MAXIMAL LENGTH SEQUENCE MEASUREMENTS - A CONTEMPORARY METHODOLOGY FOR WINDOWS® SYSTEMS.

R. Walker, Consultant

1. INTRODUCTION.

Audio and acoustic measurements based on the use of pseudo-random noise (PN) have been available for some years now [1]. This paper presents a brief review of the theoretical basis of PN-based measurements. It then demonstrates how freely-available (public-domain) library packages for Fourier transformation and the integrated audio facilities included in nearly every modern personal computer can be used the carry out the measurements.

2. IMPULSE RESPONSE MEASUREMENT.

For a linear, time-invariant system the impulse response (IR) theoretically contains all of the information about the system. The IR is a convenient starting point for the extraction of more familiar responses, such as the response magnitude as a function of frequency (amplitude response) or the response decay as a function of time (e.g. reverberation time).

In principle, to obtain a system IR an infinite amplitude, zero-duration signal can be applied to the system under test and the resulting time-domain response recorded. In practice, implementing that method directly leads to severe problems with system dynamic range and measurement signal-noise ratios [2]. No real system can respond linearly to an infinite input, even if it could be created, introducing (at least) amplitude distortion. In addition, the total energy in the output signal is limited. Improvements to the signal noise ratio can be obtained by averaging many responses [3], but the amplitude limitations will always remain.

The principles of IR measurement with pseudo-random (deterministic, but noise-like) signals are well known [4]. They involve the use of real and practical signals, i.e. ones that are constrained in amplitude and duration, to excite the system under test. The resulting system response can be processed arithmetically to derive a close approximation to the infinite IR. The problems of amplitude non-linearity can be significantly reduced (thought they may still need consideration). The signal-noise ratio problems can also be reduced by the extended nature of the excitation, in which the average power in the test signal is increased (for finite amplitude) and can thus contain much more total energy than a short pulse.

In practice, the finite bandwidth, sampling frequency and resolution of modern digital signal generation and system response recording methods introduce additional limitations. However, those limitations are always calculable and can, in most applications, be arranged to provide adequate performance for the task, but they must always be taken into account when interpreting results of measurements. For example, the finite record length sets the frequency resolution – the true resolution cannot be finer than the reciprocal of the total record length. In the frequency domain, sampling sets the upper limit of the frequency range to one-half of the sampling frequency. Within these well-understood limitations, the IR contains all of the information about the system under test. Further processing of the IR can be carried out to extract any other parameters or functions [e.g. 5].

3. IMPULSE RESPONSE MEASUREMENT USING PSEUDO-NOISE TEST SIGNALS AND CORRELATION.

The test signal needs to be both relatively long in duration and contain all of the frequency components needed for the measurement. If the test signal has an auto-correlation function which is (or is nearly enough) an impulse then the system IR can be calculated arithmetically by correlation of the system response with the original test signal.

One common PN test signal is a maximal-length sequence (MLS) of length *N* samples, where *N* is of the form $2^n - 1$, *n* being integer. They are easily generated in hardware in real time or by a simple calculation as stored sequences, of enormous lengths if needed[1]. For system testing they have some highly desirable properties. The sequence values are limited to ±1 only, giving a peak/rms ratio of 1.0, which is the lowest possible, and easy generation by binary logic circuits. All of the frequency components they contain are the same amplitude but of pseudo-random phase. They have normalised auto-correlation function values of unity for zero shift and 1/*N* for all other shifts. Thus, their auto-correlation function is close to the required rectangular impulse. For *N* = 1023, 1/*N* has a value of -60 dB, which is good enough for many acoustic applications. Typically, values of *N* of about 4,095 to 65,535 would be used in practice, gives a values for 1/*N* ranging from -72 dB to -96 dB, good enough for nearly all acoustic or electronic applications.

Historically, one of the difficulties with this approach has been actually carrying out the required correlation operations. In the past, it was extremely time consuming. As recently as 25 years ago, this author was using expensive, dedicated hardware to process 1024-point samples in a second or so. In principle, a 1024-point correlation requires 1024 summations, each involving 1024 multiply-accumulate operations, or about 2 million arithmetical operations. Today that would be trivial (< 5 ms). However, practically useful correlation lengths up to 64,000 points would still take a significant fraction of a second, even on a modern processor. It is likely that a 64k-point correlation could take a modern processor up to 6 s to execute, if done using the direct method.

Over the years, algorithms have been devised to reduce processing times for signal transformation, especially for carrying out the Fourier transform. Those algorithms break up a long transform into shorter segments. It is much faster to process many short transforms and re-assemble the results than to carry out the longer transform directly. Carl Friedrich Gauss around 1805 already knew such algorithms. Various limited forms were rediscovered several times throughout the 19th and early 20th centuries. In 1965, Cooley and Tukey [6] published the modern form of the Fast Fourier Transform (FFT) for electronic computers. In 1972, this author was using the related, but arithmetically simpler, Fast Hadamard Transform (FHT) to process real-time video signals (110 Mb/s) for bit-rate reduction [7, 8]. Today, the FFT and its close relative the Fast Cosine Transform are ubiquitous, widely used in video and audio coding (MP3, AAC, DTT, digital video, mobile phones, etc.) and in many other fields.

In MLS measurements, these fast transform algorithms can be used to implement the correlation function. The convolution of two signals can be achieved by multiplying their Fourier transforms and then using the Inverse FFT (IFFT) to return the result back to the time domain. If the complex conjugate of the Fourier transform of the second signal is used then the final result is the cross-correlation. Fig. 1 shows the operation schematically. If $x(t)$ is the original test signal and $y(t)$ the recorded system response then $corr(x(t),y(t))$ is the wanted impulse response. It should be noted that the MLS sequence length of 2^n-1 does not fit exactly with these FFT processes, which are significantly better matched to lengths of 2^n. The missing sample is usually filled by zero padding.

[1] A 2^{32}-1 sequence, easily generated with 32-bit integer arithmetic, would take nearly 25 hours to reproduce at 48 kHz sampling frequency.

That has no significant effect in practice, but it is needed to keep the processing optimised. In references to sequence lengths in the remainder of this paper the padded length will be intended.

All of this processing was used by Borish and Angell in 1982[1] and subsequently incorporated into MLSSA™, developed by Douglas Rife and John Vanderkooy in the late 1980's [4, 5]. It has been much used and emulated ever since. Because of processor limitations, early systems used the FHT rather than the FFT as it was simpler to implement and quicker in operation [4]. Many still do. However today, with modern processors and maths co-processors that take no more time to do multiplication than addition, the FHT has no practical speed advantage.

4. SYSTEM REQUIREMENTS.

To implement an MLS measurement system requires a signal generator to create the MLS signal, an audio output system to drive the system under test and recording record to acquire the resulting response, a means to carry out the correlation and a means of displaying the final results.

Data storage is required for the audio signals and the intermediate, transform-domain data. If the audio data is represented as 16-bit words and the (complex) transform data by (two) 32-bit words, the total storage required is of the order of 38*N* bytes, where *N* is the MLS length. For *N* = 4096, that amounts to 156 kB, which is trivial today. Even for *N* = 262,144, which would be considered to be a long sequence for most acoustic purposes, the total would be 'only' about 10 MB[1]. A sequence that long would take about 5 seconds to reproduce once (at 48 kHz sampling frequency) and would give a resolution of around 0.2 Hz in the frequency domain, or about 100,000 discrete values over the normal audio spectral range.

5. IMPLEMENTATION IN WINDOWS® BASED COMPUTERS.

5.1. Signal Generator and Audio I/O.

With modern desktop or laptop computers, replay and recording of audio signals is an integral part of the multimedia system. Standardised procedure calls can set up, output and re-record audio signals in high enough quality for room acoustic measurements and most other audio measurements. In principle, the operations are trivial[2].

5.2. Correlation.

Writing FFT routines, especially efficient ones, is non-trivial. Usually, for the highest possible processing speeds, at least some of the code needs to be written in assembly language, or some other language that allows cycle-by-cycle control of the central processor unit (cpu). That leads to potential incompatibilities between processors and the prospect of different optimum coding for the different cpu types that might be encountered.

One option would be to accept a less-than-optimum solution. Today's processors are so fast that adopting this approach might be acceptable. After all, there would be no significant difference if a human user had to wait 10 ms instead of 1 ms for a result. However, that approach would, even today, lead to noticeable delays for longer transforms. It would also restrict the potential speed of automated measurement systems.

[1] In practice, additional storage would be needed for graphical display of the results but, even then, the total system memory requirements would not be excessive for a modern computer.

[2] Though this paper is limited to Microsoft Windows-based computer systems, all of the principles can easily be applied to any other OS that includes audio I/O and for which the fft library package is available. That certainly includes UNIX, Linux and variants, MAC-OS, OS2 and ARM-based machines. In principle, the source code can be compiled on any machine for which an ANSI-standard C compiler is available.

An alternative is to use an FFT process that is self optimising. One reasonably well known example is “FFTW” (or the “Fastest Transform in the West”) [9]. This is a ‘free’[1], self-optimising FFT compiler that adjusts the run-time process coding to match itself to the system it is running on at the time - a class of self-modifying code. Fortunately, the developers have made its use very simple – as a programme library it is no more difficult to use directly than any other. Implementation of the self-optimising parts require very little additional user effort.

FFTW also incorporates a “learning” facility, so that repeated uses of a particular transform on the same system use the same optimised code. The ‘learned’ data can be passed from one invocation to another, at any time in the future, through the use of stored “wisdom” files.

6. MLS-BASED IMPULSE RESPONSE MEASUREMENTS.

6.1. General Outline.

The basic components shown in Fig. 2 are needed to carry out an MLS-based IR measurement using correlation. The test signal and the complex conjugate of its Fourier transform need only be calculated once (for any setting of the sequence length). Fig. 2 also indicates the general storage requirements at each stage of the process, relative to the MLS length.

In practice, because of the requirements for different numerical representations in different components and a slightly longer receive array length to allow for timing fluctuations (see Section 7.2), the actual processing requirements are a little more complicated, as shown in Fig. 3. Fig. 3 also gives the storage requirements at each stage of the process, relative to the MLS length. The data format *fftw_complex* is a custom-defined number format in the FFTW library. All of the function prototypes are defined with that number format. It is identically equivalent to a tw-element array of ANSI *float* variables. The FFTW package default option is to use the equivalent of double representation but, for MLS-based measurements of ‘ordinary’ audio systems, the higher resolution is not required. ANSI *float* gives a working arithmetic dynamic range of significantly more than 100 dB, even after several stages of computation.

Appendix A–1 shows an outline flow chart for the process. The theoretical basis for MLS-based IR response measurements includes the assumption that the signal is infinitely cyclically repetitive. In order to satisfy that requirement the sequence is repeated, the actual measurement being taken for the second sequence. It also implies that the sequence must be longer than the total response time of the system under test. In practice, it is usually assumed that a minimum sequence duration of about 1/3rd of the reverberation time (RT) is adequate for ordinary room acoustic measurements. In more critical cases, a length about equal to the RT is usually sufficient[2].

6.2. Programme Coding.

The appendix shows a flow chart for the sequence of operations (A–1) and gives some brief extracts illustrating elements of program coding (A–2). For clarity and simplicity, the code fragments were taken from the prototype command-line programme to avoid the complications of class member variables and functions in the illustrations. Illustration is limited to the FFT components because programming for multi-media applications is well described in standard OS library packages. The wave files need to be set up, initialised with data and then the play and record functions called as needed.

[1] Starting with version 1.3, FFTW is Free Software in the technical sense defined by the Free Software Foundation, and is distributed under the terms of the GNU General Public License. Previous versions of FFTW were distributed without fee for noncommercial use, but were not technically “free”. Non-free licenses for FFTW are also available that permit different terms of use than the GPL

[2] The detailed limitations of the method are well known and are outside the scope of this paper.

A–2.1 shows the simple way in which the pre-programmed 'plans' are set up for the FFT functions. This simple code hides all of the optimisation carried out by the FFTW package. A plan includes the locations of the input and output data arrays, the length and the optimisation strategy to be used. A-2.2 shows how a previously set up plan is used to execute the FFT. The fragment also includes timers to print out execution time data. A–2.3 shows two components, the first for opening and loading an existing "wisdom" file and the second for saving the new "wisdom" data. The new data may be more comprehensive than that loaded, depending on whether additional plans had been created in the current invocation.

7. PERFORMANCE TESTS.

7.1. Processing Times.

The execution times were measured on a two-year old ACER 4500 series laptop computer with an Intel Centrino M 1.8 GHz processor and 1GB RAM. This machine was only mid-range when purchased and would now be considered old.

Measurements were made of the time to prepare the FFT plans, carry out the actual FFTs and the complex multiplication of the transform-domain data, using a timer resolution of about 0.3 μs. Sequence lengths of 2^{12} (4,096) to 2^{20} (1,048,576) were measured. Measurements of the FFT planning time were made with and without the wisdom data. That corresponds to first use and subsequent use of any particular sequence length. It should be noted that the wisdom data is usually carried over from one use to the next, so the long preparation times would only be seen on first use of the programme. Times are in ms and were somewhat variable.

Initial measurements, no stored wisdom data.

Length (2^n)	**n=12**	**14**	**16**	**18**	**20**
Plan 1 (Forward transform 1)	250	920	3,830	22,000	123,640
Plan 2 (Forward transform 2)	0.033	0.064	0.068	0.143	0.260
Plan 3 (Inverse transform)	3.76	10.16	7.48	9.79	284
Fft1 (output signal preparation)	0.150	0.725	3.379	22.49	122.4
Fft2 (received signal processing	0.116	0.606	3.416	23.39	121.6
Multiply transform domain data	0.148	0.574	2.58	0.27	37.23
Ifft (Inverse transform)	0.119	0.588	2.626	24.01	156.4
Total for each new measurement.				≈48	≈314

Using previously stored wisdom data.

Length (2^n)	**n=12**	**14**	**16**	**18**	**20**
Plan 1 (Forward transform 1)	2.6	10.7	41.1	122	315
Plan 2 (Forward transform 2)	0.09	0.172	0.177	0.15	0.185
Plan 3 (Inverse transform)	0.101	0.185	0.186	0.08	268

This illustrates the very significant savings in preparation time as a result of the storing of the plans. In the second table, the actual FFT execution times were, as expected, essentially identical. It also illustrates that, even on this relatively modest laptop computer, the time to carry out the whole of the processing scheme for each new measurement was only about 320 ms, even for a 1 MB impulse response.

7.2. Latency.

One of the difficulties with trying to carry out real-time operations on general-purpose hardware that is being used under a multi-tasking OS is latency. Sometimes, commands do not execute at the desired time. For this measurement system, the actual rate of the audio I/O is not affected (the essence of audio I/O is a consistent sample rate). However, the start and stop delays will not be either consistent or short. For recording the system response (audio input) it is necessary to issue the start command somewhat before the proper time. The input buffer must be long enough to accommodate the advanced start so that the end point is properly recorded. On the original laptop computer, the required advance was around 280 samples at 48 kHz.

The second effect of this delay is the resulting lack of synchronicity between the playout and recording functions. In most commercial measurement systems running on general-purpose hardware that problem is overcome by using one of the two stereo channels for the measurement and the other as a hard-wired reference. That way the exact delay can be removed from the measured impulse response. In this demonstration programme the delay was removed by searching for the peak in the impulse response and adjusting the origin accordingly. That only requires one channel (or alternatively would allow two-channel measurements) and will work where it can be assumed that the largest value in the impulse response does represent the reference delay. That cannot always be assumed, so the safest method is to use a reference channel. That was not implemented in the demonstration system.

7.3. Sample Display.

Fig. 4 shows the main dialogue window from the demonstration Windows® programme. Fig 5 shows a screen shot of the frequency response measured for the laptop computer from its built-in loudspeakers to its built-in microphone. The response was essentially indistinguishable from that measured using a commercial software MLS measurement system.

8. POST-PROCESSING.

This paper is not directly concerned with the very large number of post-processing options that could be applied to the impulse response. These are well known in the literature. Only a brief list is given, mainly in order to illustrate that the IR can be used in many different ways, once it has been obtained.

8.1. Time Domain Responses.

The IR response can be processed directly in the time domain to provide time-histories of the system response. Alternatively, the time-domain responses may be pre-filtered in the frequency domain, most economically by applying appropriate frequency-weighting windows to the stored frequency data before carrying out the final IFFT calculation. In room acoustic measurements, the reverberation time can be displayed, either directly as an envelope of the (filtered) IR or by reverse (Schroeder) integration. Room reflections can be identified by using shifted short transforms, “Energy-Time-Frequency” or CDS or “Cumulative Decay Spectra”. Each of these functions has advantages and limitations.

8.2. Frequency Domain Responses.

Responses in the frequency domain can be calculated from the IR by applying the FFT again to the data. The time data can be (and usually is) restricted by applying time-domain windows to the IR. Of course, if the transform of the whole response were wanted then it would already exist in the system. In most cases, some limitation of the time domain response is needed, so the IR needs to be calculated in order to apply the time-domain window. For example, approximations to loudspeaker responses as a function of frequency can be measured in a non-anechoic environment by applying appropriate windows in the time domain to mask out unwanted data.

8.3. Combined Time and Frequency Domain Responses.

Many measurements involve combinations of time and frequency domain presentations. Examples are Energy-Time-Frequency responses (ETF), Cumulative Energy spectra and spectrograms. ETF measurements have been used for identifying early reflections in room acoustics. In particular, Modulation Transfer Functions (MTFs) can be extracted [5, 10, 11] and from them measurements such as Sound Transmission Index (STI) or RASTI can be derived [12, 13].

9. FURTHER WORK.

The development described in this paper was only continued to the point where the principle could be proved and the performance assessed. The software was mainly tested using a command-line window with separate graphical display software. A very elementary 'OOP' Windows application was also developed to provide illustrations. No work has been done on a user-friendly application, with its large list of post-processing options.

The author's main objective was to illustrate the relative ease with which the PC programme and hardware environment could be used to do MLS-based measurements. Several commercial systems are available to carry out these sorts of measurements. They also have the advantage of a wide range of post-processing options being written already by their authors. However, one of this author's objectives was to incorporate the functionality in a dynamic link library (dll). Users (who needn't have to understand the details) could then use the library in their own applications to carry out special measurement functions. That could even extend to calling measurement functions from, say, an Excel® spreadsheet.

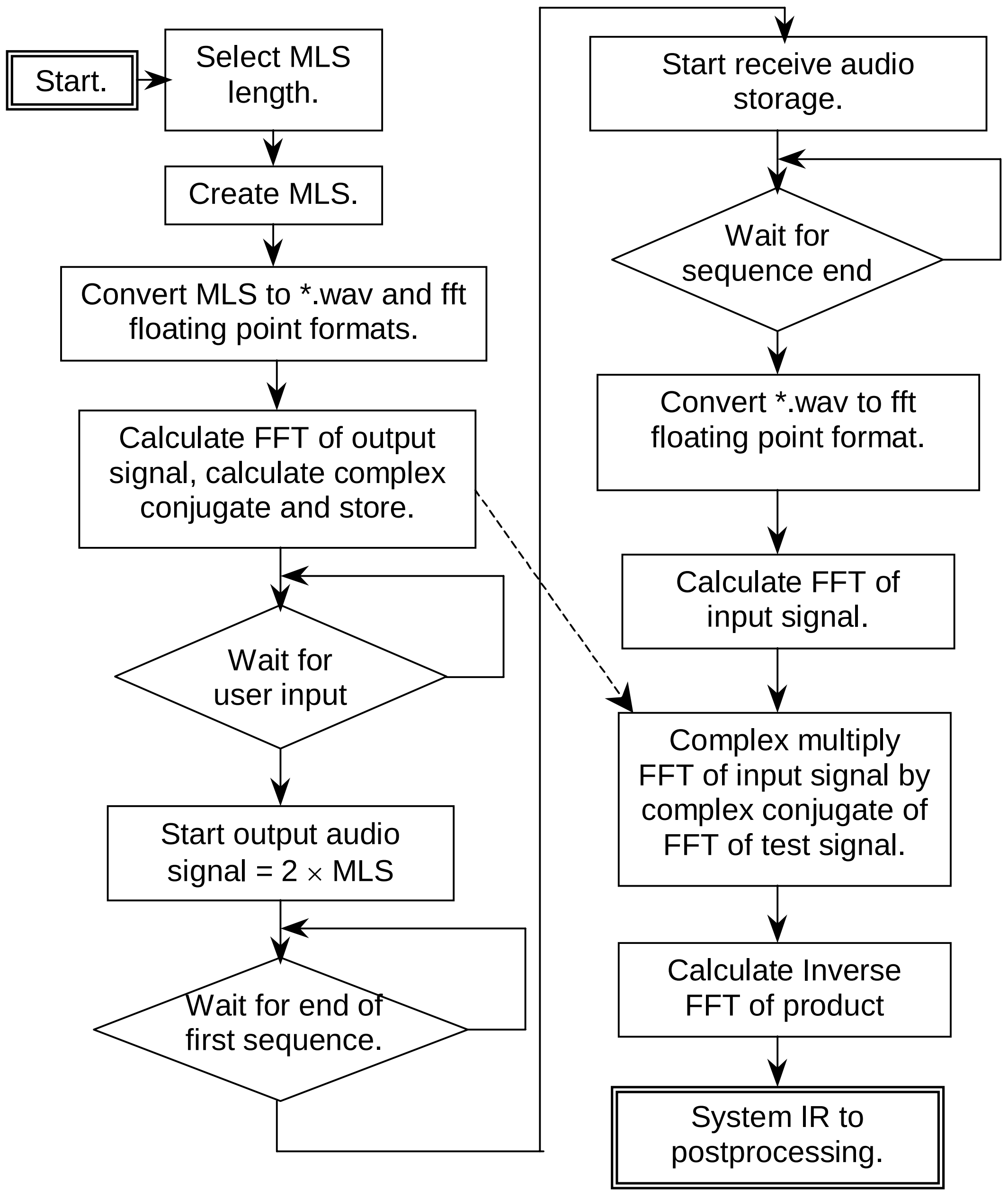
It has to be said that objective has not yet been achieved, but the existence of the code in compatible OOP form makes it a relatively simple matter of re-compiling the code with appropriate parameters and a suitable export and calling structure, thus eliminating the graphical 'front end'.

10. REFERENCES.

1. Borish, Jeffrey and James B. Angell, "An Efficient Algorithm for Measuring the Impulse Response Using Pseudorandom Noise", J. Audio Eng. Soc., vol. 31 7/83, p. 478.
2. Berman, J. M. and L. R. Fincham, "The Application of Digital Techniques to the Measurement of Loudspeakers", J. Audio Eng. Soc., vol. 25, 6/77, p. 370.
3. Fincham, L. R., "Refinements in the Impulse Testing of Loudspeakers", J. Audio Eng. Soc., vol. 33, 3/85, p. 133.
4. Rife, Douglas D. and John Vanderkooy, "Transfer-Function Measurement with Maximum-Length Sequences", J. Audio Eng. Soc., vol. 37, 6/89, p. 419.
5. Rife D. Modulation Transfer Function Measurement with Maximum-Length Sequences J. Audio Eng. Soc. Vol. 40, No. 10, October, 1992.
6. James W. Cooley and John W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19, 297-301 (1965).
7. Walker, R. Hadmard Transformation: a real time transformer for broadcast standard p.c.m television. BBC RD Report No. 1974/7 (1974).
8. Walker, R and Clarke, C.K.P. Walsh-Hadamard transformation of television pictures. BBC RD Report No. 1974/13 (1974).
9. www.fftw.org
10. Schroeder, M.R. (1981). Modulation transfer functions: definition and measurement, Acustica, 49, 179-182.
11. Studebaker, G.A. and Matesich, J.S. (1992). "A derivation of the Modulation Transfer Function", Memphis State University, Memphis, Tennessee.
12. Steeneken, H.J.M. and Houtgast, T. (1980). A physical method for measuring speech-transmission quality, Journal of the Acoustical Society of America, 67, 318-326.
13. Steeneken, H.J.M. and Houtgast, T. (1984). "A review of the MTF concept in room acoustics and its use for estimating speech intelligibility in auditoria", Institute for Perception TNO, Soesterberg, the Netherlands.

APPENDIX A – PROGRAMME CODING AND FRAGMENTS.

A-1 Flow chart.


Start.
Select MLS length.
Create MLS.
Convert MLS to *.wav and fft floating point formats.
Calculate FFT of output signal, calculate complex conjugate and store.
Wait for user input
Start output audio signal = 2 × MLS
Wait for end of first sequence.
Start receive audio storage.
Wait for sequence end
Convert *.wav to fft floating point format.
Calculate FFT of input signal.
Complex multiply FFT of input signal by complex conjugate of FFT of test signal.
Calculate Inverse FFT of product
System IR to postprocessing.

A–2 Code fragments.

A–2.1 Create FFTW plan.

```
fftwf_complex *in, *out, *in2, * out2;       // declare storage *
fftwf_plan p1, p2, pr;                       // declare fftw plans *

if(!(in = (fftwf_complex *)fftwf_malloc(sizeof(fftwf_complex) * N))) {
                printf("Failed to allocate memory for input array\n");
                return 0;
```

```
       }                                    // Allocate storage for fft input array
….

p1 = fftwf_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_MEASURE); //Create plan 1.
…
```

A–2.2 Use FFTW plan (with timers).

```
if(TIMER) start_timer(&timer1);
fftwf_execute(p1);
if(TIMER) printf("FFT 1 execution time = %f ms\n", (float)stop_timer(timer1));
//printf("Start multiplication\n");
```

A–2.3 FFTW Wisdom.

Two functions – the first called early in the application to check for wisdom file and import if there is and the second before application closedown to save the (revised) wisdom data. One bug was that the fftw functions for getting the wisdom data directly from a file did not work. The data had to be imported and exported via the string functions `fftwf_import_wisdom_from_string()` and `fftwf_export_wisdom_to_string()`.

```
UINT check_wisdom(void) {
       UINT length = 0;
       FILE * wisdom;
       if ((wisdom = fopen("wis.dat", "rb"))) {
              char *s;
              fseek(wisdom, 0, SEEK_END);
              length = ftell(wisdom);
              fseek(wisdom, 0, SEEK_SET);   //Get length of data.
              //printf("Wisdom file size = %d\n", length);
              if (length > 0 && (s = (char *)malloc(length))) {
                     fread(s, length, 1, wisdom);
                     //printf("%s\n", s);
                     if(!fftwf_import_wisdom_from_string(s)) {
                            printf("Failed to import wisdom from file\n");
                     }
                     if(s) free(s);
              }
              fclose(wisdom);
       }
       return length;
}

void save_wisdom(UINT l) {
       FILE * wisdom;
       char *p;

       p = fftwf_export_wisdom_to_string();
       //printf("Length of new wisdom = %d\n", strlen(p));

       if (strlen(p) > l) {
              if ((wisdom = fopen("wis.dat", "wb"))) {
                     fwrite(p, strlen(p), 1, wisdom);
                     fclose(wisdom);
                     //printf("New wisdom file written\n");
              }
              else {
                     printf ("Failed to open wisdom file\n");
              }
       }
       if (p) fftwf_free(p);
}
```

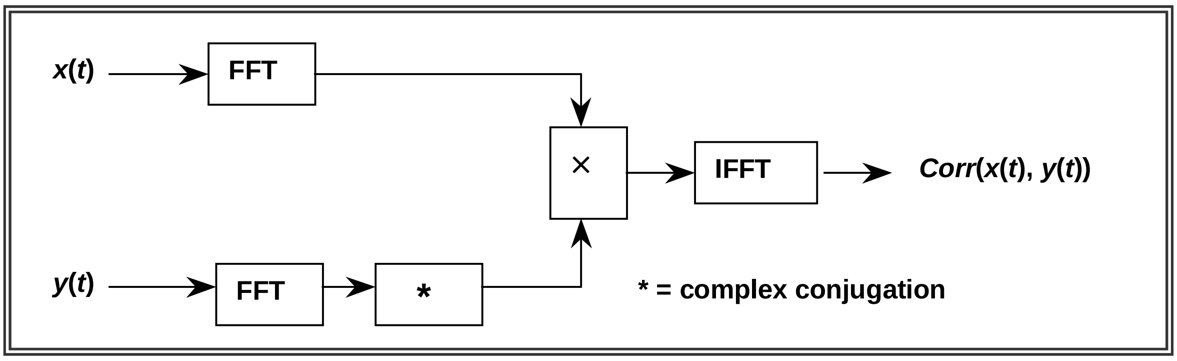
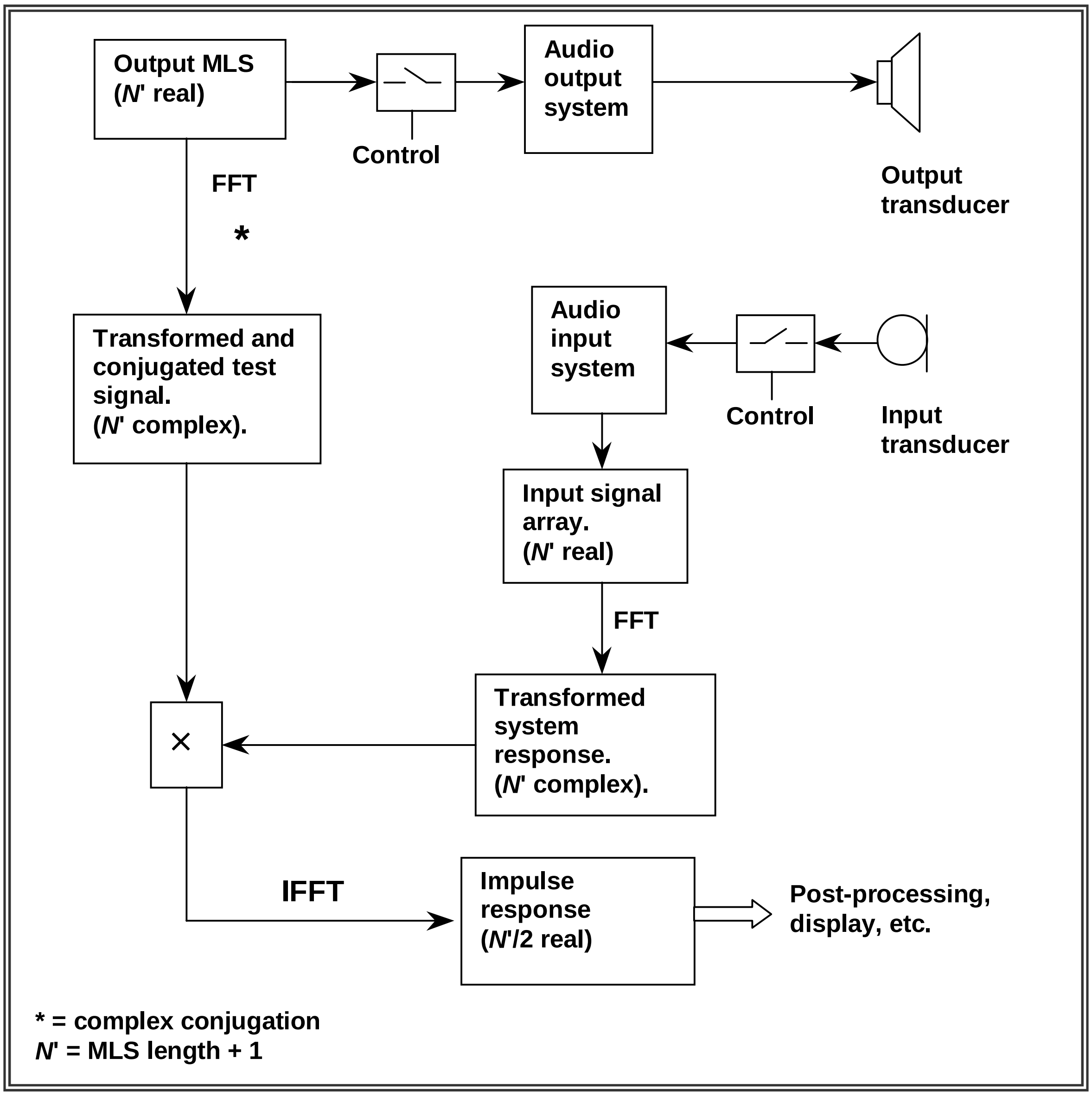


Fig. 1 Cross-correlation using Fourier transforms.



Output MLS (N' real)
Control
Audio output system
Output transducer
FFT
*
Transformed and conjugated test signal. (N' complex).
Audio input system
Control
Input transducer
Input signal array. (N' real)
FFT
$\times$
Transformed system response. (N' complex).
IFFT
Impulse response (N'/2 real)
Post-processing, display, etc.
* = complex conjugation
N' = MLS length + 1

Fig. 2. Processing schematic.

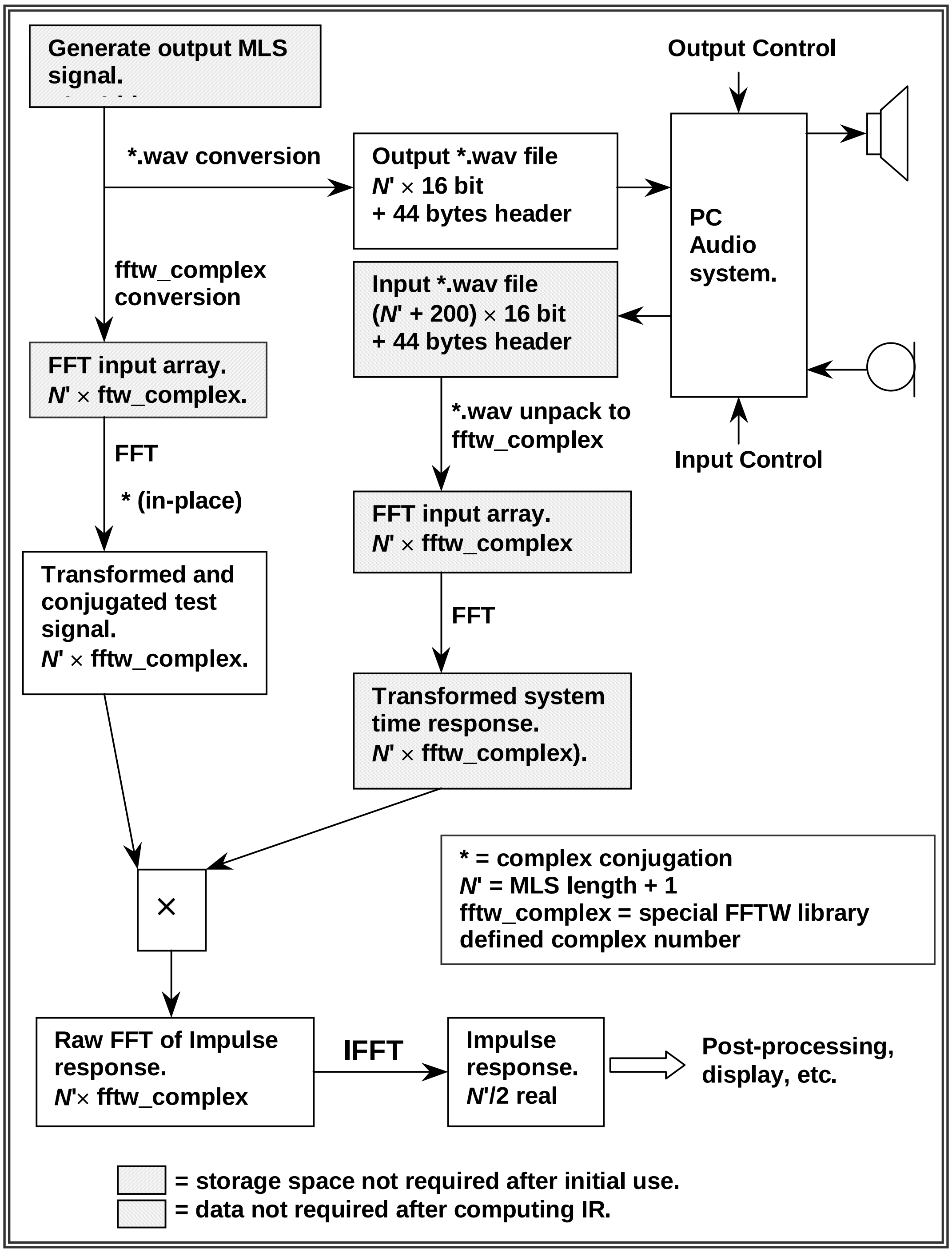
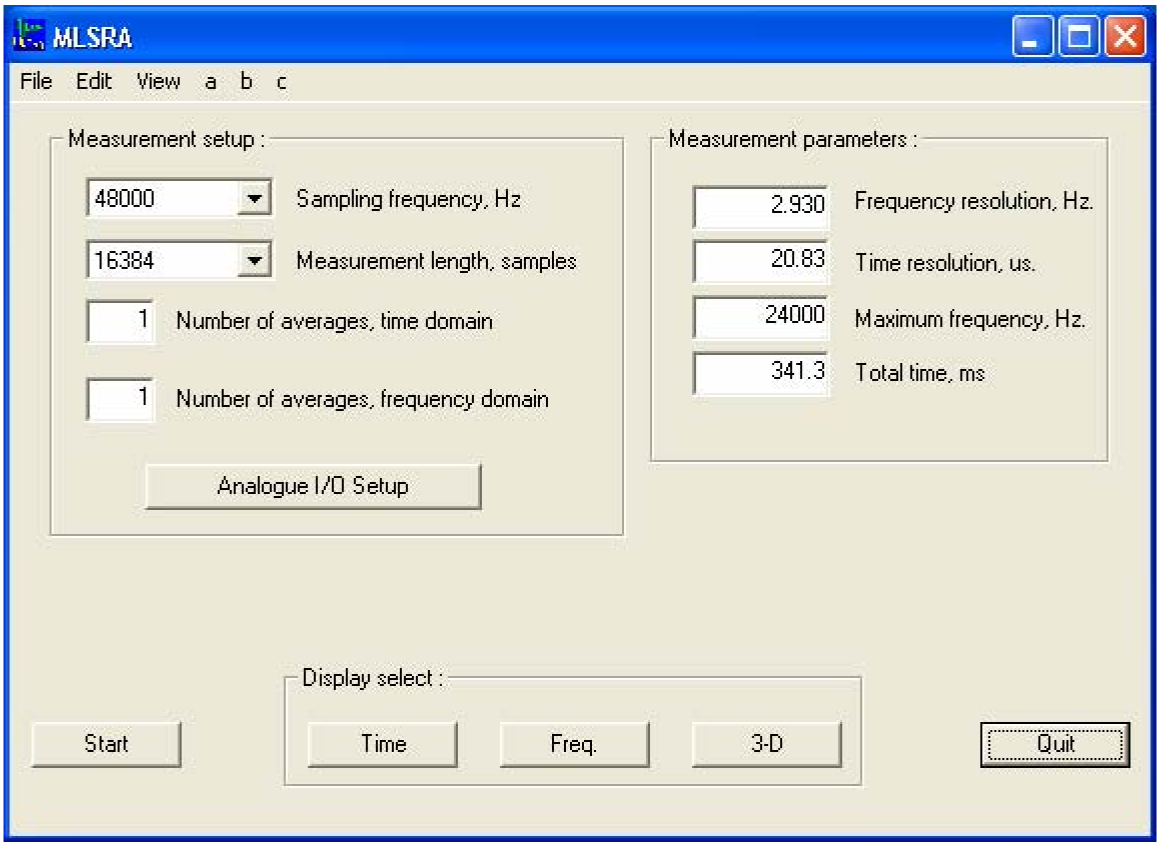


Fig. 3. FFTW processing scheme.



Measurement setup :

48000	Sampling frequency, Hz
16384	Measurement length, samples
1	Number of averages, time domain
1	Number of averages, frequency domain

Measurement parameters :

2.930	Frequency resolution, Hz.
20.83	Time resolution, us.
24000	Maximum frequency, Hz.
341.3	Total time, ms

Display select :

Fig. 4. Control screen for demonstration system.

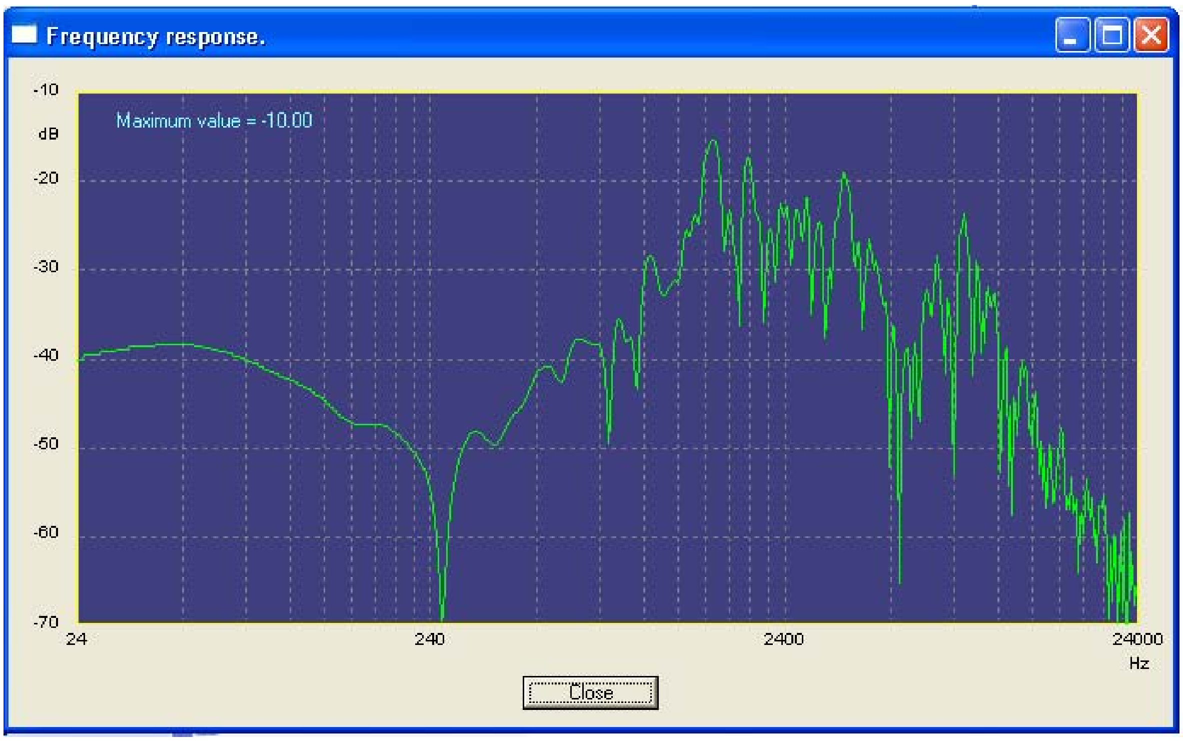


Fig. 5 Graphical result for frequency response from the demonstration Windows® system.