To the Graduate Council:

I am submitting herewith a thesis written by Michael E. Mitchell entitled "Development and Implementation of a Causal Iterative Deconvolution on the Motorola DSP56000ADS." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Dr. Paul Benjamin Crilly, Major Professor

We have read this thesis and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Requests for permission for extensive quotation from or reproduction of this thesis in whole or in parts may be granted by the copyright holder.

Signature Mike E. McRell

Date 4/30/92

Development and Implementation of a Causal Iterative Deconvolution on the Motorola DSP56000ADS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Michael E. Mitchell

May 1992

Copyright © Michael E. Mitchell, 1992
All rights reserved

Acknowledgements

I would like to thank my parents Mike and Libby Mitchell, my grandmother Mrs. Allie Kerr, and my late grandparents, Mr. Paul E. Kerr and Mrs. Margaret Mitchell without whose love, patience and support this would not have been possible. The advice and patience from my sister Kelli Aguirre, my relatives, Cindy Templeton, and the other students and staff at the University of Tennessee were a constant reminder to me of the things that are truly important important in life. Additionally, I would like to thank the members of my committee Dr. Paul Crilly, Dr. Robert E. Bodenheimer, Dr. Dan Koch, and Professor C. D. Martin. Finally, I would like to thank Bell South for their partial support of this work.

Abstract

A new method of iterative deconvolution that will be referred to as the causal iterative deconvolution (CID) method was developed and implemented on a personal computer (PC) with an 80386 microprocessor and on a Motorola DSP56000 digital signal processing (DSP) development system. The results show that the Motorola DSP system executed the CID algorithm significantly faster than the PC-based system. Furthermore, the CID technique appears to enhance the baseline and peak areas of the results at a faster rate than Jansson's method of iterative deconvolution.

The PC implementation was written in the C programming language, whereas the DSP56000 was implemented in Assembly. Convolved Gaussian-shaped functions with additive random noise were used to simulate test results from chromatographic or spectroscopic instrumentation. Analog-to-digital and digital-to-analog (A/D and D/A) converters were designed for the DSP56000 system to verify the validity of the results for analog input and output functions.

Table of Contents

CHAPTER PAGE

1. INTRODUCTION 2
2. ITERATIVE DECONVOLUTION CONCEPTS 3
    Causal Iterative Deconvolution Concepts 11
3. DIGITAL SIGNAL PROCESSING SYSTEMS 13
    Motorola Disc-Based Number Format 13
    The DSP56000 System 14
    Hardware Development 15
    DSP56001 CID Program Development 18
4. RESULTS AND CONCLUSIONS 21
    Recommendations 32
5. SUMMARY 34

BIBLIOGRAPHY 35

APPENDICES 37
    APPENDIX A. FRACTIONAL AND INTEGER ARITHMETIC 37
        Data Representations 38
        Arithmetic Operations 39
    APPENDIX B. ASSEMBLER PROGRAM FILE 41

VITA 50

List of Tables

TABLE PAGE

1. Window Width Vs. Iteration RMS Error Comparison 23

List of Figures

FIGURE PAGE

1. Impulse Response, h(t) 4
2. True Data, x(t) 5
3. Observed (Raw) Data with Added 100/1 S/N Random Noise 5
4. Jansson's Relaxation Function 7
5. Iteration Refinements 7
6. Jansson's Concept 10
7. CID Concept 11
8. CID Approach 12
9. Application Development System 15
10. Real Time Signal Processing Unit 16
11. Analog Signal Generation Unit Block Diagram 17
12. ADC0808 Signals 17
13. Retriggerable Monostable Pulse Characteristics for a 20 Wide Window 18
14. ADC0808 Schematic 18
15. DAC0808 Block Diagram 19
16. CID Analog Input/Output Flowchart 20
17. Points/Second Rate for the IBM and Motorola Systems 22
18. Convolved Input Information with S/N Ratio of 100/1 24
19. 100/1 S/N Ratio for Jansson's and CID Methods 24
20. 100/1 S/N Ratio for Points 80-155 25
21. 100/1 S/N Ratio for Points 180-320 25
22. 100/1 S/N Ratio for Points 350-440 26
23. Convolved Input Information with S/N Ratio of 50/1 26
24. 50/1 S/N Ratio for Jansson's and CID Routine 27
25. 50/1 S/N Ratio for Points 90-165 27
26. 50/1 S/N Ratio for Points 180-330 28
27. 50/1 S/N Ratio for Points 350-440 28

28. 20 Wide Window @ 500 ms/Div 29
29. 20 Wide Window @ 200 ms/Div 30
30. 30 Wide Window @ 500 ms/Div 30
31. 30 Wide Window @ 200 ms/Div 31
32. 50 Wide Window @ 1.00 s/Div 31
33. 50 Wide Window @ 500 ms/Div 32
34. 80 Wide Window @ 2.00 s/Div 32
35. 80 Wide Window @ 1.00 s/Div 33
36. Data Formats 37

Chapter 1
Introduction

A common problem encountered by researchers in such diverse fields as gas chromatography, mass spectroscopy, and medical diagnostics involves a spreading or blurring phenomena caused by the instrumentation used and characterized mathematically as the convolution of two functions obtained in the results of the research. To solve this problem, researchers must determine how to restore, enhance, de-blur or deconvolve the data from the blurred results.

Frequently, researchers desire to recover the original object given only the spread function (impulse response function of the system) and the resulting smeared data or image.[1] A complicating factor in the restoration of this ideal signal by numerical methods is the random noise that is inevitably added to the data by the act of collection or by the very nature of the information channel.[2] Additional problems may be incurred in resolving the ideal estimate due to limitations on the physical constraints of the system such as non-negativity. Non-negativity implies that the results should contain only positive real numbers (i.e., negative values may be created by the deconvolution process).

The earliest attempts at resolving the ideal function required that the Fourier transform of the observed peak be divided by the Fourier transform of the system transfer function.[1] This iterative deconvolution method has limitations due to severe degradation of the signal-to-noise (S/N) ratio.[3] The iterative deconvolution process not only restores the frequency components that enable sharp, well-defined peaks, but in some cases, enhances the noise components at a faster rate.[4]

One method of analysis that has recently proven useful for recovering blurred chromatographic and spectroscopic data is Jansson's method of iterative deconvolution.[5,6] The benefit of this method is that the algorithm is more accurate than previous iterative methods in estimating the original data and requires little *a priori* knowledge of the data characteristics. However, Jansson's method of iterative deconvolution does not have the

ability to deconvolve noisy data without significantly reducing the final estimate's S/N ratio.

Until recently, the use of Jansson's method of iterative deconvolution was limited to systems with relatively low signal sampling rates due to the computationally intensive nature of the algorithm.[2] If the inherent number of calculations or the computation time required to calculate the refined estimate could be lowered, the algorithm could be applied to a wider class of data and produce faster real time results.

A basic hypothesis proposed is that the Motorola DSP56001 system is significantly faster than an IBM PC, 80286 platform. This is due to the digital signal processing (DSP) specific architecture of the Motorola chip. For example, at a 20.48M Hz clock rate the DSP56001 can execute a 1024-point complex Fast Fourier Transform (FFT) with bit reversed data addressing in 3.39 milliseconds (ms) or a four coefficient Infinite Impulse Response (IIR) filter section in only four instruction cycles, which is the theoretical minimum for a single multiplier architecture.[7]

The use of a dedicated signal processing chip and a new iterative deconvolution algorithm may allow for faster, more accurate results. If the system is significantly faster than existing general purpose computer architectures and the required number of calculations necessary to deconvolve the information minimized, this may produce the ability to deconvolve information with a higher frequency content. For these reasons, a new algorithm was developed and implemented on the Motorola DSP56001 system to enable a wider class of data to be analyzed.

A rough estimate of the abilities and accuracy of a new deconvolution algorithm are compared to Jansson's method utilizing pre-generated disc information on an IBM PC. The algorithm is implemented on the Motorola DSP56000 application development system (ADS) using the same disc-based information to obtain a rough performance comparison to the 80286 microprocessor platform. Additionally, the Motorola system is configured with the necessary analog-to-digital and digital-to-analog (A/D and D/A) hardware to allow the DSP56000 ADS to deconvolve analog signals in real time.

Chapter 2
Iterative Deconvolution Concepts

Assuming that the system which produces the observed data is linear and time invariant, the spreading or blurring effect can be modeled by the following equation:

$$\mathbf{g} = \mathbf{h} * \mathbf{x} + \mathbf{n}. \qquad (1$$

The terms **g**, **h**, **x**, and **n** are functions of continuous time t, or discrete index i. These functions represent the observed (raw) data **g**, the impulse response function **h**, the true data **x**, and the additive noise **n**. Convolution is symbolized by the * operator. In the following examples, results shown as observed data are in fact synthesized (by convolution) from hypothetical data so that the true solution sought is in fact known beforehand.

If Figures 1 and 2 represent **h** and **x**, respectively, an example of the convolution process for Gaussian shaped system information with added random noise is shown in Figure 3.

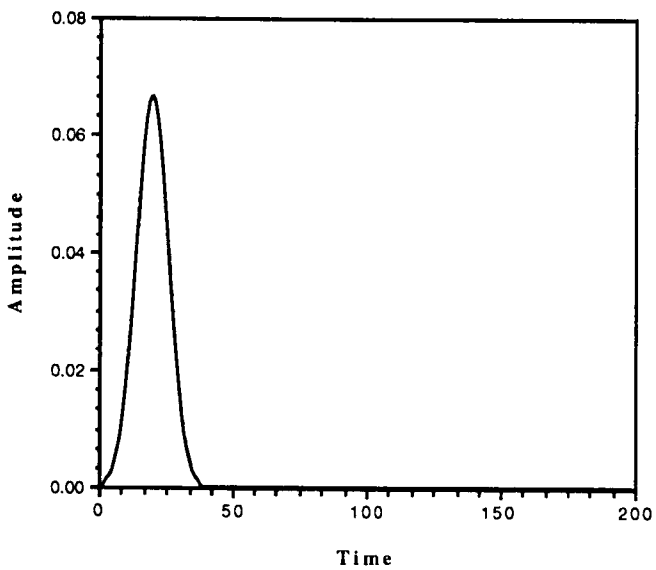


Figure 1: Impulse Response, h(t)

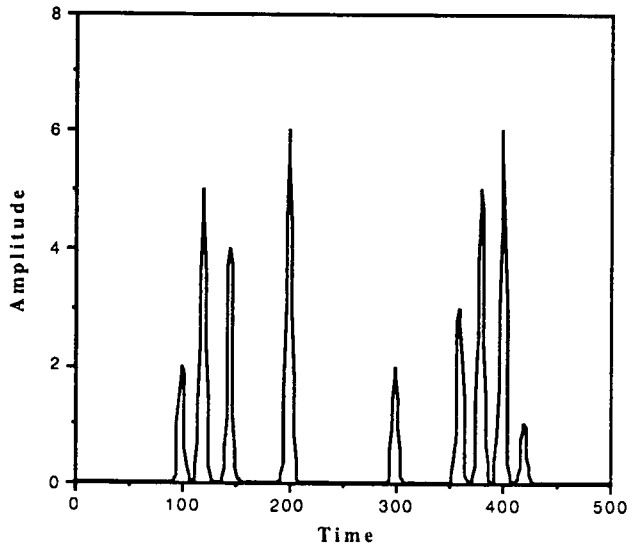


Figure 2: True Data x(t)

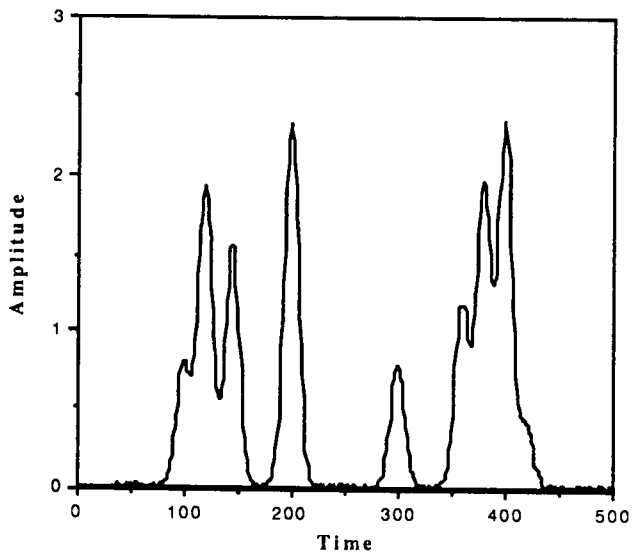


Figure 3: Observed (Raw) Data with Added 100/1 S/N Random Noise

The convolution process of Equation (**1**) degrades the resolution of Figure 3 to the extent that some of the peaks appear fused or are barely discernible. This blurring effect is caused because **x** loses frequency components when convoluted with **h**. Ideally, if these 'lost' frequency components could be recovered (i.e., if noise were absent and the system were not bandlimited), the original data of Figure 2 would result. Although this 'ideal' situation does not exist in practice, Jansson's method of iterative deconvolution has proven the ability to recover many of the frequency components necessary to enable sharp, well-defined peaks in chromatographic and spectroscopic instrumentation.[6]

As shown below, van Cittert's approach to iterative deconvolution is represented by the equation[8]

$$\hat{\mathbf{x}}^{k+1} = \hat{\mathbf{x}}^{k} + \mathbf{b}(\mathbf{g} - \mathbf{h}*\hat{\mathbf{x}}^{k}) \qquad (2$$

where * is the convolution operator, $\hat{\mathbf{x}}^{k}$ is the kth estimate of **x** (the values of the variable **x** over the full range of interest), **b** is a convergence constant, and **g** is the initial value of **x** - the observed data set. The constant **b** is selected to achieve a given rate of convergence and will depend upon the class of data processed. The algorithm calculates a new estimate $\hat{\mathbf{x}}^{k+1}$ by utilizing a correction factor $\mathbf{b}(\mathbf{g} - \mathbf{h}*\hat{\mathbf{x}}^{k})$ to adjust the kth estimate of **x**. A negative aspect of this method of analysis is that van Cittert's method linearly magnifies the image noise with each successive iteration.[2]

If the data are known *a priori* to be non-negative, then constraints can be incorporated to take advantage of this information. Schafer, et. al. incorporate the non-negativity constraint into van Cittert's method in the following manner[9]:

$$\hat{\mathbf{x}}^{k+1} = \mathbf{p}\hat{\mathbf{x}}^{k} + (\mathbf{g} - \mathbf{h} * \mathbf{p}\hat{\mathbf{x}}^{k}) \qquad (3a$$

with

$$\mathbf{p} = 1 \text{ for } \hat{\mathbf{x}}^{k} \geq 0 \qquad (3b$$

$$\mathbf{p} = 0 \text{ for } \hat{\mathbf{x}}^{\mathbf{k}} < 0.$$

The constraint operator **p** truncates the negative artifacts from the estimate during each iteration and reportedly provides better results than van Cittert's method.[4]

If an estimate of the maximum peak amplitude is known and the data are non-negative, the information can be constrained within certain limits using a relaxation function. In place of the convergence operator **b**, Jansson incorporated a relaxation-based function $\mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}$ to constrain each successive estimate between an upper and a lower bound. Jansson's equation[9]

$$\hat{\mathbf{x}}^{\mathbf{k}+1} = \hat{\mathbf{x}}^{\mathbf{k}} + \mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}(\mathbf{g} - \mathbf{h}*\hat{\mathbf{x}}^{\mathbf{k}}) \qquad (4$$

requires a recursive calculation in which the next estimate $\hat{\mathbf{x}}^{\mathbf{k}+1}$ is dependent upon the previous estimate $\hat{\mathbf{x}}^{\mathbf{k}}$ with the improvements occurring due to $\mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}(\mathbf{g} - \mathbf{h}*\hat{\mathbf{x}}^{\mathbf{k}})$. The first estimate or iteration ($\hat{\mathbf{x}}^{\mathbf{0}}$) is the input **g**. Each iteration may be considered a refinement over the entire data range of the previous estimate.

The new estimate $\hat{\mathbf{x}}^{\mathbf{k}+1}$ is improved by incorporating a nonlinear function $\mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}$, which allows minor adjustments in the low signal-to-noise regions near the baseline of a peak, and negative corrections if $\hat{\mathbf{x}}^{\mathbf{k}}$ is negative or exceeds a maximum value. The factor $\mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}$ thus transforms van Cittert's linear, unconstrained technique into a nonlinear, constrained method.[1]

Relaxation functions can have a variety of forms such as a Gaussian curve, a semicircle, or a second-order polynomial, each with an amplitude and width dependent upon the raw data. From testing, optimal results appear to be obtained from a triangular-type constraining function as shown in Figure 4. This function incorporates the following corrections to the estimate:

1) large/small corrections at high/low signal-to-noise ratios,

2) reverse corrections if the estimate has non-physical results, and

3) positive corrections within the physical limits.[10]

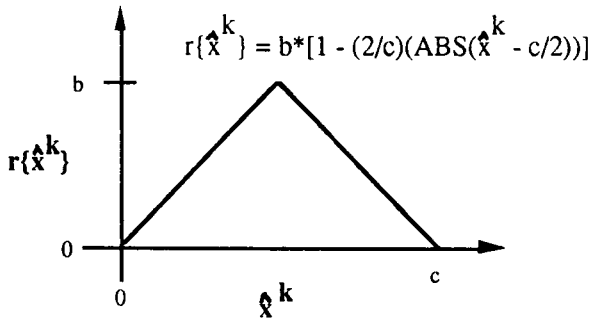


Figure 4: Jansson's Relaxation Function

Statement (1) implies that the relaxation function attempts to deconvolve the noisy data without significantly decreasing the final estimate's S/N ratio. Statements (2) and (3) constrain the resulting estimate from having negative artifacts and enhance the estimate within physical bounds.

The choice of the variable **c** in Figure 4 is dependent upon a knowledge of the signal's maximum amplitude. The variable **b** is chosen depending upon S/N ratio considerations and the desired convergence rate. Jansson's method thus requires a knowledge of the impulse response function and the maximum peak height of the ideal signal.

Figure 5 depicts the refinements that have occurred in utilizing Jansson's method for iterations 10 and 30. This information is shown to illustrate how the iterations lower the root mean square error (RMS) value to the ideal solution **x**. The graph of RMS error versus iteration number for the data set of Figure 5 indicates that Jansson's method results in the most dramatic changes within the first 20 iterations. After these first 20 iterations, the program makes minor adjustments to the overall range of data.

One approach to decreasing the iterative deconvolution program run time is to implement the convolution for the current estimate $\mathbf{h}*\hat{\mathbf{x}}^k$ only for values of time in which **h** is non-zero. The impulse response contains only a few non-zero valued points when

compared to the total range of data necessary for the deconvolution process (see Figure 1). Since the convolution estimate ($\mathbf{h}*\hat{\mathbf{x}}^{\mathbf{k}}$) used to adjust the kth estimate of **x** in the deconvolution procedure may be implemented utilizing the non-zero valued points of **h** (roughly 40 non-zero points -- Figure 1), the required computations for the convolution may be decreased. For example, the resulting modifications for a 30 iteration program over a 600-point interval saves (600-40)*30 or 16,800 unnecessary computations.

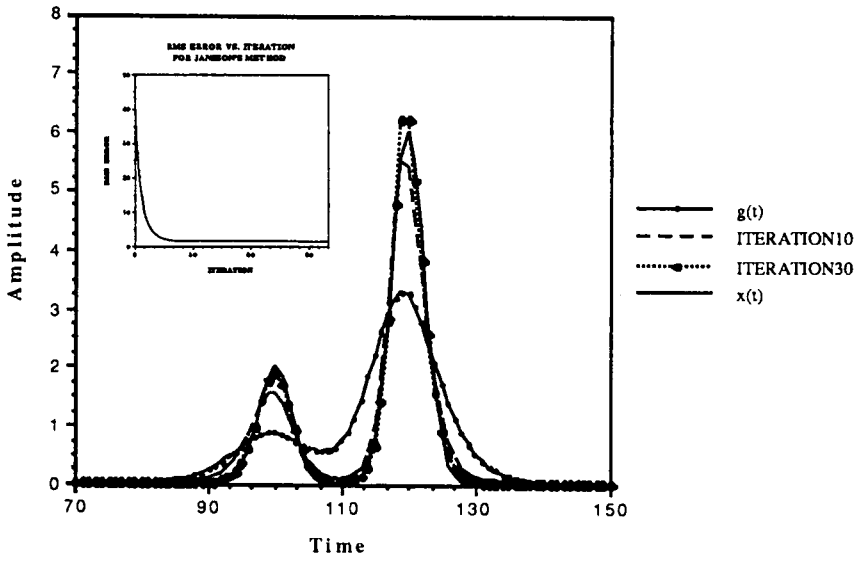


Figure 5: Iteration Refinements

If Jansson's method does not strictly adhere to $\mathbf{h}*\mathbf{x}$ of Equation (4), the resulting information diverges (i.e., $\mathbf{h}*\hat{\mathbf{x}}$ and not $\hat{\mathbf{x}}*\mathbf{h}$). Additionally, Equation (4) requires the 'offset' of the impulse response function (i.e., the number 20 is the offset value for the graph of Figure 1). Therefore, the final equation for Jansson's method should read

$$\hat{\mathbf{x}}^{\mathbf{k+1}}[\mathbf{n}] = \hat{\mathbf{x}}^{\mathbf{k}}[\mathbf{n}] + \mathbf{r}\{\hat{\mathbf{x}}^{\mathbf{k}}\}(\mathbf{g}[\mathbf{n}] - \mathbf{h}[\mathbf{m}]*\hat{\mathbf{x}}^{\mathbf{k}}[\mathbf{n-m+offset}]). \quad (5$$

Jansson's method requires that the initial estimate ($\hat{\mathbf{x}}^{\mathbf{0}}(\mathbf{n})$) equals the observed data ($\mathbf{g}(\mathbf{n})$) for a 'segment' or block of **n** points from the complete range of **i** points. Figure 6 graphically denotes the operating procedures required for the simple case of 3 points for Jansson's method. Three points of information are presampled and clocked into the top

and bottom row of registers (for iteration $\hat{x}^0$ only). The following mathematical operations are then performed to achieve one iteration:

1) The convolution sum of the bottom row of estimate registers is calculated (**linear sum operation**).

2) Information from operation 1 is used to determine the difference between the top row of registers, which contains the convolved signal value and the linear sum (**difference operation**).

3) A multiplication involving operation 2 and the calculation of the relaxation coefficient yields the corresponding estimate (**multiplication**).

4) The sum of operation 3 and the last estimate is made available before the next clock pulse (**addition/next estimate operator**).

5) A clock pulse clocks the new estimate into the bottom row of estimate registers.

The cycle continues passing the refined estimates back into the bottom row of registers for **k** iterations. At this point, all 3 of the refined estimates are available for output. In this manner, information is sampled and refined for a block of $\mathbf{n} = 3$ points.

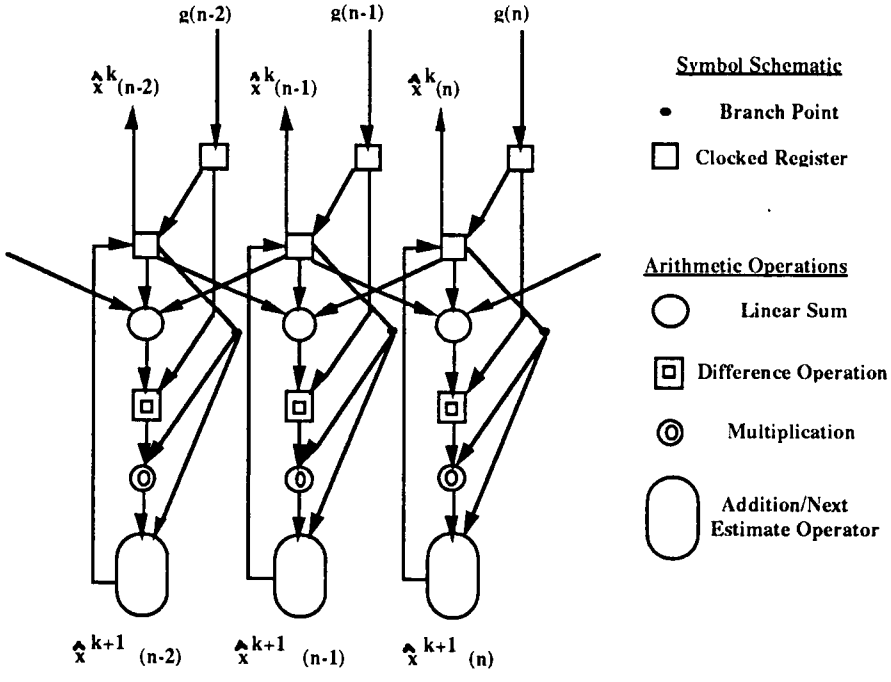


Figure 6: Jansson's Concept

In comparison to Jansson's method, the CID technique is a point-by-point refinement starting with the input point **n**. To visualize this effect, consider the 3 wide 'window' routine as shown in Figure 7. Initially all registers are cleared. As a single, new sample of information is shifted into the top row of registers from left to right (representing the convolved signal), the new **g(i)** sample is also clocked into the first register of the bottom row. Before the next clock clock pulse occurs, the corresponding arithmetic operations are carried out:

1) The convolution of the estimate registers is calculated (**linear sum operation**).

2) Information from operation 1 is used to determine the difference between the convoluted signal value and the linear sum (**difference operation**).

3) A multiplication involving operation 2 and the calculation of the relaxation coefficient yields the corresponding estimate (**multiplication**).

4) The new estimate is made available before the next clock pulse to shift the information left in the bottom row of registers (**addition/next estimate operator**).

5) Simultaneously, a clock pulse reads in new information and shifts the resulting information to the left in both the top and bottom row of registers to produce a deconvolved output.

The five preceding steps produce an estimate, which is shifted left into the next available register.

The CID method and Jansson's methods differ in how the convolution sum is used to calculate the evaluated estimate. For the preceding sequence of events of the CID method, the points are evaluated separately for different register depths of the 'window' of data points. This indicates that the CID convolution sum calculated in step (**1**) uses information based upon different levels of refinement to produce an estimate. Jansson's method in step (**1**) uses a convolution sum based upon the same level of refinement to produce an estimate. This difference in the use of the convolution sum appears to increase the rate of enhancement of the baseline, peak, and noise regions for the CID method over that of Jansson's method.

Additionally, the CID output estimate for step (**5**) is available for each consecutive clock pulse after (**n** + 1) clock pulses. This delay represents a phase shift in the resulting information. After **k** iterations of step (**5**) for Jansson's method, the estimate for all **n** points is available for output representing a phase shift in the information.

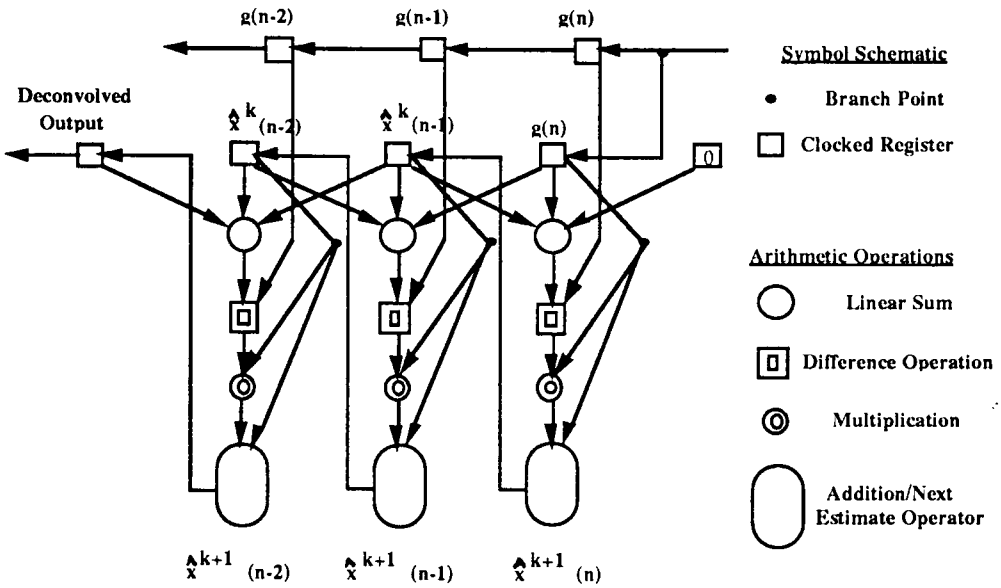


Figure 7: CID Concept

Causal Iterative Deconvolution Concepts

A causal process implies that the output is a function of the input sample for $\mathbf{n} < \mathbf{n}_0$ where $\mathbf{n}_0$ is the present time. By individually refining the present input point $\mathbf{n}_0$ and a specific number (or 'window') of previous points, the computation requirements and the sample rate remain constant regardless of the number of data points involved in the set (see Figure 8). This eliminates Jansson's requirement of sampling a segment of the signal, deconvolving the information to produce an estimate, and outputting the information at the same sampling rate before the next segment of samples occurs.

Windowing is an indication as to what extent the information is refined. The smaller the window, the less the refinement, and the less the program execution time. A

larger window implies more refinement and a slower program execution time. By calculating only the non-zero valued impulse response points (as discussed with Jansson's method) and using the 'causal windowing' technique, this considerably decreases the program execution time.

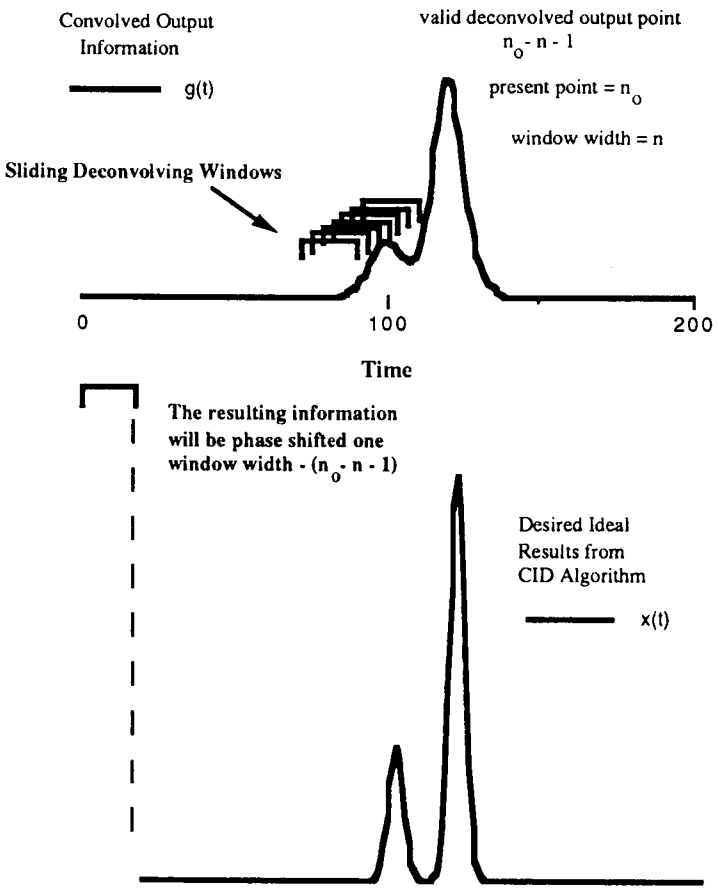


Figure 8: CID Approach

Chapter 3
Digital Signal Processing Systems

Two separate systems were used to obtain information relating to Jansson's method and the CID algorithm. These systems were: 1) an 80286 IBM PC PS2 with math coprocessor and 2) a DSP56001 application development module (ADM) with a Macintosh IIx as the host computer.

Jansson's method and the CID algorithm were implemented in the C programming language on an IBM PC to enable testing and evaluation of both algorithms utilizing pregenerated disc ASCII text files. For Jansson's method, the **g(i)** function was accessed once and processed for a various number of iterations. The CID algorithm accessed this **g(i)** information sequentially for window widths of 20, 30, 50, and 80.

An Assembly language CID program was implemented on the Motorola DSP56000 system. The Motorola platform was used in two separate programs to evaluate disc or analog information. Since the pregenerated functions were created by an IBM PC and used in the Motorola disc-based system, the Motorola program was designed to use the IBM PC ASCII text function files.

Motorola Disc-Based Number Format

For the disc implementation, the generated IBM PC information files from the convolution program consisted of data with a real number format (i.e., an integer and a fraction). A decision was made to implement a Motorola disc program that accessed the integer part as a signed integer and the fractional portion as a signed fraction. The segmentation of the ASCII file was required since the DSP56000 application development system (ADS) can request disc access information in either an integer, fractional, or hexadecimal format from an ASCII text file.[11] The IBM PC **g(i)** file was thus separated into two files: one file represented the integer and the other file represented the fractional part of the real number. For example, the number 1.285463 would have the integer number 1 stored as a signed integer in one file and the fraction .285463 stored as a signed

fraction in a separate file. This required that both files be accessed sequentially to form a concatenated number in the DSP56001 memory. Appendix A contains information relating to fractional arithmetic and data storage concepts.

The analog CID program uses the same fractional number format of Appendix A. A discussion on DSP56001 CID program development later in this chapter details how the **g(i)** function is represented as an analog signal and is used by the CID Assembly language program.

The DSP56000 System

The DSP56000 application development system (ADS) includes the necessary hardware and software for designing, debugging, and evaluating real time applications. As shown in Figure 9, the ADS consists of three main hardware components and a software development package.[12]

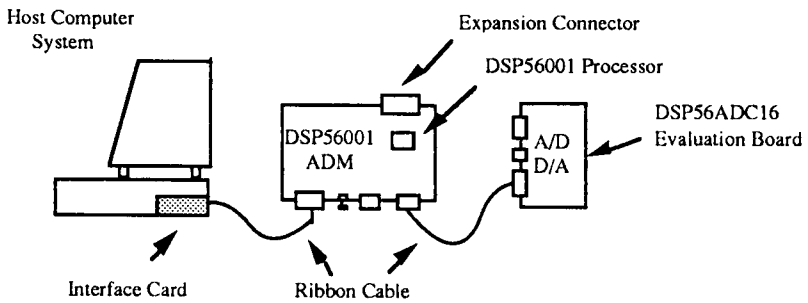


Figure 9: Application Development System

The interface card plugs into the expansion slot of a host computer motherboard. This communication link between the development software and the application development module (ADM) allows a user to develop, simulate, and test application development programs.

Every ADM contains a DSP56000 very large scale integrated (VLSI) DSP chip and the necessary interface and control circuitry for communication with the environment. This signal processing chip is the 'brain' of the ADM. The module can be configured as a slave

to the host computer system or may operate independently. If operated independently, a user program in a resident read-only-memory (ROM) chip controls the ADM.

Motorola's DSP56ADC16 evaluation board (EVB) is available as a complete serial analog-to-digital and digital-to-analog (A/D and D/A) conversion system for the ADM. Although the EVB is an added plus for most developmental work, the tandem combination of the coupling capacitor and the internal filters eliminated the use of the EVB for convolved signals. The coupling capacitors and the internal filters added additional filtering to the convolved signal before the input information reached the DSP56000 ADM.

To increase the throughput of the CID algorithm, the DSP56001 ADM was used as the signal processing system as shown in Figure 10. To accomplish this goal, several separate hardware modules were necessary to provide communication capabilities for the ADM. In addition, an analog signal generation unit was designed to produce the necessary convolved signal for analysis purposes.

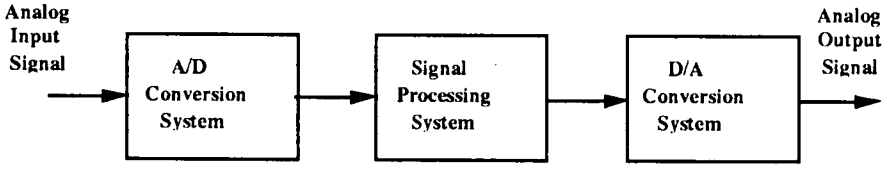


Figure 10: Real Time Signal Processing Unit

Hardware Development

The ability to generate a convolved analog signal utilized the concept of storing the pregenerated **g(i)** information in an 8-bit static memory. This allows a user to easily alter or modify the resulting signal for testing purposes. Additionally, if the static memory (specifically, an erasable programmable read only memory -- EPROM) is accessed in a sequential, loop-type fashion, the resulting output appears continuous.

A 2048x8-bit 2716 EPROM was indexed in an increasing, linear fashion by a CD4040 12-bit counter. Digital information was converted to an analog signal by an 8-bit D/A converter. The converter was configured with an op-amp to allow for signal gain

control. Figure 11 is a block diagram of the analog signal generation unit.

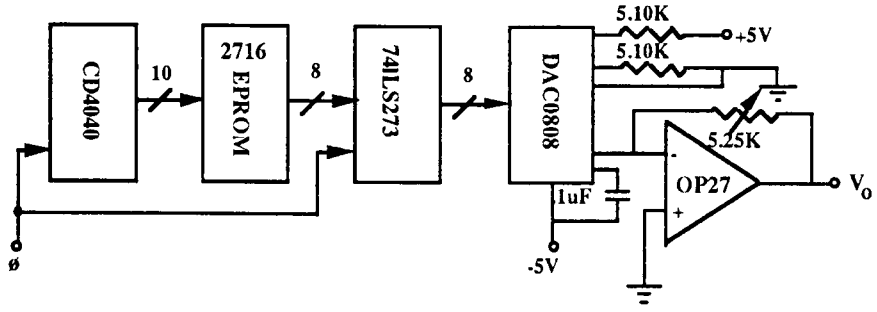


Figure 11: Analog Signal Generation Unit Block Diagram

Conversion of the analog input signal into a digital format for the ADM was accomplished using an ADC0808. The ADC0808 is a successive approximation register (SAR) type A/D converter that requires an external clock and start of conversion pulse (SOC) as detailed in the timing diagram of Figure 12. The A/D converter's SAR is reset on the rising edge of the SOC signal, while the conversion process is begun on the falling edge of the same signal. Because the ADM requires each individual sample of $\mathbf{g(i)}$ to be stored in the 2716 EPROM, the analog signal generation unit needs to be synchronized to

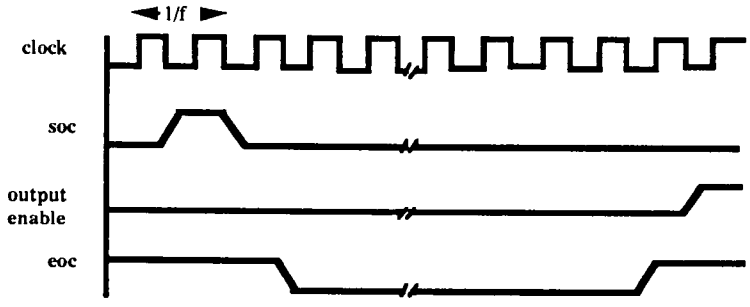


Figure 12: ADC0808 Signals

the ADC0808 converter. This requires that the rising edge of the SOC signal (i.e., reset the SAR) as shown in Figure 13 to be synchronized in time to the falling edge of the analog signal generation pulse. Additionally, the falling edge of the SOC pulse is delayed to allow the rising edge of the analog signal generation clock pulse (generated by the ADM)

time to clock the information into the 74LS273 flip-flops (see Figure 11) and stabilize before the conversion process begins. For this reason, the information for the SOC conversion signal (see Figures 12 and 13) was generated by a retriggerable monostable multivibrator. End-of-conversion (EOC) will then go high between 0 and 8 clock pulses after the falling edge of the SOC, which indicates that the analog signal has been converted to a digital value.

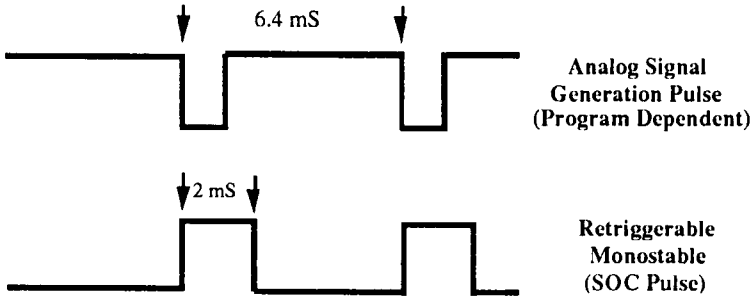


Figure 13: Retriggerable Monostable Pulse Characteristics for a 20 Wide Window

One constraint for the SOC pulse was the amount of delay time required for the falling edge of the retriggerable monostable multivibrator. This was determined by the program parameters of the ADM for the slowest CID window width algorithm. A 2 ms delay for the SOC line was more than adequate for the maximum 1 ms falling to rising edge pulse of the slowest 80 wide window CID algorithm. After a delay of 2 ms, the retriggerable monostable pulse (SOC pulse) falls, which signals the ADC0808 unit to begin the conversion process.

A second constraint was that the A/D conversion process needed to be complete for the fastest processing rate (re: a 20 wide window indicates an analog signal generation frequency pulse of 156.250 Hz). An external clock frequency of 51.917k Hz yielded a maximum A/D conversion time of

$$\text{monostable delay} + 8*(1/\text{clock frequency}) + 2\mu s$$
$$2\text{ ms} + 8*(18.89\mu s) + 2\mu s$$

or 2.153 ms, which is much less than the 6.4 ms time required for the fastest CID 20 wide window algorithm. Figure 14 is a schematic of the final A/D conversion unit shown in Figure 10.

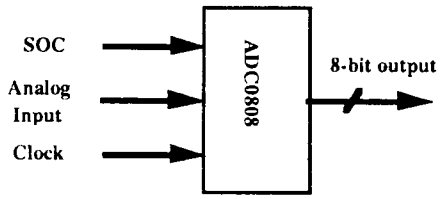


Figure 14: ADC0808 Schematic

Digital deconvolved information from the ADM was converted to an analog signal by a continuously converting DAC0808. Figure 15 is the final block diagram of the D/A conversion system necessary to obtain an analog output for Figure 10.

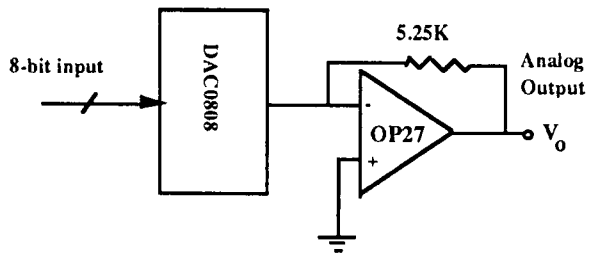


Figure 15: DAC0808 Block Diagram

These hardware modules coupled with the signal processing system and the CID application software completed the construction of the real time signal processing unit shown in Figure 10.

DSP56001 CID Program Development

The CID algorithm for the DSP56000 real time signal generation system follows the flowchart shown in Figure 16. Assembly coding for a 30 wide window is detailed in Appendix B.

Lines 1-36 clear specific sections of memory and load the impulse response file **h(i)** from disc into memory. The next eight lines of assembly code enable the necessary I/O registers of the DSP56001 to perform read/write operations through the general purpose I/O ports. An analog data point is converted and stored in a register from the ADM port. Since the number is in an 8-bit format (representative of numbers between 0 - 255), the number is multiplied by .01 and manipulated to store in the appropriate signed integer/fraction format.

A clock control bit of the output port is then cleared to allow the hardware to begin conversion of the next analog input point. While the hardware is evaluating the new analog input point, the software begins refinement estimates of the window.

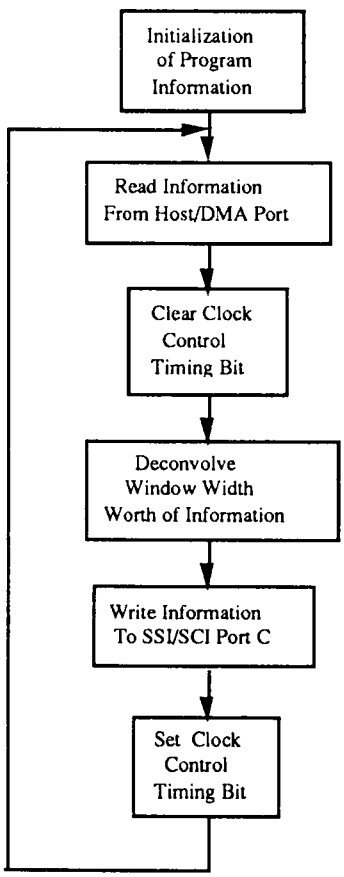


Figure 16: CID Analog Input/Output Flowchart

The previous window width estimate is shifted down in memory to allow the new input value to be placed at the top of the stack. A value at the bottom of the stack is shifted out, multiplied by a constant, and rounded to an integer for output to the D/A board. In a similar manner, the convolved data is shifted down in the stack. Information at the bottom of this stack is lost.

Line 137 is the start of the deconvolution algorithm. The **outr** loop is used to calculate the refinement of the estimate over the window width of information, while the **iner** loop calculates the convolution estimate for each specific point of the window.

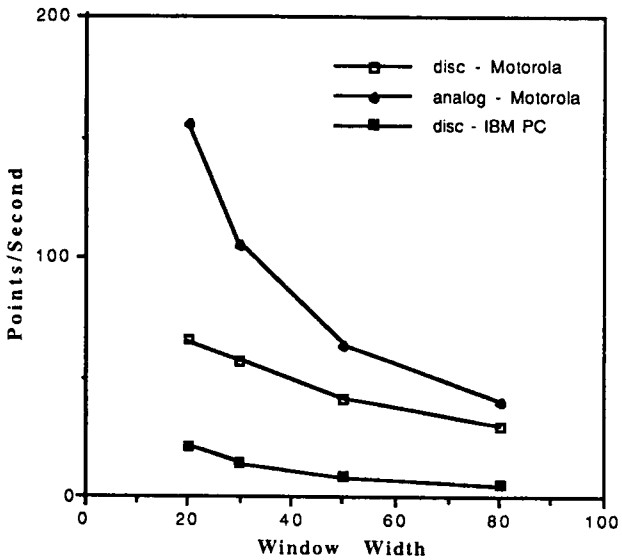
Lines 171 - 212 calculate the necessary relaxation function value for each point. The clock control bit is cleared when the convolution estimate for a specific point is calculated. The algorithm runs continuously or until the user prompts the system to stop.

An interesting point to consider is the effect that truncation has on the resulting information. Calculation of the relaxation function (b*(1 - (2/c)*abs($\hat{x}^k$ - c/2)) -- see Figure 4) involves the calculation of a non-terminating value for the specific relaxation constants c=14 and b=8. Substituting these values, the equation is 8*(1 - (1/7)*abs($\hat{x}^k$ - 7)). A problem occurs since 1/7 cannot be represented as a non-terminating fraction. The value is stored as .1428571 in a 24-bit signed fraction machine. For an estimate $\hat{x}^k$ of 0, the fractional program evaluates (abs(0 - 7))*(.1428571) as the real number 1.0000004. When the program attempts to calculate the refinement estimate $\hat{x}^{k+1}$, the machine calculates 8*.0000004 as the relaxation function refinement necessary to be added to the current estimate $\hat{x}^k$. Lines 242-259 offset this problem by checking for the value 1.0000004 and forcibly truncating the information to 1.0000000. For values of the current estimate greater than 0, the error is allowed to accumulate.

Chapter 4
Results and Conclusions

The operating times for the DSP56001 versus the IBM PC PS2 systems for various CID window widths are shown in Figure 17. Since one processor is fractionally based and programmed in Assembly language, while the other is a floating point C implementation, a common ground to compare the unbiased operating times does not exist. Such a comparison also would be biased since the disc operating system delay is introduced into the measurement process.

A hardware implementation improved the performance for the CID routine over a disc implementation. For comparison purposes, the DSP56001 real time signal generation unit was an average of 7.33 times faster than the IBM PC disc-based system for an evaluation of the same number of data points.



Motorola System:
Disc
27 ms access time
20.48M Hz CPU

IBM 80286 with math coprocessor:
Disc
30 ms access time
10M Hz CPU

Figure 17: Points/Second Rate for the IBM and Motorola Systems

Since the properties of the CID windowing algorithm are similar to Jansson's method, Table 1 was developed using RMS error comparison techniques and the same relaxation function coefficients for both algorithms without noise added to the final signal.

Here again the information is biased. A comparison of the CID algorithm to Jansson's method is unequal. Implied similarities exist, but the operating concepts are different.

Table 1: Window Width vs. Iteration RMS Error Comparison

CID Method		Jansson's Method	
Window Width	RMS Error	Iteration #	RMS Error
20	50.454002	4	52.615498
		5	42.703045
30	25.993851	7	29.517477
		8	25.060200
50	17.766672	10	18.745420
		11	16.495428
80	11.289536	14	11.979785
		15	10.983976

Using the same relaxation coefficients for both methods gives diversely different results, as is shown in Figures 18 - 27. A comparison was made on the IBM PC for Jansson's and the CID methods using random additive noise since many deconvolution techniques produce excellent results without this rather disturbing influence. This method of comparison shows that by using the same relaxation function coefficients (b=8 and c=14) for both algorithms the noise is enhanced at a faster rate for the CID method than for Jansson's. Manipulation of these constants supposedly would yield better results and faster convergence properties for the CID algorithm. Additionally, note that the CID 50 wide window routine is quicker to enhance the baseline and peak areas of the graphs as well as the additive noise than Jansson's method for 10 iterations for the same data sets. The reason for using 10 iterations of Jansson's method and a 50 wide CID routine is based upon the RMS error comparison results shown in Table 1.

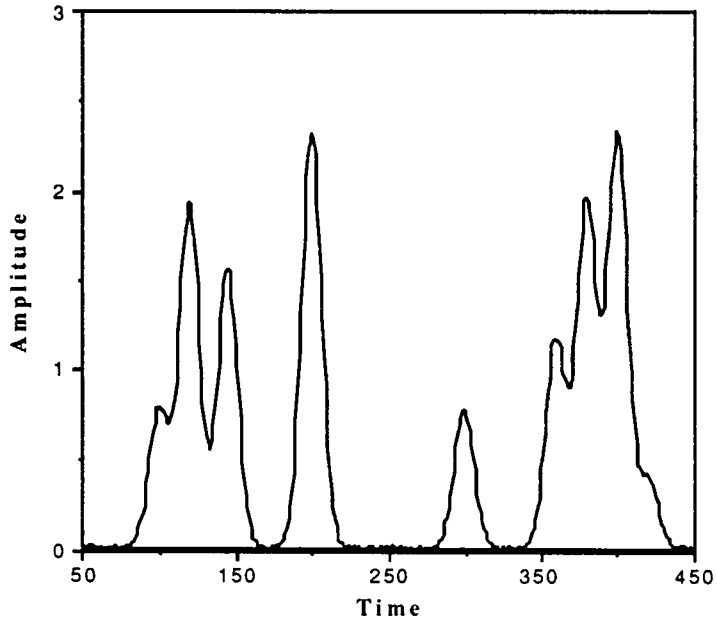


Figure 18: Convolved Input Information with S/N ratio of 100/1

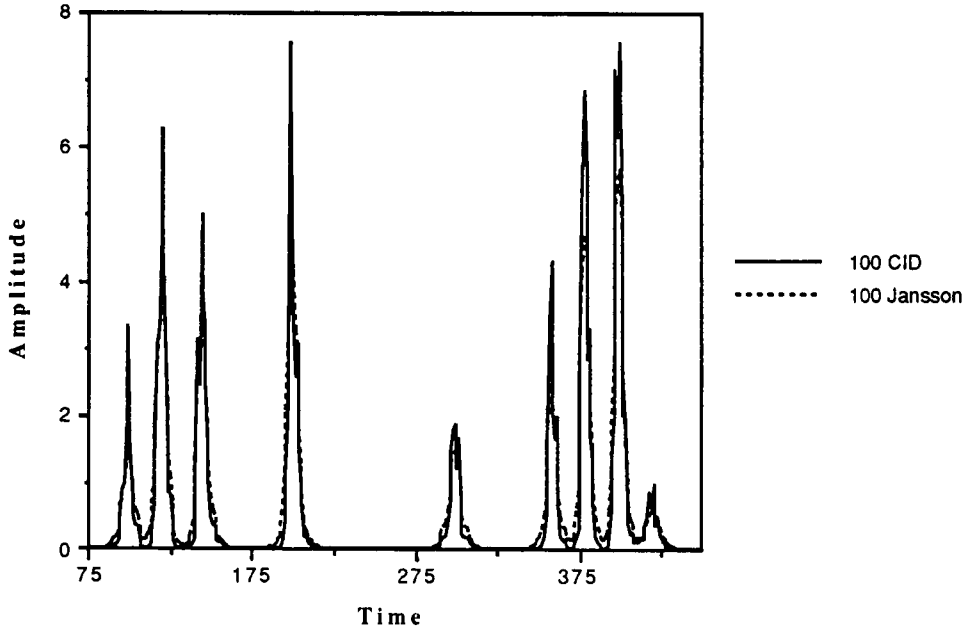


Figure 19: 100/1 S/N Ratio for Jansson's and CID Methods

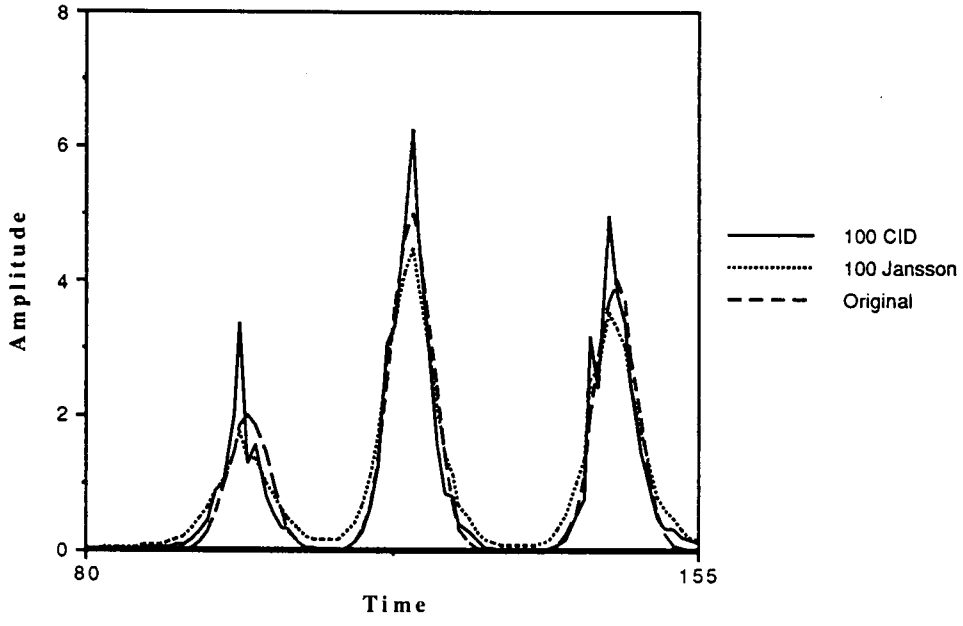


Figure 20: 100/1 S/N Ratio for Points 80-155

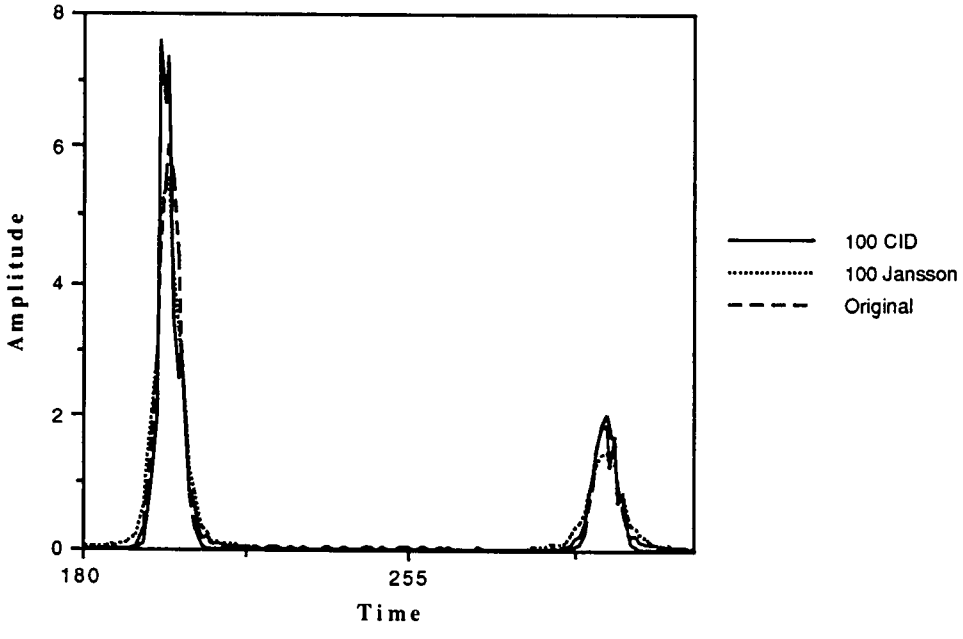


Figure 21: 100/1 S/N Ratio for Points 180-320

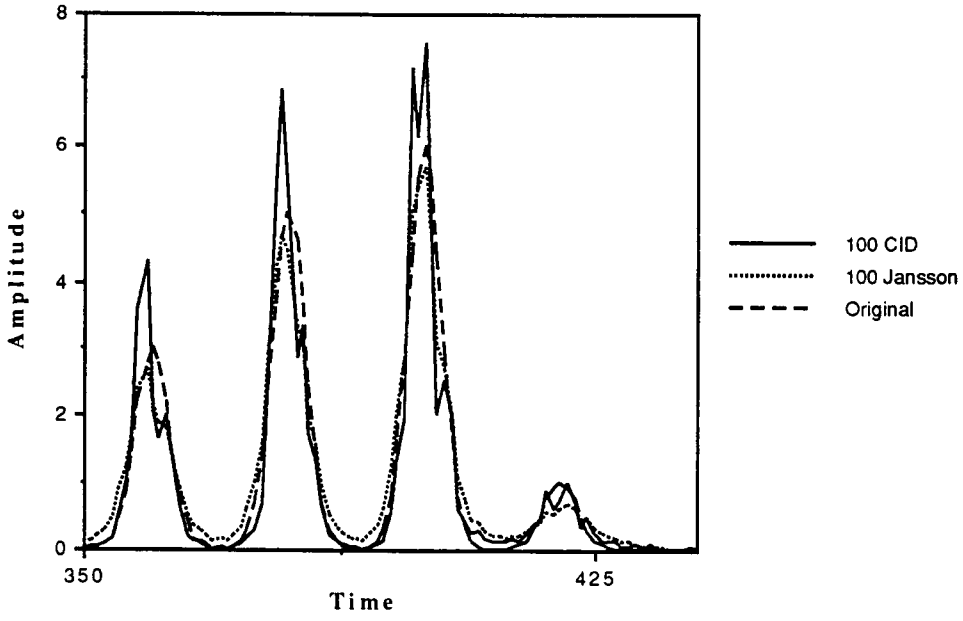


Figure 22: 100/1 S/N Ratio for Points 350-440

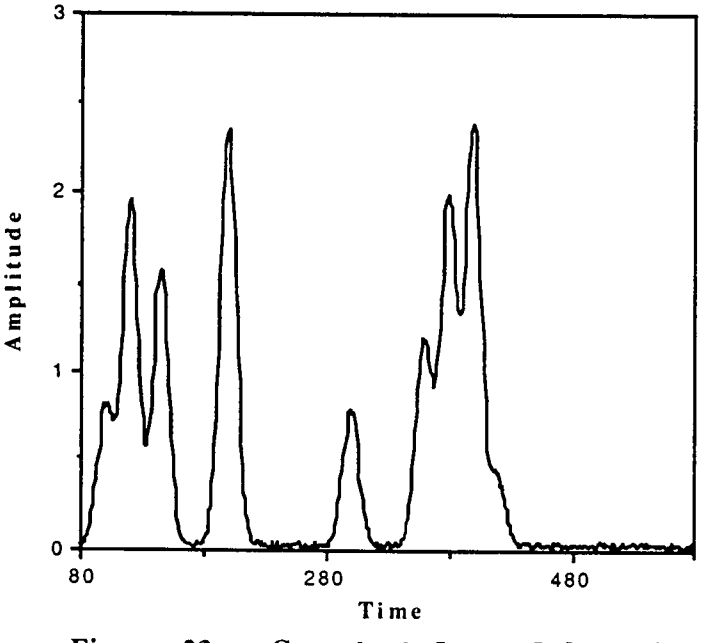


Figure 23: Convolved Input Information with S/N Ratio of 50/1

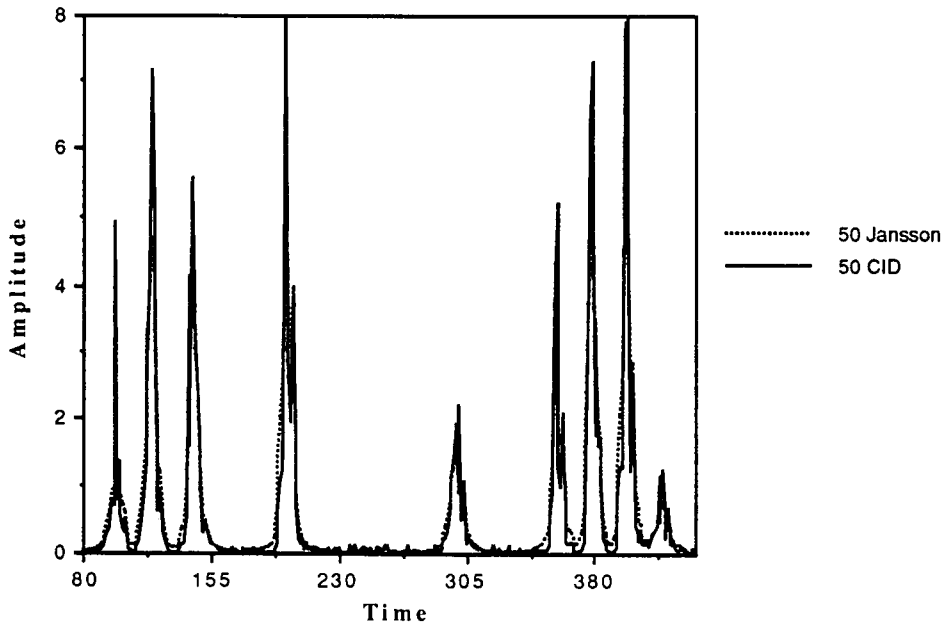


Figure 24: 50/1 S/N Ratio for Jansson's and CID Routine

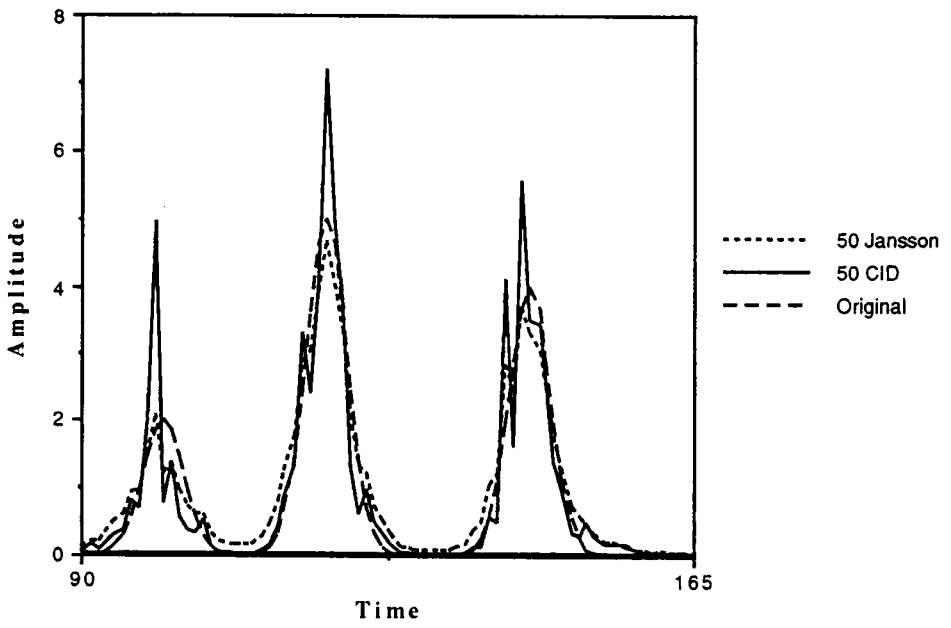


Figure 25: 50/1 S/N Ratio for Points 90-165

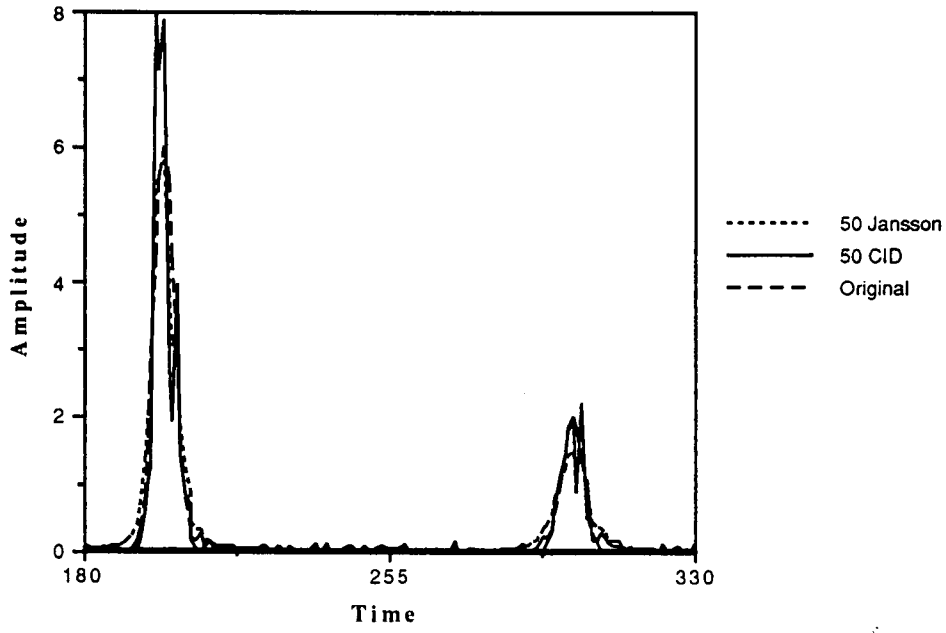


Figure 26: 50/1 S/N Ratio for Points 180-330

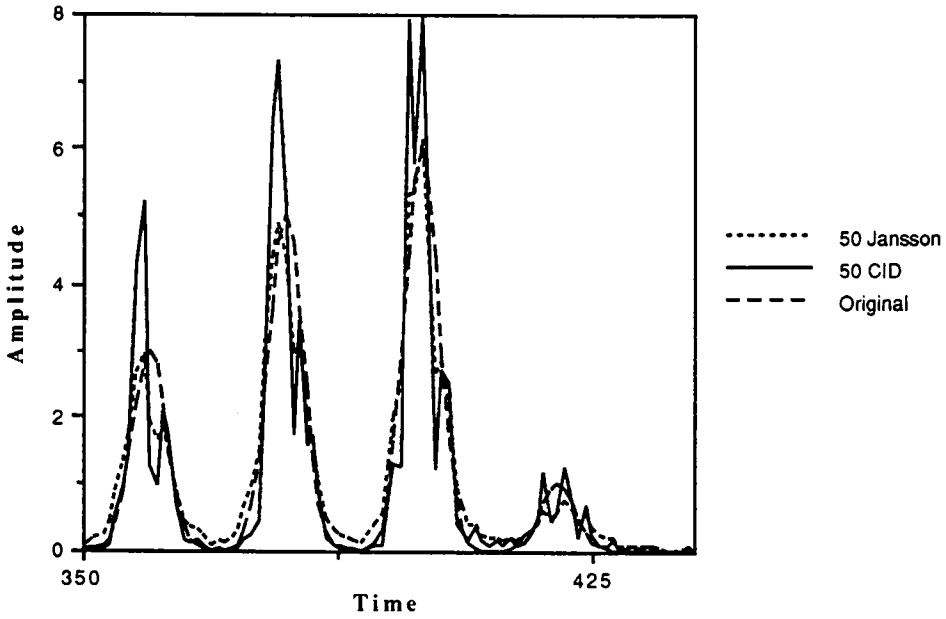


Figure 27: 50/1 S/N Ratio for Points 350-440

The DSP56000 real time signal processing system shown in Figure 10 was the only tool used to evaluate information for throughput rates of 20, 30, 50, and 80 CID window widths. These throughput rates were previously detailed in Figure 17. For a graphical evaluation, Figures 28 - 35 are presented from raw data taken from a 100M Hz HP5401A digitizing oscilloscope. Note that the external noise (S/N < 100) often fragments one peak into several peaks. Additionally note that the phase shift increases as the CID window's width is increased.

The following oscilloscope channel settings were taken for 2 volts/division measurements with a 3 volt offset in the resulting information. Amplitude comparisons of the resulting information do not appear to be scaled correctly because the input/output information was scaled to take advantage of the maximum amplitude ranges of the A/D and D/A converters.

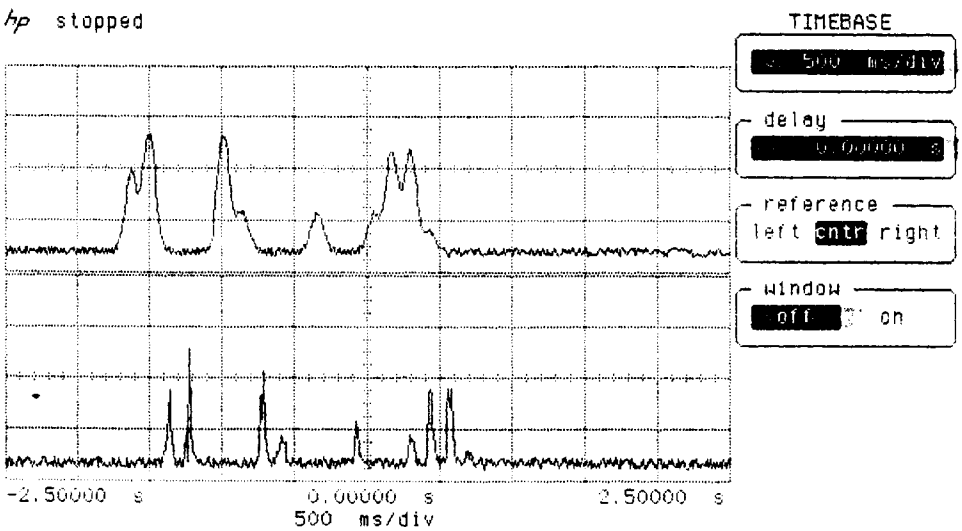


Figure 28: 20 Wide Window @ 500 ms/Div

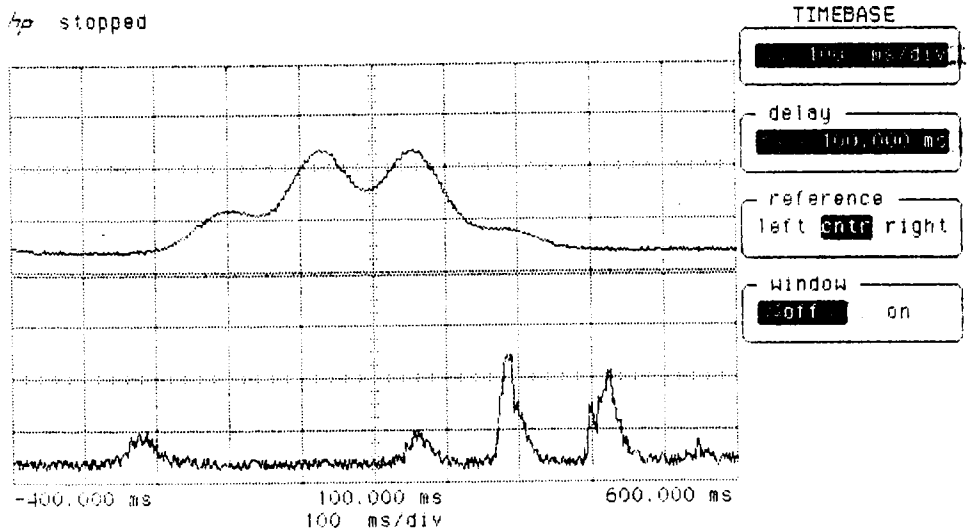


Figure 29: 20 Wide Window @ 200 ms/Div

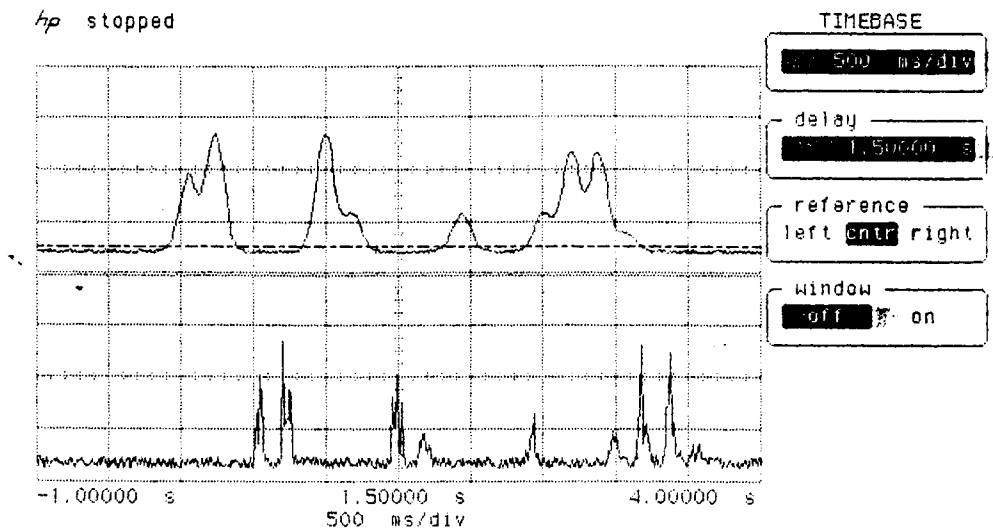


Figure 30: 30 Wide Window @ 500 ms/Div

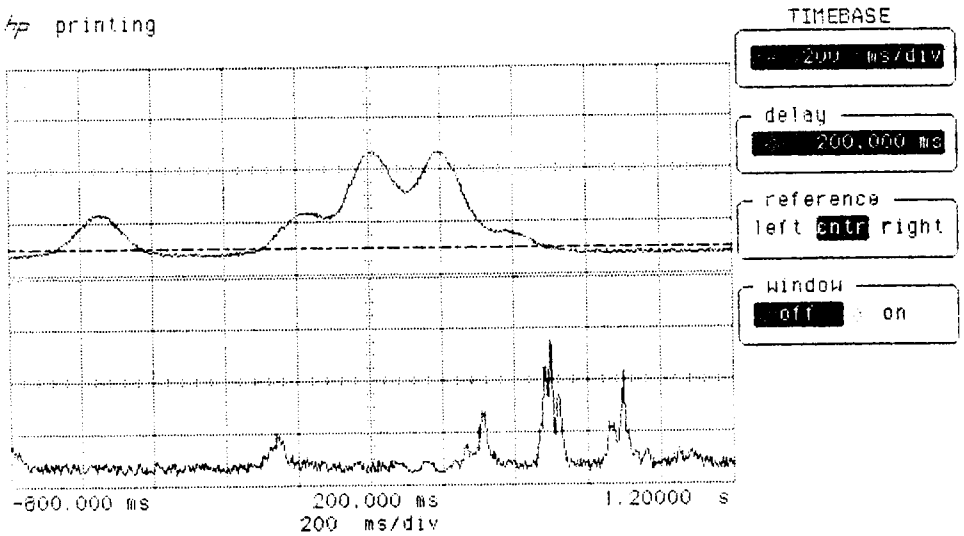


Figure 31: 30 Wide Window @ 200 ms/Div

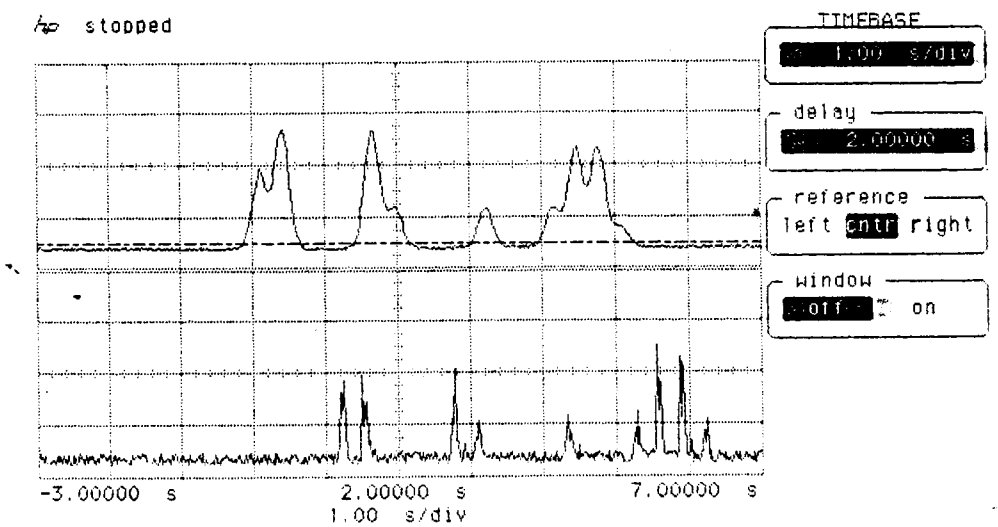


Figure 32: 50 Wide Window @ 1.00 s/Div

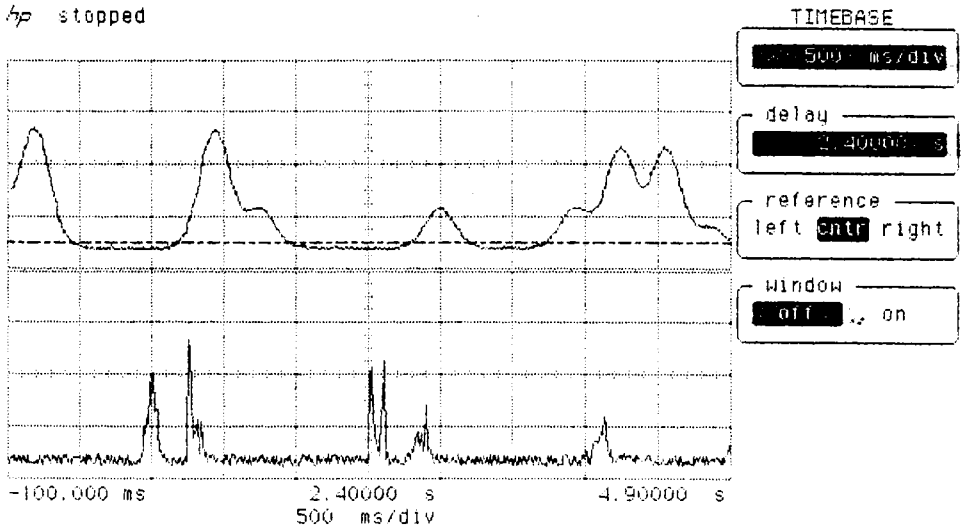


Figure 33: 50 Wide Window @ 500 ms/Div

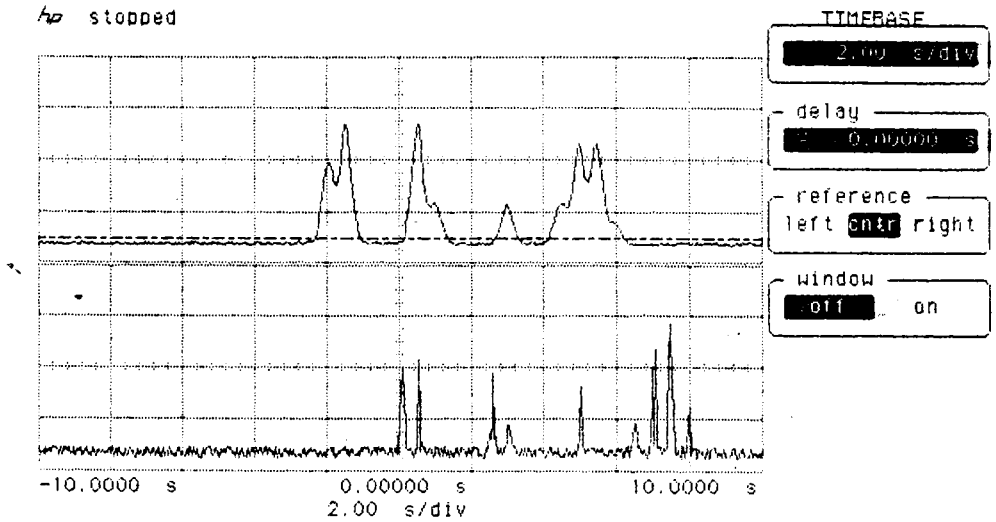


Figure 34: 80 Wide Window @ 2.00 s/Div

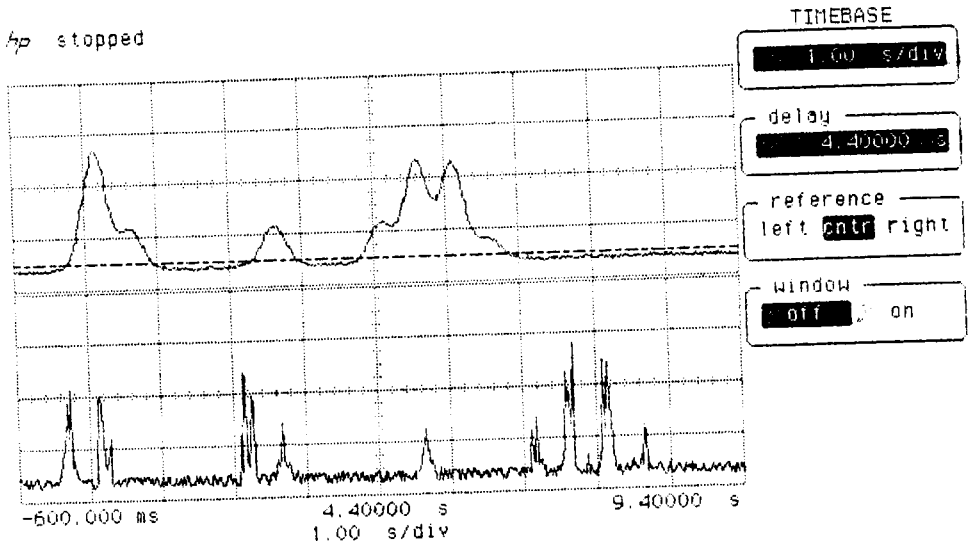


Figure 35: 80 Wide Window @ 1.00 s/Div

Recommendations

Further investigations of the new Causal Iterative Deconvolution (CID) algorithm are necessary to fully evaluate the benefits and properties of this method. Because the noise and the baseline/peak areas appear to be enhanced at a faster rate with this algorithm than with Jansson's method, a new set of relaxation coefficients are necessary to obtain the optimal S/N ratio from the algorithm.

Since many forms of instrumentation define a peak as containing 3 to 8 points (i.e., gas chromatography), the necessary window widths could be scaled down appropriately to yield even faster results. The pregenerated peak information used to test this new method

consisted of as many as 40 or more points to define a peak and therefore resulted in lower output speeds.

Additionally, a program implementation that takes better advantage of the DSP56001's signal-processing specific architecture would be to the user's benefit. If the program had utilized a 24-bit number representation instead of the 48-bit implementation (due to the 24-bit integer, 24-bit fraction required) the program may have run with a 50 to 100% decrease in the computation time. A 24-bit number implementation may allow the algorithm to take advantage of the multiply and accumulate (MAC) single line instruction and possibly the modulo-addressing capabilities of the DSP56001 architecture.

Chapter 5
Summary

The Motorola DSP56000 system appears to have a speed advantage over a conventional desktop IBM PC computer, which resulted in decreased solution times when using a CID signal processing algorithm. Although some of the above results were obtained from computer-generated arrays, the CID routine was applied to an analog signal smeared by a convolution process. Due to the speed increases over an IBM PC model 50Z 80286 computer, this form of iterative deconvolution might be applied to a wider class of data other than the Gaussian-approximated information previously described.

The restoration of the lost high frequency components for an 80 wide window width were comparable to RMS error measurements for a non-causal solution with 14 to 15 iterations. A projected window width of 3 times the impulse response function width is necessary to achieve RMS error values of less than 6%. CID analog implementation on the DSP56001 suffered from truncation of information due to the 8-bit A/D and D/A converters and a lack of a more defined set of relaxation function parameters for various S/N ratios.

Although Jansson's method has already proven the ability to deconvolve convolved Gaussian data, a more in-depth analysis of the convergence properties, noise analysis, and inherent characteristics of the CID algorithm is needed. The scope of this thesis was not to totally evaluate the benefits or inefficiencies of the CID algorithm; however, a rough estimate was given of the abilities, time-effectiveness, and simplicity of the algorithm. This work is designed to stress the need for implementing a self-developed CID algorithm on the DSP56001ADS.

Bibliography

Bibliography

[1] Frieden, B. R., Huang, T. S., et al, "Picture Processing and Digital Filtering," Springer-Verlag Berlin Heidelburg, 1975, p. 226.

[2] Jansson, Peter, A., "Superresolving Neural Network for Deconvolution," *Proc. IJCNN*, Jan. 1990.

[3] Crilly, P. B., "A Comparative Study of Relaxation Based Iterative Deconvolution Methods," *22nd SSST*, 1990.

[4] Crilly, P. B., "Signal Restoration Using a Modified Version of Jansson's Iterative Deconvolution Method," *Proc. IEEE, 1990 Southeastcon*, pp. 723-727.

[5] Jansson, P. A., *Deconvolution with Applications in Spectroscopy*, New York: Academic Press, 1984, ch. 3-4, pp. 67-134.

[6] Crilly, P. B., "Numerical Deconvolution of Gas Chromatographic Peaks Using Jansson's Method," *J. Chemometrics*, vol. 1, pp. 79-90, April 1987.

[7] El-Sharkawy, Mohamed, *Real Time Digital Signal Processing Applications with Motorola's DSP56000 Family*, Prentice Hall, ch. 1-2, 1990.

[8] van Cittert, P. H., "Z. Physik," vol. 69, p. 298, 1931.

[9] Schafer, R., Mersereau, R. and Richards, M., "Constrained Iterative Restoration Algorithms," *Proc. IEEE*, vol. 69, pp. 432-450, 1981.

[10] Crilly, Paul, Benjamin, "Numerical Deconvolution of Overlapped Gas Chromatographic Peaks Using Jansson's Method," Ph.D. Dissertation, New Mexico State University, May 1987, p. 47.

[11] Motorola Inc., "DSP5600ADS Application Development System User's Manual," Motorola 1989, p. 47.

[12] Motorola Inc., "Application Development System Reference Manual," Motorola 1989, pp. 2-6.

[13] Motorola Inc., "Fractional and Integer Arithmetic Using the DSP56000 Family of General-Purpose Digital Signal Processors," Motorola 1989, pp. 1-17.

Appendices

Appendix A
Fractional and Integer Arithmetic Concepts

The DSP56000 family of general-purpose DSPs directly supports fractional data formats and indirectly supports integer data formats. Motorola contends that a fractional data representation was chosen for the following reasons:

1) The most significant product (MSP) of a multiplication has the same format as the input and can be immediately used as an input to the multiplier without a shifting operation.

2) The least significant product (LSP) of a multiplication can be rounded into the MSP naturally, i.e., without having to handle a changing exponent.

3) All floating-point formats use fractional mantissas.[13]

Data Representations

A fractional data representation is used for all arithmetic operations in DSP56000/1 processors. Weighting of the bits is shown in Figure 36 for information in the fractional, integer, and real number formats.

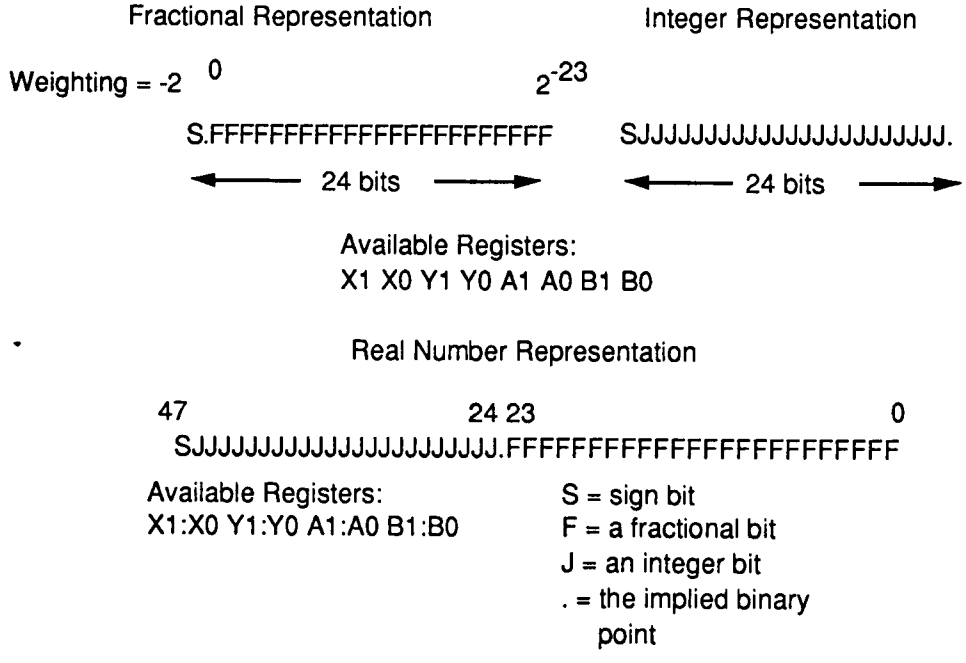


Figure 36: Data Formats

The DSP56001 views all data as left-hand justified in the fractional storage format. Integer data formats are manipulated in the same manner with the format characteristics as the inhibiting factor. Concatenation of 24-bit registers allows for maximum word lengths of 48-bits. For words and long words, values of $800000 and $800000000000 represent -1.0 in the twos-complement fractional representation. The most positive twos-complement word is $7fffff (0.9999998 decimal representation), while the most positive long word is $7fffffffffff (0.999999999999993). Additionally, the A and B accumulators have 8-bit extension registers, A2 and B2. This extension allows for word growth for a range of +255.999999999999993 and -250.0.

Twos-complement integer data representation is right-hand justified in that the binary word is interpreted as having a binary point after the least significant bit (LSB). A range of $-2^{N-1} < I < 2^{N-1} - 1$ is available for integer representations where the MSBs are used as sign extension bits. Care must be taken in manipulating integer data in that the DSP56000/1 naturally left-hand justifies data (re: assumes the data is fractional).

Arithmetic Operations

The normal operations needed to carry out the CID algorithm required addition, subtraction, and multiplication routines for numbers stored in a split signed integer/fraction format. Addition and subtraction require the same type of operation. For two real numbers to be added, a left shift of the fractional portion is required to remove the sign of both fractions. Subtraction requires the same loss of sign procedure for the fractional information.

The multiplication of two real numbers involves four basic steps:

1) multiplication of two integers.
2) multiplication of two fractions.
3) two multiplications involving an integer and a fraction.
4) summing the above multiplications.

For the four basic steps, consider the following problems.

1) Two signed integers in registers x1 and y1 are multiplied, and the result is stored in accumulator A. If the contents of registers x1, y1, and A are interpreted as fractions, the result is correct. If the contents of x1, y1, and A are interpreted as integers, a shift is required immediately following the multiplication to obtain the correct results. This shift moves the LSB out of the accumulator and adds a sign-extension bit in the MSB, re:

```
mpy x1,y1,a
asr a.
```

2) The multiplication of two signed fractions in registers x0 and y0 result in a 48-bit signed fraction in an accumulator register. To represent this number as a 24-bit signed fraction, the results must be shifted right 24-bits. The following assembly code yields the correct results:

```
mpy x0,y0,b
rep  #24
asr a.
```

3) A signed integer in x1 multiplied by a signed fraction in y0 results in a 48-bit signed real number in the accumulator:

```
mpy x1,y0,a.
```

A signed integer occupies the 24 most significant bits of a, and an unsigned fraction occupies the least significant 24-bits of a.

4) The summation of the above multiplications may be implemented by the following operation:

```
add   a,b
```

where the fractional portion of the A and B registers is assumed to be unsigned.

To simplify the process, the multiplication was augmented using absolute values. By initially storing the sign of both numbers in memory, a simple check was a determinant as to whether the resulting value was positive or negative.

APPENDIX B
ASSEMBLER PROGRAM FILE

```
                opt   nopp,nofc
            ;*********************************
            ;***      Real Time Iterative      ***
            ;***        Deconvolution          ***
            ;***       30 point window         ***
            ;****        Rick Mitchell         ****
            ;*****         12/31/91            ****
            ;*****    Rev. for 30 pt window   *******
            ;*********************************

            ;organize and read in data

            ;load function to be deconvolved

X:0000              org   x:00
X:0000      tblx    ds    900

Y:0000              org   y:00
Y:0000      tbly    ds    900

P:0040              org   p:$40

            ;clr estimate & input table

P:0040 370000       move  #0,r7
P:0041 61F400       move  #450,r1
   0001C2
P:0043 200013       clr   a
P:0044 06C281       do    #450,lpin
   000047
P:0046 8A2700       move  a,x:(r7)     a,y:(r1)
P:0047 BA7900       move  a,x:(r1)+    a,y:(r7)+
            lpin

P:0048 377800       move  #120,r7                ;start address
P:0049 2E0100       move  #1,a                   ;x memory
P:004A 44F400       move  #$010029,x0            ;ld x:(120) with table
   010029
P:004C 0BF080       jsr   $e006
   00E006

            ;set up port c for output

P:004E 08F4A1       movep #>0,x:$ffe1
   000000
P:0050 08F4A3       movep #>$1ff,x:$ffe3
   0001FF

            ;set up port b for input

P:0052 08F4A0       movep #0,x:$ffe0
   000000
```

```
46    P:0054 08F4A2       movep  #0,x:$ffe2
        000000
47
48
49
50    1.428571E-001 inv_7  equ        .142857142
51
52              ;start of the convolution routine
53              ;results will be written to

54              ;memory location l:400-900
55
56    P:0056 3AC800       move   #200,n2                ;offset for g
57    P:0057 34B400       move   #180,r4                ;for jclr instr.
58
59    P:0058 000000 thend nop
60    P:0059 000000       nop
61    P:005A 000000       nop
62
63              ;*********************************
64              ;     read data in
65              ;*********************************
66
67    P:005B 56F000       move   x:$ffe4,a
        00FFE4
68    P:005D 44F400       move   #>$ff,x0
        0000FF
69    P:005F 200046       and    x0,a
70    P:0060 21C400       move   a,x0
71
72              ;*********************************
73              ;     write out clock control bit #8
74              ;*********************************
75
76    P:0061 0AA508       bclr   #8,x:$ffe5
77
78              ;multiply input value by .01
79
80    P:0062 45F400       move   #$0147ae,x1
      -  0147AE
81    P:0064 2000A0       mpy    x0,x1,a
82    P:0065 218700       move   a1,y1
83    P:0066 2C0000       move   #0,a1
84    P:0067 200022       asr    a
85    P:0068 210600       move   a0,y0
86
87    P:0069 373200       move   #50,r7                 ;set operating point
88
89              ;shift old data down
90
91    P:006A 061980       do     #25,tche               ;.5*oper. point
        000073
```

```
92      P:006C 45E700        move  x:(r7),x1
93      P:006D 4CE700        move  y:(r7),x0          ;shift estimate
94      P:006E 476700        move  y1,x:(r7)          ;window down
95      P:006F 4E5700        move  y0,y:(r7)-
96      P:0070 47E700        move  x:(r7),y1
97      P:0071 4EE700        move  y:(r7),y0
98      P:0072 456700        move  x1,x:(r7)
99      P:0073 4C5700        move  x0,y:(r7)-
100           tche
101     P:0074 45E700        move  x:(r7),x1
102     P:0075 4CE700        move  y:(r7),x0
103     P:0076 476700        move  y1,x:(r7)
104     P:0077 4E6700        move  y0,y:(r7)
105
106           ;multiply by 31
107     P:0078 47F400        move  #>31,y1
          00001F
108     P:007A 2000F0        mpy   y1,x1,a            ;intxint
109     P:007B 200022        asr   a

110     P:007C 0618A0        rep   #24
111     P:007D 200032        asl   a
112     P:007E 2000C8        mpy   y1,x0,b            ;fractxint
113     P:007F 200010        add   b,a
114     P:0080 200011        rnd   a
115     P:0081 218500        move  a1,x1
116     P:0082 210400        move  a0,x0
117
118     P:0083 457000        move  x1,x:$ffe5
          00FFE5
119
120     P:0085 47B200        move  x:(50),y1          ;setup data for
121     P:0086 4EB200        move  y:(50),y0          ;g(t) shift
122     P:0087 35FA00        move  #250,r5
123
124     P:0088 061980        do    #25,texc
          000091
125     P:008A 45E500        move  x:(r5),x1
126     P:008B 4CE500        move  y:(r5),x0          ;shift input
127     P:008C 476500        move  y1,x:(r5)          ;window down
128     P:008D 4E5500        move  y0,y:(r5)-         ;51st value
129     P:008E 47E500        move  x:(r5),y1          ;will be lost
130     P:008F 4EE500        move  y:(r5),y0
131     P:0090 456500        move  x1,x:(r5)
132     P:0091 4C5500        move  x0,y:(r5)-
133           texc
134     P:0092 476500        move  y1,x:(r5)
135     P:0093 4E6500        move  y0,y:(r5)
136
137           ;start of deconvolution
138
139     P:0094 321400        move  #20,r2             ;o.pt - wind width
```

```
140     P:0095 225600       move  r2,r6                 ;r6 destroyed inner lp.
141     P:0096 3E1400       move  #20,n6                ;imp resp width
142
143     P:0097 061F80       do    #31,outr              ;wind + 1
          000171
144
145     P:0099 55EE00       move  x:(r6+n6),b1          ;int of est
146     P:009A 59EE00       move  y:(r6+n6),b0          ;fract of est
147     P:009B 0B6E37       btst  #23,x:(r6+n6)
148                   .if   <cs>
151     P:009E 20002E       abs   b
152                   .endi
154     P:009F 21A700       move  b1,y1
155     P:00A0 212600       move  b0,y0
156
157     P:00A1 200013       clr   a
158     P:00A2 547000       move  a1,x:(181)            ;clear out
          0000B5
159     P:00A4 5C7000       move  a1,y:(181)            ;sum value
          0000B5
160     P:00A6 20001B       clr   b                     ;b is sum of conv
161     P:00A7 317800       move  #120,r1
162
163     P:00A8 062980       do    #41,iner              ;convolution start
          0000D0
164
165     P:00AA 44D900       move  x:(r1)+,x0            ;h(t) with sign
166
167     P:00AB 2000C0       mpy   x0,y1,a               ;intxfract

168     P:00AC 547000       move  a1,x:(183)            ;store result
          0000B7
169     P:00AE 587000       move  a0,y:(183)            ;as real number
          0000B7
170     P:00B0 2000D0       mpy   x0,y0,a               ;fractxfract
171     P:00B1 0618A0       rep   #24
172     P:00B2 200022       asr   a
173     P:00B3 200032       asl   a
174     P:00B4 45F000       move  x:(183),x1
          0000B7
175     P:00B6 4CF000       move  y:(183),x0
          0000B7
176     P:00B8 200020       add x,a
177     P:00B9 200009       tfr   a,b
178     P:00BA 0B6E37       btst  #23,x:(r6+n6)
179                   .if   <cs>
182     P:00BD 20003E       neg   b
183                   .endi
185     P:00BE 200001       tfr   b,a
186     P:00BF 55F000       move  x:(181),b1
          0000B5
187     P:00C1 59F000       move  y:(181),b0
```

```
          0000B5
188       P:00C3 200018      add   a,b                    ;save result in b
189       P:00C4 557000      move  b1,x:(181)
          0000B5
190       P:00C6 597000      move  b0,y:(181)
          0000B5
191
192       P:00C8 20561B      clr   b     (r6)-
193
194       P:00C9 55EE00      move  x:(r6+n6),b1
195       P:00CA 59EE00      move  y:(r6+n6),b0
196       P:00CB 0B6E37      btst  #23,x:(r6+n6)
197                  .if   <cs>
200       P:00CE 20002E      abs   b
201                .endi
203       P:00CF 21A700      move  b1,y1
204       P:00D0 212600      move  b0,y0
205
206       P:00D1 000000 iner   nop
207
208                ;*********************************
209                ;   write out clock control bit #8
210                ;*********************************
211
212       P:00D2 0AA528      bset  #8,x:$ffe5
213
214       P:00D3 200013      clr   a
215       P:00D4 55F000      move  x:(181),b1              ;sum stored in
          0000B5
216       P:00D6 59F000      move  y:(181),b0              ;real # format
          0000B5
217       P:00D8 557000      move  b1,x:(182)              ;move sum
          0000B6
218       P:00DA 597000      move  b0,y:(182)              ;into storage
          0000B6
219       P:00DC 58E200      move  y:(r2),a0
220       P:00DD 200032      asl   a
221       P:00DE 54E200      move  x:(r2),a1
222       P:00DF 45F400      move  #>7,x1                  ;xest - 7
          000007
223       P:00E1 240000      move  #0,x0
224       P:00E2 200024      sub   x,a
225       P:00E3 200026      abs   a
226       P:00E4 218500      move  a1,x1
227       P:00E5 2C0000      move  #0,a1
228       P:00E6 200022      asr   a                       ;replace sign
229       P:00E7 210400      move  a0,x0
230       P:00E8 46F400      move  #inv_7,y0               ;div by 7
          124925
231       P:00EA 2000E0      mpy   y0,x1,a                 ;intxfract
232       P:00EB 2000D8      mpy   y0,x0,b                 ;fractxfract
```

```
233     P:00EC 0618A0        rep   #24
234     P:00ED 20002A        asr   b
235     P:00EE 20003A        asl   b
236     P:00EF 200010        add   b,a                          ;result in a
237     P:00F0 200009        tfr   a,b
238     P:00F1 556400        move  b1,x:(r4)
239     P:00F2 2D0000        move  #0,b1
240     P:00F3 20002A        asr   b
241     P:00F4 596400        move  b0,y:(r4)
242                  .if   x:(r4) <EQ> #>$1     AND   y:(r4) <LE
> #>$3
252     P:0101 290000        move  #0,b0
253     P:0102 55E400        move  x:(r4),b1
254                  .else
257     P:0105 20003A        asl   b
258     P:0106 55E400        move  x:(r4),b1
259                  .endi
261     P:0107 212400        move  b0,x0
262     P:0108 21A500        move  b1,x1
263     P:0109 56F400        move  #>$1,a                       ;1 - intm
          000001
264     P:010B 200024        sub   x,a
265     P:010C 200009        tfr   a,b
266     P:010D 546400        move  a1,x:(r4)                    ;x:(r4) sign of
267     P:010E 0B6437        btst  #23,x:(r4)                   ;1 - intm
268                  .if   <cs>                                 ;if negative
271     P:0111 20002E        abs   b                            ;make positive
272                  .endi
274     P:0112 21A500        move  b1,x1
275     P:0113 2D0000        move  #0,b1
276     P:0114 20002A        asr   b
277     P:0115 212400        move  b0,x0
278
279     P:0116 46F400        move  #>8,y0                       ;8*(1 - intm)
          000008
280     P:0118 2000E0        mpy   x1,y0,a                      ;intxint
281     P:0119 200022        asr   a                            ;create sign
282     P:011A 0618A0        rep   #24
283     P:011B 200032        asl   a
284     P:011C 2000D8        mpy   x0,y0,b                      ;fractxint
285     P:011D 200010        add   b,a                          ;real numb in a
286     P:011E 547000        move  a1,x:(181)
          0000B5
287     P:0120 2C0000        move  #0,a1
288     P:0121 200022        asr   a
289     P:0122 587000        move  a0,y:(181)
          0000B5
290
291     P:0124 20001B        clr   b

292     P:0125 200013        clr   a
293     P:0126 59F000        move  y:(182),b0                   ;in real
```

```
          0000B6
294       P:0128 55F000      move  x:(182),b1          ;numb format
            0000B6
295       P:012A 58EA00      move  y:(r2+n2),a0
296       P:012B 200032      asl   a                   ;convert to rn form
297       P:012C 54EA00      move  x:(r2+n2),a1
298       P:012D 200014      sub   b,a
299
300       P:012E 200009      tfr   a,b
301       P:012F 5C6400      move  a1,y:(r4)           ;y:(r4) sign of
302       P:0130 0B6477      btst  #23,y:(r4)          ;g[n] - sum
303                    .if   <cs>                      ;move g[n] -sum
306       P:0133 20002E      abs   b                   ;into y in
307       P:0134 21A700      move  b1,y1               ;signed int/fract
308       P:0135 2D0000      move  #0,b1               ;format
309       P:0136 20002A      asr   b
310                    .else
313       P:0139 21A700      move  b1,y1
314       P:013A 2D0000      move  #0,b1
315       P:013B 20002A      asr   b
316                    .endi
318       P:013C 212600      move  b0,y0
319       P:013D 45F000      move  x:(181),x1
            0000B5
320       P:013F 4CF000      move  y:(181),x0
            0000B5
321
322       P:0141 2000D8      mpy   x0,y0,b             ;fractxfract
323       P:0142 0618A0      rep   #24
324       P:0143 20002A      asr   b
325       P:0144 20003A      asl   b
326       P:0145 2000C0      mpy   x0,y1,a             ;fractxint
327       P:0146 200010      add   b,a
328       P:0147 2000E8      mpy   x1,y0,b             ;intxfract
329       P:0148 200010      add   b,a
330       P:0149 2000F8      mpy   x1,y1,b             ;intxint
331       P:014A 20002A      asr   b                   ;create sign
332       P:014B 0618A0      rep   #24
333       P:014C 20003A      asl   b
334      -P:014D 200010      add   b,a
335
336       P:014E 547000      move  a1,x:(181)          ;l:(181) is
            0000B5
337       P:0150 587000      move  a0,y:(181)          ;yn(g[n] - sum)
            0000B5
338       P:0152 200013      clr   a                   ;in real # format
339       P:0153 20001B      clr   b
340       P:0154 50E400      move  x:(r4),a0           ;determine
341       P:0155 200032      asl   a                   ;correct sign +/-
342       P:0156 59E400      move  y:(r4),b0
343       P:0157 20003A      asl   b
344       P:0158 290000      move  #0,b0
```

```
345     P:0159 280000          move  #0,a0
346     P:015A 200010          add   b,a
347     P:015B 546400          move  a1,x:(r4)
348     P:015C 54F000          move  x:(181),a1
          0000B5
349     P:015E 58F000          move  y:(181),a0
          0000B5

350     P:0160 0A6480          jclr  #0,x:(r4),psr
          000163
351     P:0162 200036          neg   a
352     P:0163 59E200  psr     move  y:(r2),b0
353     P:0164 20003A          asl   b                    ;lose sign bit
354     P:0165 55E200          move  x:(r2),b1
355     P:0166 200010          add   b,a                  ;yn = yn + xest
356
357     P:0167 546200          move  a1,x:(r2)            ;store value
358     P:0168 2C0000          move  #0,a1                ;back to xest
359     P:0169 200022          asr   a                    ;create sign
360     P:016A 586200          move  a0,y:(r2)
361     P:016B 0A6297          jclr  #23,x:(r2),pos4
          00016E
362     P:016D 0A6277          bset  #23,y:(r2)           ;negative
363     P:016E 205A13  pos4    clr   a     (r2)+
364     P:016F 20001B          clr   b                    ;instruction not
365     P:0170 225600          move  r2,r6                ;available yet
366     P:0171 000000          nop
367     P:0172 000000  outr    nop
368     P:0173 000000          nop
369     P:0174 200013          clr   a
370     P:0175 20001B          clr   b
371
372     P:0176 0C0058          jmp   thend
373     P:0177 000000          nop
374     P:0178 000000          nop
375     P:0179 000000          nop
376                    end

0  Errors
0  Warnings
```

Vita

Mike Mitchell was born in Kingsport, Tennessee, on July 26, 1966. After graduating from Sullivan Central High School with an emphasis in science and mathematics, he moved to Knoxville, Tennessee, for undergraduate studies at the University of Tennessee. He was awarded a B.S. degree in Electrical and Computer Engineering with an emphasis in Digital and Analog systems design in May 1990.

In the summer of 1990, he was a summer intern for CTI/Siemans in Knoxville. His efforts were directed towards high speed analog/digital test equipment design for the bucket electronics of the Positron Emission Tomography (PET) medical imaging scanner and C programming.

He joined the graduate program in Electrical Engineering at the University of Tennessee in the fall of 1990. As a graduate research assistant, he focused his research efforts on the development and implementation of a new form of iterative deconvolution. Other duties included a teaching assistantship for the senior-level 68HC11 microprocessor course.

Also during this time, Mr. Mitchell worked for the Electrical and Computer Aided Design section of Philips Consumer Electronics (Magnavox) in Knoxville to implement file management programs in the shell script, awk, and C programming languages.

The author's permanent mailing address is 1888 Shipley Ferry Road, Kingsport, TN 37663.

This thesis was typed by the author on the MacWrite II word-processing system of the Macintosh IIx of the Communications, Information, and Signal Processing (CISP) Group of the Electrical Engineering Department.