# Appendix II

# The Multi-Channel Spectrum Analyzer

*A. M. Peterson, K. S. Chen*

*Stanford University, Stanford, California*

**Abstract**

The Multi-Channel Spectrum Analyzer *(MCSA)* is a special-purpose digital signal processor. Its main function is to filter a wide-band signal into many narrower bands, so that each of the output bands has a bandwidth that is a better match to the signal being searched for.

The basic *MCSA* provides simultaneous output bandwidths of approximately 1 Hz, 32 Hz, 1024 Hz and 74 kHz over a spectrum that is about 8 MHz wide. The input to the *MCSA* consists of a complex signal sampled at 10 MHz, and the outputs consist of either complex samples or power (square-law detected) samples. In addition, the *MCSA* provides an accumulator for taking the integral of the power of the output bands for periods up to 1000 seconds.

The *MCSA* hardware is constructed using wire-wrap technology. The implementation of the hardware is done with the aid of a computer program developed specifically for the design of the *MCSA*. Care has been taken in the *MCSA* design to ensure that engineering tradeoffs do not adversely affect the performance of the system.

## General Description

Instead of using a single large Fast Fourier Transform *(FFT)*, the *MCSA* derives its narrow bands by cascading two stages of *digital bandpass filters* with moderate sized Discrete Fourier Transforms *(DFT)* . *FFT* operations do not yield convenient signals for deriving the intermediate bandwidths that the *MCSA* delivers. Furthermore, it is possible to provide better Radio Frequency Interference *(RFI)* rejection with the bandpass filter technique. An *FFT* has a worst-case sidelobe (adjacent bin) response that is only 13 dB below the response of the main lobe. A bandpass filter

can be designed to give more than 70 dB of adjacent channel rejection.

The first bandpass filter splits the input signal into 112 bands, each approximately 74 kHz wide. Each of these 74 kHz-wide signals is then filtered by a second bandpass filter which further subdivides the signal into 72 bands. Each of the resultant bands is about 1024 Hz wide.

The 1024 Hz signals are then fed either to a 36-point *DFT* or to a 1152-point *DFT* to form the final 32 Hz or 1 Hz outputs, respectively.

Each of these bandwidths (1 Hz, 32 Hz, 1024 Hz and 74 kHz) is available as an output of the *MCSA*. The magnitude squared value of each output sample is computed and is available as the square-law detected power output. Except for the 74 kHz bandwidth, the complex signals from the other bands are also available as outputs.

The memory required to do *real-time* transforms is equal to twice the size of the transform – one block of memory being required to buffer a block of data while the second block is having the transform operations done to it. By using relatively small transform sizes in the final *DFTs*, the need for a main memory that is twice the size of the *MCSA* bandwidth can be avoided. Thus, a substantial saving in memory cost is made by applying the *bandpass filter – DFT* technique compared to a single large *FFT*, offsetting the extra cost needed to implement the bandpass filters.

Internal to the *MCSA* , the sampling frequency of a given signal is kept a fraction larger than its analysis bandwidth to prevent additional noise from aliasing into the band of interest. If this were not done, it would be impossible to avoid a substantial degradation of the signal-to-noise ratio (*SNR*) near the extremes of the band, however sharp the cutoff of the filter is. A sharper filter reduces the size of the region that gets severe aliasing, but could not prevent a 3 dB loss of *SNR* at the edge of the band due to aliasing. The sampling frequencies used were chosen such that for each filter, the additional aliased noise is kept below 0.1 dB.

The use of oversampling causes the *DFT* transform sizes to come out to be non-power-of-two's. This is not a disadvantage, however. Through the use of newly developed *DFT* algorithms (called the Generalized Winograd *DFT* Algorithms), the computational efficiencies achieved are comparable with that of power-of-two *FFT* algorithms. Hardware complexity has increased (more

hardware is needed in some cases, faster hardware is needed in others) because of oversampling. This, however, is unavoidable unless a substantial loss in *SNR* can be tolerated.

**Bandpass Filter # 1**

The first bandpass filter (see Appendix II-A) operates on a complex input that is sampled at 10 MHz. Each of the *real-imaginary* pair of samples is made up of a pair of 8-bit binary numbers taken by an *A-D Convertor subsystem* that precedes the *MCSA*.

Before being given to the bandpass filter, each 8-bit sample is further quantized into a 4-bit representation to reduce the arithmetical complexity of this very high-speed stage. It was found that, with a signal that has Gaussian statistics, the loss in *SNR* due to the quantization of the signal to 4 bits is no more than 0.05 dB. Coarse quantization of a signal has been a neccessity in many systems that are required to operate at high speeds. However, careful analyses have shown that very low *SNR* losses can indeed be achieved without having to quantize the signal into very fine levels, providing the quantization parameters are held to some given values (see Appendix II-B).

The first Bandpass Filter is implemented as two identical banks of filters. Alternate 74 kHz passbands appear at the outputs of either of the two filters — the even 74 kHz-bands appearing at the outputs of the first filter bank and the odd 74 kHz-bands appearing at the outputs of the second filter bank. The basic filter bank separates its input into 72 bands, of which 16 are discarded, retaining 56 of the bands as outputs. The 16 discarded bands represent the part of the input signal that has substantial amounts of noise aliased into it by the sampling process.

The input signal, following the 4-bit quantization, is applied directly to the input of the first filter bank. Before being applied to the second filter bank, the input signal is shifted in frequency by 74 kHz. The frequency shifting is achieved by the multiplication of the input signal by a complex sinusoid at the appropriate frequency, done digitally. The output of the frequency shifter, quantized to 4 bits, is then applied to the input of the second filter bank.

Each of the two filter banks consists of a $288^{th}$ order Finite Impulse Response *(FIR)* filter and a 72-point DFT. The 72-point DFT is implemented with an 8-point *FFT* followed by a 9-point

*DFT.*

The 8-point *FFT* is realized with hard-wired pipelined logic circuits. A table-lookup Read-Only Memory *(ROM)* is used to implement the only multiplication, the scaling of a value by $\sqrt{2}$, required to perform an 8-point *FFT*. The number representation at this stage (and successive stages) is kept at 16 bits.

The 9-point *DFT* is implemented with 8 special-purpose programmable processors (see section on *DFT Processor*) working in parallel. A single micro-programmed controller controls all the 8 *DFT Processor* (also the 8 *DFT Processor* in the other bank of the filter).

The multiplication by the filter weights of the *FIR* filter is done with table lookups using *ROMs*. The *FIR* filter taps are partitioned into 4 sets of taps. One table is used for each of these sets of coefficients. For each table, a datum from a filter tap (4 bits) and the index of one of 72 filter coefficients (7 bits) form the address to the table. The output of the table-lookup is a 16-bit number which represents the product of the datum and the filter coefficient pointed to by the 7-bit index. This implementation resulted in substantial savings in hardware which would otherwise be needed if actual multiplications were performed.

As a consequence of using the table lookup technique to perform multiplications, the input data need not be the result of a uniform quantization of the input signal. Each 4-bit datum can be merely some known *representation* of the signal value, the actual conversion of the representation to its actual value taking place within the table-lookup process itself. Non-uniform quantization provides both a better *SNR and* less susceptibility of the *SNR* to changes in the system gain.

**Bandpass Filter # 2**

Each of 112 74 kHz-wide signals at the output of the first Bandpass Filter is further filtered into bands that are 1024 Hz wide. Conceptually, this is done with 112 bandpass filters. In actual implementation, this is done with only 28 separate filters, each filter capable of performing the task of bandpass filtering 4 different 74 kHz-wide signals.

Each of the second stage Bandpass Filters consists of a $2016^{th}$ order *FIR* filter together with

a 144-point *DFT*.

The *FIR* filters in the $2^{nd}$ Bandpass Filter are implemented with actual multiplications rather than with lookup tables. The representation of the signal at this point is a 16-bit quantity, making any lookup table prohibitively large. The summation of the 14 active taps (equation (A.2) of Appendix II-A) is performed within an integrated circuit *Multiplier-Accumulator (MAC)*. Since the *MAC* maintains a 35-bit internal accumulator, no precision is lost through the arithmetical operations even though the number of active taps is high. This enables us to implement filters which have stop-band rejections that are better than 70 dB.

The 144-point *DFT* is performed by microprogramming the same special-purpose processor that is used in the first Bandpass Filter. The transform algorithm consists of a 16-point *DFT* followed by a 9-point *DFT*.

The signal at the input of this stage is oversampled by a factor of 2. Thus, of the 144 bands available at the output of each of the $2^{nd}$ Bandpass Filters , half are discarded, retaining only 72 bands, each 1024 Hz wide.

**Main Memory**

The main memory is logically partitioned into blocks of 384k bytes (96k complex words, 16-bit real and 16-bit imaginary components) each. Each block provides enough memory to process a 74 kHz slice of the spectrum in real-time. The entire 8 MHz bandwidth (8,257,536 channels, at the 1 Hz resolution) is covered with 112 memory banks, making up approximately 43M bytes (10.75 million complex words) of memory. The memory uses dynamic random-access memory circuits. Data are written into the memory from the output of the second bandpass filter (bandwidth of 1024 Hz). The stored data are then read out of the memory and forwarded to the final *DFT Processors*, in the order that they are expected. Thus, the function of the main memory can be envisioned as a permutation operation on the data.

**36-point and 1152-point Discrete Fourier Transforms**

Two sets of *DFT Processors* are used to filter the 1024 Hz signals into the final 32 Hz and 1 Hz

bins. Fourteen processors are used in parallel to perform the transforms required to produce the 32 Hz outputs, and 28 processors are required for the transforms which result in the 1 Hz output bins.

The first set of *DFT Processors* is programmed to implement 36-point Fourier Transforms. The 1024 Hz signals are oversampled by a factor of 1.125. After the 36-point transform, four of the output bins are discarded, leaving 32 bins, each covering 32 Hz of bandwidth. The second set of processors is programmed to perform 1152-point transforms. Only 1024 of the output bins, representing 1024 Hz worth of bandwidth at the resolution of 1 Hz, are retained.

**The DFT Processor**

A special-purpose microprogrammable processor, which we have called the *DFT Processor*, is used thoughout the *MCSA* to implement the various transforms. In the $1^{st}$ Bandpass Filter, it is programmed to perform a 9-point *DFT*. In the $2^{nd}$ Bandpass Filter, it is programmed to perform a 144-point *DFT*. For the final transforms, the *DFT Processor* is programmed for either a 36-point *DFT* or a 1152-point *DFT*.

Although designed with the efficient implementation of the Generalized Winograd *DFT* algorithms in mind, the *DFT Processor* can also be programmed to perform other arithmetic-intensive tasks.

The *DFT Processor* is designed as a pipelined processor, with 2 *Arithmetic-Logic Units (ALU)* and a hardware *Multiplier-Accumulator (MAC)*. The processor *cycle-time* of the *DFT Processor* is 167 ns, corresponding to the processor clock frequency of 6 MHz. The two *ALU's* and the *MAC* are capable of concurrent operation — with proper programming, this gives the *DFT Processor* the equivalent capability of an 18 MIPS (million-instructions per second) machine. The processing elements themselves need perform no input-output operations — the local memory is doubly buffered; while data are being read into one section of the memory from the previous stage, the data in the second section of the local memory are processed by the arithmetical elements. The *DFT Processor* outputs are handled in a similar manner. All the input-output operations are

performed concurrent to actual computations; all compute cycles are therefore usable.

The program (microcode) for the *DFT Processor* resides in random-access memories *(RAM)* on a *DFT Controller* module. A single *DFT Controller* is used to control the *DFT Processors* that perform an identical task (such as the 28 *DFT Processors* performing the 1152-point *DFT*). The microcode for each *DFT Controller* can be *downloaded* from floppy-disks by an LSI-11/23 microcomputer, used as the *MCSA* controller. It is possible to change algorithms in the field by reloading the microcode. The *DFT Processors* doing the 1152-point transforms, for example, may be reprogrammed to process the 1024 Hz signals in a different way.

**Design Aids**

Much of the tedious work in the *MCSA* design is alleviated by a computer program which produces the wire-wrap tables. The program has access to a data-base of all the integrated circuit types used in the *MCSA*. A designer using this program could ignore the details of the integrated circuits such as physical pin positions, input loadings and output drive capabilities. The aim of this program is to allow a designer to concentrate on the functions of a board rather than having to keep track of every single detail. It provides the designer a tool analogous to what a high-level language provides a computer programmer.

The input to the program is a file of circuit descriptions. The integrated circuits used are declared, much like variable declarations in computer programming languages, giving their types and locations on the wire-wrap board. After that, each node in the design is defined by the symbolic name of the integrated circuit, as defined by its declaration, and the mnemonic representing a given pin on the integrated circuit, defined by a data-base entry for the particular integrated circuit type. The actual location on the board of each pin is kept track of by the program. To relocate an integrated circuit on a board, for example, one simply changes its declaration line, and the input file is resubmitted to the program.

In addition to symbolic module names, symbolic signal names can also be declared. The wire-wrap program creates interconnections between nodes that reference the same signal name.

A macro facility is provided for making repetitive commands, such as the wiring of data and

address buses, less laborious. In addition to having the designer spend less time on defining the circuit, errors are minimized and checking is also simplified.

The program provides an output file that can be written onto a magnetic tape which can be directly processed by an automatic wire-wrapping machine. In addition to the output tape, the program provides a printout of the circuit board layout, a cross-reference listing and a signal run list. The latter items aid in the debugging of the design in addition to providing a uniform set of documentation for the *MCSA*. Thus, from a file of circuit descriptions typed in by the designer, the result is a completely wired board from a wire-wrap machine.

The wire-wrap program is implemented in the C Programming Language and currently runs on the DECSYSTEM-20, using about 150k bytes of memory.

# Appendix II-A

## Digital Bandpass Filter

The Digital Bandpass Filter in the *MCSA* is implemented by combining the operations of a *Finite Impulse Response filter* with an *inverse DFT*.

An $n^{th}$ order Finite Impulse Response *(FIR)* filter (also known as a Transversal filter or a Tapped delay-line filter) consists of a delay-line of length $n$. Each of the delay stages has a tap brought out and multiplied by a gain constant. These weighted taps are then summed together at a single common node, forming the output of the *FIR* filter.

Consider an input sequence to the *FIR* filter that is in the form of a single impulse at time $t_0$. As time increases, the impulse appears at successive taps of the delay-line. Since the output is simply the sum of the weighted taps, the response of the *FIR* filter to a single impulse is just the time-ordering of the weights of the *FIR* filter, which, by definition is the *impulse response* of the filter. Notice that the impulse response is identically zero before time $t_0$ and after time $t_{n-1}$ ($n-1$ time units later) — which leads to the term *Finite* Impulse Response filter.

For an arbitrary input sequence $x_\nu$, the output sequence of an $n^{th}$ order *FIR* filter with weights $h_\mu$ , $\mu = 0, 1, \ldots, n-1$ is given therefore by

$$y_\nu = \sum_{\mu=0}^{n-1} x_{\nu-\mu} h_\mu. \qquad (A.1)$$

A linear time-invariant filter is uniquely determined by its *frequency response,* which is just the Fourier transform of its *impulse response.* Thus, by an appropriate choice of the weights for the *FIR* filter, we can approximate various filter responses — in particular, Lowpass filter responses, which are of primary interest here.

Various techniques exist for the determination of the *FIR* filter weights (impulse response) for realizing lowpass filters. The filters in the *MCSA* were designed with the *Remez algorithm* and the application of *Dolph-Čebyšev* windows.

Now, consider an *FIR* filter of order $nm$, for integers $n$ and $m$, defined by its filter weights

$h_\lambda$, $\lambda = 0, 1, \ldots, nm-1$.

Given an input sequence $x_\nu$, we define $y_\nu^\kappa$ , $\kappa = 0, 1, \ldots, n-1$ by

$$y_\nu^\kappa = \sum_{\mu=0}^{m-1} x_{\nu-\mu n-\kappa}\, h_{\mu n+\kappa}. \qquad (A.2)$$

This is a slight variant from the simple *FIR* filter in that there are $n$ outputs, each computed as a sum of a *subset* of the taps of the delay-line. Each of the $m$ components of the impulse response $h_\mu$ in equation (A.2) is termed an *active* tap of the filter. From this, we take the $n^{th}$ order inverse *DFT* of $y^\kappa$, to obtain the sequence $z^\kappa$, i.e.,

$$z_\nu^\kappa = \sum_{\mu=0}^{n-1} y_\nu^\mu\; e^{i\frac{2\pi\mu\kappa}{n}}. \qquad (A.3)$$

Defining

$$H_\mu^\kappa = h_\mu\, e^{i\frac{2\pi\mu\kappa}{n}}, \qquad (A.4)$$

equations (A.2) and (A.3) can be combined and rewritten as

$$z_\nu^\kappa = \sum_{\mu=0}^{nm-1} x_{\nu-\mu}\, H_\mu^\kappa. \qquad (A.5)$$

Notice that equation (A.5) has the same form as equation (A.1), except for a different impulse response for each index $\kappa$. From equation (A.4), we see that each of the impulse responses $H^\kappa$ is simply the impulse response of the *prototype* filter, $h$, that has been translated in the frequency domain by the amount $\frac{\kappa}{n}$.

Thus, if the *prototype* filter were a lowpass filter, each output signal $z^\kappa$ would be a bandpass-filtered output of the input signal $x$, the location of the passband being a function of $\kappa$. If the *prototype* lowpass filter has a bandwidth of $\frac{1}{n}$, then the $\kappa$ outputs would represent passbands which are non-overlapping and span the bandwidth of the input signal $x$.

Furthermore, given that each band is precisely $\frac{1}{n}$ in width, by undersampling each of the output signals by a factor of $\frac{1}{n}$, each signal is folded precisely into a lowpass (baseband) signal that is $\frac{1}{n}$ wide, but representing its original bandpass. This implies that the entire arithmetical

operation need be done only once each $n$ time units. The computational complexity of an $n$ band Bandpass Filter has thus been reduced approximately to that of a single *FIR* filter together with an $n$-point inverse *DFT*, at a computational rate of one iteration per $n$ time units.

It should be pointed out that the inverse *DFT* is used here in an operational sense and not in the manner the inverse *DFT* is usually interpreted to be. Notice from equation (A.5) that even though the input signal has gone through a *DFT*, the signal $z^{\kappa}$, for each $\kappa$, remains a *time-domain* signal. I.e., for a given $\kappa$, $z_n^{\kappa}$, $z_{n+1}^{\kappa}$, $z_{n+2}^{\kappa}$, ... form a time sequence.

## Appendix II-B

## Signal-to-Noise Ratio Loss due to Quantization

Given a signal with Gaussian statistics that gets quantized to a 1 bit representation (i.e., only the polarity of the signal is preserved), it has been shown that there is no loss of spectral information, except for a degradation in the signal-to-noise ratio (SNR) of about 2 dB. This well-known fact has been applied to systems which require high speed arithmetical operations.

Here, we shall derive the SNR loss for a general $n$-bit quantizer, given that the signal has Gaussian statistics. We shall also require that the quantizer be linear (but not neccessarily uniform). Other constraints will be introduced at the appropriate times.

Let $x(t)$ be a Gaussian process. Then, the random variables $x(t_1)$ and $x(t_2)$ are jointly Gaussian with the joint probability density function of

$$f(x_1, x_2) = \frac{1}{2\pi\sigma^2\sqrt{1-\rho_x(\tau)}} \exp\left(-\frac{x_1^2 + x_2^2 - 2\rho_x(\tau)x_1x_2}{2(1-\rho_x^2(\tau))\sigma^2}\right) \tag{B.1}$$

$$\text{where} \quad x_1 = x(t_1), \quad x_2 = x(t_2), \quad \tau = |t_1 - t_2|.$$

Without any loss of generality, we can assume the variance $\sigma^2$ to be unity. If we also assume that $\rho_x(\tau) \ll 1$ for $\tau \neq 0$, then equation (B.1) can be approximated as

$$f(x_1, x_2) = \frac{1}{2\pi} \exp\left(-\frac{x_1^2 + x_2^2 - 2\rho_x(\tau)x_1x_2}{2}\right). \tag{B.2}$$

The second assumption implies that the power of any coherent signal is much smaller than the total power of the process $x(t)$.

Let $\hat{x}(t)$ be the process which results from passing $x(t)$ through a quantizing function $\Xi(.)$. I.e., $\hat{x}(t) = \Xi(x(t))$. We shall impose the restriction that $\Xi(.)$ be a bounded function. The Autocorrelation function $R_{\hat{x}}(\tau)$ of the process $\hat{x}(t)$ can then be written as

$$R_{\hat{x}}(\tau) = \int_{-\infty}^{\infty} dx_1 \int_{-\infty}^{\infty} dx_2 \, \Xi(x_1)\Xi(x_2) f(x_1, x_2). \tag{B.3}$$

In the limit $\rho_x(\tau) \to 0$, and using equation (B.2),

$$\begin{aligned} R_{\hat{x}}(\tau) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} dx_1 \int_{-\infty}^{\infty} dx_2 \, \Xi(x_1)\Xi(x_2)(1 + \rho_x(\tau)x_1x_2) \exp\left(-\frac{x_1^2 + x_2^2}{2}\right) \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} dx_1 \int_{-\infty}^{\infty} dx_2 \, \Xi(x_1)\Xi(x_2) \exp\left(-\frac{x_1^2 + x_2^2}{2}\right) \\ &\quad + \frac{\rho_x(\tau)}{2\pi} \int_{-\infty}^{\infty} dx_1 \int_{-\infty}^{\infty} dx_2 \, x_1\Xi(x_1) \, x_2\Xi(x_2) \exp\left(-\frac{x_1^2 + x_2^2}{2}\right). \end{aligned} \tag{B.4}$$

For an odd-valued quantizer function $\Xi(.)$, the first term on the right hand side of equation (B.4) vanishes, leaving

$$R_{\hat{x}}(\tau) = \frac{2}{\pi} \left( \int_0^{\infty} dx \, x \, \Xi(x) \, \exp\left(-\frac{x^2}{2}\right) \right)^2 \rho_x(\tau). \tag{B.5}$$

By definition,

$$\begin{aligned} R_{\hat{x}}(0) &= \int_{-\infty}^{\infty} dx \, \Xi^2(x) f(x) \\ &= \sqrt{\frac{2}{\pi}} \int_0^{\infty} dx \, \Xi^2(x) \exp\left(-\frac{x^2}{2}\right). \end{aligned} \tag{B.6}$$

From equations (B.5) and (B.6), we obtain the normalized autocorrelation function

$$\rho_{\hat{x}}(\tau) = \frac{R_{\hat{x}}(\tau)}{R_{\hat{x}}(0)} = \sqrt{\frac{2}{\pi}} \frac{\left( \int_0^{\infty} dx \, x \, \Xi(x) \exp\left(-\frac{x^2}{2}\right) \right)^2}{\int_0^{\infty} dx \, \Xi^2(x) \exp\left(-\frac{x^2}{2}\right)} \rho_x(\tau). \tag{B.7}$$

Thus, we see that the normalized autocorrelation function $\rho_{\hat{x}}(\tau)$ of the quantized process has changed with respect to autocorrelation function $\rho_x(\tau)$ of the original signal by a constant factor

$$\xi = \frac{\rho_{\hat{x}}(\tau)}{\rho_x(\tau)} = \sqrt{\frac{2}{\pi}} \frac{\left( \int_0^{\infty} dx \, x \, \Xi(x) \exp\left(-\frac{x^2}{2}\right) \right)^2}{\int_0^{\infty} dx \, \Xi^2(x) \exp\left(-\frac{x^2}{2}\right)} \tag{B.8}$$

The power spectrum $S_x(\omega)$ of a signal $x(t)$ is simply the Fourier Transform of its autocorrelation function. By normalizing the autocorrelation function in the above equations, we have kept the

total power of the process constant, independent of the quantizer function. We see from equation (B.8), that by doing this, the autocorrelation function, thus, the power spectrum, of a quantized process has suffered a loss in gain with respect to the total power (signal + noise) of the process, i.e., by quantizing a signal, we have suffered a factor of $\xi$ loss in the SNR.

Given any bounded function $\Xi(.)$, we can compute $\xi$ from equation (B.8). A 1-bit quantizer can be described by $\Xi(x) = \text{sgn}(x)$. With this, equation (B.8) evaluates to $\frac{2}{\pi}$, which is precisely the value previously obtained for the SNR loss for the hardlimiting quantizer.

For quantizer laws that are piecewise-constant (such as A-D convertors), equation (B.8) can be further simplified. Let a piecewise-constant quantizer be described by

$$\Xi(x) = \Xi_i \qquad \text{for } x \in (\sqrt{2}x_{i-1}, \sqrt{2}x_i),\ i = 1, 2, \ldots, n, \text{ and ,}$$

$$\Xi(x) = -\Xi(-x) \qquad \text{for } x < 0$$

$$\text{where } 0 = x_0 < x_1 < \ldots < x_{n-1} < x_n = \infty.$$

Then the SNR loss is given by

$$\xi = \frac{2}{\pi} \frac{\left(\sum_{i=1}^{n} \Xi_i \left[\exp(-x_{i-1}^2) - \exp(-x_i^2)\right]\right)^2}{\sum_{i=1}^{n} \Xi_i^2 \left[\text{erf}(x_i) - \text{erf}(x_{i-1})\right]} \qquad (B.9)$$

where erf(.) is the Error Function.

Given an $m$ −bit uniform quantizer (i.e., $x_n - x_{n-1} = \alpha$ for $i = 1, 2, \ldots, 2^{m-1}$), we can vary the quantizer stepsize, $\alpha$, to obtain a minimum $\xi$ from equation (B.9). This would then represent the optimal uniform quantizer for a signal with Gaussian statistics with a variance $\sigma^2 = 1$. (The optimal quantizer stepsize for a non-unit variance signal is scaled accordingly.)

The following is a table of the SNR losses sustained by optimal $m$ −bit quantizers:

| m | SNR Loss |
|---|---|
| 1 | 1.961 dB |
| 2 | 0.550 dB |
| 3 | 0.166 dB |
| 4 | 0.050 dB |
| 5 | 0.015 dB |
| 6 | 0.005 dB |
| 7 | 0.001 dB |

It should be noted that an ordinary A-D Convertor does not satisfy the constraint that the quantizer law be an odd-valued function unless a bias equal to half a bit is introduced.