The Static and Dynamic Characterization of the MC68HC11A8's Analog to Digital Converter

by

Jeffrey Charles Daniels

B.S, Kansas State University, 1984

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

Donald A. Lenhert
____
Major Professor

Table of Contents

List of Figures

List of Tables

Acknowledgments

My gratitude goes to Motorola Semiconductor Group for funding this project and to the Department of Electrical and Computer Engineering for their patience and understanding to help the completion of this project come to an end. Special thanks goes to Dr. Lenhert for his guidance and prodding to get the spoken words down on paper in an understandable form. Most of all, I would like to thank my parents for their teachings which gave me the will to make it though graduate school, my wife who gave me the reason why, and finally my son whose learning mind gave hope and promise for all good things to come.

1.0 Introduction

Automated test equipment for integrated circuit (IC) manufacturing in mass production applications is expensive to purchase and to maintain. This expense contributes to the proportionality between the cost of an IC and its testing time. As ICs become more complex, as with the case of microprocessors, it soon becomes cost prohibitive to test all combinations of inputs, outputs, and functions available.

Motorola has developed a fast, low power microcomputer, the MC68HC11A8 (HC11), which has an elaborate timer system, two serial communications interfaces, parallel input/output (I/O) configurations, and a unique feature of a onboard 8 bit, successive approximation analog to digital converter (A/D) with sample and hold.[1] The HC11 has endless possibilities for control applications using its timer system, serial communication interfaces, I/O configurations, and its eight bit A/D. One control application and a more in-depth description of the HC11 is outlined in Draving[4] where the HC11 is used as a controller for low power, precision A/D converters.

For an A/D used in a critical application, extensive testing is necessary to ensure conversion results to be within the manufacturer's specifications. As stated by Doerfler[2], testing of even low resolution A/Ds can take several hours to complete. This presents a problem for Motorola. To keep the HC11 at a competitive price, testing time must be kept to a minimum thus eliminating the possibility of extensive testing its A/D. The purpose of this thesis is to statically and dynamically characterize the HC11's A/D and to present the testing procedures used in this characterization.

2.0 The Testing System

In order to test the HC11's A/D, a test system was developed consisting of a M68HC11EVB evaluation board (EVB), an interface board, a Hewlett Packard (HP) 9845B computer with a parallel interface, an IBM PCXT equipped with a modified (pull up resistors on inputs and outputs) 24 bit Parallel Digital I/O Interface Model PIO12 Metra-byte board, an HP 3878A digital voltage meter (DVM), an HP 3325A function generator, and an eighteen bit digital to analog converter (DAC) as shown in figure 2-1.

The EVB is a small, compact, low cost tool for development of HC11 based target system equipment.[5] This board provides host computer down loading capabilities which allows the use of a cross assembler running on an IBM PCXT, eight kilobytes (8k) of user RAM, 8k of EPROM, and a monitor/debugging program called BUFFALO (Bit Users Fast Friendly Aid to Logical Operations). The EVB provides access to all 52 pins of its HC11 via a 60 conductor flat ribbon cable.

The EVB is well suited for the testing system in figure 2-1 except for the lack of a bypass capacitor on the HC11's power and ground pins. This problem was corrected by the installation of a 10uF, tantalum capacitor across the HC11's V_{DD} and V_{SS} pins on the underside of the EVB board.

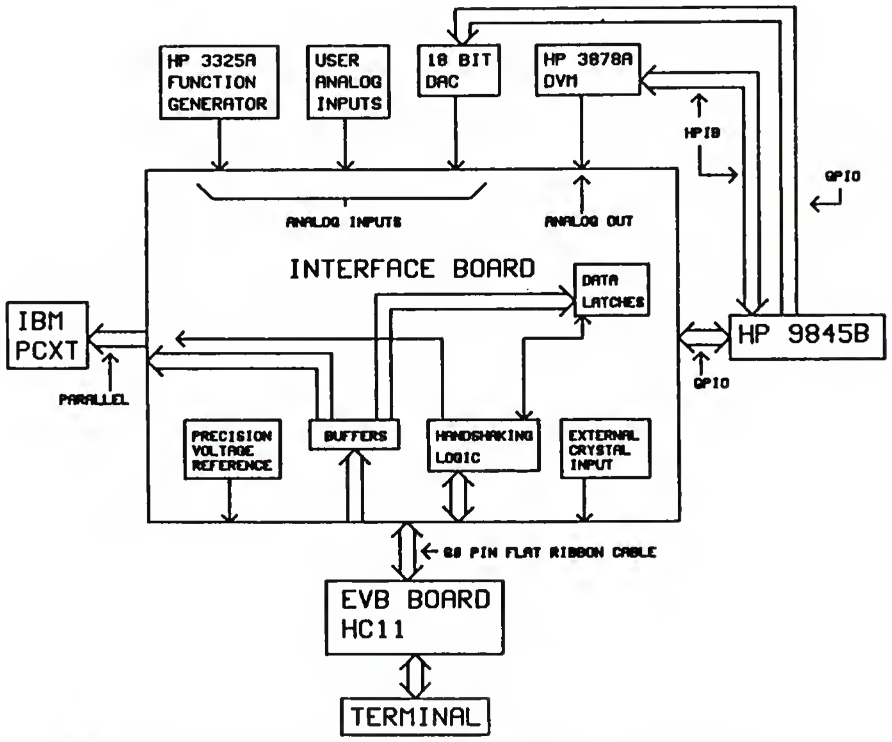

Figure 2-1. Static and Dynamic Testing Configuration

The interface board of the test system, shown in figure 2-2, provides buffering for the HC11's inputs and outputs, data latches, handshaking logic to the HP 9845B, a stable voltage reference for the V_{RH} input, and a circuit to provide an external source for the HC11's EXTAL and XTAL pins to allow the user to lower the standard operating frequency of the EVB.

The buffers are used to protect the HC11 from being overdriven thus causing possible damage. 74HC373 unidirectional 8 bit data latches were chosen for the buffers and also the data latches on the interface board. Used as buffers, the 74HC373s were operated in transparent mode to allow the outputs to follow the inputs with no need for a clock input. The 74HC373s used as data latches on the interface board used a pulse output from a pin (PA4) on the HC11's PORTA to set and hold the data to be read by the HP 9845B or IBM PCXT. Figure 2-2 shows high and low byte data latches although the only use for the high byte is to establish 0's on the top eight data lines on the GPIO interface.

The handshaking logic between the HC11 and HP 9845B is a 7474 D flip flop with preset and clear inputs. The HC11 waits until the flip flop is set before sending data to the HP 9845B by latching the data into the data latches with a pulse on PA4 and clearing the D flip flop with a pulse on PA6. When a conversion result is latched into

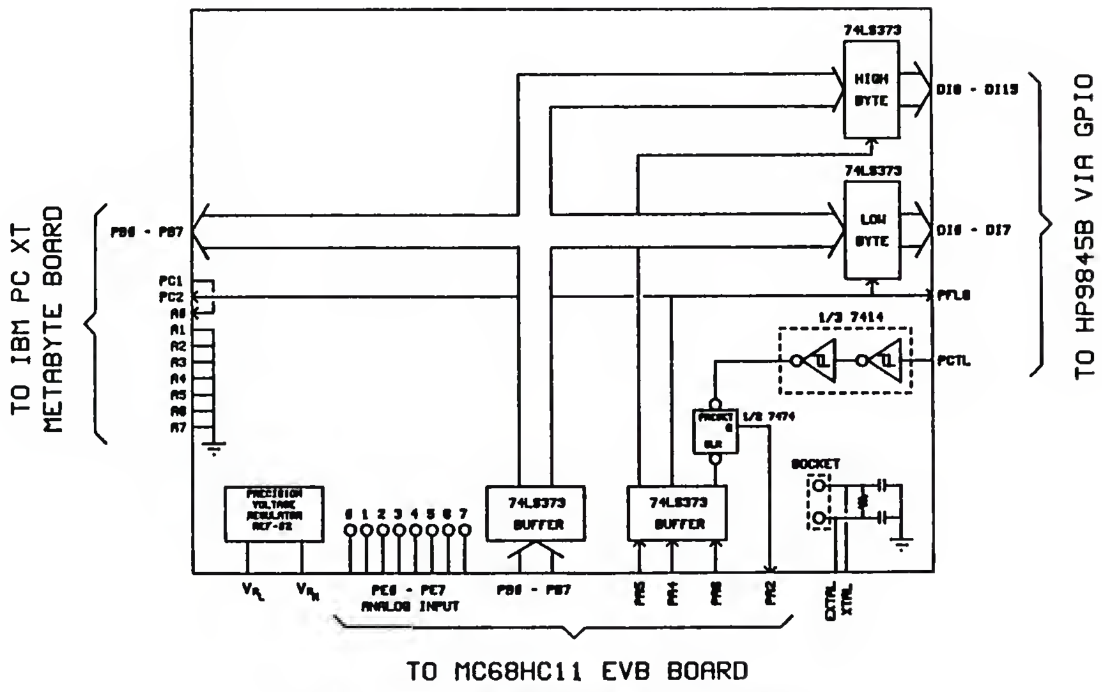

Figure 2-2. Interface Board Configuration

the latches, PA4 also pulses the HP 9845B to indicate valid data. When the HP 9845B wants data, it sets the flip flop and then waits for a pulse on its PFLG pin to accept the available data.

The handshaking method used with the IBM PCXT and the HC11 is simpler than that used with the HP 9845B. When the HC11 has data available, the conversion result is latched with a pulse being sent, via PA4, to the IBM PCXT. For the IBM PCXT to receive data, it waits for a pulse from the HC11, takes the data and then waits for another pulse.

The HP 3878A DVM is controlled by the HP 9845B via an HPIB interface and is used to measure voltages and return the results to the HP 9845B for the analysis of testing results. The HP 3325A function generator is controlled manually by the user to provide a precision sine wave used as one of the analog inputs to the HC11's A/D. Also, an 18 bit DAC, built and tested to 16 bit linearity by Holdeman[6], provides a ramp input to the HC11's A/D. A precision voltage reference shown in figure 2-3, provides a stable voltage of +5 volts to be used as the input of V_{RH} pin on the HC11. Finally, the clock frequency of the EVB can be changed by changing a jumper on the EVB and placing a suitable crystal on the interface board.

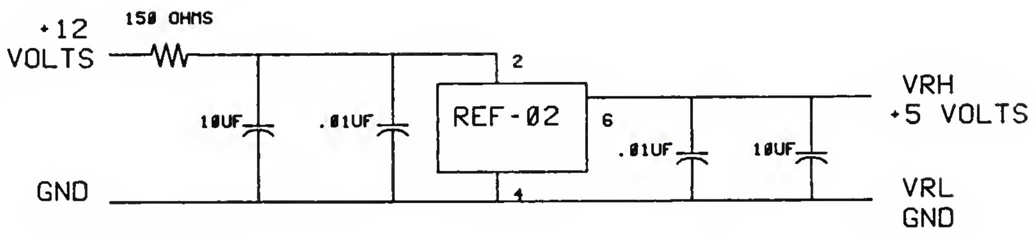

Figure 2-3. Precision Voltage Reference Circuit

3.0 Theory of Operation of the HC11's A/D[1]

The A/D provides ten inputs to the user, of which, eight are analog inputs (AN0 - AN7) with two being dedicated for use as reference voltages (V_{RL} and V_{RH}). The voltage range for V_{RL} and V_{RH} is zero and five volts respectively. Motorola documentation states that the A/D is ratiometric. This implies that an analog input equal to V_{RH} converts to $FF (full scale) and an input equal to V_{RL} converts to $00, with no over or under flow indication.

The A/D is clocked by one of two sources, the HC11's E clock or an internal RC oscillator. With the E clock rate greater than 1 MHz, each A/D conversion is accomplished in thirty two E clock cycles. For E clock rates less than 1 MHz, the A/D is designed to be clocked by the internal RC oscillator enabled by setting a bit (CSEL) in the OPTION register. The RC timer, when enabled, operates at about 1.5 MHz.

In a small period of time, 128 E clock cycles, the A/D can perform four conversions on user specified analog inputs, either AN0 - AN3 or AN4 - AN7. The four conversion results are placed in four A/D Result Registers, ADR1 through ADR4. The first conversion is placed in ADR1, the second in ADR2 and so on. The A/D conversion process is initiated by a write to the A/D Control/Status Register (ADCTL) with valid results in ADR1 in 32 E clock cycles, ADR2 in 64, ADR3 in 96, and ADR4 in 128. Each time a conversion is initiated, the A/D system performs four conversions and then stops or continues depending upon its configuration.

Control of the inputs to the A/D is determined by the configuration of the A/D Control/Status Register (ADCTL). Figure 3-1 displays the ADCTL and its description.

B7	B6	B5	B4	B3	B2	B1	B0
CCF	-	SCAN	MULT	CD	CC	CB	CA

Bit 7,CCF	Conversion Complete Flag - This bit is a read only status indicator that becomes set when all Result Registers contain valid results. When a conversion is initiated, by a write to ADCTL, this bit is cleared automatically and then becomes set when valid results are found in the Result Registers.
Bit 6,	Not implemented. Reads as zero.
Bit 5,SCAN	Continuous Scan Control - With this bit cleared, the A/D performs four conversions and places the results in the Result Registers. When this bit is set, the A/D performs conversions in a round robin fashion with the Result Registers being updated as data becomes available.
Bit 4,MULT	Multiple Channel/Single Channel Control When this bit is cleared, the A/D is configured to perform four consecutive conversions on a single input channel as specified by the four channel bits in the ADCTL, CA through CD (bits 0-3). When this bit is set, the A/D is configured to perform a conversion on each of four channels with each Result Register corresponding to one channel.
Bit 3,CD	Channel Select D
Bit 2,CC	Channel Select C
Bit 1,CB	Channel Select B
Bit 0,CA	Channel Select A - These four bits select one of sixteen possible analog inputs to the A/D. Of these sixteen, only eight are available to the user for external inputs. When the multiple input mode is selected, Bit 4,MULT is set, the two least significant bits, CB and CA have no meaning because a group of four channels are each converted once with their results placed in the Result Registers. Table 3-1 summarizes the input channels selected by the channel select bits.

Figure 3-1. A/D Control/Status Register (ADCTL)

CD	CC	CB	CA	Channel Signal	Result in ADRx if MULT = 1
0	0	0	0	AN0	ADR1
0	0	0	1	AN1	ADR2
0	0	1	0	AN2	ADR3
0	0	1	1	AN3	ADR4
0	1	0	0	AN4	ADR1
0	1	0	1	AN5	ADR2
0	1	1	0	AN6	ADR3
0	1	1	1	AN7	ADR4
1	0	0	0	Reserved	ADR1
1	0	0	1	Reserved	ADR2
1	0	1	0	Reserved	ADR3
1	0	1	1	Reserved	ADR4
1	1	0	0	V_{RH} Pin	ADR1
1	1	0	1	V_{RL} Pin	ADR2
1	1	1	0	V_{RH} /2	ADR3
1	1	1	1	Reserved	ADR4

Table 3-1. Analog to Digital Channel Assignments

By analyzing Table 3-1, it appears that the A/D system has sixteen inputs with four control lines. Actually, the A/D does have sixteen analog inputs of which only eight are user inputs. The last four shown in Table 3-1 are internal reference points with the prior four being reserved for future use.

Single Channel Operation

Single channel operation is accomplished by clearing bit 4 of the ADCTL. This configuration causes the A/D to perform four conversions of a single input channel selected by the four Channel Select bits (CD - CA) and place the results in the four Results Registers.

Multiple Channel Operation

Multiple channel operation is accomplished by setting bit 4 of the ADCTL. This configuration causes the A/D to perform four conversions of the group of four input channels selected by the Channel Select bits CD and CC. In this configuration the Channel Select bits CB and CA have no meaning.

Scan Control

The Scan configuration refers to how many A/D conversions are performed after a write to the ADCTL. By clearing bit 5 of the ADCTL, the A/D is configured to perform four conversions and then stop all activity. With bit 5 set, the A/D performs conversions continually with new conversion results being placed in the Result Registers as they become available.

Using the A/D

To use the A/D converter, it must be supplied power. This power up procedure is accomplished by setting bit 7 of the OPTION Register. To set up a mode of operation for the A/D and to initiate a conversion, a write to the ADCTL is necessary. After a period of time, two methods are possible to ensure that valid results are found in the Result Registers. If E clock frequencies are greater than 1 MHz and the RC oscillator is not enabled, the user can just execute a delay loop until 128 E clock cycles have passed. This, according to Motorola specifications, is the time required for all Result Registers to contain valid results. Also, once the conversion is initiated, a loop that checks the CCF bit and exits on its high state will ensure valid conversion results exist. If the A/D is clocked by the RC oscillator, regardless of E clock frequency, the method for checking for valid results is the bit test of the CCF or with a very long delay loop (> 128 microseconds).

4.0 Static Testing of the HC11's A/D

The system configuration used to test the HC11's A/D with static inputs is shown in figure 4-1. The basic procedure for most static testing methods has the HP9845B tell the 18 bit DAC to setup a constant output voltage to the interface board, have the DVM measure the voltage and return the result, and finally tell the HC11 to perform an A/D conversion and return the result. This loop continues until the amount of data desired is collected. Finally, the calculation of errors is performed on the HP9845B and then plotted if desired. The software for all three static testing methods for the HP9845B and HC11 are shown in Appendix A.

Testing Procedure

Three preliminary testing methods for static analog input conditions were developed and used in obtaining data presented in this thesis. These procedures were repeated in different operational modes of the HC11's A/D and at different E clock frequencies.

A mode of operation is defined as the byte, in Hex, which is written to the ADCTL to initiate an A/D conversion. Four mode combinations were used in collecting data for this thesis. The modes are:

four-conversion, single input, input channel 1 (01),
four-conversion, multiple input (10),
continuous-conversion, single input, input channel 1 (21),
continuous-conversion, multiple input (31).

For example, a four-conversion, single input, input

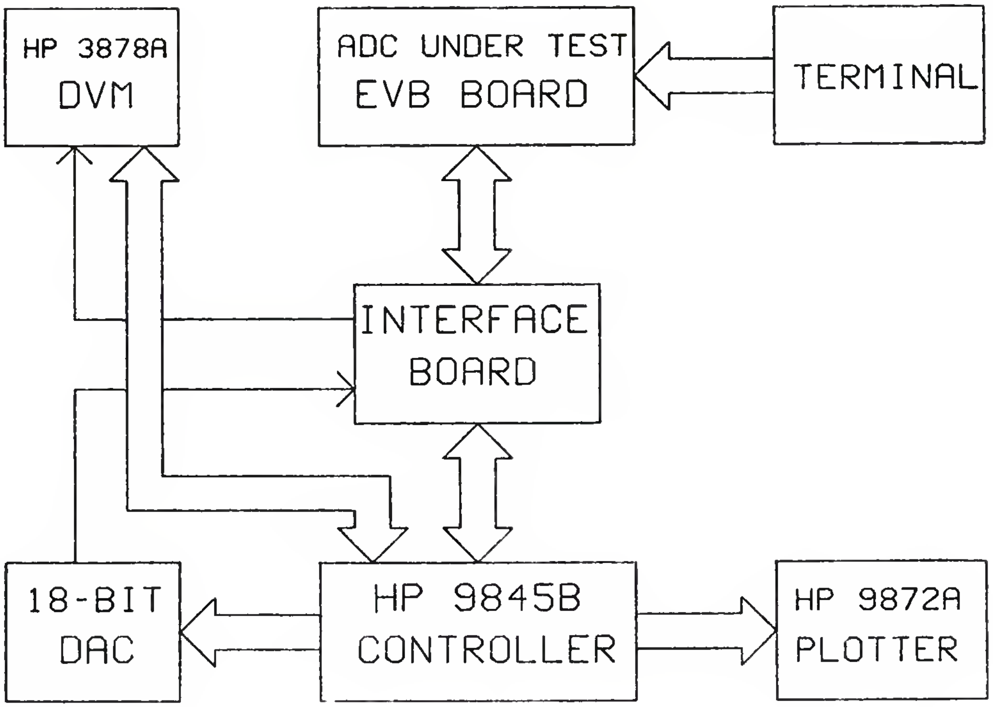

Figure 4-1. Static Testing System

channel 1 (01) mode, configures the A/D system to convert the analog signal found at AN1 four times and place the results in the Result Registers (ADR1 - ADR4). With four-conversion, multiple input (10), as the mode, the A/D system converts the analog signal at AN0 with the result being placed in ADR1, the conversion of AN1 in ADR2, AN2 in ADR3, and AN3 in ADR4. The two other modes used in testing, continuous-conversion, single input, input channel 1 (21) and continuous-conversion, multiple input (31), are similar to the 01 and 10 modes except that the A/D is configured to perform continuous conversions. Note that when multiple inputs are configured, the input channel is not specified in the description. These different modes are each used in the three methods used in static testing the A/D.

Method 1

This method uses a user specified mode for the HC11's A/D. The HC11 receives a signal from the HP, starts a conversion, waits until the CCF bit is set and then outputs the four Result Registers through the interface board to the HP for display purposes. The input signal used for this test was V_{RH}, +5 volts. The HP reads the DVM, signals the HC11, and then reads in four conversion results which are then displayed on the screen along with the DVM reading. This test continues until the user aborts it.

Method 2

This method is very similar to Method 1, except for the testing of the CCF. This method executes a long delay loop after a conversion is initiated. When the delay loop is completed, the HC11 transfers the contents of the Result Registers to the HP for screen display.

Method 3

This method consists of using a static histogram testing procedure developed by Doerfler.[2] This method uses the HP9845B to control an 18 bit Digital to Analog Converter (DAC), a precision digital voltage meter (DVM) along with receiving data from the HC11. The basic theory of this method is to increment the 18 bit DAC by one fourth of an HC11 least significant bit (LSB). Which is:

$$LSB = (V_{RH} - V_{RL}) / 2^{**}Resolution$$

where: V_{RH} is the high reference voltage
V_{RL} is the low reference voltage
Resolution is the number of bits of the A/D converter.

For testing methods used in collecting data, an LSB for the HC11 equals 19.53 mV.

At each step of the 18 bit DAC, ten A/D conversions are performed with the conversion results used to construct a histogram. From this histogram, differential and integral non-linearity errors are estimated. Along with the histogram data, DVM readings are taken at each step to be used to calculate total errors. Offset and gain errors are calculated by a method that uses end point transitions.[3] These transitions are used to calculate the slope of the transfer function of the A/D. Also, this slope is used to adjust the DVM readings to remove gain and offset errors. Using the adjusted DVM readings, integral non-linearity errors are then calculated at the transition points of the A/D. Further explanation of this end point transition method is provided in Appendix C. Finally, a search of the data files is performed to find missing codes and non-monotonic behavior.

Of the two possible methods for checking for valid data in the Results Registers, checking the CCF bit and executing a sufficiently long delay, the data produced for this thesis for method 3 uses the check of the CCF bit to indicate when a conversion is complete.

5.0 Static Testing Results

Exhaustive static testing of all the HC11s acquired has not been completed at this time, but several HC11s from different lots, from the mask B96D, have undergone the tests previously described. In the analysis that follows, errors of the HC11's A/D will usually be described in terms of an LSB.

Method 1

This method has the HC11 initiate an A/D conversion, checks for the high state of the CCF bit, and then outputs the contents of the Result Registers to the HP for display. The E clock frequency for this method was 2 MHz and the input voltage was V_{RH}, +5 volts. Four HC11s, provided by Motorola, were tested, and each produced similar results.

With the RC timer disabled, using the two, four-conversion modes (01 and 10), Result Register three showed errors in the range of 20 to 60 LSBs with the other Result Registers having the expected result of 255. The transfer functions of the A/D reading from the four Result Registers of the four-conversion, single input, input channel 1 (01) mode are shown in Figure 5-1 for a 2 MHz E clock and Figure 5-2 for a 500 kHz E clock. Readings from Result Register 3 from four other HC11s are shown in Figure 5-3. These figures clearly show the error in Result Register 3 in the form of a D.C. offset in the

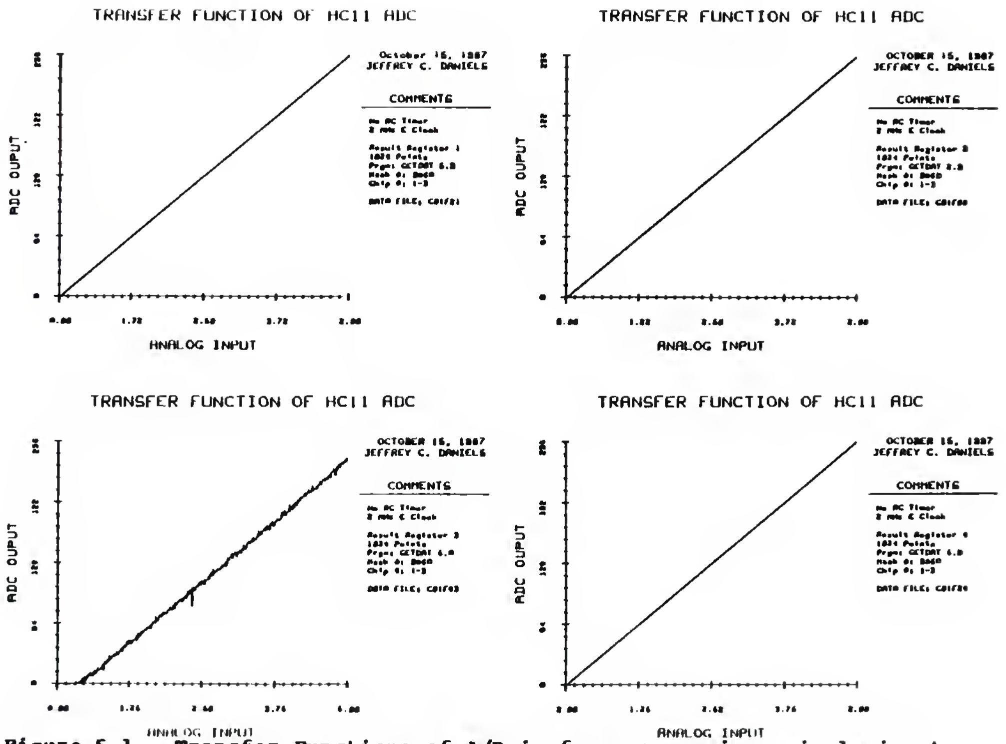

Figure 5-1. Transfer Functions of A/D in four conversion, single input, input channel 1. Checking state of CCF bit

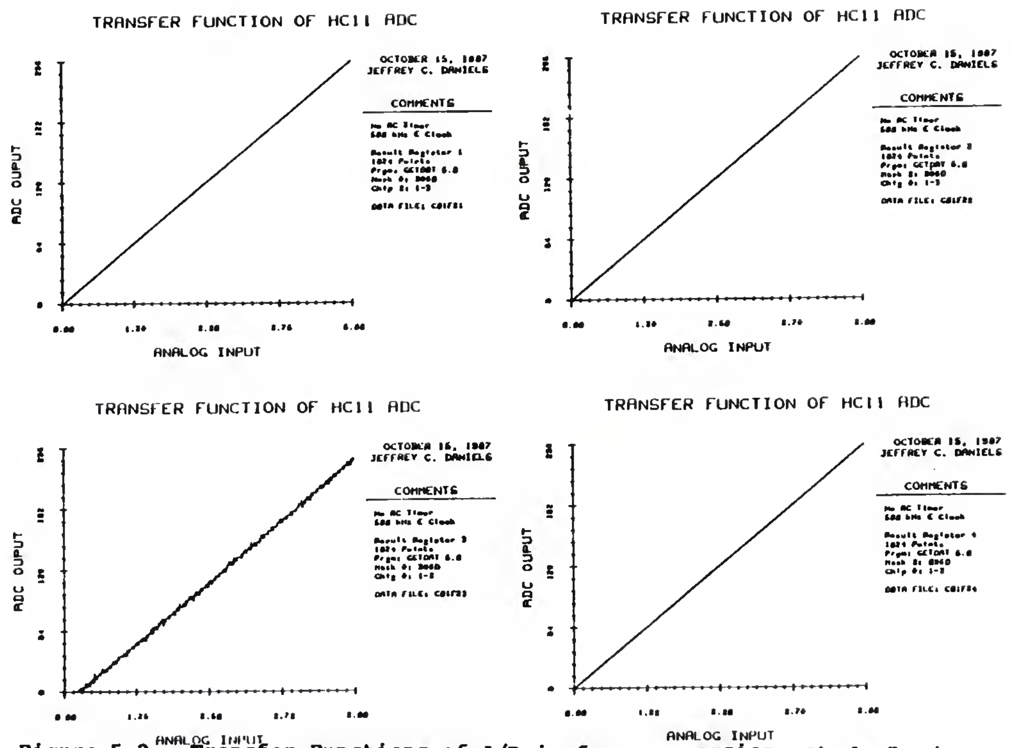

Figure 5-2. Transfer Functions of A/D in four conversion, Single Input input channel 1. Checking state of CCF bit

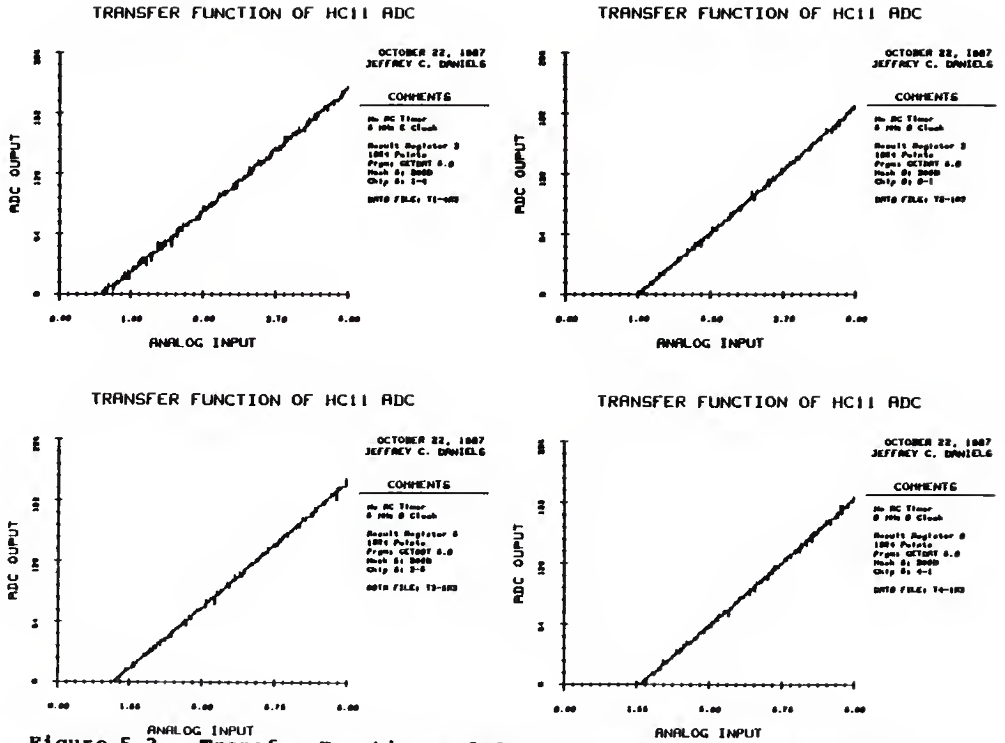

Figure 5-3. Transfer Functions of four HC11s in four conversion, single input, input channel 1. Checking state of CCF

input channel when the analog input is sampled for the conversion for Result Register 3 which varies for different HC11s. Also, as seen from comparing Figures 5-1 and 5-2, the D.C. offset becomes slightly smaller for lower E clock frequencies.

Continuous-conversion modes (21 and 31), had a solid 255 for Result Registers 1 and 3 but 2 and 4 displayed an erratic nature between 255 and a result ranging from 33 to 55 LSBs lower depending on which chip was tested. Also, there appeared to be no correlation in the errors between 2 and 4.

With the RC timer enabled, every mode tested displayed at least 1 or 2 LSBs of noise on all Result Registers. Large noise spikes were present but occurred only every few seconds. The Result Registers with the large noise spikes varied with different HC11s.

Method 2

This method, similar to method 1, executes a long delay after a conversion is started and then outputs four conversion results to the HP. The length used for the delay was 450 clock cycles. This length, according to specifications is over 3.5 times the length needed for Result Registers to contain valid results. For this method the E clock frequency was 2 MHz and 500 kHz with testing performed on several HC11s.

When the RC timer is disabled, the two, four-conversion modes (01 and 10), produced errors in Result Register 2 ranging from 12 to 72 LSBs depending on which chip was tested with the other Result Registers having the desired result of 255. The transfer functions of the A/D reading from the four Result Registers of the four-conversion, single input, input channel 1 (01) mode are shown in Figure 5-4 for a 2 MHz E clock and a 500 kHz E clock in Figure 5-5. These figures clearly show the error in Result Register 2 in the form of a D.C. offset that decreases slightly with E clock rates in the input channel when the analog input is sampled for the conversion for Result Register 2. In continuous-conversion modes (21 and 31), Result Registers 2 and 4 displayed a toggling action between 255 and a value between 183 and 236 with each HC11 being a different value.

With the RC timer enabled, noise was apparent on all outputs of at least 1 LSB with large noise spikes of up to 80 LSBs occurring occasionally in all modes with each HC11 having different characteristics.

Method 3

This method determines total errors using a ramp input, differential and integral non-linearity errors using a histogram procedure[2], integral non-linearity errors, gain, and offset errors using an end point procedure[3] and also the number of missing codes and occur-

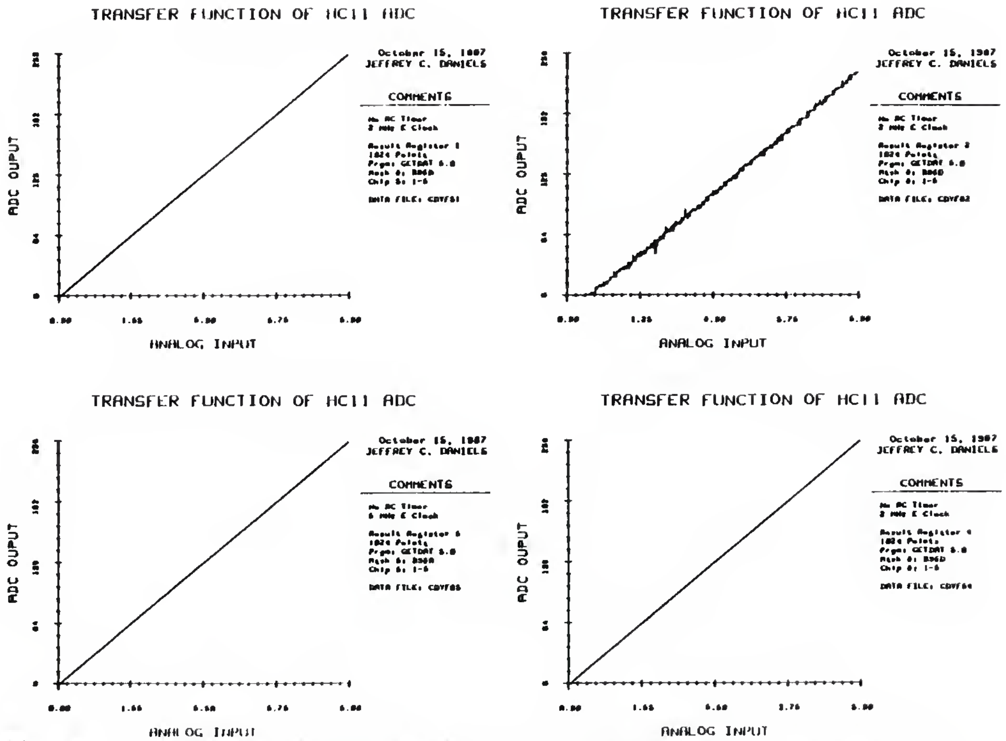

Figure 5-4. Transfer Functions of A/D in four conversion, single input, input channel 1. Executing long delay

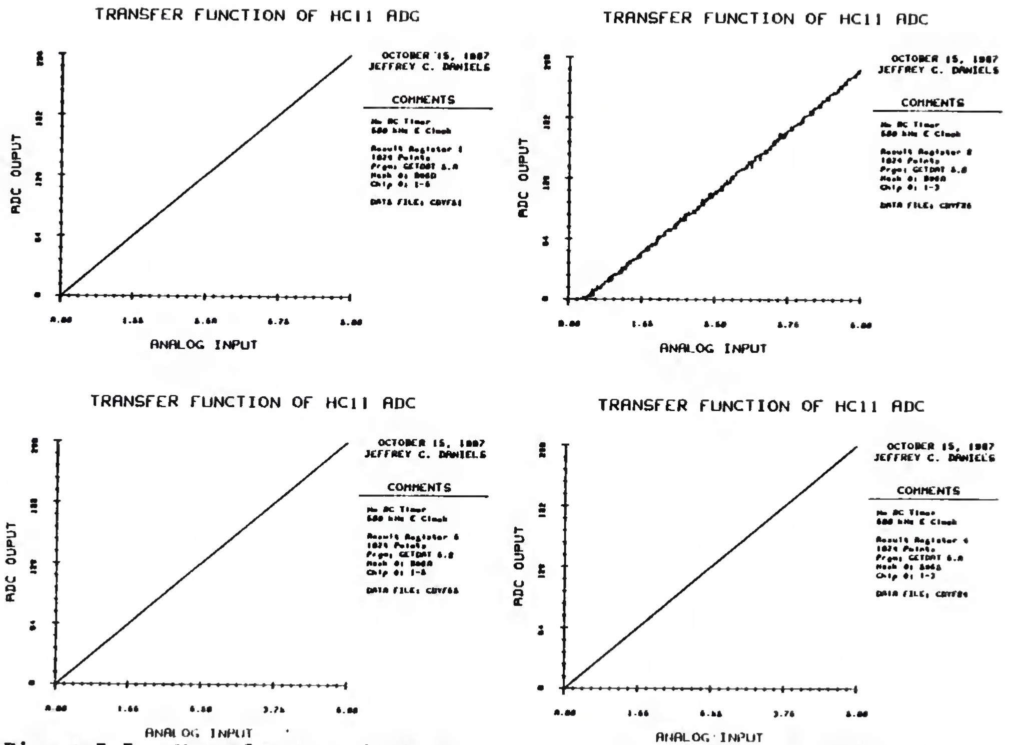

Figure 5-5. Transfer Functions of A/D in four conversion, single input, input channel 1. Executing long delay

rences of non-monotonic behavior. This method was tested on one HC11 with an E clock frequency of 2 MHz and 500 kHz with the RC timer enabled and disabled. Four A/D mode combinations used to test the A/D and produce the plots in Figures 5-6 through 5-21 were:

four-conversion, single input, input channel 1 (01),
four-conversion, multiple input (10),
continuous-conversion, single input, input channel 1 (21),
and continuous-conversion, multiple input (31).

Each figure contains plots of a single A/D mode configuration with the left plots corresponding to 2 MHz E clock rates and the right a frequency of 500 kHz. The only difference in the A/D configuration for top and bottom plots in each figure, is the RC timer is enabled in bottom plots and disabled in top plots.

Total errors for the output of the A/D are shown in figures 5-6, 5-7, 5-8, and 5-9. The top plots (RC timer disabled) in these figures are all very similar in showing an offset of approximately -1 LSBs. The bottom plots (RC timer enabled) show an extreme presence of conversion result errors caused by the RC timer. These errors have magnitudes in excess of + or - 4 LSBs in some instances. The left plots, an E clock rate of 2 MHz, display larger total errors than the right plots, an E clock rate of 500 kHz. These factors imply that the magnitude of total errors has a strong correlation with increasing E clock rates and also whether the RC timer is enabled.

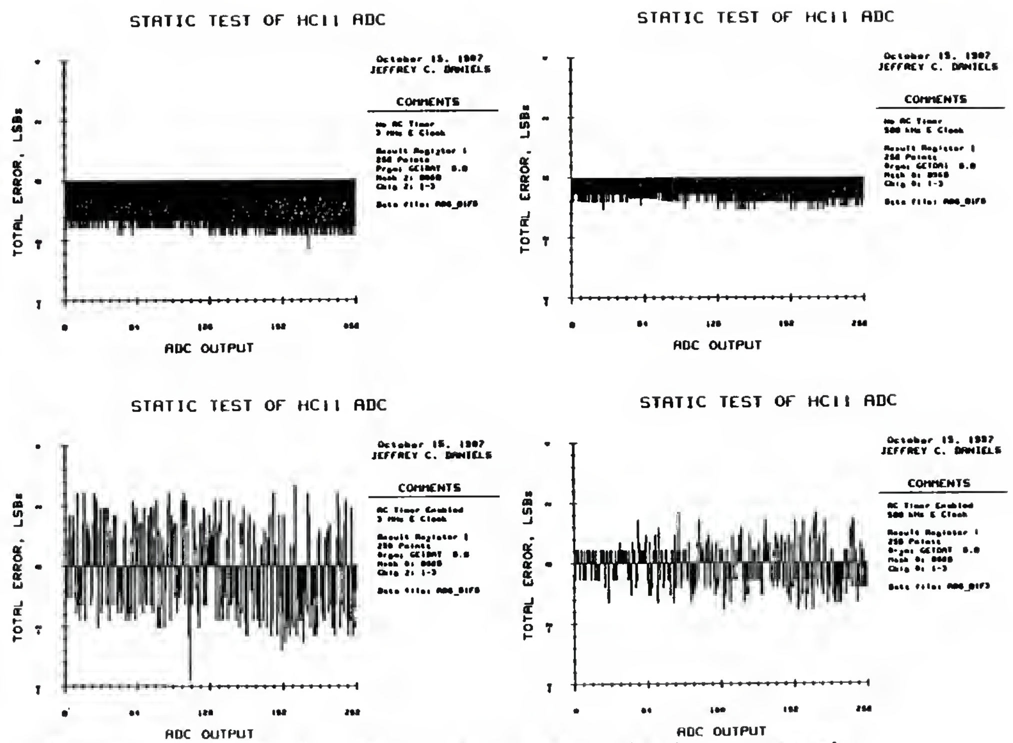

Figure 5-6. Total errors of the A/D using the four conversion, single input, input channel 1 mode.

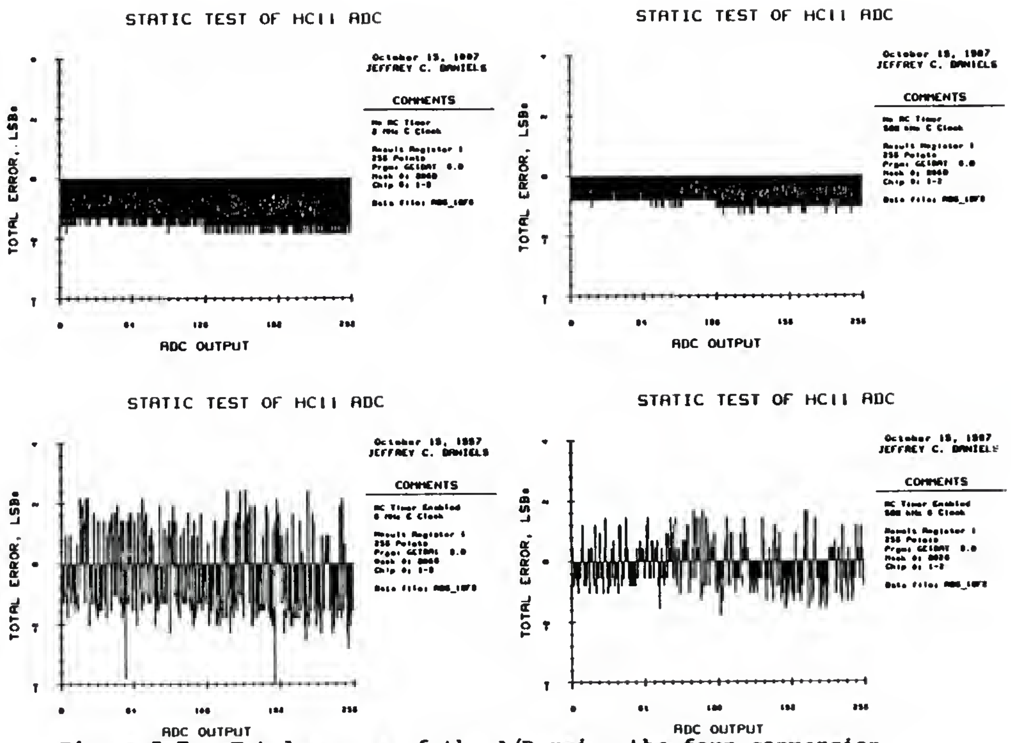

Figure 5-7. Total errors of the A/D using the four conversion, multiple input mode.

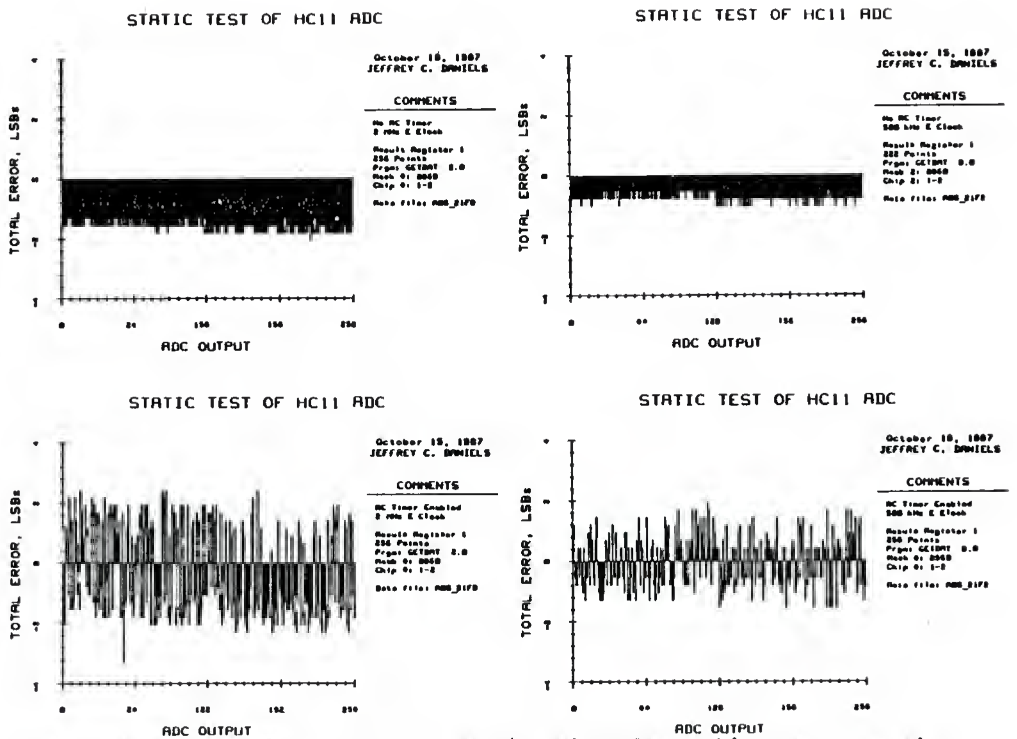

Figure 5-8. Total errors of the A/D using the continuous conversion, single input, input channel 1 mode.

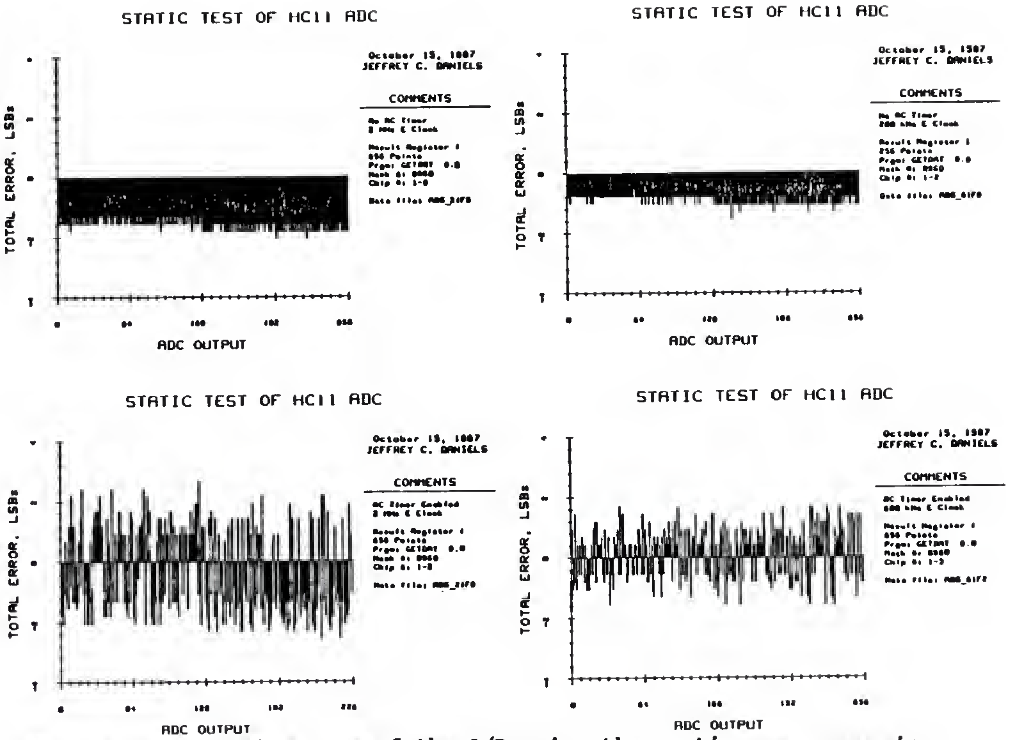

Figure 5-9. Total errors of the A/D using the continuous conversion, multiple mode.

The number of occurrences of non-monotonic behavior was found by searching the total error data files. With the RC timer disabled (top plots), the number of occurrences of non-monotonic behavior was less than three in 1024 points using a ramp input between between 0 and 5 volts. With the RC timer enabled (bottom plots), non-monotonic behavior was apparent well over 300 times in 1024 points between 0 and 5 volts. This increase in non-monotonic behavior with the RC timer enabled over when the RC timer is disabled indicates the RC timer causes errors to occur in conversion results.

Histogram differential and integral non-linearity errors are shown in Figures 5-10 through 5-17. The top plots (RC timer disabled), the bottom plots (RC timer enabled), the left plots (2 MHz E clock), and the right plots (500 kHz E clock) show errors that fall well within Motorola specifications with only a slight increase in errors with the RC timer enabled and with higher E clock frequencies. However, it should be noted that the histogram procedure uses many data points in differential and integral non-linearity error calculations. Using many data points causes the noise to be averaged out, thus allowing the test to be of the actual A/D transition points. These eight figures (5-10 through 5-17), show that the A/D has transition points within specifications

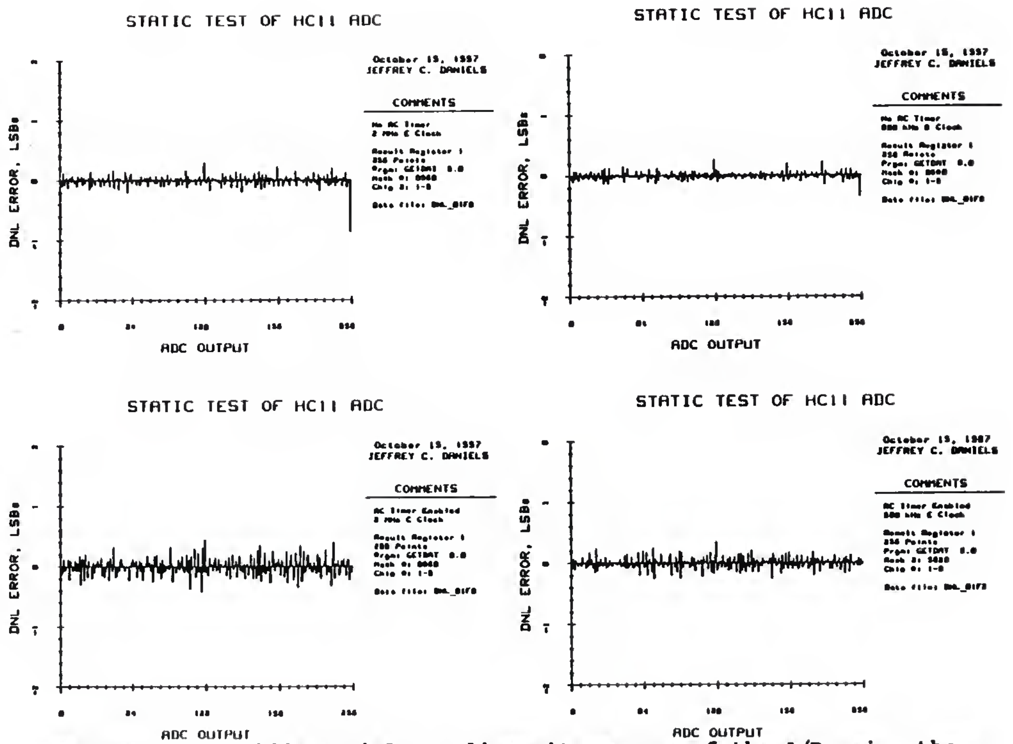

Figure 5-10. Differential non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode.

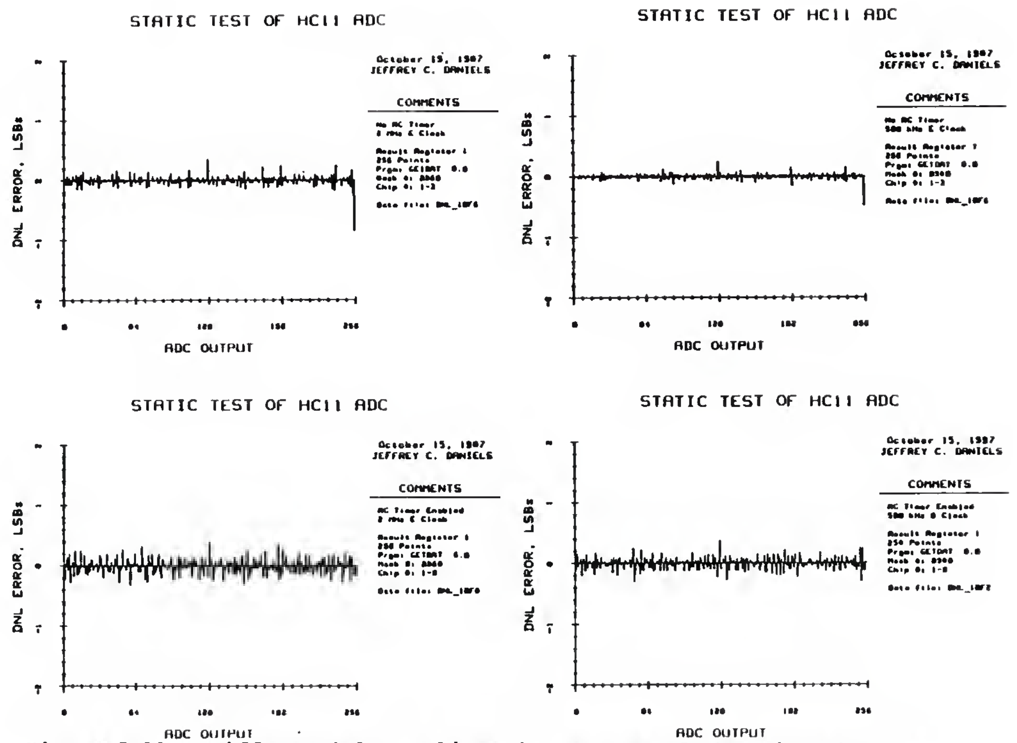

Figure 5-11. Differential non-linearity errors of the A/D using the four conversion, multiple input mode.

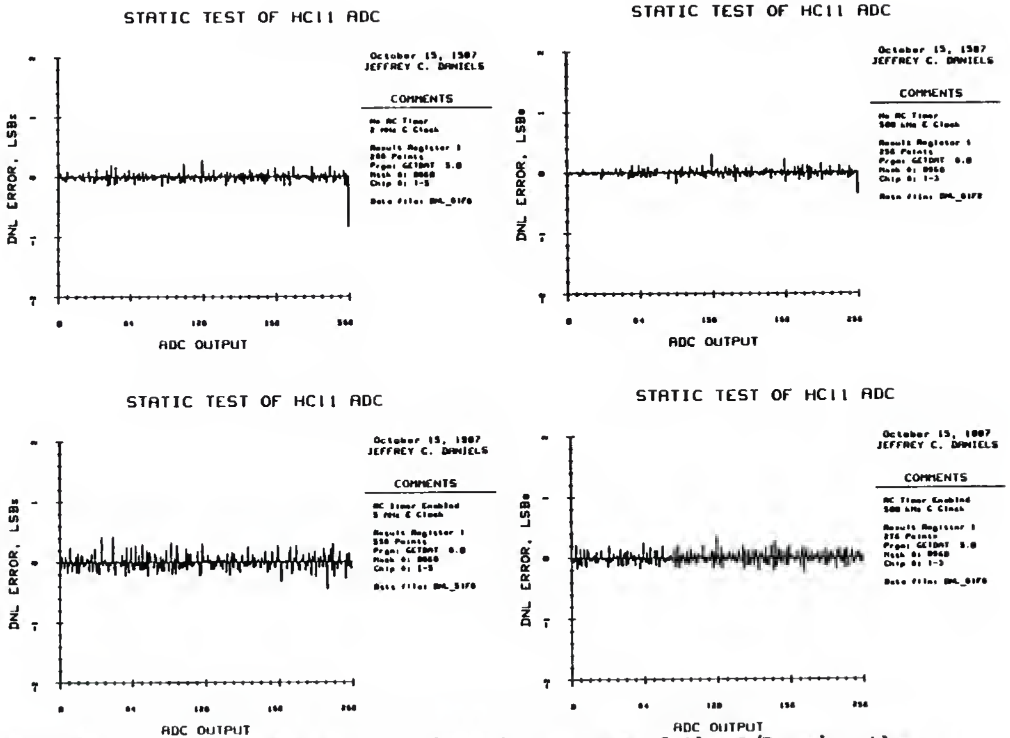

Figure 5-12. Differential non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode.

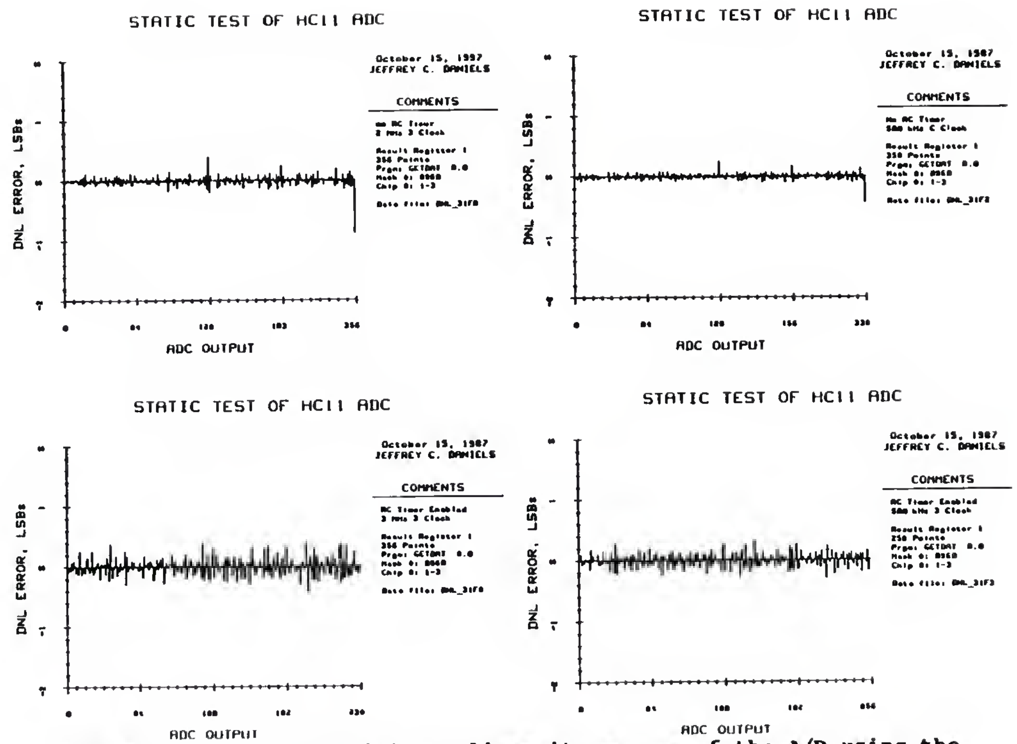

Figure 5-13. Differential non-linearity errors of the A/D using the continuous conversion, multiple mode.

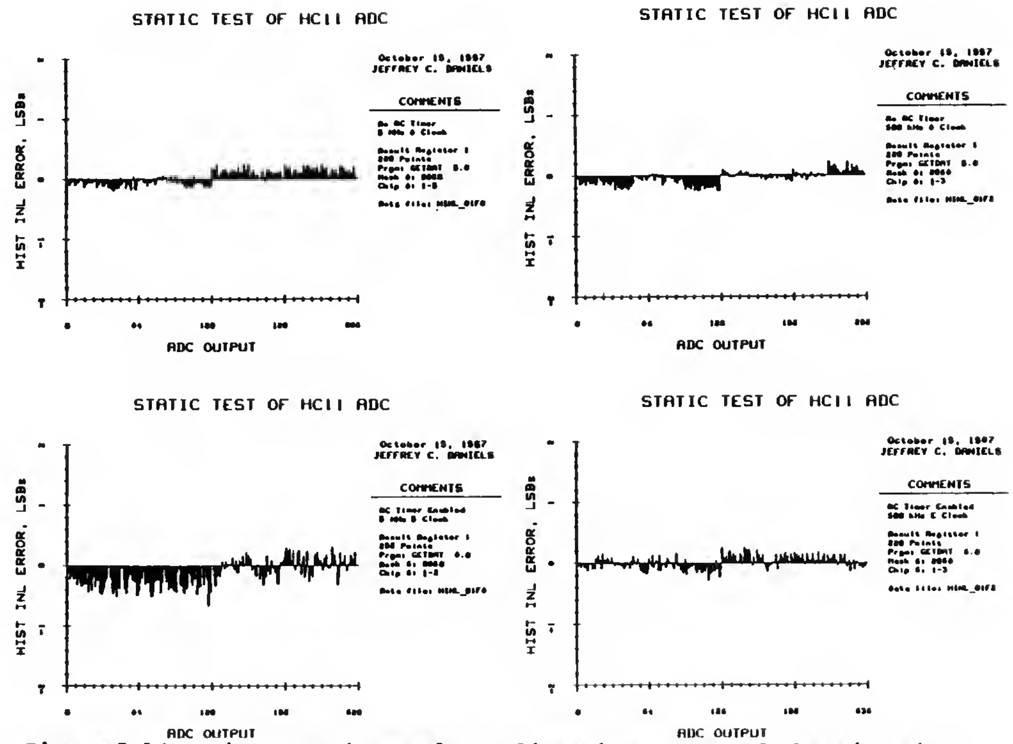

Figure 5-14. Histogram integral non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode.

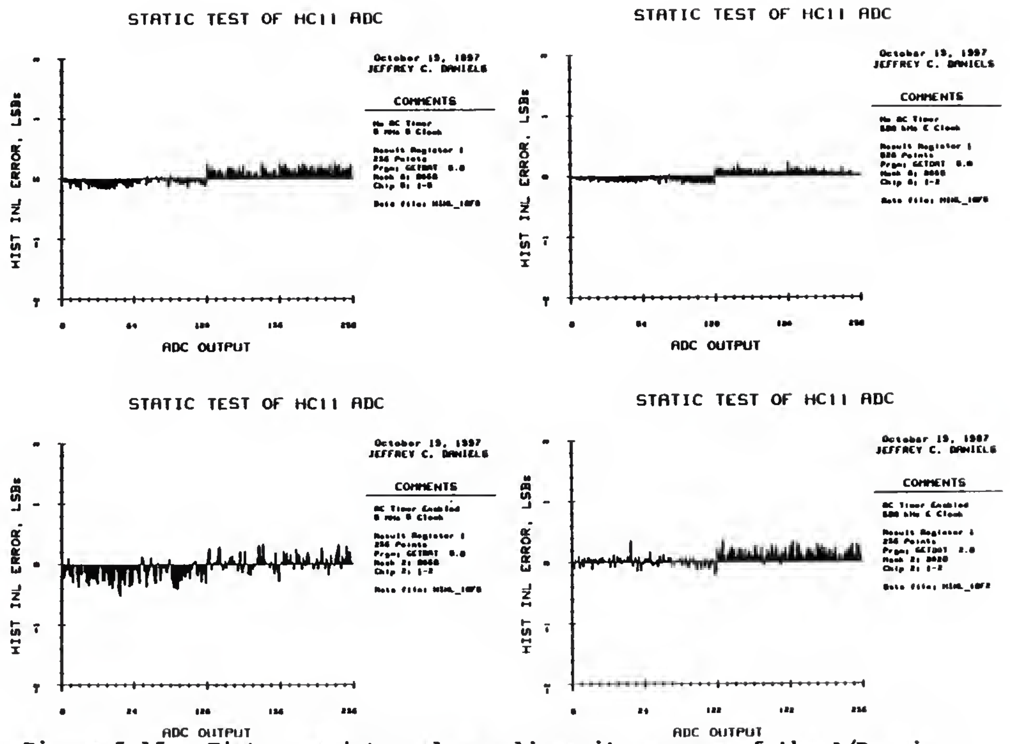

Figure 5-15. Histogram integral non-linearity errors of the A/D using the four conversion, multiple input mode.

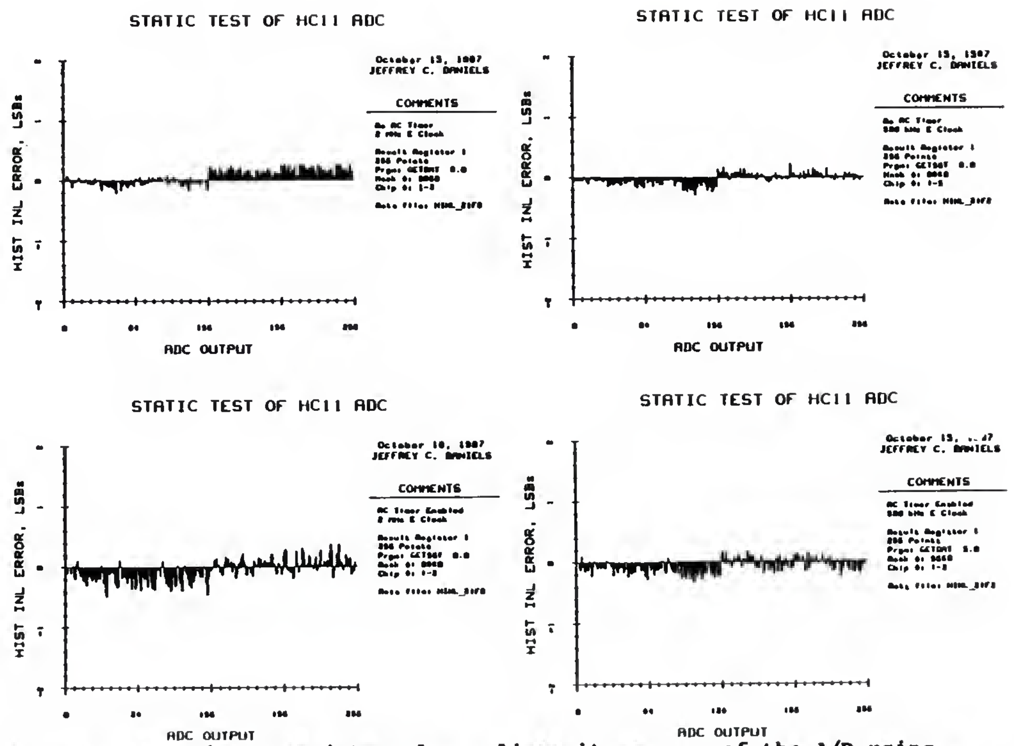

Figure 5-16. Histogram integral non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode.

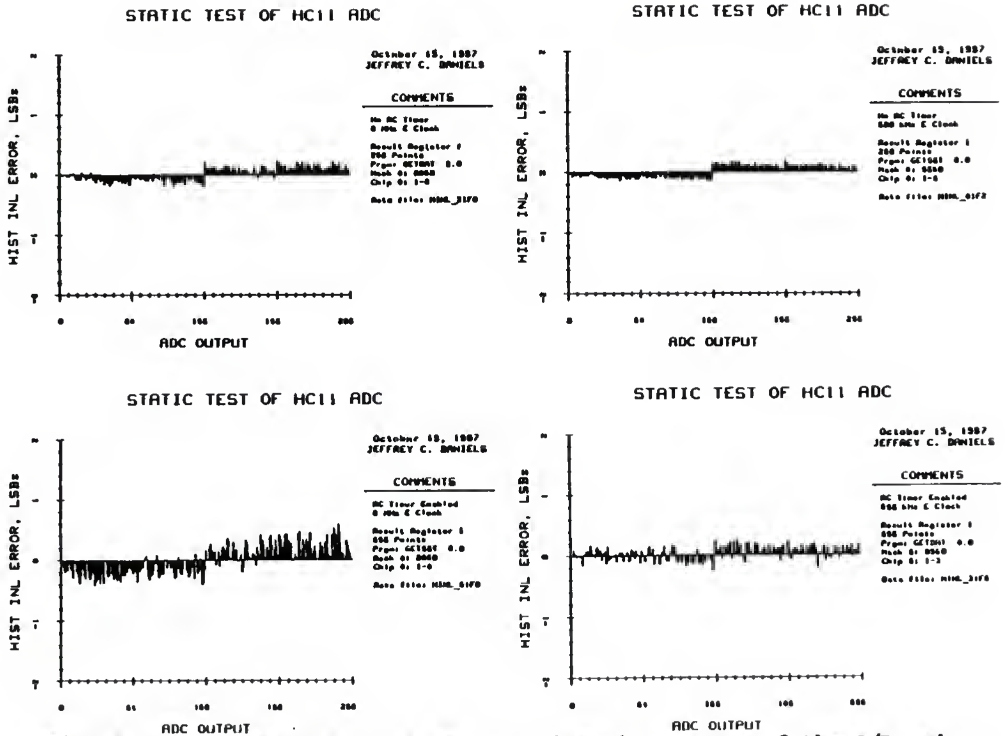

Figure 5-17. Histogram integral non-linearity errors of the A/D using the continuous conversion, multiple mode.

but these tests show no presence of errors found in other static tests.

No missing codes, checked by searching the histogram data for a bin count of zero, were found in any A/D configuration modes regardless of whether the RC timer was enabled or disabled.

The end point integral non-linearity errors are shown in Figures 5-18 through 5-21. With the RC timer disabled (top plots), these errors fell well within specifications but it must be noted that gain and offset errors had been removed prior to error calculation. Again, a slight decrease in errors is seen with a decrease in frequency. Due to reasons discussed in Appendix C, Data for end point integral non-linearity with the RC timer enabled was not taken at this time.

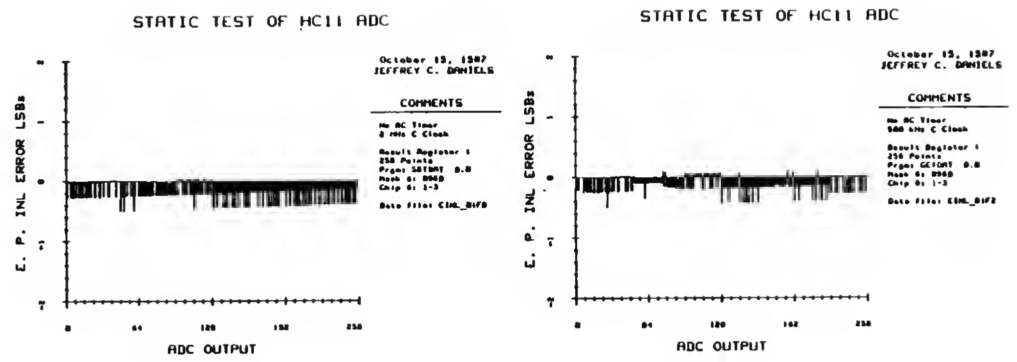

Figure 5-18. End Point integral non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode.

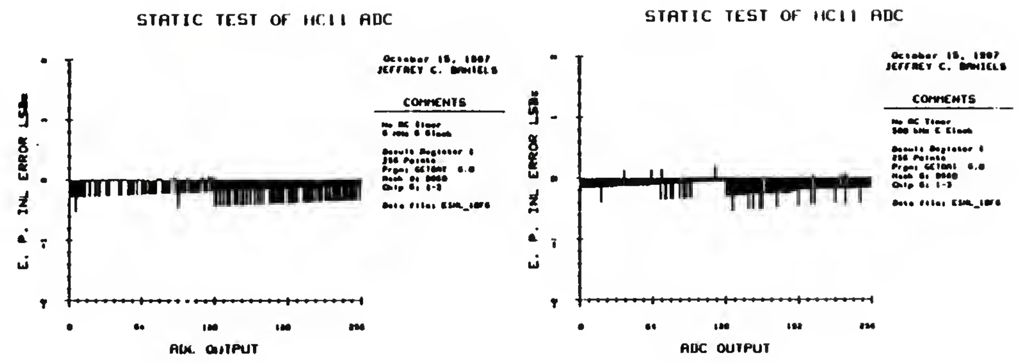

Figure 5-19. End Point integral non-linearity errors of the A/D using the four conversion, multiple input mode.

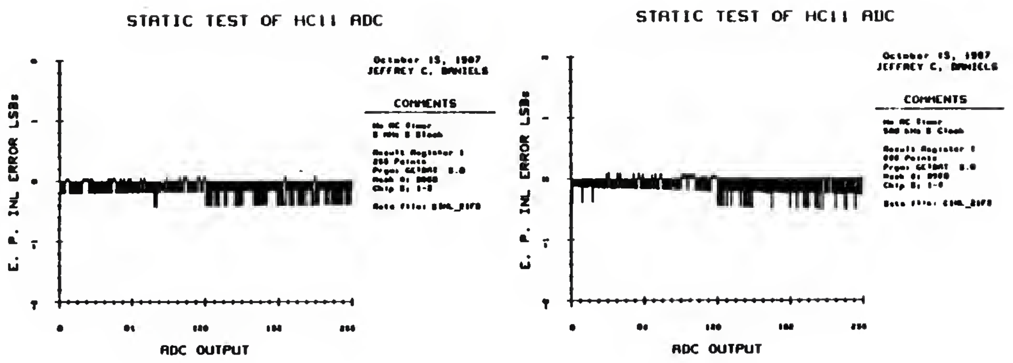

Figure 5-20. End Point integral non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode.

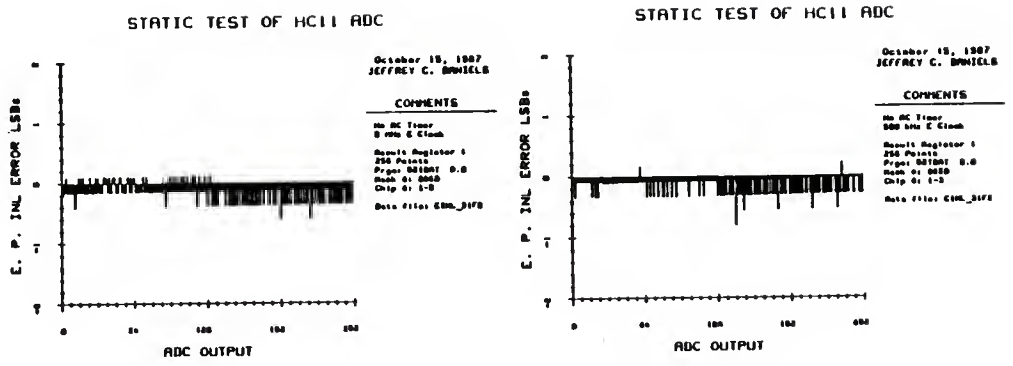

Figure 5-21. End Point integral non-linearity errors of the A/D using the continuous conversion, multiple mode.

Conclusion

Method 1, checking the CCF bit for the end of conversion, led to the discovery of large offset error in the results obtained from Result Register 3. Method 2, executing a large delay loop to allow the A/D conversions to be completed, displayed a large offset in Result Register 2. The magnitude of the offset in Methods 1 and 2 was different in each HC11 tested and decreased as the frequency was decreased. This pattern sensitivity can be verified by using two programs that are provided in Appendix A. These programs perform a quick check of conversion results with a logic probe or logic analyzer. They were written to eliminate the need for an elaborate testing setup for quick testing of the A/D. Using Method 3, total error calculations indicated the constant presence of small offsets, -1 LSB, and extreme conversion result errors when the RC timer was enabled. Also, Method 3's histogram procedure, which averages out noise, yielded results showing that the transition point errors of the A/D were well within Motorola specifications.

6.0 Dynamic Testing of the HC11's A/D

Testing an A/D with varying, dynamic, inputs is an excellent way to help characterize an A/D's actual performance in real world situations. The system configuration used to test the HC11's A/D with dynamic inputs is shown in figure 6-1. The basic testing procedure for most dynamic testing methods is to have the HC11 perform equally spaced A/D conversions on a precision sine wave. A conversion is performed, the result written to PORTB of the HC11, a pulse is sent to the IBM PCXT and then the loop is repeated. The IBM PCXT waits for a pulse, reads the data and then waits for another pulse. The IBM continues to receive data until the desired number of results is acquired. Finally, missing code existence and errors in the form of integral and differential non-linearity are calculated and then plotted. The software for IBM PCXT and the HC11 are shown in Appendix B.

The precision sine wave used in dynamic testing is generated from an HP3325A function generator. With a signal to noise ratio in excess of 60 dB, the HP3325A provides waveforms with adequate spectral purity when considering the eight bit resolution of the HC11's A/D.

The sampling frequency used in dynamic testing methods was desired to be as fast as the HC11 could perform A/D conversions. The HC11 can theoretically

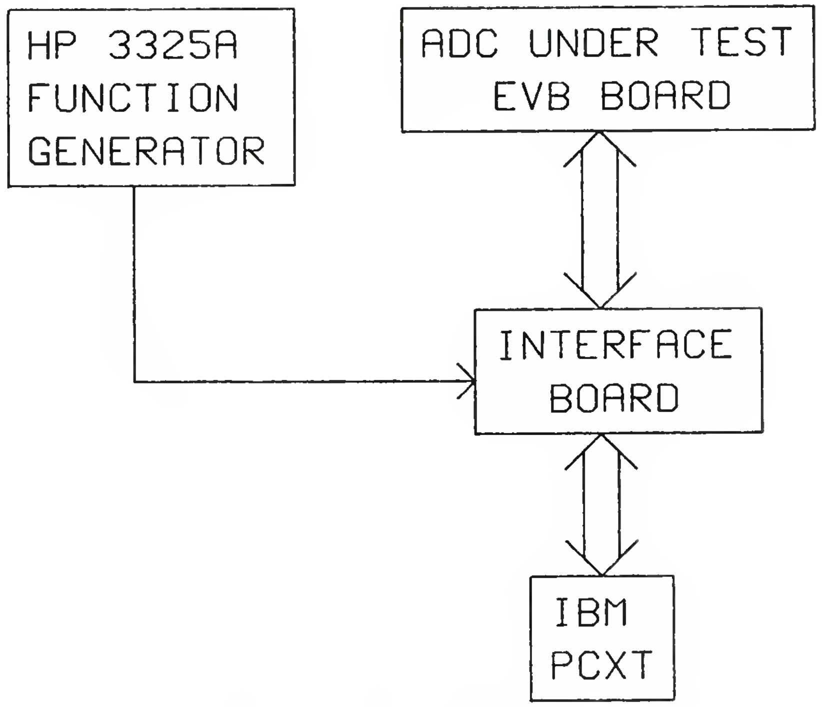

Figure 6-1. Dynamic Testing System.

perform conversions in excess of 62 kilohertz (kHz). However, due to the architecture of the IBM PCXT and its compatibles, the maximum sampling rate without missing results was determined through trial and error to be slightly larger than 4 kHz. Therefore, a sampling rate of 4 kHz was chosen.

The frequency for the sine wave to be sampled was chosen to 1406 Hz because the FFT procedure works best when an integral number of periods of the input signal are present in the number of samples taken.

Testing Procedure

Two testing methods were used to evaluate the dynamic performance of the HC11's A/D. These methods, developed by Doerfler[2], use a histogram procedure to find missing codes and to estimate differential non-linearity errors and a fast fourier transform (FFT) procedure to estimate integral non-linearity errors and to give an indication of overall system noise. These methods were tested on several HC11s at different E clock frequencies and with the A/D in four different modes of operation. These modes are:

four-conversion, single input, input channel 1 (01)
four-conversion, multiple input (10)
continuous-conversion, single input, input channel 1 (21)
and
continuous-conversion, multiple input (31).

Also, each mode of operation and E clock frequency was tested with the RC timer enabled and disabled.

Method 1

Method 1 consists of using a histogram procedure developed by Doerfler[2]. Briefly, this procedure uses a large number of data points to produce an estimate of differential non-linearity errors and to find missing codes. The data points are used to construct a histogram from which a cumulative histogram procedure is used to estimate the A/D's transition points. The transition point estimates now can be used to calculate differential non-linearity errors. Finally, a search of the histogram data files for bins with a count of zero determines where, if any, the missing codes occur.

For this method to produce meaningful estimates of differential non-linearity errors, enough data points must be collected to ensure that a proper distribution is obtained in the histogram. The minimum number of data points, npts, needed for B bit precision and 100(1-a) percent confidence is given by[2]

$$\text{npts} = \frac{Z_{a/2}^{2} \ \text{pi} \ 2^{N-1}}{B^{2}}$$

where $Z_{a/2}$ is found in a standard normal distribution table, and N is the resolution of the A/D converter. For example, for an 8-bit A/D, the number of data points required to estimate the differential non-linearity error to within 0.1 bit with a 99% confidence requires 265600 samples.

Missing codes, found by searching the histogram data files for bins with a count of zero, are determined with the same accuracy and confidence level as differential non-linearity errors.

To properly produce the distribution of a sine wave in the histogram data, a sufficient number of points must be collected and also the sine wave must have the proper amplitude and offset. For the HC11's 8-bit A/D with 0 and +5 volt reference voltages, the input sine wave must have a 2.5 volt d.c. offset and a peak to peak amplitude of slightly larger than the reference voltage (i.e. 5.05 volts). This voltage ensures that all A/D codes have a chance to be exercised including the end bins. Ideally this method can be used to calculate the size of an offset if it is present. This becomes a problem because it requires that the input sine wave be exactly centered about the offset of 2.5 volts used in testing the HC11's A/D. Therefore the exact calculation of the offset present will not be performed but the differences in the offset between different operational modes and E clock frequencies can be observed because all of the data was collected in one period of time.

An important characteristic to keep in mind about this method is that the formation of a histogram causes noise to be averaged out and become non-detectable. With the noise averaged out, the test results of the transition points are free from the influence of noise in the system.

Method 2

Method 2, also developed by Doerfler[2], uses a fast fourier transform (FFT) procedure to estimate integral non-linearity errors and also give an indication to overall system noise. Briefly, this method uses a power of 2, equally spaced data points (4096) of a spectrally pure sine wave. The data points are windowed with a Von Hann window to help eliminate spectral leakage and then an FFT is performed with the resulting spectrum being plotted. Integral non-linearity errors appear as harmonics of the fundamental frequency, the frequency of the input sine wave. These harmonics are aliased into the frequency window which is one half the sampling frequency. Further considerations for this method include an overall indication of overall system noise by raising the noise floor in the FFT output spectrum.

The harmonics of the fundamental frequency are folded back into the frequency window due to aliasing. In order to show exactly where they appear, a software simulation was performed. A 1406.25 Hz sine wave and its first eight harmonics were sampled at 4000 Hz with the amplitude of each harmonic being reduced by 20 dB to make it possible to detect the respective harmonic in frequency window.

The fractional input frequency was chosen for simulation purposes to ensure that the samples produced contained an integral number of periods in the number of points taken. Also, care must be taken in choosing the sampling frequency and the frequency of the input sine wave so that the harmonics are not aliased into the peak of the fundamental frequency, thus becoming undetectable. The results of this simulation are shown in figure 6-2 with the exact frequencies tabulated in table 6-1.

Harmonic	Frequency (in Hertz)
0	1406.25
1	1187.5
2	218.75
3	1625.0
4	968.75
5	437.5
6	1843.75
7	750.0
8	656.25

Table 6-1. Location of Harmonics in the Frequency window

The dynamic range of an N-bit converter is known to be[2]

$$\text{dynamic range} = 20\log_{10}(2^N) = 6.02N \text{ dB.}$$

This equation implies that if the amplitude of the highest harmonic is less than 6.02N, then the integral nonlinearity is less than 1 LSB. Also, the number of bits of integral linearity can be calculated by dividing the amplitude of the highest harmonic by 6.02.

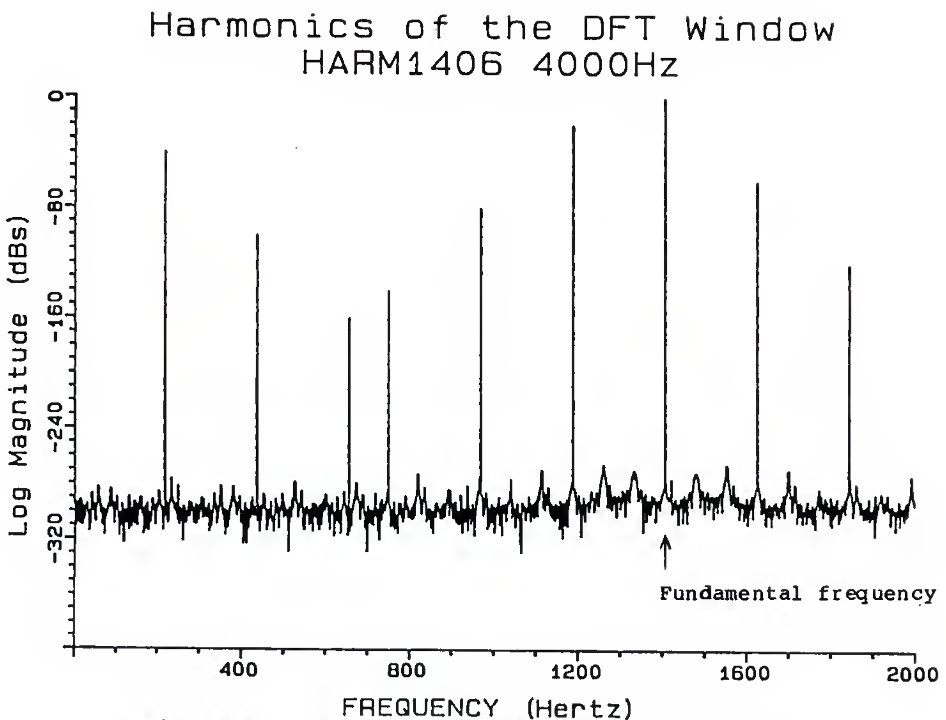

Figure 6-2. **Harmonics of the DFT Window**
1406.25 Hz sine wave sampled at 4000 Hz.

To determine the noise floor of a perfect 8-bit A/D sampling a 1406.25 sine wave at 4000 Hz, another software simulation was performed. The spectrum produced is shown in figure 6-3 and will be used in the analysis of this method's results obtained from the HC11's A/D.

The amplitude of the input sine wave for testing using Method 2 was a peak to peak voltage of 4.8 volts with a offset of 2.5 volts. The 4.8 volt input was used to eliminate the possibility of clipping which causes frequencies to appear in the output spectrum that actually do not exist.

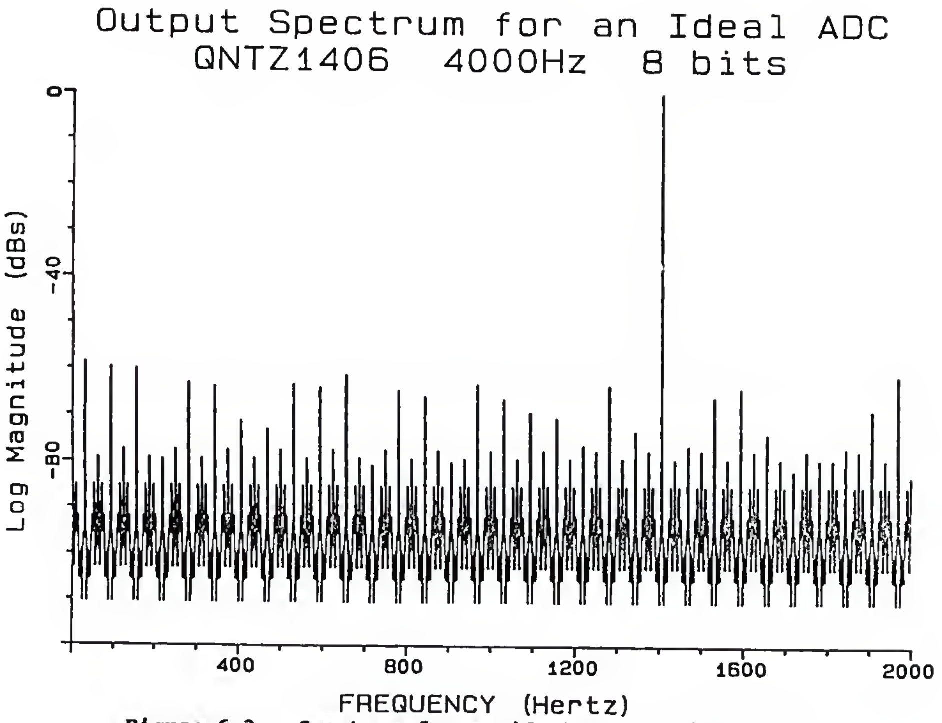

Figure 6-3. Spectrum for an ideal 8-bit A/D
1406.25 Hz sine wave sampled at 4000 Hz.

7.0 Dynamic Testing Results of the HC11's A/D

Due to time limitations, the dynamic testing of all HC11s acquired has not been completed at this time but one HC11 with the number 1-3 has been tested with the two methods described in chapter 6 in all possible modes and E clock frequencies both with, and without the RC timer.

Method 1

Method 1, consisting of constructing a histogram using many data points to find missing codes and to produce a cumulative histogram to calculate differential non-linearity errors at the A/D's transition points was performed on one HC11 at four A/D operational modes and at two E clock frequencies. The plots for these tests are shown in figures 7-1 to 7-8 with the top plots from data with a 2 MHz E clock and the bottom plots from a 500kHz E clock.

For all different operational modes of the HC11's A/D the Histogram Data plots with the RC timer disabled with a 2 MHz E clock frequency indicates the presence of a small offset because the number of data points in the last bin in the histogram is very small in comparison to other plots with the RC timer enabled at a 2 MHz E clock (top plots) and at an E clock frequency of 500 kHz (bottom plots).

Differential non-linearity errors are well within Motorola specifications in most combinations of

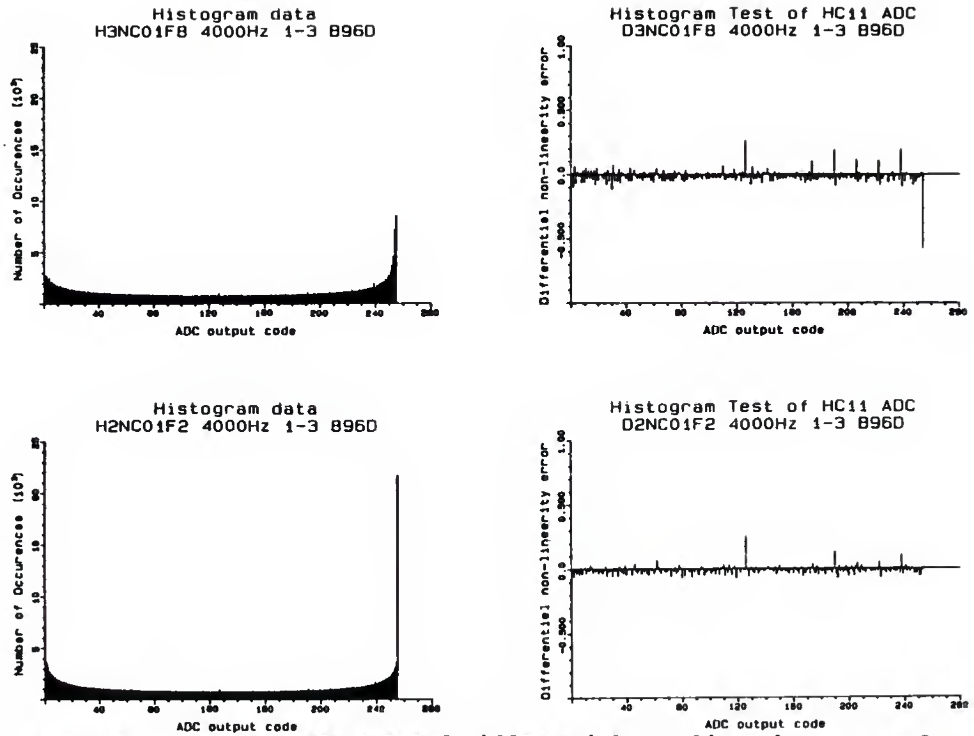

Figure 7-1. Histogram data and Differential non-linearity errors for A/D. Four conversion, single input, input channel 1. RC timer disabled.

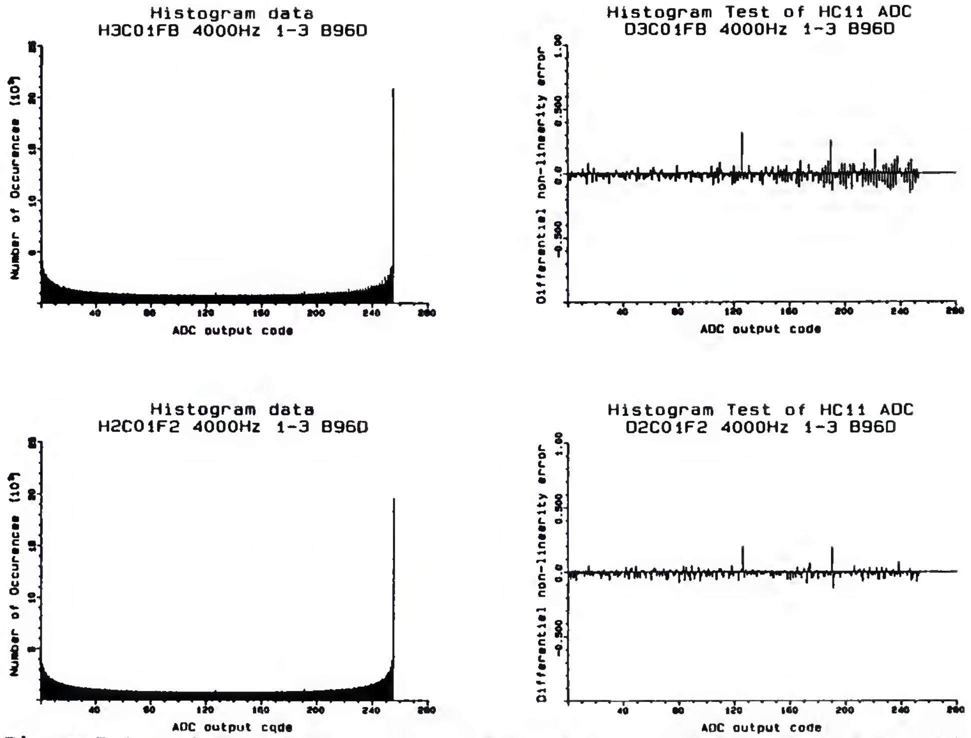

Figure 7-2. Histogram data and Differential non-linearity errors for A/D. Four conversion, single input, input channel 1. RC timer enabled.

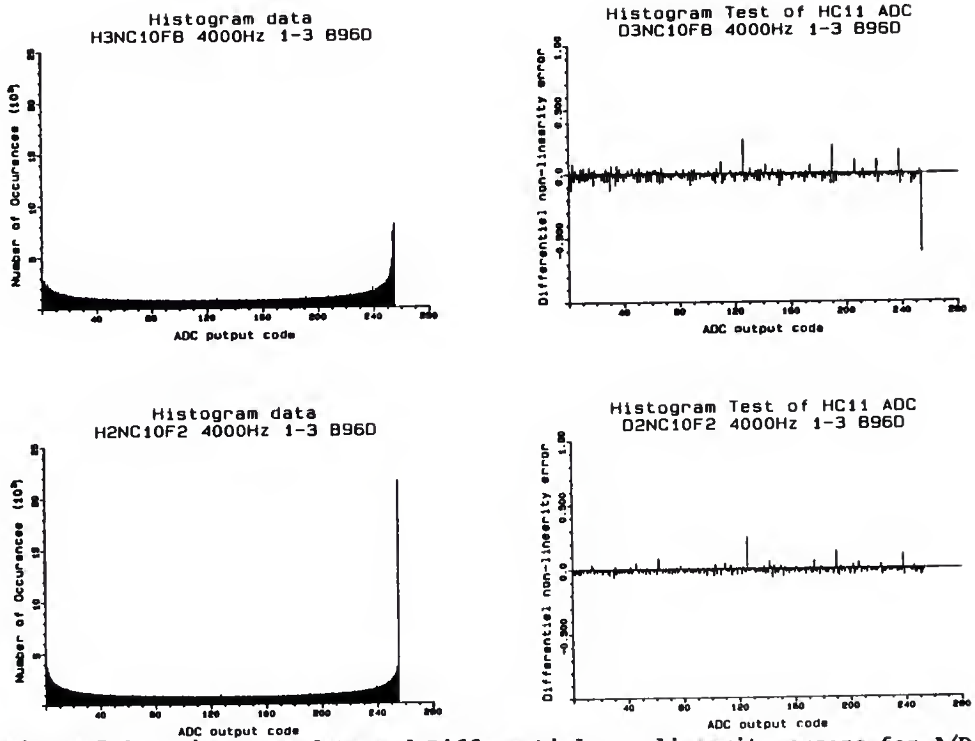

Figure 7-3. Histogram data and Differential non-linearity errors for A/D. Four conversion, multiple input. RC timer disabled.

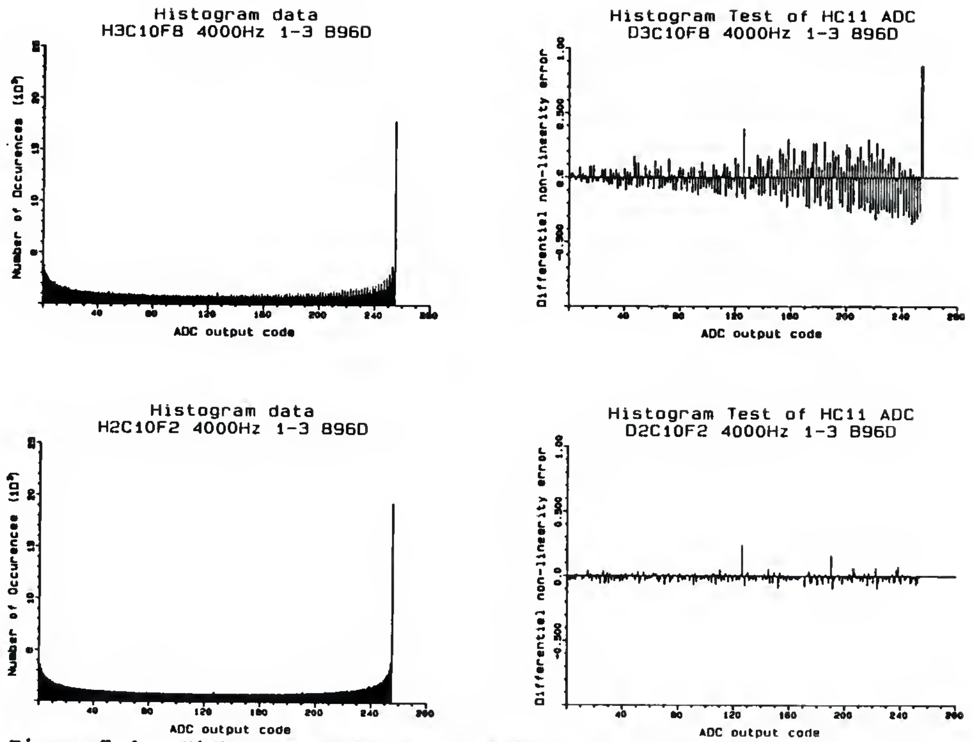

Figure 7-4. Histogram data and Differential non-linearity errors for A/D. Four conversion, multiple input. RC timer enabled.

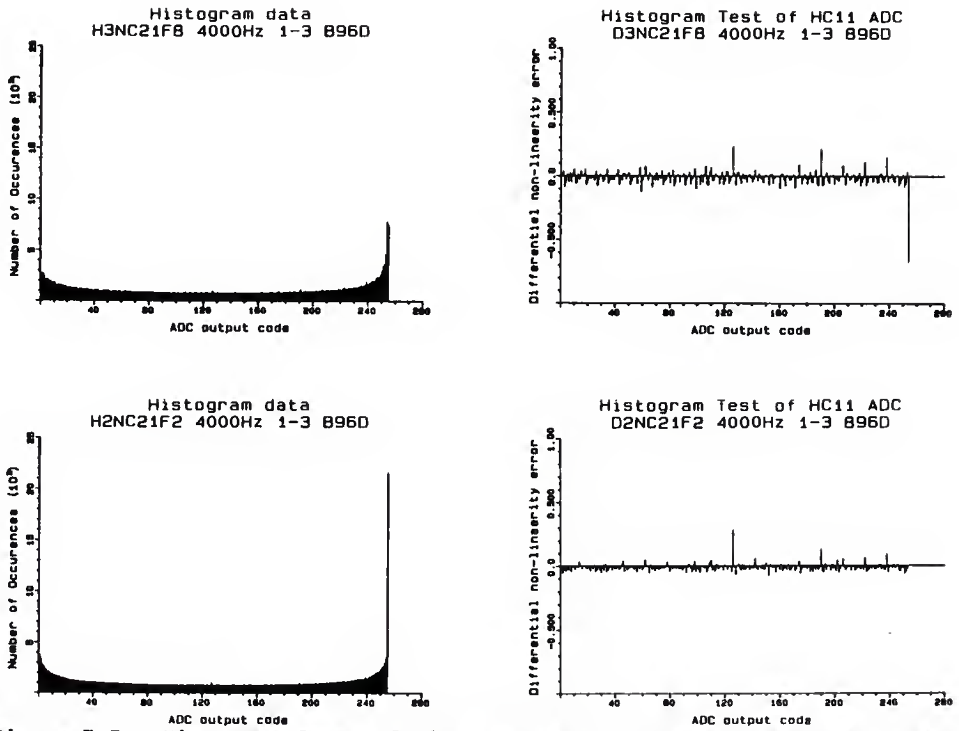

Figure 7-5. Histogram data and Differential non-linearity errors for A/D. Continuous-conversion, single input, input channel 1. RC timer disabled.

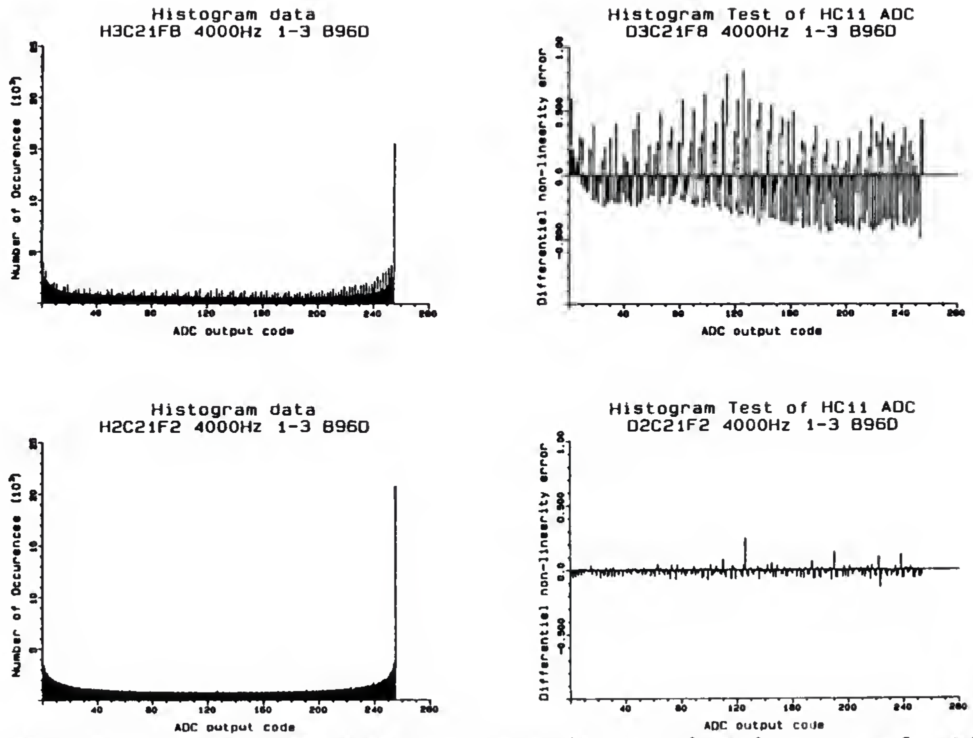

Figure 7-6. Histogram data and Differential non-linearity errors for A/D. Continuous-conversion, single input, input channel 1. RC timer enabled.

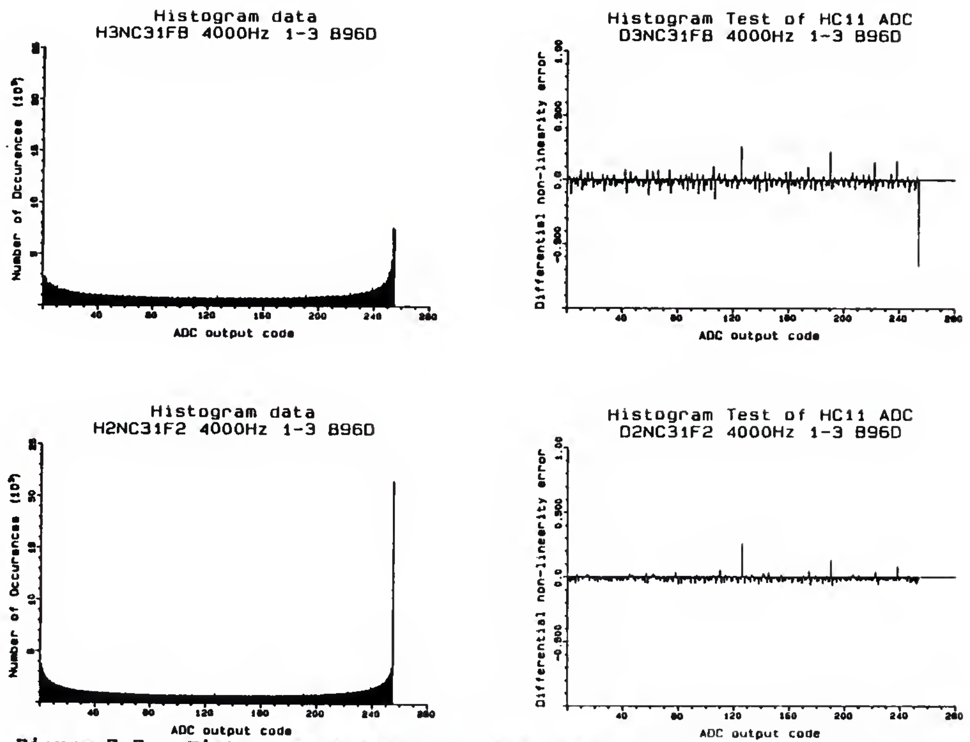

Figure 7-7. **Histogram data and Differential non-linearity errors for A/D. Continuous-conversion, multiple input. RC timer disabled.**

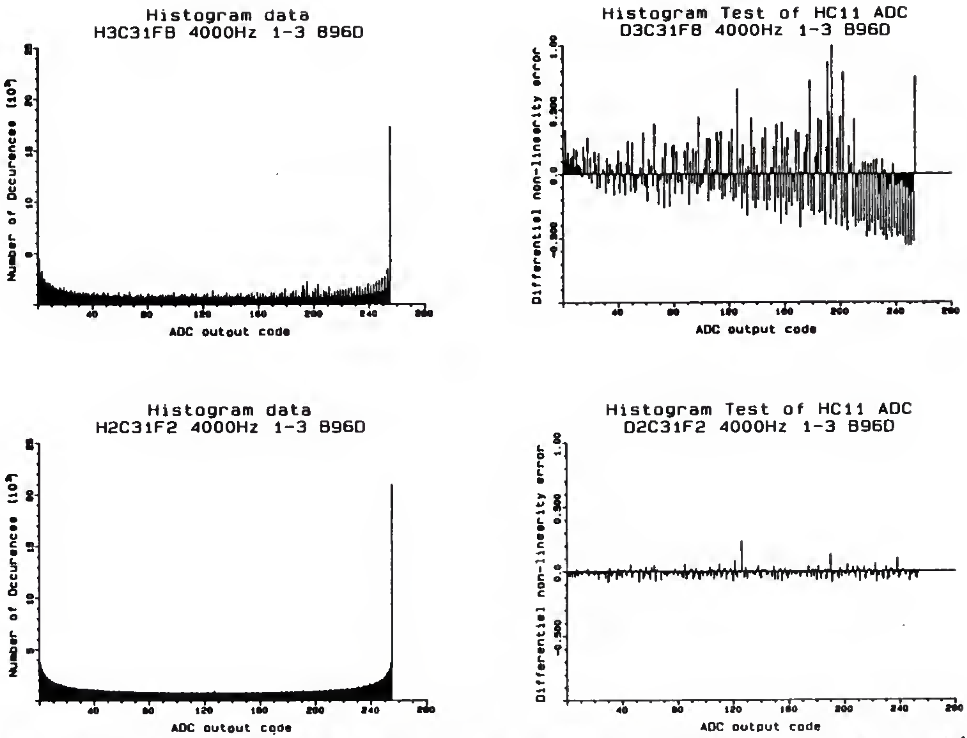

Figure 7-8. **Histogram data and Differential non-linearity errors for A/D. Continuous-conversion, multiple input. RC timer enabled.**

operational modes, E clock frequencies, and with the RC timer enabled and disabled except for continuous conversion modes at an E clock frequency of 2 Mhz (top plots) with the RC timer enabled. This exception is shown in figures 7-6 and 7-8. In all cases, the differential non-linearity errors increase in magnitude with an increase in frequency.

No missing codes were found in the histogram data files for any combination of A/D operational mode and E clock frequency with or without the RC timer enabled or disabled.

Method 2

Integral non-linearity errors, calculated to within one LSB from the harmonics in output FFT spectrum are shown in figures 7-9 to 7-12 with the RC timer enabled in the bottom plots and the 2 MHz E clock results on the left. Also, an indication of system noise can be observed as well.

Integral non-linearity errors seem to be within Motorola specifications in all combinations of A/D operational modes, E clock frequencies, and with the RC timer enabled and disabled except for the continuous-conversion, multiple input mode at a 2 MHz E clock frequency with the RC timer enabled, figure 7-12.

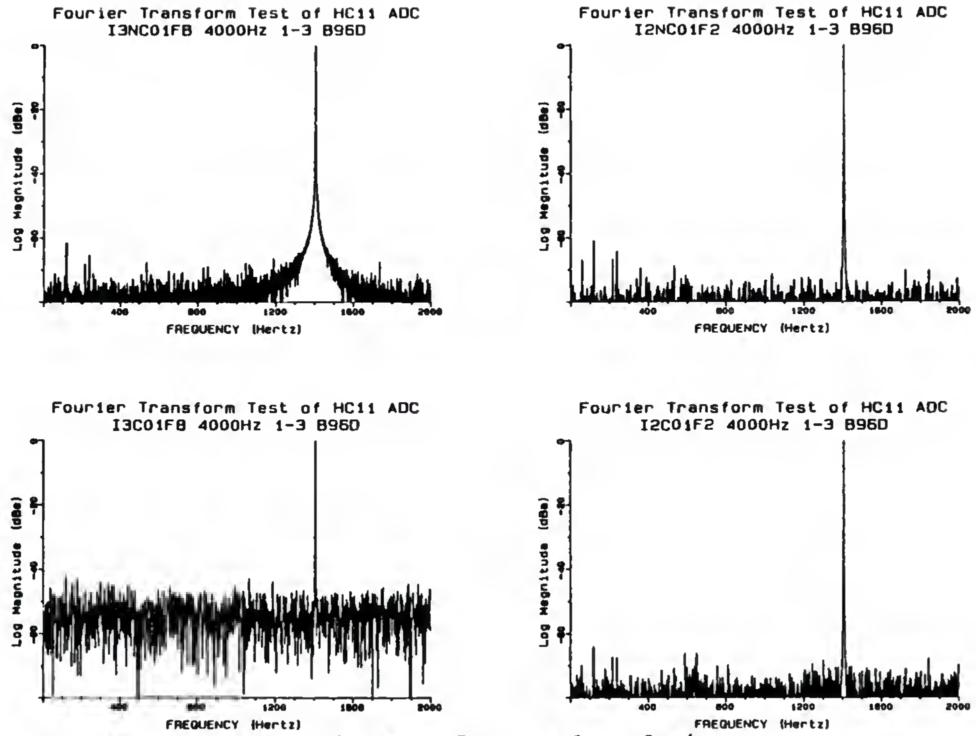

Figure 7-9. Fourier Transform Results of A/D.
Four-conversion, single input, input channel 1

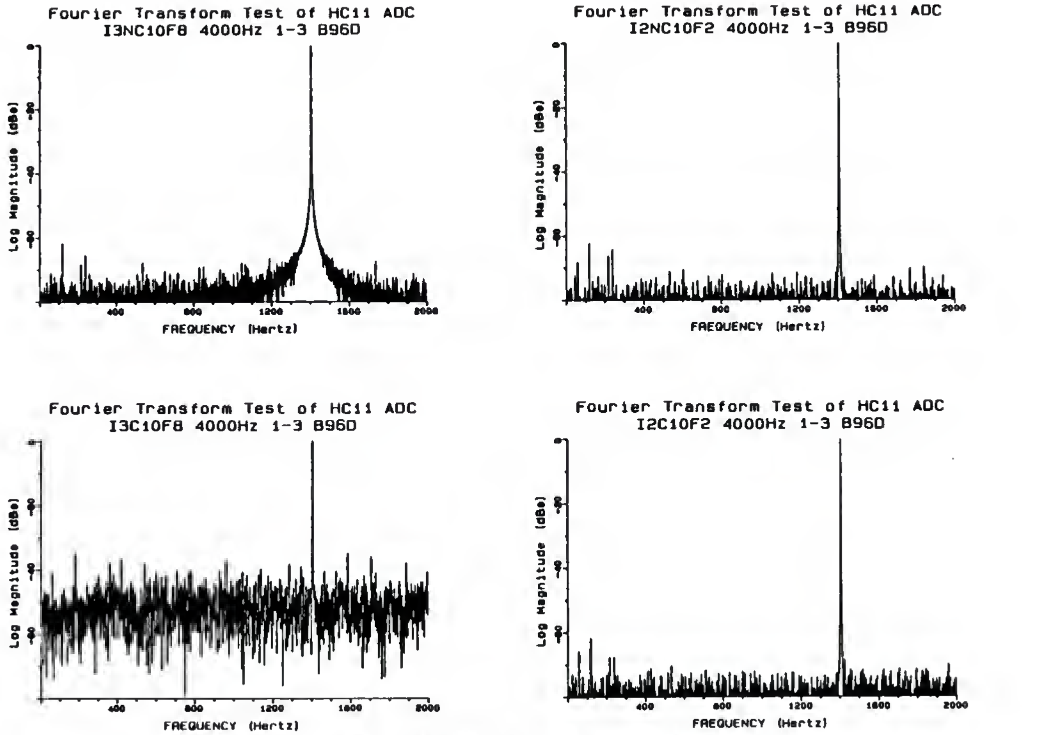

Figure 7-10. Fourier Transform Results of A/D. Four-conversion, multiple input.

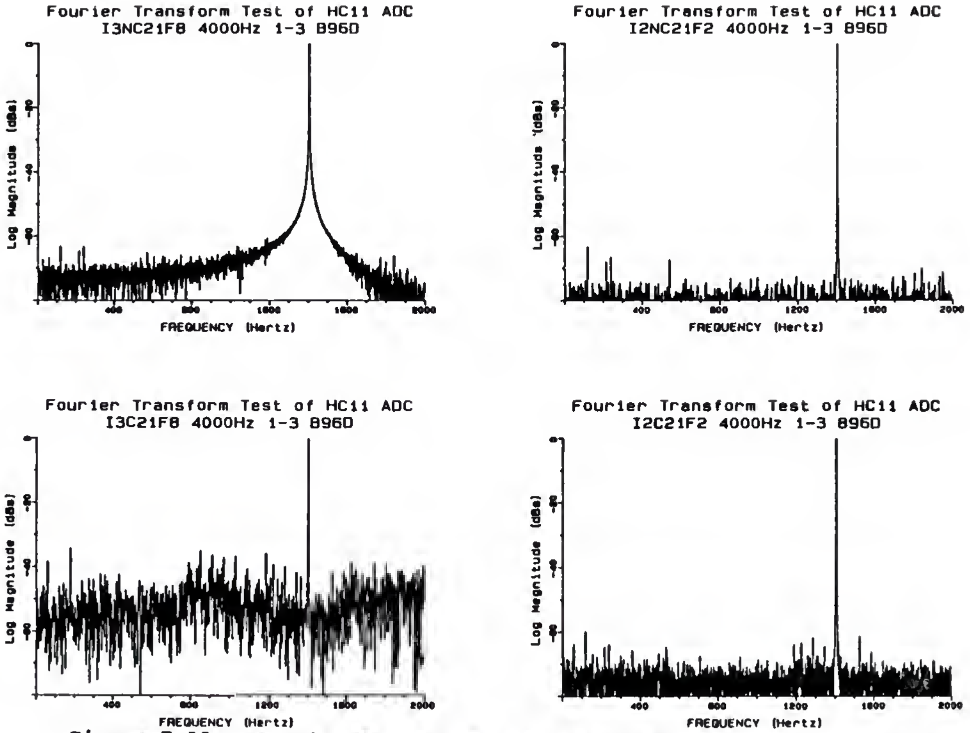

Figure 7-11. Fourier Transform Results of A/D.
Continuous-conversion, single input, input channel 1

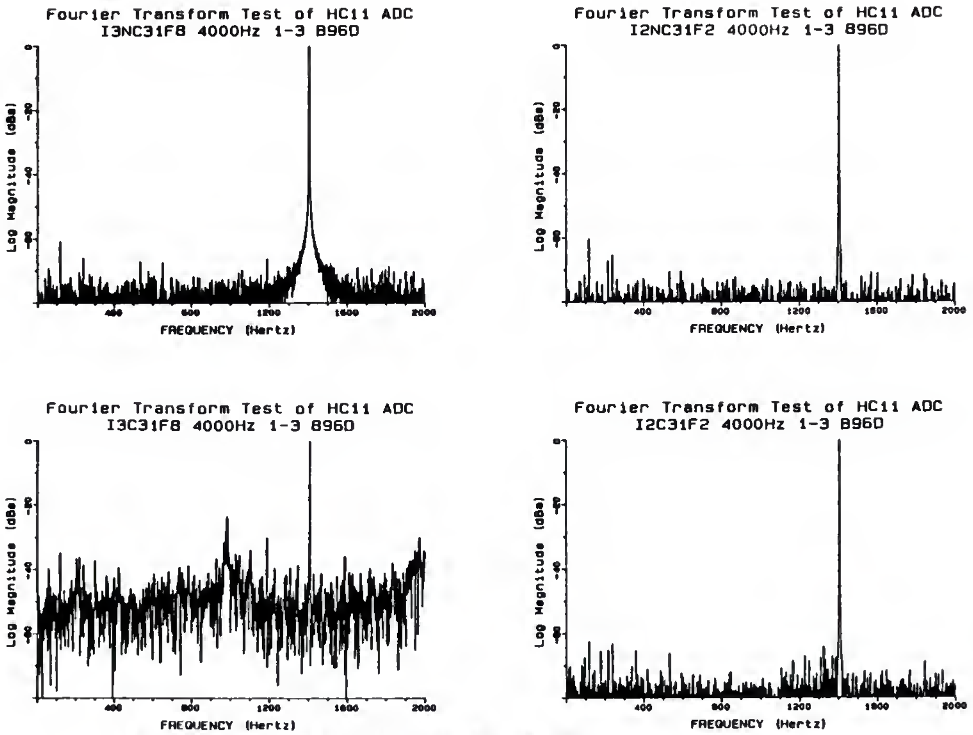

Figure 7-12. Fourier Transform Results of A/D.
Continuous-conversion, multiple input.

Overall system noise, indicated by the raising of the noise floor in comparison of an ideal 8-bit A/D, figure 6-3 is not apparent except in results obtained with the RC timer enabled at a 2 MHz E clock frequency.

Conclusion

Although no missing codes were found in any histogram test performed, small offsets were apparent in histogram plots with E clock frequencies of 2 MHz with the RC timer disabled. Differential non-linearity errors, with noise averaged out, are very small and within specifications except in the case of continuous-conversion modes with an E clock frequency of 2 MHz with the RC timer enabled. Integral non-linearity errors are within specifications except for results with the RC timer enabled at 2 MHz E clock frequencies. Overall system noise is apparent in all configurations at 2 MHz E clock rates with the RC timer enabled.

8.0 Summary and Recommendations

Static and dynamic testing of the HC11s for this thesis indicates the possibility of the HC11's A/D operating within specifications in future mask releases if the problems found during testing are corrected. In all cases of static testing, when noise and offsets are removed from the conversion results, the A/D falls within specifications for differential and integral non-linearity errors. In dynamic testing, all errors discovered had been previously found using static tests indicating the HC11's A/D has no large scale dynamic sensitivities. This indicates that dynamic testing need not be included in production floor testing of the HC11's A/D for this mask. If mask changes occur in the future, a dynamic characterization should be performed with several HC11s to ensure that dynamic sensitivities are not introduced with the mask change. Again, if dynamic sensitivities are not found in the newer mask, then dynamic testing need not be performed on the production floor.

The most informative test presented in this thesis was the total error determination in Method 3 of the static testing procedure. This test consists of using 1024 step, ramp input between 0 and +5 volts. Using a precision digital voltage meter, each step voltage was measured and then compared with the HC11's A/D conversion to yield the total errors in conversion results. This test gave indications of no missing codes, non-monotonic behavior, constant offsets, and the presence of RC timer induced errors.

Two programs that give indications of noise and pattern sensitivities are provided in Appendix A. These programs eliminate the need of an elaborate testing setup and require only a logic probe or logic analyzer.

Further testing recommendations for the HC11's A/D are to isolate the pattern sensitivity between Method 1 and Method 2 of the Static Testing Procedure, find the source of the constant offset found in Method 3 of the Static Procedure, and determine the cause of RC timer induced errors found in most testing methods used in this thesis. Due to the lack of wafer level testing facilities at this university, Motorola should probe the HC11 at the wafer level to find the source of the pattern sensitivities, the constant offsets, and the RC timer induced errors found in testing methods used in this thesis.

Appendix A

```
*****************************************************
*
*  SOURCE FILE:   qkccfck.src
*
*  DESCRIPTION:   This program provides a quick
*                 check of the HC11's A/D
*                 conversion process using a check
*                 of the CCF bit to ensure valid
*                 results are in the Result
*                 Registers.  The user needs to
*                 provide the following inputs
*                 prior to execution.
*
*            Location in
*                RAM
*
*                00 Hex    the configuration of
*                          the A/D that is
*                          written to the ADCTL.
*
*            01 - 02 Hex   the address of the
*                          Result Register to be
*                          checked.
*
*             VRH          a precision 5 volt
*                          reference.
*
*             VRL          tied to ground.
*
*                 Also, the user must supply an
*                 analog input to be converted if
*                 the A/D is configured to convert
*                 an external input.
*                 The result of the conversion is
*                 written to PORTB and can be
*                 checked using a logic probe.
*
*                 The RC timer can be enabled to
*                 clock the A/D system by changing
*                 the instruction
*                                oraa #$80
*                          to
*                                oraa #$c0.
*
*                 This program continually
*                 executes until the user aborts
*                 it.
*
*  AUTHOR:        Jeffrey C. Daniels    9-3-87
*                 Kansas State University
*
*****************************************************
```

```
PORTB  equ $1004   PORTB address

       org $c000

       ldaa $1039  Power up A/D system.
       oraa #$80   RC timer off.
       staa $1039

       ldx  $01    Have index register point to
*                  the Result Register of choice.

CONVRT ldaa $00    Initiate conversion.
       staa $1030

CHECK  ldab $1030  Check to see if conversion in
       bpl  CHECK  done.

       ldaa 0,X    Load result.
       staa PORTB  Store result to PORTB.

       bra CONVRT  Do another conversion.
```

```
*****************************************************
*
*  SOURCE FILE:   qkdlyck.src
*
*  DESCRIPTION:   This program provides a quick
*                 check of the HC11's A/D
*                 conversion process using the
*                 execution of a long delay loop
*                 to ensure valid results are in
*                 the Result Registers.  The user
*                 needs to provide the following
*                 inputs prior to execution.
*
*            Location in
*                RAM
*
*                00 Hex    the configuration of
*                          the A/D that is
*                          written to the ADCTL.
*
*            01 - 02 Hex   the address of the
*                          Result Register to be
*                          checked.
*
*             VRH          a precision 5 volt
*                          reference.
*
*             VRL          tied to ground.
*
*                Also, the user must supply an
*                analog input to be converted if
*                the A/D is configured to convert
*                an external input.
*                The result of the conversion is
*                written to PORTB and can be
*                checked using a logic probe.
*
*                The RC timer can be enabled to
*                clock the A/D system by changing
*                the instruction
*                                oraa #$80
*                          to
*                                oraa #$c0.
*
*                This program continually
*                executes until the user aborts
*                it.
*
*  AUTHOR:        Jeffrey C. Daniels    9-3-87
*                 Kansas State University
*
*****************************************************
```

```
PORTB   equ $1004   PORTB address
DELAY   equ  76

        org $c000

        ldaa $1039  Power up A/D system.
        oraa #$80   RC timer off.
        staa $1039

        ldy  $01    Have index register point to
*                   the Result Register of choice.

CONVRT  ldaa $00    Initiate conversion.
        staa $1030

        ldx #DELAY  Wait 450 clock cycles.
LOOP    dex
        bne  LOOP

        ldaa 0,Y    Load result.
        staa PORTB  Store result to PORTB.

        bra CONVRT  Do another conversion.
```

```
**********************************************
*
*  Source file: ckccf4.src
*
*  This program was written to statically
*  test the HC11 microprocessor.
*
*  RC oscillator is disabled.
*  Place ADC conversion mode in location 00
*  before running the program.
*
*  8-11-87
*
*  Jeffrey C. Daniels
*
*  Revisions: 8-11-87 Created from stget4.src.
*
************************************************

ADR0    EQU $1031
PORTB   EQU $1004
RESPTR  EQU $0001

         ORG   $C000

         LDY   #$1000    Point to port A.

         BCLR 0,Y $40   Clear flip flop.
         BSET 0,Y $40

         CLR   $1004     Clear PORT B
         BSET 0,Y $20    Latch PORT B into
         BCLR 0,Y $20    High Byte of Data Latch

         LDX   #ADR0     Initialize pointer to
         STX   RESPTR    result register.

         LDAA $1039      Enable ADC - -
         ORAA #$80       Power up ADC
         STAA $1039      RC timer off.

         LDAB $00
         STAB $1030      Initiate conversion.

*  Start of main loop *

NTREDY   LDAA 0,Y        Check to if HP has sent
         ANDA #$04       a pulse to start
         BEQ  NTREDY     conversion.

         LDX  RESPTR     Check to see if conversion
         CPX  #ADR0      has already been done.
         BNE   NXTOUT     If so, jump to NXTOUT.


         LDAB  $00
         STAB  $1030      Initiate conversion.
CHECK    LDAB  $1030      Check if conversion
         BPL   CHECK      is done.


NXTOUT   LDX   RESPTR
         LDAA  0,X
         STAA  PORTB      Load result

         BSET  0,Y $10    Store result in Low
         BCLR  0,Y $10    Byte of data latch.

         BCLR  0,Y $40    Clear flip flop.
         BSET  0,Y $40

         INC   RESPTR+1   Increment pointer to next
         LDX   RESPTR     Result register and then
         CPX   #ADR0+4    check if four results have
         BNE   CONT       been outputted to PORTB.

         LDX   #ADR0      If 4 results have been
         STX   RESPTR     sent then update result
*                         pointer.

CONT     BRA   NTREDY
```

```
*********************************************
*
*  Source file: ckdly4.src
*
*  This program was written to statically
*  test the HC11 microprocessor.
*
*  RC oscillator is disabled.
*  Place ADC conversion mode in location 00
*  before running the program.
*
*  8-11-87
*
*  Jeffrey C. Daniels
*
*  Revisions: 8-11-87 Created from sttest4.src.
*
*************************************************

ADR0    EQU $1031
PORTB   EQU $1004
RESPTR  EQU $0001
LOOPNO  EQU 76

         ORG  $C080

         LDY  #$1000    Point to port A.

         BCLR 0,Y $40  Clear flip flop.
         BSET 0,Y $40

         CLR  $1004     Clear PORT B
         BSET 0,Y $20   Latch PORT B into
         BCLR 0,Y $20   High Byte of Data Latch

         LDX  #ADR0     Initialize pointer to
         STX  RESPTR    result register.

         LDAA $1039     Enable ADC - -
         ORAA #$80      Power up ADC
         STAA $1039     RC timer off.

         LDAB $00
         STAB $1030     Initiate conversion.

*  Start of main loop *

NTREDY   LDAA 0,Y       Check to if HP has sent
         ANDA #$04      a pulse to start
         BEQ  NTREDY    conversion.
```

```
        LDX   RESPTR     Check to see if conversion
        CPX   #ADR0      has already been done.
        BNE   NXTOUT     If so, jump to NXTOUT.

        LDAB  $00

        STAB  $1030      Initiate conversion.

        LDX #LOOPNO      Wait 450 clock cycles.
DELAY   DEX
        BNE DELAY

NXTOUT  LDX   RESPTR
        LDAA  0,X
        STAA  PORTB      Load result

        BSET 0,Y $10     Store result in Low
        BCLR 0,Y $10     Byte of data latch.

        BCLR 0,Y $40     Clear flip flop.
        BSET 0,Y $40

        INC   RESPTR+1   Increment pointer to next
        LDX   RESPTR     Result register and then
        CPX   #ADR0+4    check if four results have
        BNE   CONT       been outputted to PORTB.

        LDX   #ADR0      If 4 results have been
        STX   RESPTR     sent then update result
*                        pointer.

CONT    BRA   NTREDY
```

```
*********************************************
*
*  Source file: stnrc2.src
*
*  This program was written to statically
*  test the HC11 microprocessor
*  utilizing the HP test system that
*  Steve Draving developed.
*
*
*  RC oscillator is disabled.
*  Place ADC conversion mode in location 00
*  before running the program.
*  Place which result register to read in
*  location 01 and 02 which will be read into
*  the X register.
*  Continous conversions.
*
*  12Mar87
*
*  Jeffrey C. Daniels
*
*  Revisions: 6-25-87 This program created from
*                     jdst5.src.
*
*             6-29-87 Created from stnrc.src to
*                     take out interupt structure.
*
*************************************************

        ORG  $C200

        LDY  #$1000    Point to port A.

        BCLR 0,Y $40  Clear flip flop.
        BSET 0,Y $40

        CLR  $1004     Clear PORT B
        BSET 0,Y $20   Latch PORT B into
        BCLR 0,Y $20   High Byte of Data Latch

        LDAA $1039     Enable ADC - -
        ORAA #$80      Power up ADC
        STAA $1039     RC timer off.

        LDAB $00
        STAB $1030     Initiate conversion.

NTREDY  LDAA 0,Y       Check to if HP has sent
        ANDA #$04      a pulse to start
        BEQ  NTREDY    conversion.
```

```
           LDAB $00
           STAB $1030      Initiate conversion.
CHECK      LDAB $1030      Check if conversion
           BPL  CHECK      is done.

           LDX  $01
           LDAA 0,X
           STAA $1004      Load result

           BSET 0,Y $10    Store result in Low
           BCLR 0,Y $10    Byte of data latch.

           BCLR 0,Y $40    Clear flip flop.
           BSET 0,Y $40

           BRA  NTREDY
```

```
*********************************************
*
*  Source file: strc2.src
*
*  This program was written to statically
*  test the HC11 microprocessor
*  utilizing the HP test system that
*  Steve Draving developed.
*
*
*  RC oscillator is Enabled.
*  Place ADC conversion mode in location 00
*  before running the program.
*  Place which result register to read in
*  location 01 and 02 which will be read into
*  the X register.
*  Continous conversions.
*
*  17 Jun 87
*
*  Jeffrey C. Daniels
*
*  Revisions: 6-25-87 This program created from
*                     jdst6.src.
*
*             6-29-30 Created from strc.src to
*                     take out interupt structure.
*
*********************************************

        ORG  $C300

        LDY  #$1000     Point to port A.

        BCLR 0,Y $40    Clear flip flop.
        BSET 0,Y $40

        CLR  $1004      Clear PORT B
        BSET 0,Y $20    Latch PORT B into
        BCLR 0,Y $20    High Byte of Data Latch

        LDAA $1039      Enable ADC - -
        ORAA #$c0       Power up ADC
        STAA $1039      RC timer on.

        LDAB $00
        STAB $1030      Initiate conversion.

NTREDY  LDAA 0,Y        Check to see if HP has
        ANDA #$04       sent a pulse to start
        BEQ  NTREDY     conversion.
```

```
         LDAB $00
         STAB $1030      Initiate conversion.
CHECK    LDAB $1030      Check if conversion
         BPL  CHECK      is done.

         LDX  $01
         LDAA 0,X
         STAA $1004      Load result

         BSET 0,Y $10    Store result in Low
         BCLR 0,Y $10    Byte of data latch.

         BCLR 0,Y $40    Clear flip flop.
         BSET 0,Y $40

         BRA  NTREDY
```

```
10    ! ********************************************************************
20    ! **                          ROFOUR            ver 2.0              **
30    ! ********************************************************************
40    ! **                                                                 **
50    ! **         This program reads the four Result Registers of the     **
60    ! **     68HC11's onboard ADC and displays them to the screen.  The  **
70    ! **     DVM readings are also displayed on the screen.  None of the **
80    ! **     information is stored in arrays or stored in files.  This   **
90    ! **     program provides a quick check to verify that the 4 Result  **
100   ! **     Registers are functioning properly.                         **
110   ! **                                                                 **
120   ! **         Jeff's STGET4.OBJ program is used to drive the 68HC11.  **
130   ! **                                                                 **
140   ! **     MICHAEL PANKRATZ                 04 AUGUST 1987             **
150   ! **                                                                 **
160   ! ********************************************************************
170   !
180   PRINTER IS 16
190   PRINT PAGE
200   PRINT TAB(30);"ADC DATA AQUISITION"
210   PRINT TAB(25);"Testing the 4 Result Registers"
220   PRINT LIN(3)
230   !
240 Initialize:    !
250   OUTPUT 709;"F1RANST2Z1"           ! Initialize DVM
260   RESET 3                           ! Initialize GPIO for ADC
270   RESET 2                           ! Initialize GPIO for precision DAC
280   !
290   BEEP
300   DISP "Press any key to begin data acquision."
310   ON KBD GOTO Get_data ,ALL
320 Wait:    GOTO Wait
330   !
340 Get_data:   !
350   !
360   PRINT "Press any key to abort."
370   ON KBD GOTO Exit ,ALL             ! Quit if any key is pressed
380   !
390 Loop_here: !
400       TRIGGER 709                   ! Start the DVM
410 Rd:   STATUS 709;Stat
420       IF BIT(Stat,0)<>1 THEN Rd     ! Wait till ready
430       ENTER 709;Dvm                 ! Read DVM
440       !
450       Chan1=READBIN(3)              ! Read result register 1
460       DISP Dvm;TAB(20);Chan1;TAB(30);Chan2;TAB(40);Chan3;TAB(50);Chan4
470       Chan2=READBIN(3)              ! Read result register 2
480       DISP Dvm;TAB(20);Chan1;TAB(30);Chan2;TAB(40);Chan3;TAB(50);Chan4
490       Chan3=READBIN(3)              ! Read result register 3
500       DISP Dvm;TAB(20);Chan1;TAB(30);Chan2;TAB(40);Chan3;TAB(50);Chan4
510       Chan4=READBIN(3)              ! Read result register 4
520       DISP Dvm;TAB(20);Chan1;TAB(30);Chan2;TAB(40);Chan3;TAB(50);Chan4
530       !
540   GOTO Loop_here
550   !
560 Exit:   !
570   DISP "Program terminated."
580   BEEP
590   END
```

```
10    ! ***************************************************************
20    ! **                        GETDAT              ver 5.0          **
30    ! ***************************************************************
40    ! **                                                             **
50    ! **        This program collects data from the ADC under test   **
60    ! **   (the 68HC11's onboard ADC) and writes it to a file.       **
70    ! **   The file can be read by PLOT and PRTDAT but only the first **
80    ! **   array (the raw data) will be read.  The bin count array and **
90    ! **   other data will be ignored.                               **
100   ! **                                                             **
110   ! **        The file format is as follows.  The first element in **
120   ! **   the file is the raw data array size.  It is followed by the **
130   ! **   array of data itself.  Next is the size of the bin count  **
140   ! **   array, followed by the array itself.  Then the reference  **
150   ! **   voltage, Vref, and the ADC resolution, Res, are stored.   **
160   ! **   Finally, 4 transtion points are stored for gain and offset **
170   ! **   error calculations.  They are 0-->1, 6-->7 & 7-->8 (arbitrary **
180   ! **   for step width), and 254-->255.                           **
190   ! **                                                             **
200   ! **        The program calls Vtrans(Pre_tog, Res, Vref), a function **
210   ! **   to find the transtion point voltages.                     **
220   ! **                                                             **
230   ! **        Version 2.0 change the DNL error method from the      **
240   ! **   theoretical method (using transition points) to the       **
250   ! **   histogram method.  Now only 4 trenstion points are teken  **
260   ! **   to figure offset and gain errors. Instead, the ADC under  **
270   ! **   test is read 10 times and the results are placed into bins. **
280   ! **                                                             **
290   ! **        Versions 3.0 and 3.5 were modifications made to the  **
300   ! **   toggle function.  These improvements are outlined in the  **
310   ! **   header for the function (see below).                      **
320   ! **                                                             **
330   ! **        Version 4.0 was a modification made in conjunction with **
340   ! **   a change made in the HC11 program.  The delay to allow the **
350   ! **   DAC to set up was taken out of the HC11 program and put in **
360   ! **   this program.  That simply involved changing two existing **
370   ! **   delay loops from 5ms to 100ms.                            **
380   ! **                                                             **
390   ! **        Version 5.0 first determines the minimum and maximum **
400   ! **   ADC output values and uses these in the TOGFN instead of  **
410   ! **   sending 0->1 end 254->255.  This will allow testing of    **
420   ! **   converters which do not operate over their full range.    **
430   ! **   Also, trensition points 6->7 and 7->8 are not taken since **
440   ! **   they are no longer used by ERROR. The two variables are   **
450   ! **   are now used to store the minimum and maximum ADC count   **
460   ! **   values that were sent into the toggle function.           **
470   ! **                                                             **
480   ! **   MICHAEL PANKRATZ                 05 AUGUST 1987           **
490   ! **                                                             **
500   ! ***************************************************************
510   !
520   DIM Input(1,1024)                  ! Data array
530   DIM Bin(1,256)                     ! Bin count array
540   DIM Vtr(3)                         ! Transition voltage array
550   !
560   !
570 Constants:     !
580   Vref=5                             ! ADC reference voltage
590   Res=8                              ! ADC resolution (in bits)
```

```
600   Semples=1024                        ! Number of semples teken
610   Vmin=0                              ! Velid values: 0 to Vref
620   Vmex=Vref                           ! Velid values: 0 to Vref > Vmin
630   !
640   !
650   PRINTER IS 16
660   PRINT PAGE
670   PRINT TAB(33);"ADC DATA AQUISITON"
680   PRINT TAB(20);"Deta for Offset, Gein, INL, end DNL errors"
690   PRINT LIN(3)
700   !
710   !
720 Initielize:    !
730   OUTPUT 709;"F1RAN5T2Z1"             ! Initializa DVM
740   RESET 3                             ! Initielize GPIO for ADC
750   RESET 2                             ! Initielize GPIO for precision DAC
760   FOR I=0 TO 2^Ras-1                  ! Initialize bin count arrey
770      Bin(0,I)=I
780      Bin(1,I)=0
790   NEXT I
800   !
810   !
820 Open_file:    !
830   REDIM Input(1,Semples-1)
840   REDIM Bin(1,2^Ras-1)
850   !
860   BEEP
870  ! Detfile$="601F2"
880   EDIT "Enter fileneme for the dete:",Detfile$
890   ASSIGN #1 TO Detfile$,Stet
900   IF Stet=0 THEN PURGE Detfile$       ! Delete if file elreedy exits
910   Records=INT((Semples+2^Res)/16)+1  ! Celculete number of records
920   CREATE Detfile$,Records             ! Creete the dete file
930   ASSIGN #1 TO Datfile$               ! Open the file
940   !
950   !
960   BEEP
970   DISP "Press eny key to bagin dete acquision."
980   ON KBD GOTO Get_trens ,ALL
990 Weit:    GOTO Weit
1000  !
1010  !
1020 Get_trans:    !
1030     DISP
1040     PRINT "Finding trensitions:"
1050     Dvel=0
1060     GOSUB Set_dec
1070     Vtr(1)=READBIN(3)                ! Determine min ADC output velue
1080     Cmin=Vtr(1)
1090     Vtr(0)=FNVtrens(Cmin,Res,Vref)  ! Voltege of 1st trensition point
1100     !
1110     Dvel=262143
1120     GOSUB Set_dec
1130     Vtr(2)=READBIN(3)                ! Determine mex ADC output velua
1140     Cmex=Vtr(2)-1
1150     Vtr(3)=FNVtrens(Cmex,Res,Vref)  ! Voltega of lest trensition point
1160  BEEP
1170  !
1180  !
1190  Dmin=INT(Vmin*2^18/Vref)
```

```
1200  Dmax=INT(Vmax*2^18/Vref)
1210  Vstep=(Vmax-Vmin)/(Samples-1)
1220  Dstep=INT(Vstep*2^18/Vref)          ! Step size to incr. the DAC
1230  !
1240  PRINT
1250  PRINT "Sampling";Samples;"points between";Vmin;"and";Vmax;"volts."
1260  PRINT "Press any key to abort."
1270  !
1280  !
1290  ON KBD GOTO Exit ,ALL               ! Quit if any key is pressed
1300  FOR I=0 TO Samples-1
1310     DISP "Conversion #";I
1320     Dval=I*Dstep+Dmin                ! Digital value for this sample
1330     GOSUB Set_dac                    ! Set the DAC with value in Dval
1340     !
1350     TRIGGER 709                      ! Start the DVM
1360 Rd: STATUS 709;Stat
1370     IF BIT(Stat,0)<>1 THEN Rd        ! Wait till ready
1380     ENTER 709;Input(0,I)             ! Read DVM
1390     !
1400     Input(1,I)=READBIN(3)            ! Read ADC under test
1410     Index=Input(1,I)
1420     Bin(1,Index)=Bin(1,Index)+1
1430     !
1440     FOR J=1 TO 9                     ! Update the bins
1450        Index=READBIN(3)
1460        Bin(1,Index)=Bin(1,Index)+1
1470     NEXT J
1480  NEXT I
1490  DISABLE                             ! Turns off key-abort function
1500  !
1510  !
1520 Write_file:  !
1530  PRINT #1;Samples,Input(*)           ! Store data
1540  PRINT #1;2^Res,Bin(*)
1550  PRINT #1;Vref,Res,Vtr(*)
1560  !
1570  !
1580 Exit:   ASSIGN #1 TO *               ! Close the file
1590  DISP "Program terminated."
1600  BEEP
1610  WAIT 750
1620  ENABLE
1630  GOTO Constants
1640  END
1650  !
1660  !
1670 Set_dac:     ! Value is passed in through Dval to set the DAC
1680     High=INT(Dval/65536)            ! High byte for the DAC
1690     Mid=INT(Dval/256)-256*High      ! Middle byte for DAC
1700     Low=Dval-65536*High-256*Mid     ! Low byte for the DAC
1710     !
1720     WRITE BIN 2;High,High+1024,High  ! \
1730     WRITE BIN 2;Mid,Mid+512,Mid      ! - Send word to the DAC
1740     WRITE BIN 2;Low,Low+256,Low,0    ! /
1750     !
1760     WAIT 100                        ! 100ms delay
1770  RETURN
1780  !
1790  !
```

```
1800  !
1810  ! ****************************************************************
1820  ! **                        TOGFN        (TOGSUB  ver 4.0)       **
1830  ! ****************************************************************
1840  ! **                                                             **
1850  ! **      This function determines the input voltage just before **
1860  ! **   and after a toggle point of the ADC under test (in this   **
1870  ! **   case, the 68HC11's on-board converter).  The full 18 bits **
1880  ! **   resolution of the precision DAC are used to obtain as     **
1890  ! **   accurate results as possible.  The results are averaged   **
1900  ! **   and returned in the function name.  The format is:       **
1910  ! **                                                             **
1920  ! **           FNVtrans(Togpt, Res, Vref)                        **
1930  ! **                                                             **
1940  ! **           where:                                            **
1950  ! **              Togpt - the desired ADC output value BEFORE    **
1960  ! **                      the transition.  0 <= Togpt <= (2^Res)-2 **
1970  ! **              Res   - the resolution of the ADC under test   **
1980  ! **                      (in bits)                              **
1990  ! **              Vref  - the ADC reference voltage              **
2000  ! **                                                             **
2010  ! **   Note:  The DVM and GPIO busses must be initialized before **
2020  ! **          this function is called.                           **
2030  ! **                                                             **
2040  ! **        Version 2.0 added a lower resolution binary pre-search **
2050  ! **   which dramatically reduced the search time for a toggle   **
2060  ! **   point.                                                    **
2070  ! **        Version 3.0 made a few changes to give the function  **
2080  ! **   the capability to catch missing code.  The function then  **
2090  ! **   then gives the users the option to enter a new toggle point **
2100  ! **   (or the same one to check for repeatability) or to terminate **
2110  ! **   the program.  Returns a 999 to indicate missing code error. **
2120  ! **                                                             **
2130  ! **        Version 4.0 made a couple more changes to help fight **
2140  ! **   missing code errors by checking the ADC output after coming **
2150  ! **   out of the binary search and kicking it back up there if the **
2160  ! **   ADC value was below the entered toggle point value.       **
2170  ! **                                                             **
2180  ! **   MICHAEL PANKRATZ       24 JUN 1987                        **
2190  ! **                                                             **
2200  ! ****************************************************************
2210  DEF FNVtrans(Togpt,Res,Vref)
2220  ! ****************************************************************
2230  !
2240  !
2250 Init:   !
2260  PRINT Togpt;"-->";Togpt+1
2270  Cvolt=0                               ! Current DAC output voltage
2280  Cadc=0                                ! Current ADC output value
2290  Pvolt=0                               ! Previous DAC output voltage
2300  Padc=0                                ! Previous ADC output value
2310  !
2320  !
2330  Vmin=Togpt*(Vref/2^Res)               ! Starting input voltage to ADC
2340  Dmin=INT(Vmin*2^18/Vref)              ! Start value for DAC
2350  Vstep=Vref/2^(Res+1)                  ! 1/2 LSB of the ADC under test
2360  Dstep=INT(Vstep*2^18/Vref)            ! Step size of DAC
2370  !
2380  !
2390  Dac=Dmin
```

```
2400  Bin_res=2^(18-13)                      ! Set min step size to 13 bit res.
2410 Bin:!  Binary search for ADC toggle point (to 13 bit resolution)
2420      IF Dstep<Bin_res THEN Seq          ! Take the DAC to 13 bit resolution
2430      Dac=Dac+Dstep                      ! DAC input value
2440      IF Dac>262143 THEN Dac=262143      ! Prevent over-ranging the DAC
2450 Lp2:GOSUB Set_dac                       ! Set the DAC
2460      Cadc=READBIN(3)                    ! Get current ADC value
2470      DISP "DAC input:";Dac,"ADC output:";Cadc
2480      IF Cadc<=Togpt THEN Bin            ! Did toggle occur?
2490      Dac=Dac-Dstep                      ! Set DAC to before toggle point
2500      IF Dac<0 THEN Dac=0                ! Prevent under-ranging the DAC
2510      IF Dstep>=Bin_res THEN Dstep=INT(Dstep/2) ! Divide step size by 2
2520  GOTO Lp2
2530  !
2540  !
2550 Seq:   ! Sequential search of toggle point (18 bit resolution)
2560  Dstep=1                                ! Set DAC step size to 1
2570  IF Cadc=Togpt THEN Lp3                 ! Make sure ADC value is good
2580      Dstep=Bin_res                      ! Set DAC step size to min value
2590      GOTO Bin                           ! Try again
2600  !
2610 Lp3:Dac=Dac+Dstep
2620      GOSUB Set_dac                      ! Set the DAC
2630      Padc=Cadc                          ! Store previous value of ADC
2640      Cadc=READBIN(3)                    ! Get current ADC value
2650      DISP "DAC input:";Dac,"ADC output:";Cadc
2660      IF Cadc>Padc THEN Exit             ! Did toggle occur?
2670  GOTO Lp3
2680  !
2690  !
2700 Exit: !
2710  GOSUB Read_dvm                         ! Current DAC output
2720  Cvolt=Voltage
2730  Dac=Dac-Dstep
2740  GOSUB Set_dac
2750  GOSUB Read_dvm                         ! Previous DAC output
2760  Pvolt=Voltage
2770  Ave=(Cvolt+Pvolt)/2                    ! Average to find transition voltage
2780  !
2790  !
2800  IF (Cadc=Padc+1) AND (Cadc=Togpt+1) THEN Ok !No missing codes
2810  PRINTER IS 16                          ! Output device is the screen
2820  FOR X=1 TO 3
2830     BEEP
2840     WAIT 250
2850  NEXT X
2860  PRINT "Missing Code!"
2870  PRINT "Actual results: ";Padc;"-->";Cadc
2880  PRINT "Desired results:";Togpt;"-->";Togpt+1
2890  Err$="N"
2900  EDIT "Enter a New toggle point or Terminate (N/T)?",Err$
2910  IF UPC$(Err$[1,1])="T" THEN Term
2920 Inp: INPUT "Enter desired ADC output BEFORE the toggle",Togpt
2930  IF (Togpt<0) OR (Togpt>2^Res-2) THEN Inp
2940  GOTO Init
2950 Term:  Ave=999
2960 Ok: RETURN Ave
2970  FNEND
2980  !
2990  !
```

```
3000  !
3010 Set_dac: ! Calculates DAC input word and sends it to the DAC
3020  Dval=Dac                            ! Digital value for this sample
3030  High=INT(Dval/65536)                ! High byte for the DAC
3040  Mid=INT(Dval/256)-256*High          ! Middle byte for the DAC
3050  Low=Dval-65536*High-256*Mid         ! Low byte for the DAC
3060  WRITE BIN 2;High,High+1024,High     ! \
3070  WRITE BIN 2;Mid,Mid+512,Mid         ! -  Set the DAC
3080  WRITE BIN 2;Low,Low+256,Low,0       ! /
3090  WAIT 100                            ! Pause for 100 msec
3100  RETURN
3110  !
3120  !
3130  !
3140 Read_dvm:   ! Takes one DVM reading
3150  !  It returns the value in 'Voltage'
3160  TRIGGER 709                         ! Begin reading DVM
3170 Wait:  STATUS 709;Dvmstat
3180  IF BIT(Dvmstat,0)<>1 THEN Wait      ! Is it finished?
3190  ENTER 709;Voltage                   ! Read the value from the DVM
3200  RETURN
```

```
10    ! ****************************************************************
20    ! **                         DMAIL        The Transmitter       **
30    ! ****************************************************************
40    ! **                                                            **
50    ! **      This program was written to run on an HP 9845B.  It   **
60    ! **   reads in a raw data file created by GETDAT and sends it over **
70    ! **   the HP-IB bus to a HP 9236 computer.  There are several  **
80    ! **   things which must be noted:                              **
90    ! **                                                            **
100   ! **   1.  The 9845B must be configured as the controller (it is **
110   ! **       normally in this mode), and the 9236 must be in the   **
120   ! **       non-controller mode.  This requires changing a jumper **
130   ! **       on the motherboard of the 9236 (jumper should connect **
140   ! **       the left two prongs).  Be sure to put the jumper back **
150   ! **       to its original position when the transfer is completed. **
160   ! **                                                            **
170   ! **   2.  The 9236 must have its version of DMAIL (the receiver) **
180   ! **       running first, before this program is started, or it **
190   ! **       will miss some of the data.                          **
200   ! **                                                            **
210   ! **   3.  The DMAILer only sends over raw data files.  If it   **
220   ! **       is desired to send a BASIC program, use EMAIL.       **
230   ! **                                                            **
240   ! **                                                            **
250   ! **   Written by HEWLETT-PACKARD                               **
260   ! **   Modified by Michael Penkretz       20 July 1987          **
270   ! **                                                            **
280   ! ****************************************************************
290   DIM Data1(1,1024),Data2(1,256),Vtr(3)  ! The data arrays
300   Size=81                              ! The size of a raw data file
310   !
320   !
330   PRINTER IS 16
340   PRINT PAGE
350   PRINT TAB(35);"HP DATA MAILER"
360   PRINT TAB(22);"Data transfer from a 9845B to a 9236"
370   PRINT LIN(3)
380   !
390   !
400 Input:  BEEP
410   Datafile$="A01F8"                    ! Input file name
420   EDIT "Enter name of file to transfer (must be ASCII):",Datafile$
430   ASSIGN #1 TO Datafile$,Stat
440   IF Stat<>1 THEN Cont                 ! Does file exist?
450      PRINT "File ";Datafile$;" does not exist."
460      PRINT
470      BEEP
480      WAIT 200
490   GOTO Input
500   !
510 Cont: !
520   OUTPUT 720;Datafile$,Size            ! Send over the file name & size
530   !
540   !
550 Read_file:  !
560      DISP "Reading data from file..."
570      READ #1;Size1
580      REDIM Data1(1,Size1-1)
590      READ #1;Data1(*)                  ! Read in first array
```

```
600       REAO #1;Siza2
610       REOIM Oata2(1,Size2-1)
620       REAO #1;Oata2(*)               ! Raad in second array
630       REAO #1;Vraf,Ras,Vtr(*)        ! Raad in raamaining data
640    !
650    !
660 Sand_data:    !
670       OISP "Sanding data over HPI8..."
680       OUTPUT 720;Sizal               ! Sand siza of 1st array
690       FOR I=0 TO 1
700          FOR J=0 TO Sizal-1
710             OUTPUT 720;Oatal(I,J)    ! Sand 1st array
720          NEXT J
730       NEXT I
740       OUTPUT 720;Size2               ! Size of 2nd array
750       FOR I=0 TO 1
760          FOR J=0 TO Siza2-1
770             OUTPUT 720;Oata2(I,J)    ! Send 2nd array
780          NEXT J
790       NEXT I
800       OUTPUT 720;Vraf                ! Sand remaining valuas
810       OUTPUT 720;Res                     !
820       FOR I=0 TO 3
830          OUTPUT 720;Vtr(I)
840       NEXT I
850 Eoj:      !
860    ASSIGN #1 TO *                    ! Closa input fila
870    OISP "Oata transfar complated."
880    8EEP
890    ENO
```

```
10    ! ***************************************************************
20    ! **    for the 9236       DATAMAIL           The Receiver        **
30    ! ***************************************************************
40    ! **                                                              **
50    ! **        This program was written to run on the HP 9236 computer.  **
60    ! **   It receives data files created by GETDAT over the HPIB       **
70    ! **   bus and stores it on disk.  The first item transfered over   **
80    ! **   the HPIB bus is the file name and size so this program does  **
90    ! **   not need to prompt the user for any information.  There are  **
100   ! **   several things which must be observed for the Data Mailer to **
110   ! **   operate properly:                                            **
120   ! **                                                              **
130   ! **   1.  The 9845B must be configured as the controller (it is    **
140   ! **       normally in this mode), and the 9236 must be in the      **
150   ! **       non-controller mode.  This requires changing a jumper    **
160   ! **       on the motherboard of the 9236 (jumper should connect    **
170   ! **       the left two prongs).  Be sure to put the jumper back    **
180   ! **       to its original position when the transfer is completed. **
190   ! **                                                              **
200   ! **   2.  The 9236 must have its version of DATAMAIL (the Receiver) **
210   ! **       running first, before the Transmitter program on the     **
220   ! **       9845B is started or some of the data will get lost.      **
230   ! **                                                              **
240   ! **   3.  The EMAILer can only send ASCII formated files.  If it   **
250   ! **       is desired to send a BASIC program, this program will    **
260   ! **       not work.  Use EMAIL2 to send programs.                  **
270   ! **                                                              **
280   ! **                                                              **
290   ! **   Written  by HEWLETT-PACKARD                                  **
300   ! **   Modified by Michael Penkratz              20 July 1987       **
310   ! **                                                              **
320   ! ***************************************************************
330   DIM Data1(1,1024),Data2(1,256),Vtr(3)
340   !
350   !
360   OUTPUT 2;CHR$(255)&CHR$(75);        ! Clear the screen
370   PRINT TABXY(33,2);CHR$(136);"HP DATA MAILER"
380   PRINT TAB(22);CHR$(138);"Data transfer from a 9845B to a 9236"
390   PRINT
400   PRINT
410   PRINT
420   !
430   !
440   ASSIGN @Ifile TO 7                  ! Set up HPIB like an input file
450   !
460   DISP "Waiting for data."
470   ENTER @Ifile;Outfile$;Fsize         ! Read filename & size off the HPIB
480   DISP "Receiving filename."
490   BEEP
500   !
510 Open_file:   !
520       ON ERROR GOSUB File_exists
530       CREATE BDAT Outfile$,Fsize
540       ASSIGN @Ofile TO Outfile$       ! Open output file on disk
550       OFF ERROR
560   !
570   !
580 Read_data:  !
590       BEEP
```

```
600      DISP "Receiving data..."
610      ENTER @Ifile;Size1                    ! Read in size of 1st array
620      REDIM Data1(1,Size1-1)
630      FOR I=0 TO 1
640         FOR J=0 TO Size1-1
650            ENTER @Ifile;Data1(I,J)    ! Read in first array
660         NEXT J
670      NEXT I
680      ENTER @Ifile;Size2                    ! Size of 2nd array
690      REDIM Data2(1,Size2-1)
700      FOR I=0 TO 1
710         FOR J=0 TO Size2-1
720            ENTER @Ifile;Data2(I,J)    ! Read in second array
730         NEXT J
740      NEXT I
750      ENTER @Ifile;Vref;Res                 ! Read in remaining data
760      FOR I=0 TO 3
770         ENTER @Ifile;Vtr(I)
780      NEXT I
790   !
800   !
810 Write_data:   !
820      DISP "Writing data to a file..."
830      OUTPUT @Ofile;Size1;Data1(*)
840      OUTPUT @Ofile;Size2;Data2(*)
850      OUTPUT @Ofile;Vref;Res;Vtr(*)
860   !
870   !
880 Eof:   !
890      ASSIGN @Ofile TO *                    ! Close the files
900      ASSIGN @Ifile TO *
910      DISP "Data transfer complete."
920      PRINT CHR$(140);"Data is stored in ";CHR$(136);Outfile$
930      GOTO Exit
940   !
950   !
960   !
970 File_exists:  !  Purges the file if it already exits
980      IF ERRN=54 THEN
990         PURGE Outfile$
1000        RETURN
1010     ELSE
1020        PRINT CHR$(137);CHR$(130)
1030        PRINT "Terminal Error!"
1040        PRINT "Error code";ERRN
1041        PRINT CHR$(128);CHR$(139);
1042        STOP
1050     END IF
1060     !
1070     !
1080 Exit:BEEP
1090  PRINT CHR$(139)
1100  END
```

```
10    ! ******************************************************************
20    ! **     for the 9236            ERROR              ver 5.236       **
30    ! ******************************************************************
40    ! **                                                               **
50    ! **      This program takes the data file created by GETDAT and   **
60    ! **  calculates the offset error, gain error, integral nonlin-    **
70    ! **  earity, differential nonlinearity, and absolute errors.  The **
80    ! **  nonlinearity and absolute errors are written to files which  **
90    ! **  may be read by PLOT if desired.                              **
100   ! **                                                               **
110   ! **       INL error is calculated using the method outlined in    **
120   ! **  ANALOG-DIGITAL CONVERSION HANDBOOK, 3rd ED (Analog Devices), **
130   ! **  pp 317-330.                                                  **
140   ! **                                                               **
150   ! **       DNL error is calculated using the histogram method      **
160   ! **  outlined in D. Doerfler's thesis, p 34.                      **
170   ! **                                                               **
180   ! **       ABS error is the difference between the ADC output and  **
190   ! **  the ideal straight line.                                     **
200   ! **                                                               **
210   ! **       Offset and Gain errors are also calculated using the    **
220   ! **  methods obtained from the A-D CONVERSIONS HANDBOOK, pp 317-330.**
230   ! **                                                               **
240   ! **       Version 4.0 added a loop to check for non-monotonicities **
250   ! **  and missing code.  Counters keep track of the number of      **
260   ! **  occurrences and the last element is also stored.  The number **
270   ! **  of occurencee and the value of the last occurences are then  **
280   ! **  printed out.                                                 **
290   ! **                                                               **
300   ! **       Version 5.0 added Alpha & Beta, correction factors, to   **
310   ! **  help eliminate offset and gain errors from the data. It now  **
320   ! **  calculates INL using 2 methods: endpoint (with Alpha & Beta) **
330   ! **  and the histogram method from Doerfler's thesis.  Also fixed **
340   ! **  program to check for true missing code by detecting any      **
350   ! **  empty bins read in from the raw data file.  And added a few  **
360   ! **  bells and whistles to make I/O a little more friendly.       **
370   ! **  Also using middle 2 Vtrans variables to store ADC values for **
380   ! **  the first and last transitions.                              **
390   ! **                                                               **
400   ! **                                                               **
410   ! **  MICHAEL PANKRATZ                        11 August 1987       **
420   ! **                                                               **
430   ! ******************************************************************
440   !
450   DIM Data(1,1024),Abs(1,1024)          ! Must make these arrays larger if
460   DIM Bin(1,256),Dnl(1,256)             !   testing an ADC with more than
470   DIM Inl(1,256),Hinl(1,256)            !   8 bits.
480   DIM Vtrans(3),Non_mono(1),Code(1)
490   Old=0                                 ! An error flag
500   !
510   !
520   PRINTER IS 1
530   PRINT CHR$(12);CHR$(140);
540   PRINT TAB(20);"OFFSET, GAIN, INL, DNL, & ABSOLUTE ERRORS"
550   PRINT CHR$(136)
560   PRINT
570   PRINT
580   !
590   !
```

```
600   ON ERROR GOTO No_file
610   CAT
620 Inp1: BEEP
630   Infile$="A01F2"
640   OUTPUT 2;Infile$;
650   INPUT "Enter input filename:",Infile$
660   ASSIGN @Ifile TO Infile$               ! Open input file
670   OFF ERROR
680   GOTO Inp2
690   !
700 No_file:   !
710      IF ERRN=56 THEN
720         PRINT CHR$(137);"File ";Infile$;" does not exist."
730         PRINT CHR$(136)
740         BEEP
750         WAIT .75
760         GOTO Inp1
770      ELSE
780         PRINT CHR$(137);CHR$(130)
790         PRINT "Error";ERRN;"occured while opening ";Infile$
800         PRINT CHR$(128);CHR$(139)
810         STOP
820      END IF
830   !
840   !
850 Inp2:  !
860   Pos=LEN(Infile$)-3                     ! Compute output filenames
870   Name$=Infile$[Pos]
880   Outfile1$="EINL_"&Name$                ! End point INL
890   Outfile2$="HINL_"&Name$                ! Histogram INL
900   Outfile3$="DNL_"&Name$                 ! Histogram DNL
910   Outfile4$="ABS_"&Name$                 ! Total Error
920   OUTPUT 2;Outfile1$;
930   INPUT "Enter filename for end point INL error:",Outfile1$
940   OUTPUT 2;Outfile2$;
950   INPUT "Enter finlename for histogram INL error:",Outfile2$
960   OUTPUT 2;Outfile3$;
970   INPUT "Enter filename for histogram DNL error:",Outfile3$
980   OUTPUT 2;Outfile4$;
990   INPUT "Enter filename for absolute error:",Outfile4$
1000  !
1010  !
1020  ENTER @Ifile;Samples                    ! Read in data
1030  REDIM Data(1,Samples-1),Abs(1,Samples-1)
1040  ENTER @Ifile;Data(*)
1050  !
1060  ENTER @Ifile;Osamples
1070  REDIM Bin(1,Osamples-1),Dnl(1,Osamples-1)
1080  REDIM Inl(1,Osamples-1),Hinl(1,Osamples-1)
1090  ENTER @Ifile;Bin(*)
1100  ENTER @Ifile;Vref;Res;Vtrans(*)
1110  !
1120  !
1130 Constants: !
1140  Vmin=0
1150  Vmax=Vref
1160  Lsb=Vref/2^Res
1170  Vf=Vtrans(0)                          ! Voltage to cause first transition
1180  Vl=Vtrans(3)                          ! Voltage to cause last tensition
1190  Cf=Vtrans(1)+1                        ! Count after first transition
```

```
1200  Cl=Vtrans(2)                               ! Count after last transition
1210  !
1220  IF (Cf<>INT(Cf) AND Cl<>INT(Cl)) OR (Cf=0 AND Cl=0) THEN
1230     Cf=1                                    ! If processing an old data file,
1240     Cl=255                                  !    assign default values to
1250     Old=1                                   !    Cf & Cl and set a flag
1260  END IF
1270  !
1280  !
1290  !
1300  PRINT
1310  PRINT "Calculating offset and gain errors."
1320  !  Offset error
1330  Voffset=Vf+(Vf-Vl)/(Cl-Cf)+Lsb/2
1340  Coffset=Voffset/Lsb
1350  !
1360  !
1370  !  Gain error
1380  Vgain=Vref-2*Lsb+Vf-Vl
1390  Cgain=Vgain/Lsb
1400  !
1410  !
1420  !
1430  PRINT "Calculating absolute error."
1440  Amax=-65536
1450  Amin=65536
1460  FOR X=0 TO Samples-1                      ! Absolute error
1470     Abs(0,X)=Data(0,X)
1480     Abs(1,X)=Data(1,X)-(Data(0,X)-Vmin)*2^Res/(Vmax-Vmin)
1490     IF Abs(1,X)>Amax THEN
1500        Amax=Abs(1,X)
1510        Max2=Abs(0,X)
1520     END IF
1530     IF Abs(1,X)<Amin THEN
1540        Amin=Abs(1,X)
1550        Min2=Abs(0,X)
1560     END IF
1570  NEXT X
1580  !
1590  !
1600  !
1610  PRINT "Correcting raw data."
1620  Alpha=Lsb*(Cl-Cf)/(Vl-Vf)                 ! Data correction constants to
1630  Beta=Lsb*((Vl*Cf-Vf*Cl)/(Vl-Vf)-1/2)      !   eliminate offset & gain errs
1640  !
1650  FOR I=0 TO Samples-1                      ! Correct raw voltage data
1660     Data(0,I)=Alpha*Data(0,I)+Beta
1670 NEXT I
1680  !
1690  FOR I=0 TO 3                               ! Correct the transition voltages
1700     Vtrans(I)=Alpha*Vtrans(I)+Beta
1710  NEXT I
1720  !
1730  !
1740  !
1750  PRINT "Calculating end point integral nonlinearity error."
1760  Slope=254/(Vtrans(3)-Vtrans(0))
1770  Imax=-65536
1780  Imin=65536
1790  Prev=0
```

```
1800  I=1
1810  FOR X=0 TO Samplas-1                      ! Intagral Nonlinearity Error
1820     IF Oata(1,X)>Prav THEN
1830        Inl(0,I)=Oata(1,X)
1840        Inl(1,I)=Oata(1,X)-Slope*(Oata(0,X)-Lsb/2)-1
1850        IF Inl(1,I)>Imax THEN
1860           Imax=Inl(1,I)
1870           Max1=Inl(0,I)
1880        ENO IF
1890        IF Inl(1,I)<Imin THEN
1900           Imin=Inl(1,I)
1910           Min1=Inl(0,I)
1920        ENO IF
1930        Prev=Oete(1,X)
1940        I=I+1
1950     ENO IF
1960  NEXT X
1970  !
1980  !
1990  !
2000  PRINT "Calculeting histogram integrel nonlinaarity arror."
2010  Eno=0
2020  FOR X=1 TO Oaamples-2
2030     Eno=Eno+Bin(1,X)
2040  NEXT X
2050  Eno=Eno/(Osamples-2)
2060  !
2070  Slopa=0
2080  FOR X=1 TO Oaamplas-2
2090     Slope=Slope+8in(1,X)
2100  NEXT X
2110  Slopa=Slope/(Eno*(Osamplas-2))
2120  !
2130  Hmax=-65536
2140  Hmin=65536
2150  FOR X=0 TO Osamplas-1
2160     Hinl(0,X)=X
2170     Hinl(1,X)=0
2180     FOR Y=0 TO X-1
2190        Hinl(1,X)=Hinl(1,X)+8in(1,Y)
2200     NEXT Y
2210     Hinl(1,X)=(Hinl(1,X)-8in(1,0))/Eno+(1-X)*Slopa
2220     IF Hinl(1,X)>Hmax THEN
2230        Hmax=Hinl(1,X)
2240        Mex3=Hinl(0,X)
2250     ENO IF
2260     IF Hinl(1,X)<Hmin THEN
2270        Hmin=Hinl(1,X)
2280        Min3=Hinl(0,X)
2290     ENO IF
2300  NEXT X
2310  !
2320  !
2330  !
2340  PRINT "Calculating histogram diffarantial nonlinaarity error."
2350  Omax=-65536
2360  Omin=65536
2370  Eno=10*2^10/Osamples                       ! OAC rasolution is 2^10
2380  !                                            AOC resolution is 2^8
2390  Eno0=Eno/2                                 ! ENO for first bin (1/2 tha normal)
```

```
2400  Eno1=3/2*Eno                        ! ENO for last bin (3/2 the normal)
2410  FOR X=0 TO Osamples-1               ! Differential Nonlinearity Error
2420     Onl(0,X)=X
2430     Onl(1,X)=Bin(1,X)/Eno-1
2440     IF X=0 THEN Onl(1,X)=Bin(1,X)/Eno0-1          ! Calculate first bin
2450     IF X=Osamples-1 THEN Onl(1,X)=Bin(1,X)/Eno1-1   ! Calculate last bin
2460     IF Onl(1,X)>Omax THEN
2470        Omax=Onl(1,X)
2480        Bmax=Onl(0,X)
2490     END IF
2500     IF Onl(1,X)<Omin THEN
2510        Omin=Onl(1,X)
2520        Bmin=Onl(0,X)
2530     END IF
2540  NEXT X
2550  !
2560  !
2570  !
2580  PRINT "Searching for non-monotonic values and missing code."
2590  Non_monotonic=0                     ! Non-monotonic error counter
2600  Missing_code=0                      ! Missing code counter
2610  Prev=0
2620  FOR X=0 TO Samples-1
2630     IF Data(1,X)<Prev THEN           ! Check for non-monotonicities
2640        Non_monotonic=Non_monotonic+1
2650        Non_mono(0)=Data(0,X)         ! Stores the last occurrence
2660        Non_mono(1)=Data(1,X)
2670     END IF
2680     Prev=Data(1,X)
2690  NEXT X
2700  !
2710  FOR X=0 TO Osamples-1
2720     IF Bin(1,X)=0 THEN               ! Check for missing code
2730        Missing_code=Missing_code+1
2740        Code(0)=Data(0,4*X)           ! Stores the last occurrence
2750        Code(1)=Bin(0,X)
2760     END IF
2770  NEXT X
2780  !
2790  !
2800  !
2810 Print_results: BEEP
2820  Out$="PRINTER"
2830  OUTPUT 2;Out$;
2840  INPUT "List device (SCREEN/PRINTER):",Out$
2850  IF UPC$(Out$[1,1])="P" THEN PRINTER IS 701
2860  !
2870 Maxform:   IMAGE "     Maximum: ", 3D.3D, " LSBs at bit ", 3D
2880 Minform:   IMAGE "     Minimum: ", 3D.3D, " LSBs at bit ", 3D
2890  !
2900  PRINT TAB(20);"ERROR CALCULATIONS"
2910  IF UPC$(Out$[1,1])="P" THEN
2920     PRINT
2930  ELSE
2940     PRINT CHR$(138)
2950  END IF
2960  PRINT
2970  PRINT "Input Filename: ";Infile$
2980  PRINT
2990  PRINT
```

```
3000  Cgain=PROUND(Cgain,-3)                    ! Round data (3 places)
3010  Coffsat=PROUND(Coffset,-3)
3020  Alpha=PROUND(Alpha,-5)
3030  Beta=PROUND(Beta,-5)
3040  PRINT "Gain Error:   ";Cgain;"LSBs";TAB(40);"Alpha: ";Alpha
3050  PRINT "Offsat Error: ";Coffsat;"LSBs";TAB(40);"Bata:  ";Bata
3060  PRINT
3070  PRINT "End point Intagral Nonlinaarity Error ------  Fila: ";Outfile1$
3080  PRINT USING Maxform;Imax,Max1
3090  PRINT USING Minform;Imin,Min1
3100  PRINT
3110  PRINT "Histogram Intagral Nonlinaarity Error ------  Fila: ";Outfile2$
3120  PRINT USING Maxform;Hmax,Max3
3130  PRINT USING Minform;Hmin,Min3
3140  PRINT
3150  PRINT "Histogram Diffarantial Nonlinaarity Error --  File: ";Outfile3$
3160  PRINT USING Maxform;Dmax,Bmax
3170  PRINT USING Minform;Dmin,Bmin
3180  PRINT
3190  PRINT "Total Error ---------------------------------  Fila: ";Outfile4$
3200  Amax=PROUND(Amax,-3)                      ! Round data
3210  Amin=PROUND(Amin,-3)
3220  Max2=PROUND(Max2,-4)
3230  Min2=PROUND(Min2,-4)
3240  PRINT "     Maximum: ";Amax;"LSBs at";Max2;"v"
3250  PRINT "     Minimum: ";Amin;"LSBs at";Min2;"v"
3260  PRINT
3270  IF Old THEN PRINT "Old data fila, dafault and count valuas used:"
3280  Vf=PROUND(Vf,-4)
3290  Vl=PROUND(Vl,-4)
3300  PRINT Cf;"->";Cf+1;":  ";Vf;"v";TAB(40);Cl-1;"->";Cl;":  ";Vl;"v"
3310  !
3320  PRINT
3330  PRINT "Bin 0:";Bin(1,0);TAB(40);"Bin 255:";Bin(1,255)
3340  !
3350  IF Non_monotonic THEN
3360     PRINT
3370     PRINT "Non-monotonicity occurrad";Non_monotonic;"times."
3380     Non_mono(0)=PROUND(Non_mono(0),-4)
3390     PRINT "Tha last occurranca was";Non_mono(1);"at";Non_mono(0);"v"
3400  END IF
3410  !
3420  IF Missing_coda THEN
3430     PRINT
3440     PRINT "Missing coda occurred";Missing_coda;"timas."
3450     Coda(0)=PROUND(Coda(0),-4)
3460     PRINT "Tha last occurranca was";Code(1);"et about";Coda(0);"v"
3470  END IF
3480  !
3490 Ejact:  IF UPC$(Out$[1,1])="P" THEN PRINT CHR$(12)
3500  PRINTER IS 1
3510  !
3520  !
3530 File:   !
3540  ON ERROR GOSUB Fila_axists
3550  !
3560  Outfila$=Outfila1$                          ! Writa end point Integrel Error
3570  Racords=INT(Dsamples/16)+1
3580  CREATE BDAT Outfile1$,Records
3590  ASSIGN @Dfila TO Outfila1$
```

```
3600  OUTPUT @Ofile;Osamples;Inl(*)
3610  ASSIGN @Ofile TO *
3620  !
3630  Outfile$=Outfile2$                    ! Write histogram Integral Error
3640  Records=INT(Osamples/16)+1
3650  CREATE BDAT Outfile2$,Records
3660  ASSIGN @Ofile TO Outfile2$
3670  OUTPUT @Ofile;Osamples;Hinl(*)
3680  ASSIGN @Ofile TO *
3690  !
3700  Outfile$=Outfile3$                    ! Write histogram Differential Err
3710  Records=INT((Osamples)/16)+1
3720  CREATE BDAT Outfile3$,Records
3730  ASSIGN @Ofile TO Outfile3$
3740  OUTPUT @Ofile;Osamples;Dnl(*)
3750  ASSIGN @Ofile TO *
3760  !
3770  Outfile$=Outfile4$                    ! Write Absolute Error
3780  Records=INT(Samples/16)+1
3790  CREATE BDAT Outfile4$,Records
3800  ASSIGN @Ofile TO Outfile4$
3810  OUTPUT @Ofile;Samples;Abs(*)
3820  ASSIGN @Ofile TO *
3830  GOTO Exit
3840  !
3850  !
3860 File_exists:   ! Purge file if it already exists
3870      IF ERRN=54 THEN
3880         PURGE Outfile$
3890         RETURN
3900      ELSE
3910         PRINT CHR$(137);CHR$(130)
3920         PRINT "Error";ERRN;"occured when writing to ";Outfile$
3930         PRINT CHR$(128);CHR$(139)
3940         STOP
3950      END IF
3960  !
3970  !
3980 Exit:  !
3990  ASSIGN @Ifile TO *                    ! Close file
4000  !
4010  BEEP
4020  PRINT CHR$(139)
4030  DISP "Program terminated."
4040 !WAIT .75
4050 !DISP "Loading CRUNCH program."
4060 !LOAD "CRUNCH"
4070  END
```

```
10    ! ******************************************************************
20    ! **      for the 9236          CRUNCH                ver 3.236     **
30    ! ******************************************************************
40    ! **                                                                **
50    ! **      This program takes the data file created by ERROR and     **
60    ! **  streamlines the data by finding the relative maximums and     **
70    ! **  minimums and throws the rest of the data away.  The output    **
80    ! **  file is 1/4 the size of the input file (256 points compared   **
90    ! **  to 1024).  The output file can be read by PLOT if a graph     **
100   ! **  is desired. Version 3.0 puts X axis in terms of ADC output.   **
110   ! **                                                                **
120   ! **  MICHAEL PANKRATZ                      13 July 1987            **
130   ! **                                                                **
140   ! ******************************************************************
150   !
160   DIM Inl(1,1024),Out(1,256)
170   !
180   !
190   PRINTER IS 1
200   PRINT CHR$(12);                          ! Clear screen
210   PRINT TAB(27);CHR$(138);"CRUNCH ABSOLUTE ERROR DATA"
220   PRINT
230   PRINT
240   PRINT
250   !
260   !
270   Infile$="ABS_01F2"
280 Inp1: !
290   BEEP
300   ON ERROR GOTO No_file
310   OUTPUT 2;Infile$;
320   INPUT "Enter input INL error filename:",Infile$
330   ASSIGN @Ifile TO Infile$              ! Open input file
340   OFF ERROR
350   GOTO Inp2
360   !
370 No_file:    !
380       IF ERRN=56 THEN
390          PRINT CHR$(137);"File ";Infile$;" does not exist."
400          PRINT CHR$(138)
410          BEEP
420          WAIT .75
430          GOTO Inp1
440       ELSE
450          PRINT CHR$(137);CHR$(130)
460          PRINT "Error";ERRN;"occured when opening ";Infile$
470          PRINT CHR$(128);CHR$(139)
480          STOP
490       END IF
500   !
510   !
520 Inp2:  !
530   Outfile$="C"&Infile$
540   OUTPUT 2;Outfile$;
550   INPUT "Enter cruched output filename:",Outfile$
560   !
570   !
580   ENTER @Ifile;Samples                     ! Read input file size
590   REDIM Inl(1,Samples-1)
```

```
600   Samples2=INT(Samples/4)
610   REDIM Out(1,Samples2-1)
620   ENTER @Ifile;In1(*)                     ! Read in the data
630   ASSIGN @Ifile TO *
640   !
650   !
660   PRINT "Crunching..."
670   FOR I=0 TO Samples2-1
680      Out(1,I)=0
690      FOR J=0 TO 3
700         IF ABS(In1(1,I*4+J))<ABS(Out(1,I)) THEN Nxt
710            Out(1,I)=In1(1,I*4+J)        . ! Put max value in 2nd array
720            Out(0,I)=I                    ! Make X axis the ADC output
730 Nxt: NEXT J
740   NEXT I
750   !
760   !
770 Write_data:   !
780   ON ERROR GOSUB File_exists
790   Records=INT((Samples2+1)/16)+1
800   CREATE BDAT Outfile$,Records           ! Create new file
810   ASSIGN @Ofile TO Outfile$
820   OUTPUT @Ofile;Samples2;Out(*)          ! Write data to file
830   GOTO Exit
840   !
850 File_exists:   !
860      IF ERRN=54 THEN
870         PURGE Outfile$
880         RETURN
890      ELSE
900         PRINT CHR$(137);CHR$(130)
910         PRINT "Error ";ERRN;"occured when writing to ";Outfile$
920         PRINT CHR$(128);CHR$(139)
930         STOP
940      END IF
950   !
960   !
970 Exit:   !
980   PRINT CHR$(139);
990   ASSIGN @Ifile TO *                     ! Close files
1000  ASSIGN @Ofile TO *
1010  BEEP
1020  DISP "Program terminated."
1030  END
```

Appendix B

```
*****************************************************
*
*  SOURCE FILE:    dynrcf8.src
*
*  FUNCTION:       Program.
*
*  DESCRIPTION:
*
*   This program does A/D conversions in a loop
*   that causes the HC11's ADC to have a sampling
*   rate of 4.0 kHz.  The A/D conversions are
*   outputed to PORTB.  This result is picked up
*   by the IBM PCXT and is stored in a data file
*   for later processing.
*
*   The A/D configuration is stored in location
*   00.  The channel converted is channel one and
*   the result is read from from ADR3 which is
*   found at memory location $1034.
*
*   This program is for use with an 8 MHz crystal
*   on the EVB board.
*
*  DOCUMENTATION
*  FILES:           None.
*
*  ARGUMENTS:       A/D configuration is stored in
*                   location 00 prior to execution.
*
*  RETURN:          None.
*
*  FUNCTIONS
*  CALLED:          None.
*
*  AUTHOR:          Jeffrey C. Daniels
*
*  DATE CREATED:    7-28-87
*
*  REVISIONS:       This program was created from
*                   dynrc.src.
*
*****************************************************

ADR0    equ $1031
ADR1    equ $1032
ADR2    equ $1033
ADR3    equ $1034
PORTA   equ $1000
PORTB   equ $1004
LOOPNO  equ 76
```

```
              org   $c400
              ldaa  $1039       Power up A/D.
              oraa #$80
              staa  $1039

              ldy   #PORTA      Point inx to PORTA
              bset 0,y $20

              ldaa  $00         Put A/D configuration
*                               accumulator A.

CONVERT       staa  $1030       Initiate conversion.

              ldx   #LOOPNO
DELAY         dex               Wait for conversion to
              bne   DELAY       be completed.

              ldab  ADR3
              stab  PORTB

              bclr  0,y $20     Send pulse to IBM that
              bset  0,y $20     data is ready.

              ldx   #$00        Waste 10 clock cycles.
              ldx   #$00
              nop
              nop

              bra   CONVERT
```

```
**************************************************
*
*  SOURCE FILE:    dyrcf8.src
*
*  FUNCTION:       Program.
*
*  DESCRIPTION:
*
*   This program does A/D conversions in a loop
*   that causes the HC11's ADC to have a sampling
*   rate of 4.0 kHz.  The A/D conversions are
*   outputed to PORTB.  This result is picked up
*   by the IBM PCXT and is stored in a data file
*   for later processing.
*
*   The A/D configuration is stored in location
*   00.  The channel converted is channel one and
*   the result is read from from ADR4 which is
*   found at memory location $1034.
*
*   This program is for use with an 8 MHz crystal
*   on the EVB board.
*
*  DOCUMENTATION
*  FILES:           None.
*
*  ARGUMENTS:       A/D configuration is stored in
*                   location 00 prior to execution.
*
*  RETURN:          None.
*
*  FUNCTIONS
*  CALLED:          None.
*
*  AUTHOR:          Jeffrey C. Daniels
*
*  DATE CREATED:    7-28-87
*
*  REVISIONS:       This program was created from
*                   dyrc.src.
*
**************************************************

ADR0    equ $1031
ADR1    equ $1032
ADR2    equ $1033
ADR3    equ $1034
PORTA   equ $1000
PORTB   equ $1004
LOOPNO  equ 76
```

```
          org   $c500
          ldaa  $1039        Power up A/D.
          oraa #$c0
          staa  $1039

          ldy  #PORTA        Point inx to PORTA
          bset 0,y $20

          ldaa  $00          Put A/D configuration
*                            accumulator A.

CONVERT   staa  $1030        Initiate conversion.

          ldx   #LOOPNO
DELAY     dex                Wait for conversion to
          bne   DELAY        be completed.

          ldab  ADR3
          stab  PORTB

          bclr  0,y $20      Send pulse to IBM that
          bset  0,y $20      data is ready.

          ldx   #$00         Waste 10 clock cycles.
          ldx   #$00
          nop
          nop

          bra   CONVERT
```

```
*****************************************************
*
*  SOURCE FILE:    dynrcf2.src
*
*  FUNCTION:       Program.
*
*  DESCRIPTION:
*
*   This program does A/D conversions in a loop
*   that causes the HC11's ADC to have a sampling
*   rate of 4.0 kHz.  The A/D conversions are
*   outputed to PORTB.  This result is picked up
*   by the IBM PCXT and is stored in a data file
*   for later processing.
*
*   The A/D configuration is stored in location
*   00.  The channel converted is channel one and
*   the result is read from from ADR2 which is
*   found at memory location $1033.
*
*   This program is for use with an 2 MHz crystal
*   on the EVB board.
*
*  DOCUMENTATION
*  FILES:           None.
*
*  ARGUMENTS:       A/D configuration is stored in
*                   location 00 prior to execution.
*
*  RETURN:          None.
*
*  FUNCTIONS
*  CALLED:          None.
*
*  AUTHOR:          Jeffrey C. Daniels
*
*  DATE CREATED:    7-28-87
*
*  REVISIONS:       This program was created from
*                   dynrc.src.
*
*****************************************************

ADR0   equ $1031
ADR1   equ $1032
ADR2   equ $1033
ADR3   equ $1034
PORTA  equ $1000
PORTB  equ $1004
LOOPNO equ 15
```

```
                org    $c600
                ldaa   $1039         Power up A/D.
                oraa  #$80
                staa   $1039

                ldy   #PORTA         Point inx to PORTA
                bset 0,y $20

                ldx   #LOOPNO        Put delay loop length
                stx   $01            at locations 01 and 02.

                ldaa   $00           Put A/D configuration
*                                    accumulator A.

CONVERT         staa   $1030         Initiate conversion.

                ldx    $01
DELAY           dex                  Wait for conversion to
                bne    DELAY         be completed.

                ldab   ADR2
                stab   PORTB

                bclr   0,y $20       Send pulse to IBM that
                bset   0,y $20       data is ready.

                bra    CONVERT
```

```
*****************************************************
*
*  SOURCE FILE:   dyrcf2.src
*
*  FUNCTION:      Program.
*
*  DESCRIPTION:
*
*   This program does A/D conversions in a loop
*   that causes the HC11's ADC to have a sampling
*   rate of 4.0 kHz.  The A/D conversions are
*   outputed to PORTB.  This result is picked up
*   by the IBM PCXT and is stored in a data file
*   for later processing.
*
*   The A/D configuration is stored in location
*   00.  The channel converted is channel one and
*   the result is read from from ADR2 which is
*   found at memory location $1033.
*
*   This program is for use with an 2 MHz crystal
*   on the EVB board.
*
*  DOCUMENTATION
*  FILES:           None.
*
*  ARGUMENTS:       A/D configuration is stored in
*                   location 00 prior to execution.
*
*  RETURN:          None.
*
*  FUNCTIONS
*  CALLED:          None.
*
*  AUTHOR:          Jeffrey C. Daniels
*
*  DATE CREATED:    7-28-87
*
*  REVISIONS:       This program was created from
*                   dyrc.src.
*
*****************************************************

ADR0   equ $1031
ADR1   equ $1032
ADR2   equ $1033
ADR3   equ $1034
PORTA  equ $1000
PORTB  equ $1004
LOOPNO equ 15
```

```
         org   $c700
         ldaa  $1039       Power up A/D.
         oraa #$c0
         staa  $1039

         ldy  #PORTA       Point inx to PORTA
         bset 0,y $20

         ldx  #LOOPNO      Put delay length at
         stx  $01          locations 01 and 02.

         ldaa  $00         Put A/D configuration
*                          accumulator A.

CONVERT  staa  $1030       Initiate conversion.

         ldx   $01
DELAY    dex               Wait for conversion to
         bne   DELAY       be completed.

         ldab  ADR3
         stab  PORTB

         bclr  0,y $20     Send pulse to IBM that
         bset  0,y $20     data is ready.

         bra   CONVERT
```

GETDAT.EXE Make File

```
getdat.obj : getdat.c local.h
           msc getdat;

getdat.exe : getdat.obj
           link/stack:5000 getdat;
```

DYHIS.EXE Make File

```
dyhis.obj : dyhis.c local.h
    msc dyhis;

dyhis.exe : dyhis.obj
    link dyhis;
```

DYFFT.EXE Make File

```
cadd.obj : cadd.c complex.h
         msc/AL cadd;

csub.obj : csub.c complex.h
         msc/AL csub;

cmult.obj : cmult.c complex.h
          msc/AL cmult;

cdiv.obj : cdiv.c complex.h
         msc/AL cdiv;

cexpon.obj : cexpon.c complex.h
         msc/AL cexpon;

cmplx.obj : cmplx.c complex.h
         msc/AL cmplx;

cneg.obj : cneg.c complex.h
         msc/AL cneg;

cmag.obj : cmag.c complex.h
         msc/AL cmag;

cmagsq.obj : cmagsq.c complex.h
         msc/AL cmagsq;
```

```
cmath.lib : cmagsq.obj cmag.obj cneg.obj cmplx.obj \
            cexpon.obj cdiv.obj cmult.obj csub.obj \
            cadd.obj
         lib cmath-+cadd;
         lib cmath-+csub;
         lib cmath-+cmult;
         lib cmath-+cdiv;
         lib cmath-+cexpon;
         lib cmath-+cmplx;
         lib cmath-+cneg;
         lib cmath-+cmag;
         lib cmath-+cmagsq;

fft.obj : fft.c cmath.h complex.h local.h
      msc/AL fft;

normal.obj : normal.c complex.h local.h
         msc/AL normal;

window.obj : window.c cmath.h complex.h local.h
         msc/AL window;

dyfft.obj : dyfft.c cmath.h complex.h local.h
        msc/AL dyfft;

dyfft.exe : dyfft.obj normal.obj window.obj fft.obj cmath.lib
    link dyfft normal window fft,,,cmath.lib;
```

HARMON1.EXE Make File

```
cadd.obj : cadd.c complex.h
         msc/AL cadd;

csub.obj : csub.c complex.h
         msc/AL csub;

cmult.obj : cmult.c complex.h
          msc/AL cmult;

cdiv.obj : cdiv.c complex.h
         msc/AL cdiv;

cexpon.obj : cexpon.c complex.h
         msc/AL cexpon;

cmplx.obj : cmplx.c complex.h
         msc/AL cmplx;

cneg.obj : cneg.c complex.h
         msc/AL cneg;
```

```
cmag.obj : cmag.c complex.h
         msc/AL cmag;

cmagsq.obj : cmagsq.c complex.h
         msc/AL cmagsq;

cmath.lib : cmagsq.obj cmag.obj cneg.obj cmplx.obj \
            cexpon.obj cdiv.obj cmult.obj csub.obj \
            cadd.obj
         lib cmath-+cadd;
         lib cmath-+csub;
         lib cmath-+cmult;
         lib cmath-+cdiv;
         lib cmath-+cexpon;
         lib cmath-+cmplx;
         lib cmath-+cneg;
         lib cmath-+cmag;
         lib cmath-+cmagsq;

fft.obj : fft.c cmath.h complex.h local.h
      msc/AL fft;

window.obj : window.c cmath.h complex.h local.h
         msc/AL window;

harmonl.obj : harmonl.c local.h cmath.h complex.h
   msc/AL harmonl;

harmonl.exe : harmonl.obj fft.obj window.obj cmath.lib
  link harmonl fft window,,,cmath.lib;
```

QNTZ1.EXE Make File

```
cadd.obj : cadd.c complex.h
         msc/AL cadd;

csub.obj : csub.c complex.h
         msc/AL csub;

cmult.obj : cmult.c complex.h
          msc/AL cmult;

cdiv.obj : cdiv.c complex.h
         msc/AL cdiv;

cexpon.obj : cexpon.c complex.h
         msc/AL cexpon;

cmplx.obj : cmplx.c complex.h
         msc/AL cmplx;
```

```
cneg.obj : cneg.c complex.h
         msc/AL cneg;

cmag.obj : cmag.c complex.h
         msc/AL cmag;

cmagsq.obj : cmagsq.c complex.h
         msc/AL cmagsq;

cmath.lib : cmagsq.obj cmag.obj cneg.obj cmplx.obj \
            cexpon.obj cdiv.obj cmult.obj csub.obj \
            cadd.obj
         lib cmath-+cadd;
         lib cmath-+csub;
         lib cmath-+cmult;
         lib cmath-+cdiv;
         lib cmath-+cexpon;
         lib cmath-+cmplx;
         lib cmath-+cneg;
         lib cmath-+cmag;
         lib cmath-+cmagsq;

fft.obj : fft.c cmath.h complex.h local.h
      msc/AL fft;

window.obj : window.c cmath.h complex.h local.h
         msc/AL window;

qntzl.obj : qntzl.c local.h cmath.h complex.h
   msc/AL qntzl;

qntzl.exe : qntzl.obj fft.obj window.obj cmath.h
  link qntzl fft window,,,cmath.lib;
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    getfft.c
 *
 *  FUNCTION:       main program
 *
 *  DESCRIPTION:    This program receives the data for the
 *                  fft and histogram tests for dynamic
 *                  testing from the ppi in IBM PCXT.
 *                  Various information is prompted from
 *                  the user and all information is then
 *                  stored in a user specified files in
 *                   binary format for later processing.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      None.
 *
 *  RETURN:         Binary files containing all valid
 *                  information.
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   8-3-87
 *
 *  REVISIONS:      8-3-87 This program was created from
 *                         getfft.c and gethis.c.
 *
*****************************************************************/

#include <stdio.h>
#include <conio.h>
#include "local.h"

int configuration,
    num_pts = NUM_DFT_POINTS;

int intdata[NUM_DFT_POINTS];

char filename[STRING_LEN],
     mode[MODE_LEN],
     chip_number[CHIP_NO_LEN],
     date[DATE_LEN],
     lot_number[LOT_NO_LEN];

float inp_float;
```

```
double samp_freq,
       clock_freq,
       hist[RESOLUTION];

FILE *out_file;

main()
 {
  register i,j;
  unsigned char data[NUM_DFT_POINTS];


/* Set up ppi communications.                          */

  outp(CONTROL,PPI_CONFIG);


/*****************************************************/
/* Enter data information.                           */
/*****************************************************/

/* Enter the date.                                     */

  printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
  puts("Enter the date is this form - - ");
  puts("mm-dd-yy");
  scanf("%s",date);
  printf("The date is: %s\n\n\n",date);

/* Enter chip number.                                  */

  puts("Enter the chip number of the hc11");
  scanf("%s",chip_number);
  printf("The chip number is : %s\n\n\n",
                              chip_number);

/* Enter lot number of HC11.                           */

  puts("Enter the lot number of the HC11");
  scanf("%s",lot_number);
  printf("The lot number is : %s\n\n\n",
                              lot_number);


/* Enter the clock frequency.                          */

  puts("Enter the clock frequency for the");
  puts(" test system in MHz.");
  scanf("%e",&inp_float);
  clock_freq = (double)inp_float;
  printf("The clock frequency is: %f\n\n\n",
                               clock_freq);
```

```
/* Enter the sampling frequency.                      */

  puts("Enter the sampling frequency of the HC11");
  puts("in Hertz.");
  scanf("%e",&inp_float);
  samp_freq = (double)inp_float;
  printf("The sampling freq. is: %f\n\n\n",
                               samp_freq);

  FOREVER
   {

/* Enter the mode of the HC11.                        */

    puts("Enter the mode of the HC11.");
    puts("An example - - 00");
    scanf("%s",mode);
    printf("The mode is: %s\n\n\n",mode);


/* Take data.                                         */

    puts("Set up fft input sine wave and hit ");
    puts("RETURN to start data acquisition.\n");

    getch();

/* Throw out 10 samples.                              */

    for ( i = 0; i < 10; i++ )
     inp(PORTB);

/* Take valid data.                                   */

    for( i = -1; i++ < NUM_DFT_POINTS;)
     {
      while( ! (0x01 & (int)inp(PORTA) ))
       ;
      data[i] = inp(PORTB);
     }

    puts("Some data");
    for ( i = 0; i < 20; i++ )
     printf("%i\n",(int)data[i]);
    printf("\n\n\n\n");
```

```
/* Enter output filename.                           */

    puts("Enter the fft output data filename.");
    puts("An example is:  INRC01F8.OUT");
    scanf("%s",filename);
    printf("The output filename is: %s\n\n\n",
                                     filename);


/* Convert data from character to integer.          */

    for ( i = 0; i < NUM_DFT_POINTS; i++ )
     intdata[i] = (int)data[i];

/* Write out data and information to output         */
/* file in binary format.                           */

    out_file = fopen(filename,"w+b");

    fwrite(date,sizeof(char),DATE_LEN,out_file);

    fwrite(chip_number,sizeof(char),CHIP_NO_LEN,
                                       out_file);

    fwrite(lot_number,sizeof(char),LOT_NO_LEN,
                                       out_file);

    fwrite(mode,sizeof(char),MODE_LEN,out_file);

    fwrite((char *)&clock_freq,sizeof(double),1,
                                      out_file);

    fwrite((char *)&samp_freq,sizeof(double),1,
                                      out_file);

    fwrite((char *)&num_pts,sizeof(int),1,
                                      out_file);

    fwrite((char *)intdata,sizeof(int),
                       NUM_DFT_POINTS,out_file);

    fclose(out_file);


/**************************************************/
/* Take histogram data.                           */
/**************************************************/

    puts("Set up histogram input sine wave and hit");
    puts("RETURN to start data accquisition.\n\n\n");

    getch();
```

```
/* Zero out histogram array                              */

    for( i = 0; i < RESOLUTION; i++ )
      hist[i] = 0.0;

/* Throw out 10 samples.                                 */

    for ( i = 0; i < 10; i++ )
     inp(PORTB);

/* Take valid data.                                      */

    for( j = 0; j < ( NUM_HIS_POINTS / LOOP_SIZE );
                                                  j++ )
     {
      printf(" %i",j);
      for( i = -1; i++ < LOOP_SIZE;)
       {
        while( ! (0x01 & (int)inp(PORTA) ))
        ;
        data[i] = inp(PORTB);
       }

      for( i = 0; i < LOOP_SIZE; i++ )
       {
        hist[(int)data[i]] = hist[(int)data[i]] + 1;
       }
     }

    puts("Some data");
    for ( i = 0; i < RESOLUTION; i++ )
     printf("%i %f\n",i,hist[i]);

/* Enter output filename.                                */

    puts("Enter the histogram output data filename.");
    puts("An example is:  hnrc01f8.out");
    scanf("%s",filename);
    printf("The output filename is: %s\n\n\n",filename);


/* Write out data and information to output              */
/* file in binary format.                                */

    out_file = fopen(filename,"w+b");

    fwrite(date,sizeof(char),DATE_LEN,out_file);

    fwrite(chip_number,sizeof(char),CHIP_NO_LEN,
                                    out_file);
      fwrite(lot_number,sizeof(char),LOT_NO_LEN,
                                     out_file);

      fwrite(mode,sizeof(char),MODE_LEN,out_file);

      fwrite((char *)&clock_freq,sizeof(double),1,
                                      out_file);

      fwrite((char *)&samp_freq,sizeof(double),1,
                                      out_file);

      fwrite((char *)&num_pts,sizeof(int),1,
                                      out_file);

      fwrite((char *)hist,sizeof(double),
                              RESOLUTION,out_file);

      fclose(out_file);

    }

  exit(0);

}
```

```
/*****************************************************************
 *
 *   SOURCE FILE:    dyhis.c
 *
 *   FUNCTION:       main()
 *
 *   DESCRIPTION:    This program reads in the data and
 *                   information taken in a dynamic histogram
 *                   test of the hcll ADC.  The data
 *                   manipulated with methods described by
 *                   Doerfler.
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *   ARGUMENTS:      None.
 *
 *   RETURN:         ascii file containing data and
 *                   information
 *
 *   FUNCTIONS
 *   CALLED:         none.
 *
 *   AUTHOR:         Jeffrey C. Daniels
 *
 *   DATE CREATED:   6-30-87
 *
 *   REVISIONS:      None.
 *
*****************************************************************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "local.h"

char in_filename[STRING_LEN],
     out_filename[STRING_LEN],
     mode[MODE_LEN],
     chip_number[CHIP_NO_LEN],
     date[DATE_LEN],
     lot_number[LOT_NO_LEN];

double samp_freq,
       fund_freq,
       clock_freq,
       max_mag;
```

```
double hist[RESOLUTION],
       voltage[RESOLUTION],
       diff[RESOLUTION],
       cum_his,
       lsb;

int num_pts;

FILE *in_file,
     *out_file;

main()
 {
  register i;

  puts("\n\nEnter the input filename.");
  scanf("%s",in_filename);

/* Open file to read in data, stop program if   */
/* file cannot be opened.                       */

  if(( in_file = fopen(in_filename,"r+b")) == NULL )
   {
    printf("\n\n %s could not be opened or doesn't",
           " exist.\n\n",in_filename);
    exit(1);
   }

/* Read in data and information.                */

  fread(date,sizeof(char),DATE_LEN,in_file);
printf("date = %s\n\n",date);

  fread(chip_number,sizeof(char),CHIP_NO_LEN,
                                   in_file);
printf("The chip number is %s\n\n",chip_number);

  fread(lot_number,sizeof(char),LOT_NO_LEN,in_file);
printf("The lot number is %s\n\n",lot_number);

  fread(mode,sizeof(char),MODE_LEN,in_file);
printf("The mode is %s\n\n",mode);

  fread((char *)&clock_freq,sizeof(double),1,in_file);
printf("clock_freq is %f\n\n",clock_freq);

  fread((char *)&samp_freq,sizeof(double),1,in_file);
printf("samp_freq = %f\n\n",samp_freq);

  fread((char *)&num_pts,sizeof(int),1,in_file);
printf("num_pts = %i\n\n",num_pts);
```

```
  fread((char *)hist,sizeof(double),num_pts,in_file);


/* For this algorithm the amplitude of the input  */
/* waveform is normalized to fall between -1 and  */
/* 1 volts.                                       */

  lsb = 2.0 / pow( (double)2,(double) NUM_BITS);

  cum_his = hist[0];
  voltage[0]  =  -cos(PI * cum_his / NUM_HIS_POINTS);

  for ( i = 1; i < RESOLUTION; i++ )
   {
    cum_his = cum_his + hist[i];
    voltage[i] = -cos( PI * cum_his
                           / NUM_HIS_POINTS );
    diff[i-1] = ( voltage[i] - voltage[i-1] )
                              / lsb  - 1;
   }

/* Write information and data to output file.    */

  printf("\n\nThe input filename was: %s\n\n",
                                   in_filename);

  puts("\n\n\nEnter the filename for the ");
  puts("histogram data.");
  puts("\nAn example - - a:hnr01f8. \n\n");
  scanf("%s",out_filename);

  out_file = fopen(out_filename,"w");

  fprintf(out_file,"%s %iHz %s %s\n",
              out_filename,(int)(samp_freq),
                         chip_number,lot_number);

  for ( i = 0; i < RESOLUTION; i++ )
    fprintf(out_file,"%i %f \n",i,hist[i]);
  fclose(in_file);
  fclose(out_file);

  puts("\n\n\nEnter the filename for the ");
  puts("output data.");
  puts("\nAn example - - a:dnr01f8. \n\n");
  scanf("%s",out_filename);

  out_file = fopen(out_filename,"w");

  fprintf(out_file,"%s %iHz %s %s\n",
              out_filename,(int)(samp_freq),
                         chip_number,lot_number);
```

```
 for ( i = 0; i < RESOLUTION; i++ )
   fprintf(out_file,"%i %f \n",i,diff[i]);
 fclose(in_file);
 fclose(out_file);
 exit(0);
}
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    dyfft.c
 *
 *  FUNCTION:       main()
 *
 *  DESCRIPTION:    This program reads in the data and
 *                  information taken in a dynamic test of
 *                  the hcll ADC.  The data is first
 *                  normalized to a range between 0 and 1,
 *                  windowed with a Von Honn window, taken
 *                  through a fast fourier transform, and
 *                  then the log magnitude is taken.  This
 *                  final result is then placed into an
 *                  output file to be plotted.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      None.
 *
 *  RETURN:         ascii file containing data and
 *                  information
 *
 *  FUNCTIONS
 *  CALLED:         normalize_data();
 *                  window_data();
 *                  fft();
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-9-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "cmath.h"
#include "complex.h"
#include "local.h"

DCOMPLEX input_data[NUM_DFT_POINTS],
         trans_data[NUM_DFT_POINTS],
         z[NUM_DFT_POINTS];            /* COMPLEX work array */

int data[NUM_DFT_POINTS];
```

```
char in_filename[STRING_LEN],
     out_filename[STRING_LEN],
     mode[MODE_LEN],
     chip_number[CHIP_NO_LEN],
     date[DATE_LEN],
     lot_number[LOT_NO_LEN];

double freqs[NUM_DFT_POINTS];

double samp_freq,
       fund_freq,
       clock_freq,
       max_mag;

int num_pts,
    act_len;

FILE *in_file,
     *out_file;

main()
 {
  register i;

  puts("\n\nEnter the input filename.");
  scanf("%s",in_filename);

/* Open file to read in data, stop program if    */
/* file cannot be opened.                        */

  if(( in_file = fopen(in_filename,"r+b")) == NULL )
   {
    printf("%s could not be opened or doesn't",
           " exist.\n\n",in_filename);
    exit(1);
   }

/* Read in data and information.                 */

  fread(date,sizeof(char),DATE_LEN,in_file);
printf("date = %s\n\n",date);

  fread(chip_number,sizeof(char),CHIP_NO_LEN,
                                     in_file);
printf("The chip number is %s\n\n",chip_number);

  fread(lot_number,sizeof(char),LOT_NO_LEN,in_file);
printf("The lot number is %s\n\n",lot_number);

  fread(mode,sizeof(char),MODE_LEN,in_file);
printf("The mode is %s\n\n",mode);
```

```
  fread((char *)&clock_freq,sizeof(double),1,in_file);
printf("clock_freq is %f\n\n",clock_freq);

  fread((char *)&samp_freq,sizeof(double),1,in_file);
printf("samp_freq = %f\n\n",samp_freq);

  fread((char *)&num_pts,sizeof(int),1,in_file);
printf("num_pts = %i\n\n",num_pts);

  fread((char *)data,sizeof(int),num_pts,in_file);
puts("some data");
for( i = 0; i < 20; i++)
 printf("%i\n",data[i]);

/* Transform data from integer to                    */
/* complex array.                                    */

  for ( i = 0; i < num_pts; i++ )
   input_data[i] = cmplx((double)data[i],0.0);

/* Normalize data between to an lsb.                 */

puts("Normalizing data.");
  normalize_data( input_data,trans_data,num_pts );

/* Window the input data.                            */

puts("Windowing data");
  window_data( trans_data, num_pts );

/* Perform fast fourier transform.                   */

puts("Performing fft");
  act_len = fft( trans_data, trans_data,
                 num_pts, DFT_N);

/* Find the log magnitude of frequency data.         */

puts("Finding magnitude of data.");
  max_mag = 0.0;

/* Take out dc offset.                               */

  trans_data[0].re = 1e-4;
  trans_data[1].re = 1e-4;
  trans_data[2].re = 1e-4;

  trans_data[0].im = 0.0;
  trans_data[1].im = 0.0;
  trans_data[2].im = 0.0;

  for ( i = 3; i < act_len / 2; i++ )
```

```
    {
     trans_data[i].re = cmag( trans_data[i]);
     trans_data[i].im = 0.0;

     if ( trans_data[i].re < le-5 )
       trans_data[i].re = le-5;

     if ( trans_data[i].re > max_mag )
      {
       max_mag = trans_data[i].re;
       printf("max_mag = %f at pt. %i\n",max_mag,i);
      }
    }

/* Convert results to dBs and produce                */
/* frequency arrary.                                 */

  fund_freq = samp_freq / act_len;

puts("Finding dBs.");
  for ( i = 0; i < act_len / 2; i++ )
   {
    trans_data[i].re = 20.0
                 * log10(trans_data[i].re / max_mag);
    freqs[i] = (double)i * fund_freq;
   }


/* Write information and data to output file.    */

  printf("\n\nThe input filename was: %s\n\n",
                                    in_filename);

  puts("\n\n\nEnter the filename for the ");
  puts("output data.");
  puts("\nAn example - - a:data.out \n\n");
  scanf("%s",out_filename);

  out_file = fopen(out_filename,"w");

  fprintf(out_file,"%s %iHz %s %s\n",
            out_filename,(int)(samp_freq),
                       chip_number,lot_number);

  for ( i = 0; i < act_len / 2; i++ )
    fprintf(out_file,"%f %f %i\n",freqs[i],
                               trans_data[i].re,i);
  fclose(in_file);
  fclose(out_file);

  exit(0);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    normal.c
 *
 *  FUNCTION:       VOID normalize_data(x,y,num_pts)
 *
 *  DESCRIPTION:    This function normalizes data from an
 *                  ADC to the range between 0 and 1.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX * - pointer to complex array
 *
 *                  y - DCOMPLEX * - pointer to complex array
 *
 *                  num_pts - int - number of points in
 *                                  the arrays
 *
 *  RETURN:         None.
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-9-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include <math.h>
#include "complex.h"
#include "local.h"

VOID normalize_data(x,y,num_pts)
 DCOMPLEX *x,
          *y;

 int num_pts;
 {
  register i;
  double lsb;
  lsb = (VHIGH - VLOW) / pow( (double)2, (double)NUM_BITS);
  for ( i = 0; i < num_pts; i++ )
   {
    y[i].re = x[i].re * lsb;
    y[i].im = 0.0;
   }
  return;
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    window.c
 *
 *  FUNCTION:       VOID window_data( x, num_pts )
 *
 *  DESCRIPTION:    This function windows data in the array
 *                  x with a Von Hann window.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX * - pointer to complex array
 *
 *                  num_pts - int - number of points in the
 *                                  window
 *
 *  RETURN:         None.
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-9-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include <math.h>
#include "cmath.h"
#include "complex.h"
#include "local.h"

extern DCOMPLEX z [NUM_DFT_POINTS];

VOID window_data( x, num_pts )
 DCOMPLEX *x;
 int num_pts;

 {
  register i;
  double multiplier;

/* Create a Von Hann window.                         */

  for( i = 0; i < num_pts; i++ )
   {
    multiplier = 0.5 * ( 1.0 -
                        cos(2.0 * PI * i / num_pts ));
    z[i].re = multiplier;
```

```
     z[i].im = multiplier;
    }

/* Now multiply data by the window.                        */

   for ( i = 0; i < num_pts; i++ )
    x[i] = cmult(x[i],z[i]);

   return;
  }
```

```
/**************************************************************
 *
 *   SOURCE FILE:    fft.c
 *
 *   FUNCTION:       int fft(x,y,n,inverse)
 *                     DCOMPLEX *x,
 *                              *y;
 *                     int n,inverse;
 *
 *   DESCRIPTION:    This function performs the decimation in
 *                   frequency fast fourier transform.
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *   ARGUMENTS:      x   - - pointer to DCOMPLEX input array
 *                   y   - - pointer to DCOMPLEX output array
 *                   n   - - the desired length of the DFT
 *                        ( or inverse DFT ) to be performed.
 *                   inverse - - a flag to indicate whether
 *                           a forward DFT or an inverse DFT
 *                               is to be performed
 *
 *                  equal to:     DFT     : forward DFT ( with
 *                                          multiplier of 1 )
 *                                DFT_N   : forward DFT ( with
 *                                          multiplier of 1/n )
 *                                IDFT    : inverse DFT ( with
 *                                          multiplier of 1 )
 *                                IDFT_N  : inverse IDFT ( with
 *                                          multiplier of 1/n )
 *
 *   RETURN:     actual length of the DFT ( IDFT ) performed
 *                  If the desired length, n, is an integer
 *                  power of 2, then the actual length is
 *                  equal to n.  Otherwise, the actual length
 *                  is the largest integer power of 2 which is
 *                  less than n.
 *
 *   FUNCTIONS
 *   CALLED:         DCOMPLEX cexpon();
 *
 *   AUTHOR:         Jeffrey C. Daniels
 *
 *   DATE CREATED:   6-3-87
 *
 *   REVISIONS:      None.
 *
**************************************************************/
```

```
#include <math.h>
#include "cmath.h"
#include "complex.h"
#include "local.h"

extern DCOMPLEX z[NUM_DFT_POINTS];

int fft(x,y,n,inverse)
 DCOMPLEX *x,
          *y;
 int n,inverse;

 {
  int dft_length,i,iter_num,
      j,k,l,length,
      m,num_blocks,
      offset,sign;

  double mult_fac,theta;

  DCOMPLEX tempc;

/* Find actual length of the DFT of IDFT to be performed. */

  length = 2;
  while ( length < n )
   length = length * 2;

  if ( length != n )
   length = length / 2;

/* Determine whether DFT or IDFT and also the                  */
/* multiplication factor.                                      */

  switch ( inverse )
   {
    case DFT_N:
                sign = 1;
                mult_fac = 1.0 / (double)length;
                break;
    case IDFT:
                sign = -1;
                mult_fac = 1.0;
                break;
    case DFT:
                sign = 1;
                mult_fac = 1.0;
                break;
    default:
                sign = -1;
                mult_fac = 1.0 / (double)length;
   }
```

```
/* Copy input array into output array if the pointers     */
/* are not to the same array.                              */

  if ( x != y )
   for ( i = 0; i < length ; i++ )
    y[i] = x[i];

/* Initialize variables                                    */

  offset = 0;
  iter_num = 0;
  dft_length = length;

/* Now perform the DFT or IDFT                             */

  while ( length >= 2 )
   {
    num_blocks = (int)pow( (double)2.0, (double)iter_num );
    iter_num = iter_num + 1;
    length = length / 2;
    offset = 0;

    for ( i =1; i <= num_blocks; i++ )
     {
      for ( j = 0; j < length; j++ )
       {
        m = j + offset;
        z[m] = cadd( y[m], y[m+length] );
        z[m + length] = cmult( csub( y[m], y[m + length] ),
                               cexpon( -(double)sign * PI
                                       * (double)j
                                       / (double)length ));
       }
      offset = length * 2 + offset;
     }
    for ( i = 0; i < dft_length; i++ )
     y[i] = z[i];
   }

/* Now unscramble the DFT ( or IDFT ) coefficients         */

  j = 0;
  for ( i = 0; i <= dft_length - 2; i++ )
   {
    if ( i < j )
     {
      tempc = y[j];
      y[j]  = y[i];
      y[i]  = tempc;
     }

    k = dft_length / 2 ;
```

```
      while ( k <= j )
       {
        j = j - k;
        k = k / 2;
       }

      j = j + k;
     }


/* Now multiply by the multiplication factor                  */

  for ( i = 0; i <= dft_length - 1; i++ )
   y[i] = cmult( y[i], cmplx(mult_fac,0.0) );

  return(dft_length);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    cadd.c
 *
 *  FUNCTION:       DCOMPLEX cadd(x,y)
 *                    DCOMPLEX x,y;
 *
 *  DESCRIPTION:    This function performs the addition of
 *                  the two DCOMPLEX numbers x and y.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX number
 *                  y - DCOMPLEX number
 *
 *  RETURN:         result of the DCOMPLEX addition
 *                  of x and y
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include "complex.h"

DCOMPLEX cadd(x,y)
 DCOMPLEX x,y;

 {
  DCOMPLEX z;

  z.re = x.re + y.re;
  z.im = x.im + y.im;

  return(z);
 }
```

```
/*****************************************************************
 *
 *   SOURCE FILE:     csub.c
 *
 *   FUNCTION:        DCOMPLEX csub(x,y)
 *                      DCOMPLEX x,y;
 *
 *   DESCRIPTION:     This function performs the substraction
 *                    of the two DCOMPLEX numbers x and y.
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *   ARGUMENTS:       x - DCOMPLEX number
 *                    y - DCOMPLEX number
 *
 *   RETURN:          result of the DCOMPLEX subtraction of
 *                    x and y
 *
 *   FUNCTIONS
 *   CALLED:          None.
 *
 *   AUTHOR:          Jeffrey C. Daniels
 *
 *   DATE CREATED:    6-2-87
 *
 *   REVISIONS:       None.
 *
*****************************************************************/

#include "complex.h"

DCOMPLEX csub(x,y)
 DCOMPLEX x,y;

 {
  DCOMPLEX z;

  z.re = x.re - y.re;
  z.im = x.im - y.im;

  return(z);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    cdiv.c
 *
 *  FUNCTION:       DCOMPLEX cdiv(x,y)
 *                    DCOMPLEX x,y;
 *
 *  DESCRIPTION:    This function performs the division of
 *                  the two DCOMPLEX numbers x and y.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX number
 *                  y - DCOMPLEX number
 *
 *  RETURN:         result of the DCOMPLEX division of
 *                  x and y
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
******************************************************************/

#include "complex.h"

DCOMPLEX cdiv(x,y)
 DCOMPLEX x,y;

 {
  DCOMPLEX z;

  z.re = ( x.re * y.re + x.im * y.im )
             / ( y.re * y.re + y.im * y.im );
  z.im = ( x.im * y.re - x.re * y.im )
             / ( y.re * y.re + y.im * y.im );

  return(z);
 }
```

```
/***************************************************************
 *
 *  SOURCE FILE:    cexpon.c
 *
 *  FUNCTION:       DCOMPLEX cexpon(theta)
 *                   double theta;
 *
 *  DESCRIPTION:    This function performs the operation of
 *                  exp(j * theta).
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      theta - double
 *
 *  RETURN:         the DCOMPLEX number exp( j * theta)
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
***************************************************************/

#include <math.h>
#include "complex.h"

DCOMPLEX cexpon(theta)
 double theta;

 {
  DCOMPLEX z;

  z.re = cos(theta);
  z.im = sin(theta);

  return(z);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    cmplx.c
 *
 *  FUNCTION:       DCOMPLEX cmplx(x,y)
 *                   double x,y;
 *
 *  DESCRIPTION:    This function makes a DCOMPLEX number
 *                  from the two double numbers x and y.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - double number
 *                  y - double number
 *
 *  RETURN:         the DCOMPLEX number x + jy
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include "complex.h"

DCOMPLEX cmplx(x,y)
 double x,y;

 {
  DCOMPLEX z;

  z.re = x;
  z.im = y;

  return(z);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:   cneg.c
 *
 *  FUNCTION:      DCOMPLEX cneg(x)
 *                   DCOMPLEX x;
 *
 *  DESCRIPTION:   This function performs the negation of
 *                 the DCOMPLEX number x.
 *
 *  DOCUMENTATION
 *  FILES:         None.
 *
 *  ARGUMENTS:     x - DCOMPLEX number
 *
 *  RETURN:        result of the negation of x
 *
 *  FUNCTIONS
 *  CALLED:        None.
 *
 *  AUTHOR:        Jeffrey C. Daniels
 *
 *  DATE CREATED:  6-2-87
 *
 *  REVISIONS:     None.
 *
*****************************************************************/

#include "complex.h"

DCOMPLEX cneg(x)
 DCOMPLEX x;

 {
  DCOMPLEX z;

  z.re = - x.re;
  z.im = - x.im;

  return(z);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    cmag.c
 *
 *  FUNCTION:       double cmag(x)
 *                    DCOMPLEX x;
 *
 *  DESCRIPTION:    This function finds the magnitude of the
 *                  DCOMPLEX number x.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX number
 *
 *  RETURN:         double - the magnitude of x
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#include <math.h>
#include "complex.h"

double cmag(x)
 DCOMPLEX x;

 {
  double z;

  z = sqrt( x.re * x.re + x.im * x.im );

  return(z);
 }
```

```
/*************************************************************
 *
 *  SOURCE FILE:    cmagsq.c
 *
 *  FUNCTION:       double cmagsq(x)
 *                    DCOMPLEX x;
 *
 *  DESCRIPTION:    This function finds the magnitude squared
 *                  of the DCOMPLEX number x.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      x - DCOMPLEX number
 *
 *  RETURN:         double - the magnitude squared of x
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
*************************************************************/

#include "complex.h"

double cmagsq(x)
 DCOMPLEX x;

 {
  double z;

  z =  x.re * x.re + x.im * x.im;

  return(z);
 }
```

```
/**************************************************************
 *
 *  SOURCE FILE:   cmult.c
 *
 *  FUNCTION:      DCOMPLEX cmult(x,y)
 *                   DCOMPLEX x,y;
 *
 *  DESCRIPTION:   This function performs the multiplication
 *                 of the two DCOMPLEX numbers x and y.
 *
 *  DOCUMENTATION
 *  FILES:         None.
 *
 *  ARGUMENTS:     x - DCOMPLEX number
 *                 y - DCOMPLEX number
 *
 *  RETURN:        result of the DCOMPLEX multiplication of
 *                 x and y
 *
 *  FUNCTIONS
 *  CALLED:        None.
 *
 *  AUTHOR:        Jeffrey C. Daniels
 *
 *  DATE CREATED:  6-2-87
 *
 *  REVISIONS:     None.
 *
**************************************************************/

#include "complex.h"

DCOMPLEX cmult(x,y)
 DCOMPLEX x,y;

 {
  DCOMPLEX z;

  z.re = x.re * y.re - x.im * y.im;
  z.im = x.im * y.re + x.re * y.im;

  return(z);
 }
```

```
/****************************************************************
 *
 *  SOURCE FILE:    harmonl.c
 *
 *  FUNCTION:       main()
 *
 *  DESCRIPTION:    This program is used to find the
 *                  location of harmonics in the DFT window.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      None.
 *
 *  RETURN:         ascii file containing 20 * log10 of the
 *                  frequency data.
 *
 *  FUNCTIONS
 *  CALLED:         fft(x,y,n,inverse)
 *                   DCOMPLEX *x,*y;
 *                   int n,inverse
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-15-87
 *
 *  REVISIONS:      6-23-87  Threw out Doerfler's algorithm
 *                           and used my own.
 *
****************************************************************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "cmath.h"
#include "complex.h"
#include "local.h"

DCOMPLEX x[NUM_DFT_POINTS];
DCOMPLEX y[NUM_DFT_POINTS];
DCOMPLEX z[NUM_DFT_POINTS];

double freqs[NUM_DFT_POINTS];

main()
 {
  char out_filename[STRING_LEN + 1];

  int inverse=DFT_N;
  int n;
  int act_len;
  int i,j;
  float inp_float;

  double ampli,
         delta,
         frequency,
         fund_freq,
         max_mag,
         samp_freq,
         w;

  FILE *out_file;

  puts("Enter the frequency of sine wave desired.");
  scanf("%f",&inp_float);
  frequency = (double)inp_float;
  printf("frequency = %f\n\n",frequency);

  puts("Enter the sampling frequency.");
  scanf("%f",&inp_float);
  samp_freq = (double)inp_float;
  printf("samp_freq = %f\n\n",samp_freq);

  puts("Enter the number of points desired.");
  scanf("%i",&n);
  printf(" n = %i\n\n",n);

  w = TWOPI * frequency / samp_freq;

  for ( j = 1; j < 10; j++ )
   {
    printf("%i ",j);
    ampli = pow( (double)10.0, (double)(1-j) );
    for ( i = 0; i < n; i++ )
     {
      x[i].re = x[i].re + ampli
                         * sin( j * w * i );
      x[i].im = 0.0;
     }
   }

/* Window data.                                      */

  puts(" ");
  puts("Windowing data.");
  window_data(y,act_len);

/* Perform Fast Fourier Transform.                   */

  puts("Performing FFT");

  act_len = fft(x,y,n,inverse);
  printf("Actual length = %i\n",act_len);


  puts("Finding magnitude");
  max_mag = 0.0;

/* Find the magnitude of frequency data.  */

  for ( i = 0; i < act_len; i++ )
   {
    y[i].re = cmag( y[i] );

    if( y[i].re < 1e-300 )
      y[i].re = 1e-15;

    if ( y[i].re >= max_mag )
     max_mag = y[i].re;

    y[i].im = 0.0;
   }

/* Convert results to dBs.                      */

  puts("Finding dBs.\n");

  fund_freq = samp_freq / act_len;

  for ( i = 0; i < act_len; i++ )
   {
    y[i].re = 20.0 * log10(y[i].re / max_mag);
    freqs[i] = fund_freq * i;
   }

/* Write out information to a data file.       */

  puts("Enter the output data filename.");
  scanf("%s",out_filename);

  out_file = fopen(out_filename,"w");

  fprintf(out_file,"%s %iHz \n",out_filename,
                              (int)samp_freq);

  for( i = 0; i < act_len/2; i++ )
   fprintf(out_file,"%f %f\n",freqs[i],y[i].re);

  fclose(out_file);

  exit(0);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:     qntzl.c
 *
 *  FUNCTION:        main()
 *
 *  DESCRIPTION:     This program is used to find the spectrum
 *                   of an ideal ADC.
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *  ARGUMENTS:       None.
 *
 *  RETURN:          ascii file containing 20 * log10 of the
 *                   frequency data.
 *
 *  FUNCTIONS
 *  CALLED:          fft(x,y,n,inverse)
 *                    DCOMPLEX *x,*y;
 *                    int n,inverse
 *                   window ( x,n )
 *                    DCOMPLEX *x;
 *                    int n;
 *
 *  AUTHOR:          Jeffrey C. Daniels
 *
 *  DATE CREATED:    7-15-87
 *
 *  REVISIONS:       Created from datgen.c.
 *
*****************************************************************/

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include "cmath.h"
#include "complex.h"
#include "local.h"

DCOMPLEX x[NUM_DFT_POINTS],
         y[NUM_DFT_POINTS],
         z[NUM_DFT_POINTS];

double freqs[NUM_DFT_POINTS];
```

```
main()
 {
  char out_filename[STRING_LEN + 1];

  int inverse=DFT_N;
  int n;
  int act_len;
  int i,num_bits;
  float inp_float;

  double ampli,
         frequency,
         fund_freq,
         lsb,
         max_mag,
         samp_freq,
         vhigh,vlow,
         w;

  FILE *out_file;

  puts(" ");
  puts("Enter the number of bits of the ADC");
  scanf("%i",&num_bits);
  printf("Number of bits = %i\n\n",num_bits);

  puts("Enter the high reference voltage");
  scanf("%f",&inp_float);
  vhigh = (double)inp_float;
  printf("High reference voltage = %f\n\n",vhigh);

  puts("Enter the low reference voltage");
  scanf("%f",&inp_float);
  vlow = (double)inp_float;
  printf("Low reference voltage = %f\n\n",vlow);

  puts("Enter the frequency of sine wave desired.");
  scanf("%f",&inp_float);
  frequency = (double)inp_float;
  printf("frequency = %f\n\n",frequency);

  puts("Enter the sampling frequency.");
  scanf("%f",&inp_float);
  samp_freq = (double)inp_float;
  printf("samp_freq = %f\n\n",samp_freq);

  puts("Enter the number of points desired.");
  scanf("%i",&n);
  printf(" n = %i\n\n",n);

  ampli = ( vhigh - vlow ) / 2.0;
  w = TWOPI * frequency / samp_freq;
  lsb = ( vhigh - vlow )
            / pow( (double)2.0,(double)num_bits);

  for ( i = 0; i < n; i++ )
   {
    x[i].re = ampli * sin( w * i );
    x[i].im = 0.0;
   }

/* Quantize data to NUM_BITS.                            */

  puts("Quantizing data");
  for ( i = 0; i < n; i++ )
    y[i].re = lsb * floor( x[i].re / lsb );

/* Window data.                                          */

  puts("Windowing data");
  window_data ( y,n );

/* Perform Fast Fourier Transform.                       */

  puts("Performing FFT");

  act_len = fft(y,y,n,inverse);
  printf("Actual length = %i\n",act_len);

  puts("Finding magnitude");

  max_mag = 0.0;

/* Find the magnitude of frequency data.  */

  for ( i = 0; i < act_len / 2; i++ )
   {
    y[i].re = cmag( y[i] );

    if( y[i].re < 1e-7 )
      y[i].re = 1e-7;

    if ( y[i].re > max_mag )
     max_mag = y[i].re;

    y[i].im = 0.0;
   }

/* Convert results to dBs.                               */

  puts("Finding dBs.\n");

  fund_freq = samp_freq / act_len;
  for ( i = 0; i < act_len / 2; i++ )
   {
    y[i].re = 20.0 * log10(y[i].re / max_mag);
    freqs[i] = fund_freq * i;
   }

/* Write out information to a data file.       */

  puts("Enter the output data filename.");
  scanf("%s",out_filename);

  out_file = fopen(out_filename,"w");

  fprintf(out_file,"%s  %iHz  %i bits\n",
          out_filename,(int)samp_freq,num_bits);

  for( i = 0; i < act_len / 2; i++ )
   fprintf(out_file,"%f %f    %i\n",
                              freqs[i],y[i].re,i);

  fclose(out_file);

  exit(0);
 }
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    local.h
 *
 *  FUNCTION:       include file
 *
 *  DESCRIPTION:    This file is an include file containing
 *                  various definitions used in many
 *                  functions.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      None.
 *
 *  RETURN:         None.
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-3-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#ifndef _local
#define _local

#define FALSE 0
#define TRUE  1
#define NO    0
#define YES   1

#define VOID void
#define FOREVER for(;;)


#define PI     3.141592653589793
#define TWOPI  6.283185307179586
#define RADDEG 0.017453292519943
#define DEGRAD 57.29577951308232

#define DFT    0  /* Forward DFT with multiplier of 1.0   */
#define DFT_N  1  /* Forward DFT with multiplier of 1.0/N */
#define IDFT   2  /* Inverse DFT with multiplier of 1.0   */
#define IDFT_N 3  /* Inverse DFT with multiplier of 1.0/N */

#define NUM_DFT_POINTS 4096
#define NUM_HIS_POINTS 327680
```

```
#define LOOP_SIZE       4096

#define STRING_LEN     80
#define CHIP_NO_LEN    20
#define DATE_LEN        8
#define MODE_LEN        2
#define LOT_NO_LEN      3

#define PORTA       0x380  /* Ports on IBM PCXT Metrabyte */
#define PORTB       0x381  /* Board                       */
#define PORTC       0x382
#define CONTROL     0x383
#define PPI_CONFIG 0xb6

#define VHIGH        5
#define VLOW         0
#define NUM_BITS     8
#define RESOLUTION 256

/* Funtion definitions. */
VOID window_data();
VOID normalize_data();

int fft();

#endif
```

```
/***************************************************************
 *
 *  SOURCE FILE:   cmath.h
 *
 *  FUNCTION:      include file
 *
 *  DESCRIPTION:   This file is an include file containing
 *                 the definitions for the functions
 *                 involving COMPLEX numbers.
 *
 *  DOCUMENTATION
 *  FILES:         None.
 *
 *  ARGUMENTS:     None.
 *
 *  RETURN:        None.
 *
 *  FUNCTIONS
 *  CALLED:        None.
 *
 *  AUTHOR:        Jeffrey C. Daniels
 *
 *  DATE CREATED:  6-2-87
 *
 *  REVISIONS:     None.
 *
***************************************************************/

#ifndef _cmath
#define _cmath

#include "complex.h"

DCOMPLEX cadd();
DCOMPLEX csub();
DCOMPLEX cmult();
DCOMPLEX cdiv();
DCOMPLEX cmplx();
DCOMPLEX cexpon();
DCOMPLEX cneg();

double cmag();
double cmagsq();
double cphase();
double cphased();

#endif
```

```
/*****************************************************************
 *
 *  SOURCE FILE:    complex.h
 *
 *  FUNCTION:       include file
 *
 *  DESCRIPTION:    This file is an include file containing
 *                  the definitions for the DCOMPLEX data
 *                  structure.  A DCOMPLEX number is just
 *                  a double precision complex.
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *  ARGUMENTS:      None.
 *
 *  RETURN:         None.
 *
 *  FUNCTIONS
 *  CALLED:         None.
 *
 *  AUTHOR:         Jeffrey C. Daniels
 *
 *  DATE CREATED:   6-2-87
 *
 *  REVISIONS:      None.
 *
*****************************************************************/

#ifndef _dcomplex
#define _dcomplex

typedef struct dcomplex
 {
  double re,
         im;
 } DCOMPLEX;

#endif
```

Appendix C

End Point Transition Procedure[3,7]

This procedure is an alternative to the histogram procedure for calculating integral non-linearity errors from a line that passes through the first and last transitions for the actual transfer function for an analog to digital (A/D) converter.

The following transfer function for an ideal, three bit, unipolar A/D is:

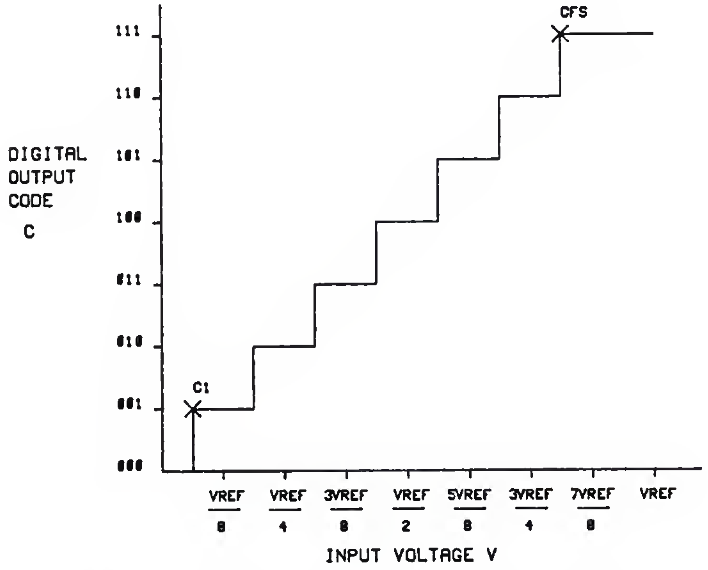

Figure C-1. Transfer function for an ideal 3-bit A/D.

where:

$V_{LSB} = V_{ref}/2^n$,

and two measurable points with the coordinates are:

C_1 - first A/D transition

$= (001, V_{ref}/16)$ or $(001, V_{ref}/2^{n+1})$

C_{FS} - last A/D transition

$= (111, 13V_{ref}/16)$ or $(2^n - 1, V_{ref} - 3V_{ref}/2^{n+1})$

Transition voltages occur at

$$V_{tran} = (V_{ref}/2^{n+1})\ (2i - 1) \quad \text{for } i = 1,2,\cdots,2^{n-1}$$

The equation of the line through C_1 and C_{FS} for an ideal A/D with some point (C,V) on the transfer function is:

$$\frac{C - C_{FS}}{C_{FS} - C_1} = \frac{V - V_{FS}}{V_{FS} - V_1}$$

or

$$\frac{C - 2^n + 1}{2^n - 2} = \frac{V - V_{ref} + (3/2^{n+1})\ V_{ref}}{V_{ref} - (3/2^{n+1})\ V_{ref} - V_{ref}/2^{n+1}}$$

manipulating, if true endpoints, we can obtain:

$$\frac{C - 2^n + 1}{2(2^{n-1} - 1)} = \frac{V - 2^{n+1} - V_{ref}2^{n+1} + 3V_{ref}}{V_{ref}2^{n+1} - 3V_{ref} - V_{ref}}$$

$$= \frac{V2^{n+1}/V_{ref} - 2^{n+1} + 3}{2^{n+1} - 4}$$

$$\frac{C - 2^n + 1}{2(2^{n-1} - 1)} = \frac{V2^{n+1}/V_{ref} - 2^{n+1} + 3}{4(2^{n-1} - 1)}$$

$$C - 2^n + 1 = V2^{n+1}/2V_{ref} - 2^{n+1}/2 + 3/2$$

This must equal the non-ideal equation. Therefore,

$$\frac{a}{V_{LSB}} = \frac{C_{FS} - C_F}{V_{FS} - V_F} \dashrightarrow a = V_{LSB} \frac{C_{FS} - C_F}{V_{FS} - V_F}$$

and

$$\frac{B}{V_{LSB}} + \frac{1}{2} = \frac{V_{FS}C_F - V_F C_{FS}}{V_{FS} - V_F}$$

yields:

$$B = \frac{V_{FS}C_F - V_F C_{FS}}{V_{FS} - V_F} - \frac{1}{2} \quad V_{LSB}$$

Thus, all voltages obtained in taking ramp data must be multiplied by a and have B added to them. This adjustment of the voltage removes gain and offset errors to from the data to then be used to calculate integral non-linearity errors at the transition points by the following equation:

$$IN(i) = \frac{V_t(i) - [\, V_t(1) + (i-1)(LSB)]}{LSB} \quad LSB$$

where:

$i = 0,1,2,\cdots,2^{n-1}$

and $LSB = V_{ref}/2^n$

The voltages corresponding to the transition points are found by searching the A/D conversion results for a transition and then integral non-linearity errors are calculated. These voltages cannot be found if the A/D conversion results have areas of non-monotonic behavior. The simple search for a transition will not yield a true transition point because of the non-monotonic behavior.

References

1. Motorola Technical Data Advance Information: MC68HC11A8 HCMOS Single-Chip Microcomputer, (Phoenix, AZ: Motorola Literature Distribution, 1985).

2. Douglas W. Doerfler, "Techniques for Testing a 15-Bit Data Acquisition System," (MS Thesis, Department of Computer and Electrical Engineering, Kansas State University, 1985).

3. Analog Devices, Analog-Digital Conversion Handbook, (Englewood Cliffs, NJ: Prentice-Hall, Inc., 1986).

4. Steven Douglas Draving, "An Evaluation of the MC68HC11A8 Single-Chip Microcomputer as a Controller for Low-Power, Precision A/D Converters," (MS Thesis, Department of Computer and Electrical Engineering, Kansas State University, 1987).

5 Motorola, "M68HC11EVB Evaluation Board User`s Manual", (First Edition, Copyright 1986 by Motorola, Inc.).

6. Dave Holdeman, "Precision Voltage Reference," (Research Report, Kansas State University, Department of Computer and Electrical Engineering, Project No. 2845, December 14, 1984).

7. Dr. Donald H. Lenhert, "Personal Research Notes for Motorola Project", (Nov. 7,1987, pp. 13-16).

The Static and Dynamic Characterization of the MC68HC11A8's Analog to Digital Converter

by

Jeffrey Charles Daniels

B.S., Kansas State University, 1984

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Abstract

The Motorola MC68HC11A8 (HC11) is a high speed, low power microcomputer with an onboard eight channel, multiplexed input, successive approximation analog to digital converter (A/D) with sample and hold. This A/D system is clocked by the HC11's E clock or by an internal RC timer. This thesis presents three static and two dynamic testing methods used to test the A/D in different configurations with the RC timer enabled and disabled and at different E clock frequencies.

Several different lots of HC11s from the mask of B96D were tested and three problems were discovered. These problems include isolated cases of errors induced by pattern sensitivity, consistent constant offsets when the A/D is operated in various operational modes, and the problem of large errors being induced when the A/D is clocked by its internal RC timer. All of the errors discovered in dynamic tests had been previously found using static tests indicating that no large scale dynamic sensitivities exist for this mask.