FUNCTION CTINTERVAL;SOURCE A"
1840    OUTPUT @Hp5371a;"MSIZE";Block_length
1850    OUTPUT @Hp5371a;"ARMING THOLDOFF;START;CHANNEL X;SLOPE NEG;DELAY";Time_h
old_off
1860    OUTPUT @Hp5371a;"INPUT;MODE,SEPARATE"
1870    OUTPUT @Hp5371a;"INPUT;SOURCE A;TRIGGER MANUAL;LEVEL -1.3;SLOPE POS"
1880    OUTPUT @Hp5371a;"INPUT;SOURCE B;TRIGGER MANUAL;LEVEL 0"
1890    OUTPUT @Hp5371a;"INPUT;SOURCE X;LEVEL 1.4"
1900    OUTPUT @Hp5371a;"*SAV,1"
1910                                                    !
1920                                                    ! SETUP #2 TO GET
1930                                                    !  FIRST AND SECOND
1940                                                    !  DATA EDGES OF TRIPOLE
1950    OUTPUT @Hp5371a;"MEAS;FUNCTION TINTERVAL"
1960    OUTPUT @Hp5371a;"*SAV,2"
1970                                                    !
1980    DISP ""
1990  SUBEND
2000                                                    !
2010                                                    !
2020     ! ************************************************************************
2030  SUB Get_raw_data(Mode_bytes)
2040     ! ************************************************************************
2050 Get_raw_data:! SUBPROGRAM TO CAPTURE RAW BINARY DATA FROM THE HP 5371A AND
2060    ! STRIP OFF THE HEADER INFORMATION. DATA IS TRANSFERRED TO THE PROC_TI
2070    ! ROUTINES THROUGH A "COMMON" CONSTRUCT.  THE DATA IS ORGANIZED INTO
2080    ! 16 BIT WORDS.
2090                                                    !
2100    REAL Num_bytes
2110                                                    ! DECLARE COMMON
2120                                                    !   VARIABLES
2130    COM /Data/ INTEGER Buff(*) BUFFER,@Hp5371a,@Controller_buf       !
2140                                                    !
2150    DISP TAB(35);"GETTING DATA"                     ! PROMPT USER
2160                                                    ! TRANSFER DATA
2170                                                    !
2180                                                    !
2190    TRIGGER @Hp5371a
2200                                                    !
2210    ENTER @Hp5371a USING "#,A";Character$           ! CHECK FOR FIRST
2220                                                    !  CHARACTER=  #
2230    IF Character$<>"#" THEN
2240      BEEP
2250      DISP "BAD FIRST CHARACTER."
2260      PRINT CHR$(128)
2270      CONTROL 1,5;139
2280      STOP
2290    END IF
2300                                                    !
2310    ENTER @Hp5371a USING "#,A";Character$           ! CHECK FOR SECOND
2320                                                    !   CHARACTER= 6
2330    IF Character$<>"6" THEN
2340      BEEP
2350      DISP "BAD SECOND CHARACTER."
2360      PRINT CHR$(128)
2370      CONTROL 1,5;139
2380      STOP
```

```
2390    END IF
2400                                                    !
2410    ENTER @Hp5371a USING "#,6A";A$                  ! GET THE NUMBER OF
2420    Num_bytes=VAL(A$)                               !   BYTES EXPECTED
2430                                                    !
2440    IF Num_bytes DIV Mode_bytes<>1 THEN             ! CHECK FOR INVALID
2450      BEEP                                          !  NUMBER OF BYTES
2460      DISP "INVALID NUMBER OF BYTES."               !  RETURNED
2470      STOP
2480    END IF
2490                                                    !
2500    RESET @Controller_buf                           ! RESET CONTROLLER BUFFER
2510    TRANSFER @Hp5371a TO @Controller_buf;COUNT Num_bytes,WAIT
2520    STATUS @Controller_buf,4;Num_bytes
2530                                                    !
2540    DISP ""
2550  SUBEND
2560                                                    !
2570                                                    !
2580       !*********************************************************************
2590  SUB Proc_ti_1(Time_1(*),Events_1(*),INTEGER Block_length)
2600       !*********************************************************************
2610 Proc_ti_1:! SUBPROGRAM TO DETERMINE CUMULATIVE EVENT AND TIME ARRAYS FOR
2620    ! THE CONTINUOUS TI MODE WITH TIME HOLDOFF ARMING. THE PROGRAM ALSO
2630    ! CHECKS FOR OVERFLOWS IN THE COUNT REGISTERS.
2640                                                    !
2650    COM /Data/ INTEGER Buff(*) BUFFER,@Hp5371a,@Controller_buf
2660                                                    !
2670    DISP TAB(30);"PROCESSING RAW DATA"
2680                                                    !
2690    INTEGER Row,Col,Interpolator,Byte,Words_per_samp
2700    REAL Ovflw_ctr_time,Ovflw_ctr_event
2710    REAL Unsigned_2_byte,Unsigned_4_byte
2720                                                    ! INITIALIZE CONSTANTS
2730                                                    !  AND VARIABLES
2740    Byte=8
2750    Ovflw_ctr_time=0
2760    Ovflw_ctr_event=0
2770    Unsigned_2_byte=2^16
2780    Unsigned_4_byte=2^32
2790    Words_per_samp=7
2800                                                    ! CONVERT FIRST EVENT AND
2810    Col=1                                           !  TIME SAMPLES
2820    Row=1
2830    Events_1(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff(R
ow,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)
2840    Time_1(Row)=((Buff(Row,Col+4)+Unsigned_2_byte*(Buff(Row,Col+4)<0))+(Buff
(Row,Col+5)<0))*Unsigned_2_byte+Buff(Row,Col+5)
2850    Time_1(Row)=Time_1(Row)*2.E-9-BINAND(31,Buff(Row,Col+6))*1.E-10
2860    Time_1(Row)=PROUND(Time_1(Row),-10)
2870                                                    !
2880                                                    !
2890                                                    ! LOOP TO CONVERT REST
2900                                                    !  OF EVENT AND TIME
2910                                                    !  SAMPLES
2920                                                    !
2930    FOR Row=2 TO Block_length+1                     ! FIGURE EVENT SAMPLES
2940      Events_1(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff
(Row,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)+Ovflw_ctr_event
2950                                                    !
```

```
2960      IF Events_1(Row)<Events_1(Row-1) THEN          ! CHECK FOR OVERFLOWS
2970        Events_1(Row)=Events_1(Row)+Unsigned_4_byte !  AND ADJUST
2980        Ovflw_ctr_event=Ovflw_ctr_event+Unsigned_4_byte
2990      END IF
3000                                                     !
3010                                                     ! FIGURE TIME SAMPLES
3020      Time_1(Row)=((Buff(Row,Col+4)+Unsigned_2_byte*(Buff(Row,Col+4)<0))+(Bu
ff(Row,Col+5)<0))*Unsigned_2_byte+Buff(Row,Col+5)+Ovflw_ctr_time
3030      Time_1(Row)=Time_1(Row)*2.E-9-BINAND(31,Buff(Row,Col+6))*1.E-10
3040                                                     !
3050      IF Time_1(Row)<Time_1(Row-1) THEN              ! CHECK FOR OVERFLOWS
3060        Time_1(Row)=Time_1(Row)+Unsigned_4_byte*2.E-9!   AND ADJUST
3070        Ovflw_ctr_time=Ovflw_ctr_time+Unsigned_4_byte
3080      END IF
3090                                                     !
3100      Time_1(Row)=PROUND(Time_1(Row),-10)
3110    NEXT Row
3120    DISP ""
3130  SUBEND
3140                                                     !
3150                                                     !
3160                                                     !
3170       !*********************************************************************
3180  SUB Proc_ti_2(Time_2(*),Events_2(*),INTEGER Block_length)
3190       !*********************************************************************
3200 Proc_ti_2:! SUBPROGRAM TO DETERMINE CUMULATIVE TIME AND EVENT ARRAYS
3210    ! FOR THE TI MODE WITH TIME HOLDOFF ARMING.  THE PROGRAM ALSO CHECKS
3220    ! FOR OVERFLOWS IN THE COUNT REGISTERS.
3230                                                     !
3240    COM /Data/ INTEGER Buff(*) BUFFER,@Hp5371a,@Controller_buf
3250                                                     !
3260    DISP TAB(30);"PROCESSING RAW DATA"
3270                                                     !
3280    INTEGER Row,Col,Interpolator,Byte,Words_per_samp
3290    REAL Ovflw_ctr_time,Ovflw_ctr_event
3300    REAL Unsigned_2_byte,Unsigned_4_byte
3310    REAL Systematic_err
3320                                                     ! INITIALIZE CONSTANTS
3330                                                     !  AND VARIABLES
3340    Byte=8
3350    Ovflw_ctr_time=0
3360    Ovflw_ctr_event=0
3370    Unsigned_2_byte=2^16
3380    Unsigned_4_byte=2^32
3390    Words_per_samp=7
3400    Systematic_err=6.00E-10
3410                                                     ! CONVERT FIRST EVENT AND
3420                                                     !  TIME SAMPLE
3430    Col=1
3440    Row=1
3450    Events_2(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff(R
ow,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)
3460    Time_2(Row)=((Buff(Row,Col+4)+Unsigned_2_byte*(Buff(Row,Col+4)<0))+(Buff
(Row,Col+5)<0))*Unsigned_2_byte+Buff(Row,Col+5)
3470    Interpolator=SHIFT(Buff(Row,Col+6),Byte)
3480    Time_2(Row)=Time_2(Row)*2.E-9-BINAND(31,Interpolator)*1.E-10
3490    Time_2(Row)=PROUND(Time_2(Row),-10)
3500                                                     !
3510                                                     ! LOOP TO CONVERT REST
3520                                                     !  OF EVENT AND TIME
```

```
3530                                                    !  SAMPLES
3540                                                    !
3550    FOR Row=2 TO Block_length                       ! FIGURE EVENT SAMPLES
3560      Events_2(Row)=((Buff(Row,Col)+Unsigned_2_byte*(Buff(Row,Col)<0))+(Buff
(Row,Col+1)<0))*Unsigned_2_byte+Buff(Row,Col+1)+Ovflw_ctr_event
3570                                                    !
3580      IF Events_2(Row)<Events_2(Row-1) THEN         ! CHECK FOR OVERFLOWS
3590        Events_2(Row)=Events_2(Row)+Unsigned_4_byte !  AND ADJUST
3600        Ovflw_ctr_event=Ovflw_ctr_event+Unsigned_4_byte
3610      END IF
3620                                                    !
3630                                                    ! FIGURE TIME SAMPLES
3640      Time_2(Row)=((Buff(Row,Col+4)+Unsigned_2_byte*(Buff(Row,Col+4)<0))+(Bu
ff(Row,Col+5)<0))*Unsigned_2_byte+Buff(Row,Col+5)+Ovflw_ctr_time
3650                                                    !
3660      IF Row MOD 2=0 THEN
3670                                                    ! PROCESS STOP
3680                                                    !  INTERPOLATOR DATA
3690        Time_2(Row)=Time_2(Row)*2.E-9-BINAND(31,Buff(Row,Col+6))*1.E-10
3700        Time_2(Row)=Time_2(Row)+Systematic_err
3710      ELSE
3720                                                    ! PROCESS START
3730                                                    !  INTERPOLATOR DATA
3740        Interpolator=SHIFT(Buff(Row,Col+6),Byte)
3750        Time_2(Row)=Time_2(Row)*2.E-9-BINAND(31,Interpolator)*1.E-10
3760      END IF
3770                                                    !
3780      IF Time_2(Row)<Time_2(Row-1) THEN             ! CHECK FOR OVERFLOWS
3790        Time_2(Row)=Time_2(Row)+Unsigned_4_byte*2.E-9!   AND ADJUST
3800        Ovflw_ctr_time=Ovflw_ctr_time+Unsigned_4_byte
3810      END IF
3820                                                    !
3830      Time_2(Row)=PROUND(Time_2(Row),-10)
3840    NEXT Row
3850                                                    !
3860    DISP ""
3870  SUBEND
3880                                                    !
3890                                                    !
3900    ! ************************************************************************
3910  SUB Det_clock(Time_2(*),Events_2(*),Clock,INTEGER Block_length,D,K)
3920    ! ************************************************************************
3930 Det_clock:! SUBPROGRAM TO DETERMINE THE AVERAGE CLOCK FROM THE DATA.  THE
3940    ! SUBPROGRAM USES THE MIDDLE PULSE OF THE TRIPOLE, USING THE DATA FROM
3950    ! THE TI MODE (SETUP NUMBER 2).
3960                                                    !
3970    DISP TAB(30);"DETERMINING AVERAGE CLOCK."
3980                                                    ! DETERMINE AVERAGE CLOCK
3990                                                    !
4000    Ptr=Block_length*2                              ! FOR SETUP#2 (TI A)
4010                                                    ! THERE ARE 2*N AVAILABLE
4020                                                    ! SAMPLES.
4030                                                    !
4040    WHILE (Events_2(Ptr)-Events_2(2)) MOD 6<>0      ! FIND A SAME POLARITY
4050      Ptr=Ptr-1                                     !  MIDDLE CLOCK PULSE
4060    END WHILE                                       !  TO USE AS AN ENDPOINT
4070                                                    !
4080    Clock_intervals=(.5*(Ptr-2))*(2*(D+1)+(K+1))    ! DETERMINE THE NUMBER OF
4090    Clock=(Time_2(Ptr)-Time_2(2))/Clock_intervals   !  INTERMEDIATE CLOCK
4100                                                    !  INTERVALS AS A
```

```
4110                                                    !  FUNCTION OF THE CODE
4120    Resolution=1.00E-10/(Block_length^.5)
4130    Clock=PROUND(Clock,INT(LGT(Resolution)))        ! ADJUST THE CALCULATION
4140                                                    !  FOR APPROPRIATE
4150                                                    !  RESOLUTION
4160    DISP ""
4170  SUBEND
4180                                                    !
4190                                                    !
4200    ! ************************************************************************
4210  SUB Drop_out_check(Events_1(*),Events_2(*),Result$,INTEGER Block_length,D,
K)
4220    ! ************************************************************************
4230 Drop_out_check:! CHECK FOR AMPLITUDE DROPOUTS FOR THE TRIPOLE DATA PATTERN.
4240    ! THIS ALGORITHM WILL ONLY WORK FOR DATA-DATA SPACING > 50NS AND
4250    ! < 100 NS.
4260                                                    !
4270    DISP TAB(25);"CHECKING FOR AMPLITUDE DROP-OUTS."
4280    Result$="NO DROPOUT"
4290    Ptr=1
4300    WHILE Ptr<=Block_length-1
4310                                                    !
4320      IF Events_1(Ptr+1)-Events_1(Ptr)<>2 THEN Dropout
4330                                                    !
4340      IF Events_2(Ptr+1)-Events_2(Ptr)<>1 THEN Dropout
4350                                                    !
4360      Ptr=Ptr+1
4370                                                    !
4380      IF Events_1(Ptr+1)-Events_1(Ptr)<>1 THEN Dropout
4390                                                    !
4400      IF Events_2(Ptr+1)-Events_2(Ptr)<>2 THEN Dropout
4410                                                    !
4420      Ptr=Ptr+1
4430    END WHILE
4440    GOTO No_dropout
4450                                                    !
4460 Dropout:                                           !
4470    BEEP
4480    PRINT "AMPLITUDE DROPOUT FOUND, IGNORE DATA BLOCK."
4490    PRINT
4500    Result$="DROPOUT"
4510                                                    !
4520 No_dropout:                                        !
4530    DISP ""
4540  SUBEND
4550                                                    !
4560                                                    !
4570     ! ************************************************************************
4580  SUB Compute_pk_shft(Time_1(*),Time_2(*),Events_1(*),Events_2(*),Clock,O_ps
_first,O_ps_mid,O_ps_last,E_ps_first,E_ps_mid,E_ps_last,INTEGER Block_length,D,K
)
4590     ! ************************************************************************
4600 Compute_pk_shft:! SUBPROGRAM TO COMPUTE PEAK SHIFT FROM THE TWO PASSES OVER
4610    ! THE DATA.
4620                                                    !
4630    DISP "CALCULATING PEAK SHIFT."
4640                                                    !
4650    Odd_first_last=0                                ! INITIALIZE VARIABLES
4660    Even_first_last=0
4670    Eo_first_last=0
```

```
4680    Eo_last_last=0
4690    Even_mid_first=0
4700    Eo_mid_mid=0
4710 Time_1:                                                !
4720    Counter_1=0                                         ! COMPUTE INTERVALS FROM
4730    Ptr=1                                               !  TIME_1 TIME STAMP DATA
4740                                                        !
4750    WHILE Ptr+7<=Block_length                           !
4760      Odd_first_last=Time_1(Ptr+5)-Time_1(Ptr)+Odd_first_last
4770      Oe_first_first=Time_1(Ptr+2)-Time_1(Ptr)+Oe_first_first
4780      Oe_first_last=Time_1(Ptr+3)-Time_1(Ptr)+Oe_first_last
4790                                                        !
4800      Counter_1=Counter_1+1
4810      Ptr=Ptr+4
4820                                                        !
4830    END WHILE
4840                                                        !
4850    Odd_first_last=Odd_first_last/Counter_1
4860    Oe_first_first=Oe_first_first/Counter_1
4870    Oe_first_last=Oe_first_last/Counter_1
4880                                                        !
4890 Time_2:                                                ! COMPUTE INTERVALS FROM
4900    Ptr=1                                               !  TIME_2 TIME STAMP DATA
4910    Counter_2=0
4920    WHILE Ptr+6<=Block_length
4930      Odd_first_mid=Time_2(Ptr+5)-Time_2(Ptr)+Odd_first_mid
4940      Oe_first_mid=Time_2(Ptr+3)-Time_2(Ptr)+Oe_first_mid
4950      Counter_2=Counter_2+1
4960      IF Counter_2>Counter_1 THEN
4970        Ptr=Block_length
4980      ELSE
4990        Ptr=Ptr+4
5000      END IF
5010    END WHILE
5020                                                        !
5030    Odd_first_mid=Odd_first_mid/Counter_2
5040    Oe_first_mid=Oe_first_mid/Counter_2
5050                                                        !
5060                                                        ! COMPUTE PEAK-SHIFT
5070                                                        !  TERMS
5080                                                        !
5090    Aconst=Odd_first_last-(((6*D)+(2*K)+8)*Clock)       ! SET UP CONSTANTS TO
5100    Bconst=Odd_first_mid-(((5*D)+(2*K)+7)*Clock)        !  MAKE PROGRAMMING
5110    Cconst=Oe_first_first-(((2*D)+(1*K)+3)*Clock)       !  EQUATIONS EASIER
5120    Dconst=Oe_first_mid-(((3*D)+(1*K)+4)*Clock)
5130    Econst=Oe_first_last-(((4*D)+(1*K)+5)*Clock)
5140                                                        !
5150    O_ps_first=(Odd_first_last+Odd_first_mid+Oe_first_first+Oe_first_mid+Oe_
first_last-((20*D)+(7*K)+27)*Clock)/(-6)                    ! COMPUTE EACH PEAK SHIFT
5160    O_ps_mid=Bconst+O_ps_first                          !  VALUE
5170    O_ps_last=Aconst+O_ps_first
5180    E_ps_first=Cconst+O_ps_first
5190    E_ps_mid=Dconst+O_ps_first
5200    E_ps_last=Econst+O_ps_first
5210                                                        !
5220                                                        !
5230    DISP ""
5240  SUBEND
5250                                                        !
5260                                                        !
```

```
5270    ! ************************************************************************
5280  SUB Asymmetry(O_first,E_first,O_mid,E_mid,O_last,E_last)
5290    ! ************************************************************************
5300 Asymmetry:! SUBPROGRAM TO CALCULATE ASYMMETRY FROM THE TRIPOLE DATA PATTERN
5310                                                    !
5320    DISP "CALCULATING ASYMMETRY."
5330                                                    ! DETERMINE ASYMMETRY FOR
5340                                                    !  TRIPOLE DATA PATTERN
5350                                                    !
5360    First_asym=(PROUND(ABS(O_first-E_first),-11))
5370    Middle_asym=(PROUND(ABS(O_mid-E_mid),-11))
5380    Last_asym=(PROUND(ABS(O_last-E_last),-11))
5390                                                    !
5400    PRINT "TRIPOLE ASYMMETRY:"
5410    PRINT "   FIRST EDGE: +/-";First_asym*1.E+9;" ns"
5420    PRINT "   MIDDLE EDGE: +/-";Middle_asym*1.E+9;" ns"
5430    PRINT "   LAST EDGE: +/-";Last_asym*1.E+9;" ns"
5440                                                    !
5450    DISP ""
5460  SUBEND
```

For more information, call your local HP sales office listed in your telephone directory or an HP regional office listed below for the location of your nearest sales office. Ask for the Electronic Instrument Department.

United States:
Hewlett-Packard Company
4 Choke Cherry Road
Rockville, MD 20850
(301) 258-2000

Hewlett-Packard Company
5201 Tollview Dr.
Rolling Meadows, IL 60008
(312) 255-9800

Hewlett-Packard Company
5161 Lankershim Blvd.
No. Hollywood, CA 91601
(818) 505-5600

Hewlett-Packard Company
2000 South Park Place
Atlanta, GA 30339
(404) 955-1500

Canada:
Hewlett-Packard Ltd.
6877 Goreway Drive
Mississauga, Ontario L4V1M8
(416) 678-9430

Japan:
Yokogawa-Hewlett-Packard Ltd.
29-21, Takaido-Higashi 3-chome
Suginami-ku, Tokyo 168
(03) 331-6111

Latin America:
Hewlett-Packard Company
3495 Deer Creek Rd.,
Palo Alto, CA 94304 USA
(415) 857-1501

Australia/New Zealand:
Hewlett-Packard Australia Ltd.
31-41 Joseph Street,
Blackburn, Victoria 3130
Melbourne, Australia
(03) 895-2895

Far East:
Hewlett-Packard Asia Ltd.
47/F China Resources Building
26 Harbour Road,
Hong Kong
(5) 833-0833

Germany:
Hewlett-Packard GmbH
Hewlett-Packard-Strasse
6380 Bad Homburg
West Germany
(49) 6172/400-0

France:
Hewlett-Packard France
Parc d'activité du Bois Briard
2, avenue du Lac
91040 Evry Cedex, France
(33) 1/60778383

United Kingdom
Hewlett-Packard Ltd.
Miller House—The Ring
Bracknell
Berkshire RG12 1XN, England
(44) 344/424898

Italy:
Hewlett-Packard Italiana S.A.
Via G. di Vittorio, 9
20063 Cernusco S/N (MI)
Milan, Italy
(39) 2/923691

Northern Europe:
Hewlett-Packard S.A.,
P.O. Box 999,
1180 AZ Amstelveen,
The Netherlands
(31) 20/437771

Southeast Europe/Africa/ Middle East:
Hewlett-Packard S.A.
1217 Meyrin 1, Geneva
Switzerland
(41) 22/989651

Or Write To:

United States:
Hewlett-Packard Company
P.O. Box 10301,
Palo Alto, CA 94303-0890

Europe/Middle East/Africa:
Hewlett-Packard Company
Central Mailing Department,
P.O. Box 529,
1180 AM Amstelveen,
The Netherlands

For all other areas:
Hewlett-Packard Company
Intercontinental Headquarters
3495 Deer Creek Rd.,
Palo Alto, CA 94304

HEWLETT PACKARD

Data Subject to Change
Printed in U.S.A.
02-5952-7928
May 1988