VALIDATION AND ENHANCEMENT OF TELETEXT SYSTEM SIMULATION

SOFTWARE USERS GUIDE

MCS File No: 8526
DSS File No: 12ST.36001-4-3095
DSS Contract No: OST84-00458
Date: March 18, 1985

Prepared by: K.W. Moulard G Wilson
for R. M. Armstrong, G. Wilson

Approved by:
S. Crozier

SUBMITTED BY:
MCS MILLER COMMUNICATIONS SYSTEMS LTD.
300 Legget Drive, Kanata, Ontario, Canada K2K 1Y5

(2)

VALIDATION AND ENHANCEMENT OF TELETEXT SYSTEM SIMULATION

SOFTWARE USERS GUIDE

MCS File No: 8526
DSS File No: 12ST.36001-4-3095
DSS Contract No: OST84-00458
Date: March 18, 1985

Prepared by: K. W. Moulard G Wilson
for (1) R. M. Armstrong, G. Wilson

Approved by: _____
S. Crozier

SUBMITTED BY:
MCS MILLER COMMUNICATIONS SYSTEMS LTD.
300 Legget Drive,
Kanata, Ontario,
Canada K2K 1Y5

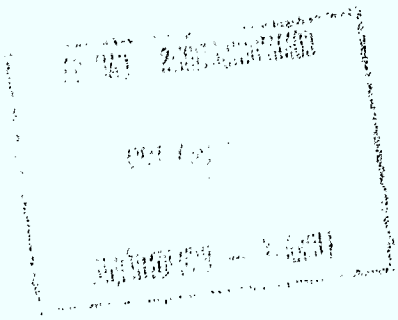

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Scope of Document

This document describes changes made to the TELIDON simulation software for the Contract DSS21ST.36100-2-4380 (MCS File No. 8342), and is intended to serve as a supplement to [1]. Only those portions of the original software which have undergone modification are discussed; the user is strongly advised to consult [1] as a first resort in all cases.

Changes were made to allow full advantage to be taken of the FPS-5105 Array Processor (AP) and VAX/750 minicomputer acquired by MCS in 1984. Using this combination, the speed of the simulation was increased by a factor of fifty times, allowing many more simulations to be performed, and many more scenarios and features of the simulation to be explored. The deliverable software includes this AP version of the software, as well as a functionally identical VAX-only implementation. In all cases, code is written in accordance with FORTRAN-77 standards.

1.2 Array Processor Utilization

The Array Processor greatly increases the speed of repetitive computations performed upon large vectors of values. For example, replacing the usual method of multiplying corresponding elements of matrices through use of a DO-loop construct with a call to the VMUL subroutine supplied in the AP math library typically cuts computation time by an order of magnitude or more. The execution time improvements associated with more complex operations, such as Fast Fourier Transforms, are even more dramatic.

The two disadvantages of the AP when compared to the VAX are (1) its inability to do disk I/O directly, and (2) the somewhat more primitive nature of the operations available on it. The first is a minor consideration; values can be transferred to and from the VAX host for output to disk, or for printing, with a minimum of difficulty. To keep execution times to a minimum, however, minimizing the number of such transfers performed is desirable. In aid of this, intermediate values calculated during the execution of the program which need not be output are left in the AP during program execution. This practice makes debugging the AP version of the program somewhat more difficult, but the increase in software development time was more than compensated for by the improved execution times noted above.

The second point forced a slight change in programming style. Vector values in the AP are referred to by the Advanced Math Library routines supplied by FPS by their base address and increment values. Throughout the AP version of the simulation these base addresses are given symbolic names (i.e. LSIGNAL refers to the location in AP memory of the base address of the SIGNAL vector). A brief summary of the Advanced Math Library routines primarily used in this simulation is provided in Table 1.

1.3 Summary of New Features

Three major functional changes, and several minor ones, have been made to the simulation. The first, and most important, is the implementation of the MK-V decoding scheme. A new bit-timing recovery scheme, and a new slicing level determination scheme, are included in this. The correlation method of bit-timing recovery has also been implemented in the new simulation, as has adaptive slicing level determination.

TABLE 1: FPS ADVANCED MATH LIBRARY SUBROUTINES USED

Routine	Result
APGET	Gets a vector from the AP memory locations specified
APPSEL	Selects which page in AP memory subsequently operations will be performed on
APPUT	Puts a vector into the AP memory locations specified
APWD/APWR/ APWAIT	Timing routine required to ensure synchronization of processing between host and AP
EVMOV	Moves a vector from one page of memory to another
CFFT	Performs forward or inverse FFT on the complex vector specified
CFFTSC	performs scaling on a complex vector after FFT operations
CVADD	Adds the two complex vectors specified
CVMUL	Multiplies the two complex vectors specified
RECT	Converts the complex vector values specified to rectangular form
VCLR	Fills the vector specified with zero values
VFIX	Converts the real vector specivied to integer
VFLT	Converts the integer array specified to real values

TABLE 1: FPS ADVANCED MATH LIBRARY SUBROUTINES USED (CONTINUED)

Routine	Result
VLIM	Clips a vector to specified upper and lower bounds
VLN	Performs the natural logarithm operation on the vector specified
VMOV	Noves a vector from one location within a page to another location in the same page
VMUL	Multiplies the two real vectors specified
VNEG	negates the vector specified
VRAND	Generates a vector of random numbers uniformly distributed on [0., 1.]
VREAL	Extracts real parts of a complex vector
VSMSA	Multiplies vector elements by one scalar and adds a second scalar to each value (linear transformation)
VSMUL	Multiplies vector elements by a scalar
VSQRT	Performs the square root operation on the vecotr specified
VSUB	Subtracts one real vector from another

Among the less important alterations, a third filter file format has been included. Setting the filter file format specifier to 2 causes the filter values corresponding to each frequency to be read from the file in complex notation (i.e. the three real numbers in each record of the filter file are read as the frequency in MHz and the real and imaginary components of the complex gain at that frequency). Also, a separation of random number generation seeds in the main program body has been effected, so that the data sequence generated for a given random seed is unique, and unaffected by the selection of bit-timing recovery method. This change was made to allow direct comparison of simulation runs performed with different bit-timing recovery schemes.

2.0 CHANGES MADE TO MAIN PROGRAM

TELSIM is composed of three main sections, one performing initialization, one generating and analyzing data, and a small third section to perform error counts. Of these, the second, which generates and analyzes data, has been largely vectorized.

2.1 Initialization

In the initialization stage, which has been modified only slightly from the original program, simulation parameters are obtained from an input file specified by the user in response to a program prompt. In addition, the function QSAMP is called to initialize an array of values used in the subsequent calculation of the Q-function. A single call is made to the VAX random number generation program to supply a real random number which can be used by the AP as a seed for its own random number generator; this call is necessitated by the difference in the format of variables the two routines used as seeds. A separate seed is also generated using the original seed for use in bit-timing recovery. The input parameters are then echoed to the output listing file.

The next stage of initialization is the creation of the filters required by the main loop of the program. This generation is done in the VAX, since there is often a requirement to read some or all of these filters from files. A new routine, NORM, is used in this stage to normalize filter values, a step necessary to ensure the correctness of the video signal to noise ratio calculations (see [2]). Another new subroutine, BTRINIT, is also called at this stage. This routine generates the frequency domain

representation of a bandpass filter to be used in the MK-V and Correlation approach to bit-timing recovery. The parameters for the impulse invariant approximation to a second-order Butterworth filter are also determined in this routine.

Calculation of gain and SNR parameters, as well as the creation of the invariant synchronization portion of the Telidon waveform, is done next. Once the gain and SNR values are known, the theoretical slicing level for the simulation parameters supplied is calculated; changes were made in this calculation to further generalize it.

After the error counter variables (KNT0 - KNT8) have been set to zero, constants required by calculations which are to be performed in the AP are stored in the arrays REALS and INTGS. While it is possible to move data from the VAX host to the Array Processor a single variable at a time, a considerable improvement in speed is obtained if a single DMA request is made, so these constants are stored in arrays which can then be moved in a single call. Data stored in arrays which are transferred to the AP using the APPUT subroutine are stored in the Array Processor sequentially in the same order as in the host. Thus, although the particular ordering of values in REALS and INTGS is unimportant, this order must correspond to the locations set in the memory table (see below). The preamble bit sequence is set in this section as well.

To improve the clarity of the program, memory locations in the AP are referred to by symbolic mnemonic names. The values associated with these mnemonics are set during this initialization stage, and are not changed during subsequent

program execution. As AP memory is divided into pages, separate memory maps are required for each page (see the internal documentation of the code). Care must be taken when changing the size of any of the arrays, or the location or number of the variables, used in the AP.

Once these memory maps have been created, the new subroutine REVERS is called to rearrange the values stored in the filter arrays in the host. This is necessitated by the differing input formats expected by the VAX and AP FFT routines. Constants and filter values are moved into the AP, and filters combined to minimize subsequent calculations.

2.2 Signal Generation and Analysis

The main loop of the program generates, then filters, adds noise to, and analyzes the Telidon signal. tHe number of times this loop is executed is set by the user in the input parameter file; each loop execution creates one burst of Telidon data. The Array Processor is used to perform all operations in this sequence of calculations except the generation of impulsive noise.

The first step is the generation of random data bits. This is performed by generating a sequence of random numbers on [0., 1.], subtracting 0.5 (so that the numbers lie in [-0.5, 0.5]), and then applying a limiting function to create a sequence values which are either +1.0 or -1.0. Parity bit generation is done by successively multiplying the first, second, third, ..., seventh bits of each simulated bytes of the data sequence. Since vector operations in the AP are done by specifying the initial address of the data to be operated on, and the increment separating successive vector elements, this parity

generation is quite simply done by moving every seventh 'bit' (in actuality stored as a word in the AP) into the position corresponding to the eighth 'bit', and then successively multiplying the sixth, fifth, ..., first bits against this value to create the parity values.

Once the bit sequence has been generated, it is sampled and scaled to produce the Telidon line signal, then transformed into the frequency domain. Pulse shape filtering is applied by multiplying the frequency-domain pulse filter against the signal values, after which synchronization and colour burst information is added. Transmission filtering and multipath filtering are applied simultaneously, these two filters having previously been compressed into one pre-noise filter (see above).

If the type of noise selected was not GAUSSIAN, impulsive noise is generated in the VAX host, and these values put in the Array Processor. Since the generation of impulse noise is not amenable to vectorization, it was felt that its generation would benefit only marginally from being done in the AP.

Gaussian noise is then generated, using the Array Processor. To do this, random numbers with a uniform distribution are first created, and then transformed to produce values with a Gaussian random distribution with specified mean and variance. These noise values are added to the impulse noise values (if any were introduced), and the result is added to the frequency-domain signal.

The signal is then passed through the receiver and receiver pulse shaping filters (which have previously been combined into one post-noise filter). If the bit-timing recovery type selected by the user is CORREL or MK-V, the operations

involved in this bit-timing recovery are then performed. The signal is then transformed to the time domain and scaled, and the imaginary part of each signal sample is zeroed. These results are then returned to the host for further processing.

The final part of the main loop is executed in the VAX. In it, data estimation is performed, following which the data sequence is extracted from the received signal, decoded, and passed through a parity checking routine. Finally, the error count for this burst is obtained.

2.3 Final Output

Once the data generation and analysis loop has been executed the number of times specified, the routine OUTRES is called to place summary error counts in the output listing file. This routine has not been changed.

3.0 CHANGES MADE TO SUBROUTINES

The following subroutines have been altered in whole or in part from their original form.

3.1 BTR

The bit-timing recovery routine has been modified to allow MK-V decoder bit-timing recovery to be performed, while still allowing MK-IV bit-timing recovery to be simulated. If the 'MK-V' option was selected in the input parameter file, the 6th positive to negative zero crossing in the data sequence is located and used to characterize the position of the first bit.

3.2 MULGEN

The previous version of MULGEN performed calculations on values which were out of the band of frequencies of interest. The upper and lower bounds on the main loop of this routine have been altered so that only those calculations having an effect upon program execution are actually performed.

3.3 RECOVERSEQ

This routine takes the word/bit timing mark from the WORDSYNC routine (described below) and uses it to recover the transmitted sequence. This routine has been modified to perform adaptive slicing, if the user has so requested.

3.4 RDFILT

The RDFILT routine, which reads filter files from disk and performs interpolation to create a full-sized array of

values representing that frequency-domain filter. The previous version of the routine allowed the user to specify as filter file formats either format 0, in which filter values were given as gain magnitude and phase, or format 1, in which filter values were given as gain magnitude and group delay. The present version also permits format 2, in which filter values are given as complex numbers.

3.5 RECEIV

The RECEIV routine is an interface between the main program and the bit-timing recovery routine RECOVR. In keeping with the changes made to the latter (described below), the parameters passed into and out of RECEIV have been altered.

3.6 RECOVR

RECOVR has been altered to permit the user to select Correlation-type bit-timing recovery. In addition, the code to handle the case in which correlation-type bit-timing recovery is not selected has been altered to accommodate MK-V bit-timing recovery.

3.7 SLICER

This routine determines the slicing level for the received Telidon signal. This can be determined in the routine either through a calculation of the average signal level over a number of bits, or through use of an approximation to the method implemented in real terminals, which uses peak detection. This routine has been altered to accommodate the 'ADAPTIVE' and 'MK-V' slicing options, as well as the 'AVERAGING' and 'IDEAL' options available previously.

3.8 WORDSYNC

This routine determines the word synchronization. The bit-timing recovery routine supplies a timing reference, which should be in the center of the first bit in the preamble. If this is the case, the simplified 'add offset' method would work, and this option is provided for simplicity in testing. However, if the sequence has been shifted in time, the word synchronization byte must be used. This routine simulates the real decoder by comparing the word-sync sequence to the original sequence to determine the most probable synchronization point.

The new version of WORDSYNC also calculates the initial sample and hold values for adaptive slicing.

3.9 ZCROSS

This routine determines the position of the zero crossings during the preamble. These measurements are used by the BTR to determine the sampling instants. The crossing point is approximated by a linear interpolation between points on opposite sides of the slicing level. These crossing points are stored in the array CROSS, with the direction of the crossing indicated by the sign of the timing mark. This routine has been modified to allow processing in accordance with MK-V decoder specifications.

4.0 NEW SUBROUTINES

The following new subroutines have been created, and added to existing software.

4.1 NORM

This routine takes as input a complex array of values and a scaling factor by which each value is to be multiplied. This routine is used to scale each filter array by the value of its center element (i.e., by its value at a frequency of 0 Hz), to facilitate calculation of signal to noise ratios.

4.2 BTRINIT

This subroutine generates the frequency domain representation of a bandpass filter used in the MK-V and Correlator approaches to bit-timing recovery. The parameters for the impulse invariant approximation to a second-order Butterworth filter are also determined in this routine. These IIR parameters are determinable for 'ADAPTIVE' and 'MK-V' slicing types.

4.3 CORBTR

This is a new bit-timing recovery routine which uses the correlation method of bit-timing recovery. The 14 bits of the preamble are used to estimate clock phase; the routine must choose between one of 5 possible clock phases. Because these first bits of the preamble are alternating +1/-1, they will be correlated with a clock runing at twice the bit rate. This routine assumes that there are 11 samples per bit, and that the possible clock phases are offsets of 0, 2, 4, 6, or 8 samples.

As input, this routine requires the windowed and amplitude limited portion of the synchronization signal that was passed through the bandpass filter. As output, it supplies the estimated sampling point of the first bit in the synchronization signal.

4.4 SLFILT

This function calculates the impulse invariant digital filter approximation to a 2nd order Butterworth filter, using the values passed in the input arrays ESTATE and SFILPAR.

4.5 REVERS

This simple routine is required because of the different input formats expected by the VAX FFT routine inherited from the previous software and the AP FFT routine supplied by Floating Point Systems. REVERS simply rearranges the frequency-domain terms of the array passed as input, so that terms are strictly ordered by increasing frequency value.

5.0 SUPPORT PROGRAMS

The following programs are external to the main simulaiton program, and were written to aid in the analysis and display of results.

5.1 CODING

This program is equivalent to the MCODING program created for the previous contract; all input and output, as well as all operations performed, are identical with the previous version.

5.1.1 ERRGEN

This program generates independent error sequences at a specified input bit error rate. The input parameter file is by default FOR004; a redefinition of this logical unit number is necessary if a different input parameter file is to be used.

The input parameter file must contain the following four parameters:

(1) Name of the output file (extension •PAR is supplied by program).

(2) Desired probability of error.

(3) Seed for random number generator (integer).

(4) Number of packets to be generated.

The file generated by this program can be analyzed directly by CODING. ERRGEN was developed to aid in the verification

of the simulation's correctness. ERRGEN requires little execution time, and thus may be reasonably operated at low error rates.

5.2 COMPRESS

This program takes error sequences generated by the TELSIM program and places them in a compressed format for use with the CODING program. When prompted, the user must supply the name of an error sequence file to be processed. the program continues processing input files until the user specifies a null file (does not enter a name). The program prompts the user for the name of the output file into which the compressed sequences are to written.

5.3 TAPEAREAD

This program reads error sequence files generated at CRC from tape and outputs the data stored in them in a compressed format suitable for use with the CODING program.

5.4 MC

This program separates the effects of word synchronization loss on the error rate expressed by a TELSIM error listing file. The program also averages error results from different simulation run performed with the same data sequence to create 95% confidence intervals for the actual error rates. The number of packets lost is also calculated and output to the user.

The program first prompts the user for the portion of the data file name which all the files to be processed have in common; this is typically the first part of the prefix portion of the file name. The user is then prompted for

the TV signal to noise ratio; this value is output to the output files created.

The program then reads all the files possessing the common portion of the file name supplied by the user and processes them. Three output files are generated: one containing the results of the interactive session, a second (PLOT1.DAT) and third (PLOT2.DAT) containing the SNR and error rate before separation as pairs (for use with the plotting program), and the SNR and error rate after separation in the same format, respectively.

The program repeats this cycle of processing until the user specifies a null file name (does not enter a string before typing <RETURN>).

5.5 SUPPLY

SUPPLY converts sampled end-to-end time-domain impulse response values to frequency-domain filter values, and then factors out the pulse shape. Given the name of a file containing the complex impulse response value to be used, the program reads in these values, performs an FFT, and queries the user for the names of three pulse shape specification filters to be factored out. These filters may be specified in a manner identical to the specification of filters in TELSIM.

During its execution, the program produces several auxiliary output files containing amplitude and group delay responses of the filter generated, both before and after factorization. For more details on these files, see the internal documentation of the program.

5.6 MAKEYE

MAKEYE reads the eye diagram data files created by the TELSIM program and uses the data they contain to create output files which, when plotted, create an eye diagram. When prompted by the program, the user must enter the name of the data file to be used as input; this will be of the form <runid>.SAV, where <runid> is the run identifier that was specified by the user in the parameter file input to the EYETEL program. The output file of MAKEYE will then be titled <runid>.EYE, and may be submitted directly to a HP7470 plotter.

5.7 FILGEN

FILGEN is similar functionally to the $FILE-GENERATOR routine supplied for the previous contract. Written in DCL (Digital Command Language), this routine aids the user in creating simulation parameter files for series of simulations.

FILGEN is invoked by typing @FILGEN; optionally, a parameter name may be specified on the same line, in which case jobs are submitted to the batch queue named. FILGEN's user interface is identical with that of the original parameter file generation routine; the only difference in output is that digits particularizing the parameter files are not separated by underscore characters ('_') as they were in the filenames generated by the previous program, since the underscore character is not recognized by the VMS operating system as a legal character in a filename definition.

In conjunction with FILGEN's use, a second command file called SYNC.COM must be present. This parameter file is

used to control synchronization of job execution, and is needed when multiple batch queues and multiple jobs are being used. Table 2 contains an example of such a command file.

5.8 FACTOR

FACTOR allows the examination of the effects of various perturbations of a sampled impulse response profile on the resulting frequency response. The program first queries the user to supply the name of a file containing complex impulse response values. The user must also supply a value for the DC offset to be added to the real parts of these values; if no DC offset is desired, the user must enter '0'. The impulse response values are then read, and the DC offset (if any) added.

The user may then elect to incorporate either Gaussian noise effects or the effects of data autocorrelation (non-white data sequence values). In both cases, the user must respond to program prompts.

The perturbed impulse response values are finally converted to the frequency domain using an FFT, and these values output to file for plotting or inspection.

5.9 IMPTEL

IMPTEL produces the end-to-end sampled impulse response of a given TELSIM configuration. Requiring an input parameter file identical to those used by TELSIM, IMPTEL constructs the various filters specified, concatenates them, and determines their time-domain impulse response profile. These values are then written to a file suitable for use in FACTOR.

TABLE 2: EXAMPLE OF SYNC.COM

```
$ ! THIS JOB IS DONE TO INITIALIZE THE SUBMITTING OF FILGEN
$ WAIT 0:02
$ EOJ
```

5.10 FILTER

FILTER is a simple utility program which reads in several filter files specified by the user, concatenates them, and writes to file their overall amplitude response characteristic. The program operates interactively, querying the user for the name(s) of the filter file(s) to be included.