The Static and Dynamic Characterization of the MC68HCllA8'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:

[Signature]
Major Professor
# Table of Contents

List of Figures ........................................ iii
List of Tables ........................................ vii
Acknowledgments ....................................... viii

1.0 Introduction .................................. 1
2.0 The Testing System ............................. 3

3.0 Theory of Operation of the HCl1's A/D .......... 9
   Single Channel Operation .......................... 12
   Multiple Channel Operation ....................... 13
   Scan Control ...................................... 13
   Using the A/D .................................... 13

4.0 Static Testing of the HCl1's A/D ................ 15
   Testing Procedure ................................ 15
   Method 1 ......................................... 17
   Method 2 ......................................... 18
   Method 3 ......................................... 18

5.0 Static Testing Results .......................... 20
   Method 1 ......................................... 20
   Method 2 ......................................... 24
   Method 3 ......................................... 25
   Conclusion ....................................... 47

6.0 Dynamic Testing of the HCl1's A/D .............. 48
   Testing Procedure ................................ 50
   Method 1 ......................................... 51
   Method 2 ......................................... 53

7.0 Dynamic Testing Results of the HCl1's A/D .... 57
   Method 1 ......................................... 58
   Method 2 ......................................... 67
   Conclusion ....................................... 73

8.0 Summary and Recommendations ................... 74

Appendix A ............................................. A-l
Appendix B ............................................. B-1
Appendix C ............................................. C-1
References ............................................. R-l
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2-1</td>
<td>Static and Dynamic Testing Configuration</td>
<td>4</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>Interface Board Configuration</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>Precision Voltage Reference Circuit</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>A/D Control/Status Register</td>
<td>11</td>
</tr>
<tr>
<td>Figure 4-1</td>
<td>Static Testing System</td>
<td>16</td>
</tr>
<tr>
<td>Figure 5-1</td>
<td>Transfer Functions of A/D in four conversion, single input, input channel 1. Checking state of CCF bit</td>
<td>21</td>
</tr>
<tr>
<td>Figure 5-2</td>
<td>Transfer Functions of A/D in four conversion, single input, input channel 1. Checking state of CCF bit</td>
<td>22</td>
</tr>
<tr>
<td>Figure 5-3</td>
<td>Transfer Functions of four HCl1s in four conversion, single input, input channel 1. Checking state of CCF</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5-4</td>
<td>Transfer Functions of A/D in four conversion, single input, input channel 1. Executing long delay</td>
<td>26</td>
</tr>
<tr>
<td>Figure 5-5</td>
<td>Transfer Functions of A/D in four conversion, single input, input channel 1. Executing long delay</td>
<td>27</td>
</tr>
<tr>
<td>Figure 5-6</td>
<td>Total errors of the A/D using the four conversion, single input, input channel 1 mode</td>
<td>29</td>
</tr>
<tr>
<td>Figure 5-7</td>
<td>Total errors of the A/D using the four conversion, multiple input mode</td>
<td>30</td>
</tr>
<tr>
<td>Figure 5-8</td>
<td>Total errors of the A/D using the continuous conversion, single input, input channel 1 mode</td>
<td>31</td>
</tr>
<tr>
<td>Figure 5-9</td>
<td>Total errors of the A/D using the continuous conversion, multiple mode</td>
<td>32</td>
</tr>
<tr>
<td>Figure 5-10</td>
<td>Differential non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode</td>
<td>34</td>
</tr>
</tbody>
</table>
Figure 5-11. Differential non-linearity errors of the A/D using the four conversion, multiple input mode .............. 35

Figure 5-12. Differential non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode ................. 36

Figure 5-13. Differential non-linearity errors of the A/D using the continuous conversion, multiple mode .............. 37

Figure 5-14. Histogram integral non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode .... 38

Figure 5-15. Histogram integral non-linearity errors of the A/D using the four conversion, multiple input mode .............. 39

Figure 5-16. Histogram integral non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode .............. 40

Figure 5-17. Histogram integral non-linearity errors of the A/D using the continuous conversion, multiple mode .............. 41

Figure 5-18. End Point integral non-linearity errors of the A/D using the four conversion, single input, input channel 1 mode .... 43

Figure 5-19. End Point integral non-linearity errors of the A/D using the four conversion, multiple input mode .............. 44

Figure 5-20. End Point integral non-linearity errors of the A/D using the continuous conversion, single input, input channel 1 mode .............. 45

Figure 5-21. End Point integral non-linearity errors of the A/D using the continuous conversion, multiple mode .............. 46

Figure 6-1. Dynamic Testing System .............. 49

Figure 6-2. Harmonics of the DFT Window 1406.25 Hz sine wave sampled at 4000 Hz .............. 55
Figure 6-3. Spectrum for an ideal 8-bit A/D
1406.25 Hz sine wave sampled at 4000 Hz . 57

Figure 7-1. Histogram data and Differential
non-linearity errors for A/D. Four
conversion, single input, input channel 1.
RC timer disabled . . . . . . . . . . . . . . . . 59

Figure 7-2. Histogram data and Differential
non-linearity errors for A/D. Four
conversion, single input, input channel 1.
RC timer enabled . . . . . . . . . . . . . . . . 60

Figure 7-3. Histogram data and Differential
non-linearity errors for A/D. Four
conversion, multiple input.
RC timer disabled . . . . . . . . . . . . . . . . 61

Figure 7-4. Histogram data and Differential
non-linearity errors for A/D. Four
conversion, multiple input.
RC timer enabled . . . . . . . . . . . . . . . . 62

Figure 7-5. Histogram data and Differential
non-linearity errors for A/D.
Continuous-conversion, single input, input channel 1. RC timer disabled . . 63

Figure 7-6. Histogram data and Differential
non-linearity errors for A/D.
Continuous-conversion, single input, input channel 1. RC timer enabled . . 64

Figure 7-7. Histogram data and Differential
non-linearity errors for A/D.
Continuous-conversion, multiple input.
RC timer disabled . . . . . . . . . . . . . . . . 65

Figure 7-8. Histogram data and Differential
non-linearity errors for A/D.
Continuous-conversion, multiple input.
RC timer enabled . . . . . . . . . . . . . . . . 66

Figure 7-9. Fourier Transform Results of A/D.
Four-conversion, single input, input channel 1 . . . . . . . . . . . . . . . . . 68

Figure 7-10. Fourier Transform Results of A/D.
Four-conversion, multiple input . . . . . 69
Figure 7-11. Fourier Transform Results of A/D. Continuous-conversion, single input, input channel 1 . . . . . . . . . . . . . . 70

Figure 7-12. Fourier Transform Results of A/D. Continuous-conversion, multiple input . . 71

Figure C-1. Transfer function of an ideal 3-bit A/D C-1
List of Tables

Table 3-1. Analog to Digital Channel Assignments . . . 12
Table 6-1. Location of Harmonics in the Frequency window . . . . . . . . . . . . 54
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. 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 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, 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 P1012 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. 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 10μF, 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
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 VRH 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 HCl1's A/D

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 HCl1'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.
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7, CCF</td>
<td>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.</td>
</tr>
<tr>
<td>6,</td>
<td>Not implemented. Reads as zero.</td>
</tr>
<tr>
<td>5, SCAN</td>
<td>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.</td>
</tr>
<tr>
<td>4, MULT</td>
<td>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.</td>
</tr>
<tr>
<td>3, CD</td>
<td>Channel Select D</td>
</tr>
<tr>
<td>2, CC</td>
<td>Channel Select C</td>
</tr>
<tr>
<td>1, CB</td>
<td>Channel Select B</td>
</tr>
<tr>
<td>0, CA</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

\[
\text{LSB} = \frac{(V_{RH} - V_{RL})}{2^{\text{Resolution}}}
\]

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. 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 CCP bit
Figure 5-2. Transfer Functions of A/D in four conversion, Single Input
input channel 1. Checking state of CCP bit
Figure 5-3. Transfer Functions of four HCIIAs 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 HClls. 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 HClls.

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 HClls.
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 HC111 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 HCll'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 HClls 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\(^2\)

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

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\textsuperscript{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 indica-
tion 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.

<table>
<thead>
<tr>
<th>Harmonic</th>
<th>Frequency (in Hertz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1406.25</td>
</tr>
<tr>
<td>1</td>
<td>1187.5</td>
</tr>
<tr>
<td>2</td>
<td>218.75</td>
</tr>
<tr>
<td>3</td>
<td>1625.0</td>
</tr>
<tr>
<td>4</td>
<td>968.75</td>
</tr>
<tr>
<td>5</td>
<td>437.5</td>
</tr>
<tr>
<td>6</td>
<td>1843.75</td>
</tr>
<tr>
<td>7</td>
<td>750.0</td>
</tr>
<tr>
<td>8</td>
<td>656.25</td>
</tr>
</tbody>
</table>

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

The dynamic range of an N-bit converter is known to be
\[
\text{dynamic range} = 20 \log_{10}(2^N) = 6.02N \text{ dB}
\]
This equation implies that if the amplitude of the highest harmonic is less than 6.02N, then the integral non-linearity 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.
Output Spectrum for an Ideal ADC
QNTZ1406  4000Hz  8 bits

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 HCl1's A/D

Due to time limitations, the dynamic testing of all HCl1s acquired has not been completed at this time but one HCl1 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 HCl1 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 HCl1'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
Fourier Transform Test of HC11 ADC
I3NC31F8 4000Hz 1-3 B96D

Fourier Transform Test of HC11 ADC
I2NC31F2 4000Hz 1-3 B96D

Fourier Transform Test of HC11 ADC
I3C31F8 4000Hz 1-3 B96D

Fourier Transform Test of HC11 ADC
I2C31F2 4000Hz 1-3 B96D

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 HCl1s for this thesis indicates the possibility of the HCl1'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 HCl1's A/D has no large scale dynamic sensitivities. This indicates that dynamic testing need not be included in production floor testing of the HCl1's A/D for this mask. If mask changes occur in the future, a dynamic characterization should be performed with several HCl1s 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 HCl1'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

ora $80
to
ora $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

lda $1039  Power up A/D system.
ora #80    RC timer off.
staa $1039

lda $1039
org $c000
ldaa $1039  Power up A/D system.
ora #80    RC timer off.
staa $1039

idx $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 ldax $00 Initiate conversion.
     staa $1030

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

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

bra CONVRT Do another conversion.
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.

ADRO 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 #ADRO 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.

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 #ADRO has already been done.
BNE NXTOUT If so, jump to NXTOUT.

LDAB $00
STAB $1030
CHECK
LDAB $1030
BPL CHECK

Initiate conversion.
Check if conversion is done.

NXTOUT
LDX RESPTR
LDA 0,X
STA PORTB
Load result

BSET 0,Y $10
BCLR 0,Y $10
BCLR 0,Y $40
BSET 0,Y $40
Store result in Low Byte of data latch.
Clear flip flop.

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

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

CONT BRA NTREDY
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.

ADRO 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 #ADRO Initialize pointer to
STX RESPTR result register.
LDAA $1039 Enable ADC --
ORAA #$80 Power up ADC
STAA $1039 RC timer off.
LDAB $00 Initiate conversion.
STAB $1030

* 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
CPX #ADR0
BNE NXTOUT

LDAB $00
STAB $1030

LDX #LOOPNO
DEX
BNE DELAY

LDX RESPTR
Load result
LDAA 0,X
STAA PORTB

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

* 
CONT BRA NTREDY

Check to see if conversion has already been done.
If so, jump to NXTOUT.

Initiate conversion.
Wait 450 clock cycles.

Load result

A-8
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.
Continuous 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 interrupt 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 —
ORA #$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.
LDAB $1030 Check if conversion
BPL CHECK is done.

LDX $01
LDA A 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

A-10
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. Continuous 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 interrupt 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 ----
ORA $c0  Power up ADC
STAA $1039  RC timer on.
LDAB $00
STAB $1030  Initiate conversion.
NTREDY
LDAA 0,Y
ANDA #$04
BEQ NTREDY

A-11
LDAB $00
STAB $1030

CHECK

LDAB $1030
BPL CHECK

Initiate conversion.

Check if conversion is done.

LDX $01
LDAA 0,X
STAA $1004

Load result

BSET 0,Y $10
BCLR 0,Y $10

Store result in Low Byte of data latch.

BCLR 0,Y $40
BSET 0,Y $40

Clear flip flop.

BRA NTREDY
This program reads the four Result Registers of the G8HC11's onboard ADC and displays them to the screen. The OVM readings are also displayed on the screen. None of the information is stored in arrays or stored in files. This program provides a quick check to verify that the 4 Result Registers are functioning properly.

Jeff's STGET4.OBJ program is used to drive the G8HC11.

MICHAEL PANKRATZ 04 AUGUST 1987
---

**GETDAT**  
ver 5.0

---

This program collects data from the ADC under test  
the 68HC11's onboard ADC) and writes it to a file.

The file can be read by PLOT and PRIDAT but only the first  
array (the raw data) will be read. The bin count array and  
other data will be ignored.

The file format is as follows. The first element in  
the file is the raw data array size. It is followed by the  
array of data itself. Next is the size of the bin count  
array, followed by the array itself. Then the reference  
voltage, Vref, and the ADC resolution, Res, are stored.

Finally, 4 transition points are stored for gain and offset  
error calculations. They are 0-1, 5-7 & 7-8 (arbitrary  
for step width), and 254-255.

The program calls Vtrans(Pre_tog, Res, Vref), a function  
to find the transition point voltages.

Version 2.0 change the DNL error method from the  
theoretical method (using transition points) to the  
histogram method. Now only 4 transition points are taken  
to figure offset and gain errors. Instead, the ADC under  
test is read 10 times and the results are placed into bins.

Versions 3.0 and 3.5 were modifications made to the  
toggle function. These improvements are outlined in the  
header for the function (see below).

Version 4.0 was a modification made in conjunction with  
a change made in the HC11 program. The delay to allow the  
DAC to set up was taken out of the HC11 program and put in  
this program. That simply involved changing two existing  
delay loops from 5ns to 100ns.

Version 5.0 first determines the minimum and maximum  
ADC output values and uses these in the TDGFN instead of  
sending 0-1 and 254-255. This will allow testing of  
converters which do not operate over their full range.

Also, transition points 6-7 and 7-8 are not taken since  
they are no longer used by ERROR. The two variables are  
now used to store the minimum and maximum ADC count  
values that were sent into the toggle function.

---

MICHAEL PANKRATZ  
05 AUGUST 1997

---

| DIM Input(1,1024) | Data array |
| DIM Bin(1,256)   | 8-bit count array |
| DIM Vtr(3)       | Transition voltage array |

**Constants:**

- Vref=5  
- Res=8  

---

A-14
Samples=1024  | Number of samples taken
Vmin=0  | Valid values: 0 to Vref
Vmax=Vref  | Valid values: 0 to Vref > Vmin

640

650 PRINTER IS 16
660 PRINT PAGE
670 PRINT TAB(33):"ADC DATA AQUISITION"
680 PRINT TAB(20):"Data for Offset, Gain, INL, and DNL errors"
690 PRINT LINK(3)
700 !
710 !
720 Initialize:  
730 OUTPUT 709:"FIRANSTZZ1"  | Initialize OVM
740 RESET 3  | Initialize GPIO for ADC
750 RESET 2  | Initialize GPIO for precision DAC
760 FOR I=0 TO 2*Res-1  | Initialize bin count array
770   Bin(0,I)=I
780   Bin(1,I)=0
790 NEXT I
800 !
810 !
820 Open_file:  
830 REDIM Input(1,Samples-1)  
840 REDIM Bin(1,2*Res-1)  
850 !
860 BEEP
870 ! Detfile$="G01F2"
880 EDIT "Enter filename for the data:",Detfile$
890 ASSIGN #1 TO Detfile$,Stat
900 IF Stat=0 THEN PURGE Detfile$  | Delete if file already exists
910 Records=INT((Samples+2*Res)/16)+1  | Calculate number of records
920 CREATE Detfile$,Records  | Create the data file
930 ASSIGN #1 TO Detfile$  | Open the file
940 !
950 !
960 BEEP
970 DISP "Press any key to begin data acquisition."
980 ON KB0 GOTO Get_trans ,ALL
990 Wait:  GOTO Wait
1000 !
1010 !
1020 Get_trans:  
1030 DISP
1040 PRINT "Finding transitions:"
1050 Dvel=0
1060gosub Set_dec
1070 Vtr(1)=READBIN(3)  | Determine min ADC output value
1080 Cmin=Vtr(1)
1090 Vtr(0)=FNVtrans(Cmin,Res,Vref)  | Voltage of 1st transition point
1100 !
1110 Dvel=2562143
1120gosub Set_dec
1130 Vtr(2)=READBIN(3)  | Determine max ADC output value
1140 Cmax=Vtr(2)-1
1150 Vtr(3)=FNVtrans(Cmax,Res,Vref)  | Voltage of last transition point
1160 BEEP
1170 !
1180 !
1190 Cmin=INT(Vmin+2^18/Vref)
1200 Dval=INT(Umax*2^16/Vref)
1210 Vstep=(Umax-Umin)/(Samples-1)
1220 Dstep=INT(Vstep*2^16/Vref) \* Step size to incr. the DAC
1230 \*
1240 PRINT
1250 PRINT "Sampling":Samples:"points between":Umin:"and":Umax:"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 \*
1340 \*
1350 TRIGGER 709 \* Start the DVM
1360 Rd: STATUS 709:Stet
1370 IF BIT(Stet,0)>1 THEN Rd \* Wait till ready
1380 ENTER 709:Input(0,1)
1390 \*
1400 Input(I,1)=READBIN(3)
1410 Index=Input(I,1)
1420 Bin(I,Index)=Bin(I,Index)+1 \* Set the DAC with value in Dval
1430 \*
1440 FOR J=1 TO 9 \* Update the bins
1450 Index=READBIN(3)
1460 Bin(I,Index)=Bin(I,Index)+1
1470 NEXT J
1480 \*
1490 DISABLE \* Turns off key-abort function
1500 \*
1510 \*
1520 Write file: \*
1530 PRINT #1;Samples,Input(*) \* Store data
1540 PRINT #1;2^2^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_dec: \* 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 \* Send word to the DAC
1730 WRITE BIN 2;Mid,Mid+512,Mid
1740 WRITE BIN 2;Low,Low+256,Low,0
1750 \*
1760 WAIT 100 \* 100ms delay
1770 RETURN
1780 \*
1790 \*
This function determines the input voltage just before and after a toggle point of the ADC under test (in this case, the 68HC11's on-board converter). The full 18 bits resolution of the precision DAC are used to obtain as accurate results as possible. The results are averaged and returned in the function name. The format is:

```
FNVtransi(Togpt, Res, Vref)
```

where:

- **Togpt** - the desired ADC output value BEFORE the transition. 0 <= Togpt <= (2^Res)-2
- **Res** - the resolution of the ADC under test (in bits)
- **Vref** - the ADC reference voltage

Note: The DVM and SPIID busses must be initialized before this function is called.

Version 2.0 added a lower resolution binary pre-search which dramatically reduced the search time for a toggle point.

Version 3.0 made a few changes to give the function the capability to catch missing code. The function then gives the users the option to enter a new toggle point (or the same one to check for repeatability) or to terminate the program. Returns a 999 to indicate missing code error.

Version 4.0 made a couple more changes to help fight missing code errors by checking the ADC output after coming out of the binary search and kicking it back up there if the ADC value was below the entered toggle point value.

MICHAEL PANKRATZ 24 JUN 1987

DEF FNVtransi(Togpt, Res, Vref)
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 Dec=Dec+Dstep ! DAC input value
2440 IF Dec>262143 THEN Dec=262143 ! Prevent over-ranging the DAC
2450 Lp2:GSUB Set_dec ! Set the DAC
2460 Cdec=READBIN(3) ! Get current ADC value
2470 DISP "DAC input:";Dec,"ADC output:";Cdec
2480 IF Cdec<Togpt THEN Bin ! Did toggle occur?
2490 Dec=Dec-Dstep ! Set DAC to before toggle point
2500 IF Dec<0 THEN Dec=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 Cdec=Togpt THEN Lp3 ! Make sure ADC value is good
2580 Dstep=8*Bin_res ! Set DAC step size to min value
2590 GOTO 8*Bin
2600 !
2610 Lp3:Dec=Dec+Dstep ! Set the DAC
2620 GSUB Set_dec ! Store previous value of ADC
2630 Padc=Cdec ! Get current ADC value
2640 Cdec=READBIN(3) ! Get current ADC value
2650 DISP "DAC input:";Dec,"ADC output:";Cdec
2660 IF Cdec-Padc THEN Exit ! Did toggle occur?
2670 GOTO Lp3
2680 !
2690 !
2700 Exit: !
2710 GSUB Read_dvm ! Current DAC output
2720 Cvolt=Voltage
2730 Dec=Dec-Dstep ! Previous DAC output
2740 GSUB Set_dec
2750 GSUB Read_dvm
2760 Pvolt=Voltga
2770 Ave=(Cvolt+Pvolt)/2 ! Average to find transition voltage
2780 !
2790 !
2800 IF (Cdec-Padc+1) AND (Cdec=Togpt+1) THEN Dk ! No missing codes
2810 Printer IS 16 ! Output device is the screen
2820 FDR X=1 TO 3
2830 BEEP
2840 WAIT 250
2850 NEXT X
2860 PRINT "Missing Code!"
2870 PRINT "Actual results:";Padc,"Expected:";Cdec
2880 PRINT "Desired results:";Togpt,"Expected:";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>2^18-2) THEN Inp
2940 GOTO Init
2950 Term: Ave=999
2960 Dk: RETURN Ave
2970 FNEND
2980 !
2990 !
A-18
3000 !
3010 Set_dac: ! Calculates DAC input word and sends it to the DAC
3020   Dval=Dec
3030   High=INT(Dval/65536) ! Digital value for this sample
3040   Mid=INT(Dval/256)-256*High ! High byte for the DAC
3050   Low=Dval-65536*High-256*Mid ! Middle byte for the DAC
3060   WRITE BIN 2:High,High+1024,High ! Low byte for the DAC
3070   WRITE BIN 2:Mid,Mid+512,Mid ! - Set the DAC
3080   WRITE BIN 2:Low,Low+256,Low ! /
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

A-19
DIM Data1(1,1024),Data2(1,256),Vtr(3) ' The data arrays

Size=81 ' The size of a raw data file

PRINT PAGE
PRINT TAB(35):"HP DATA MAILER"
PRINT TAB(22):"Data transfer from a 9845B to a 9236"
PRINT LIN(3)

Input:  BEEP

Input file name

EDIT "Enter name of file to transfer (must be ASCII):"",Datafile$

IF Stat<>1 THEN Cont ' Does file exist?

PRINT "File ":Datafile$" does not exist."

PRINT
BEEP
WAIT 200

GDID Input

Cont: 1

Send over the file name & size

Read_file: 1

Disp "Reading data from file..."

Read $1:Size

REDim Data1(1,Size-1)

Read $1:Data1(*) ' Read in first array
READ #1:Size2
READ #1:Data2(*)  ! Read in second array
READ #1:Uref,Res,Vtr(*)  ! Read in remaining data

! Send_data:
OISP "Sending data over HPIB..."
OUTPUT 720:Size1          ! Send size of 1st array
FOR I=0 TO I
   FOR J=0 TO Size1-1
      OUTPUT 720:Data1(I,J)  ! Send 1st array
   NEXT J
NEXT I

NEXT I
OUTPUT 720:Size2          ! Send 2nd array
FOR I=0 TO I
   FOR J=0 TO Size2-1
      OUTPUT 720:Data2(I,J)
   NEXT J
NEXT I

NEXT I
OUTPUT 720:Uref           ! Send remaining values
OUTPUT 720:Res
FOR I=0 TO 3
   OUTPUT 720:Vtr(I)
NEXT I

END
It was written to run on the HP 9236 computer. It receives data files created by GETDAT over the HPIB bus and stores it on disk. The first item transferred over the HPIB bus is the file name and size so this program does not need to prompt the user for any information. There are several things which must be observed for the Data Mailer to operate properly:

1. The 9845B must be configured as the controller (it is normally in this mode), and the 9236 must be in the non-controller mode. This requires changing a jumper on the motherboard of the 9236 (jumper should connect the left two prongs). Be sure to put the jumper back to its original position when the transfer is completed.

2. The 9236 must have its version of DATAMAIL (the Receiver) running first, before the Transmitter program on the 9845B is started or some of the data will get lost.

3. The EMAILer can only send ASCII formatted files. If it is desired to send a BASIC program, this program will not work. Use EMAIL2 to send programs.

Written by HEWLETT-PACKARD
Modified by Michael Pankratz 20 July 1987

DIM Data1(1,1024), Data2(1,256), Var(3)

OUTPUT 2:CHR$(255)&CHR$(75)"Clear the screen"
PRINT TAB(33,2):CHR$(139):"HP DATA MAILER"
PRINT TAB(22):CHR$(139):"Data transfer from a 9845B to a 9236"
PRINT
PRINT
PRINT
ASSIGN @Ifile TO 7 "Set up HPIB like an input file"

BEEP "Waiting for data."
ENTER @Ifile:Outfile@Fsize "Read filename & size off the HPIB"

BEEP "Receiving filename."
BEEP

Open file: "Open output file on disk"

Read_data: ""
OISP "Receiving data..."  
ENTER @file:Size1  
REDIM Deta1(I,Size1-1)  
FOR I=0 TO I  
  FOR J=0 TO Size1-1  
  ENTER @file:Deta1(I,J)  
  NEXT J  
  NEXT I  
ENTER @file:Size2  
REDIM Deta2(I,Size2-1)  
FOR I=0 TO I  
  FOR J=0 TO Size2-1  
  ENTER @file:Deta2(I,J)  
  NEXT J  
  NEXT I  
ENTER @file:Uref:Res  
READ In size of 1st array  
READ in first array  
Size of 2nd array  
READ in second array  
READ in remaining data  
WRITE data:  
OISP "Writing data to a file..."  
OUTPUT @file:Size1:Deta1(*)  
OUTPUT @file:Size2:Deta2(*)  
OUTPUT @file:Uref:Res:Utr(*)  
EOF:  
ASSIGN @file TO *  
CLOSE the files  
ASSIGN @file TO *  
OISP "Data transfer complete."  
PRINT CHR$(140):"Data is stored in ":CHR$(136):Outfile$  
GOTO Exit  
STOP  
ELSE  
PURGE Outfile$  
RETURN  
PRINT CHR$(137):CHR$(130)  
PRINT "Terminal Error!"  
PRINT "Error code:";ERRN  
PRINT CHR$(128):CHR$(139)  
STOP  
END IF  
END IF  
Purge the file if it already exits
IF ERRN=54 THEN  
PURGE Outfile$  
RETURN  
ELSE  
PRINT CHR$(137):CHR$(130)  
PRINT "Terminal Error!"  
PRINT "Error code:";ERRN  
PRINT CHR$(128):CHR$(139)  
STOP  
ENDIF  
END  
ENDIF  
Exit:BEEP  
PRINT CHR$(139)  
END
This program takes the data file created by GETDAT and calculates the offset error, gain error, integral non-linearity, differential non-linearity, and absolute errors. The non-linearity and absolute errors are written to files which may be read by PLOT if desired.

INL error is calculated using the method outlined in ANALOG-DIGITAL CONVERSION HANDBOOK, 3rd ED (Analog Devices), pp 317-330.

ONL error is calculated using the histogram method outlined in D. Doerfler's thesis, pp 34.

ABS error is the difference between the ADC output and the ideal straight line.

Offset and Gain errors are also calculated using the methods obtained from the A-D CONVERSIONS HANDBOOK, pp 317-330.

Version 4.0 added a loop to check for non-monotoncity and missing code. Counters keep track of the number of occurrences and the last element is also stored. The number of occurrences and the value of the last occurrences are then printed out.

Version 5.0 added Alpha & Beta correction factors, to help eliminate offset and gain errors from the data. It now calculates INL using 2 methods: endpoint (with Alpha & Beta) and the histogram method from Doerfler's thesis. Also fixed program to check for true missing code by detecting any empty bins read in from the raw data file. And added a few bells and whistles to make I/O a little more friendly.

Also using middle 2 Vtrans variables to store ADC values for the first and last transitions.

MICHAEL PANKRATZ 11 August 1987

DIM Data(1,1024), Abs(1,1024)!
DIM Bin(1,256),Onl(1,256)
DIM Inl(1,256),Hinl(1,256)
DIM Vtrans(3),Non_mono(1),Code(1)
Old=0

100

PRINT CHR$(12);CHR$(140):
530 PRINT TAB(20);"OFFSET, GAIN, INL, ONL, & ABSOLUTE ERRORS"
550 PRINT CHR$(136)
560 PRINT 
570 PRINT 
580 
590
ON ERROR GOTO No_file

CAT

Inp1:  BEEP

Infile$="A01F2"

OUTPUT 2;Infile$

INPUT "Enter input filename:";Infile$

ASSIGN @file TO Infile$  ! Open input file

OFF ERROR

GOTO Inp2

No_file:

IF ERR#SG THEN

PRINT CHR$(137);"File "&Infile$;" does not exist."

PRINT CHR$(135)

BEEP

WAIT .75

GOTO Inp1

ELSE

PRINT CHR$(137);CHR$(130)

PRINT "Error",ERR#;" occurred while opening ":Infile$

PRINT CHR$(129);CHR$(139)

STOP

END IF

Inp2:

Pos=LEN(Infile$)-3  ! Compute output filenames

Name=Infile$(Pos)$

Outfile1$="EINL_"&Name$

Outfile2$="HINL_"&Name$

Outfile3$="ONL_"&Name$

Outfile4$="ABS_"&Name$

OUTPUT 2;Outfile1$

INPUT "Enter filename for end point INL error:";Outfile1$

OUTPUT 2;Outfile2$

INPUT "Enter filename for histogram INL error:";Outfile2$

OUTPUT 2;Outfile3$

INPUT "Enter filename for histogram ONL error:";Outfile3$

OUTPUT 2;Outfile4$

INPUT "Enter filename for absolute error:";Outfile4$

END POINT INL

ENTER @file;Samples  ! Read in date

REGIM Oe(n,1,Samples-1),Abs(1,Samples-1)

ENTER @file;Date(*)

END POINT Oe

ENTER @file;0amples

REGIM Bin(1,0amples-1),Onl(1,0amples-1)

REGIM Inl(1,0amples-1),Hinl(1,0amples-1)

ENTER @file;Bin(*)

ENTER @file;Vref,Res,Vtrans(*)

END POINT

Constants:

Umin=0

Vmax=Vref

Lab=Vref/2-Res

Uf=Vtrans(0)

Vl=Vtrans(3)

Cf=Vtrans(1)+1

Voltage to cause first transition

Voltage to cause last transition

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, assign default values to
1240  Cl=255  ! Cf & Cl and set a flag
1250  Old=1
1260  END IF
1270  
1280  
1290  
1300  PRINT
1310  PRINT "Calculating offset and gain errors."
1320  ! Offset error
1330  Voffset=Uf+(Vf-Vl)/(Cl-Cf)+Lsb/2
1340  Coffset=Voffset/Lsb
1350  
1360  ! Gain error
1370  Vgain=Uref-2*Lsb+Vf-Vl
1380  Cgain=Vgain/Lsb
1390  
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)=Abs(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)/(U1-Uf)  ! Data correction constants to
1630  Beta=Lsb*((U1+Uf+Cl)/(U1-Uf)-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 and point integral nonlinearity error."
1760  Slope=254/(Vtrans(3)-Vtrans(0))
1770  Imax=-65536
1780  Imin=65536
1790  Prev=0

A-26
FOR X=0 TO Samples-1
    I=1
    FOR Y=0 TO X-1
        Hmin(I,Y)=Hmin(I,Y)+Bin(I,Y)
    NEXT Y
    Hmin(I,X)=Hmin(I,X)+Bin(I,X)
    IF Hmin(I,X)>Hmax THEN
        Hmax=Hmin(I,X)
        Max=Hmin(I,X)
    END IF
    NEXT I
    IF Hmin(I,X)<Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
    IF Hmin(I,X)>Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
NEXT X
PRINT "Calculating histogram differential nonlinearity error."
FOR X=0 TO Samples-2
    Slope=Slope+Bln(1,X)
NEXT X
Slope=Slope/(Samples-2)
Hmax=-65536
Hmin=-65536
FOR X=0 TO Samples-1
    Hmin(I,X)=X
    Hmin(I,0)=0
    FOR Y=0 TO X-1
        Hmin(I,X)=Hmin(I,X)+Bin(I,Y)
    NEXT Y
    Hmin(I,X)=(Hmin(I,X)-Bin(I,0))/Eno*(1-X)*Slope
    IF Hmin(I,X)>Hmax THEN
        Hmax=Hmin(I,X)
        Max=Hmin(I,X)
    END IF
    IF Hmin(I,X)<Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
    IF Hmin(I,X)>Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
NEXT X
PRINT "Calculating histogram integrall nonlinearity error."\nEno=0
FOR X=1 TO Samples-2
    Eno=Eno+Bin(I,X)
NEXT X
Eno=Eno/(Samples-2)
FOR X=1 TO Samples-2
    Slope=Slope+Bin(I,X)
NEXT X
Slope=Slope/(Samples-2)
Hmax=-65536
Hmin=-65536
FOR X=0 TO Samples-1
    Hmin(I,0)=X
    Hmin(I,X)=0
    FOR Y=0 TO X-1
        Hmin(I,X)=Hmin(I,X)+Bin(I,Y)
    NEXT Y
    Hmin(I,X)=(Hmin(I,X)-Bin(I,0))/Eno*(1-X)*Slope
    IF Hmin(I,X)>Hmax THEN
        Hmax=Hmin(I,X)
        Max=Hmin(I,X)
    END IF
    IF Hmin(I,X)<Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
    IF Hmin(I,X)>Hmin THEN
        Hmin=Hmin(I,X)
        Min=Hmin(I,X)
    END IF
NEXT X
PRINT "Calculating histogram differential nonlinearity error."
Omax=-65536
Omin=-65536
Eno=10*2^10/Samples
IF DAC resolution is 2^10
    ADC resolution is 2^8
END for first bin (1/2 the normal)
ENOl=3/2*Eno

FOR X=0 TO Osamples-1

Onl(0,X)=X

Onl(1,X)=Bin(1,X)/Eno-1

IF X=0 THEN Onl(1,X)=Bin(1,X)/Eno0-1

IF X=Osamples-1 THEN Onl(1,X)=Bin(1,X)/Enol-1

Calculate first bin

Calculate last bin

IF Onl(1,X)>Onmax THEN

Onmax=Onl(1,X)

Bmax=Onl(0,X)

END IF

IF Onl(1,X)<Onmin THEN

Onmin=Onl(1,X)

Bmin=Onl(0,X)

END IF

NEXT X

PRINT "Searching for non-monotonic values and missing code."

Non_monotonic=0

Missing_code=0

Prev=0

FOR X=0 TO Osamples-1

IF Data(1,X)<Prev THEN

Non_monotonic=Non_monotonic+1

Non_mono(0)=Data(0,X)

Non_mono(1)=Data(1,X)

END IF

Prev=Data(1,X)

NEXT X

IF Bin(1,X)=0 THEN

Missing_code=Missing_code+1

Code(0)=Data(0,4+X)

Code(1)=Bin(0,X)

END IF

NEXT X

PRINT_RESULTs: BEEP

OUT$="PRINTER"

OUTPUT 2:OUT$:

INPUT "List device (SCREEN/PRINTER):", Out$:

IF UPC$(Out$[1,1])="P" THEN PRINTER IS 70!

MAXFORM: IMAGE " Maximum: ": 30.30, " LSBs at bit ": 30

MINFORM: IMAGE " Minimum: ": 30.30, " LSBs at bit ": 30

PRINT TAB(20):"ERROR CALCULATIONS"

IF UPC$(Out$[1,1])="P" THEN

PRINT

ELSE

PRINT CHR$(13)

END IF

PRINT "Input Filename: ": Infile$

PRINT

PRINT A-28
3000 Cgain=ROUND(Cgain,-3) ! Round data (3 places)
3010 Coffset=ROUND(Coffset,-3)
3020 Alpha=ROUND(Alpha,-5)
3030 Beta=ROUND(Beta,-5)
3040 PRINT "Gain Error: ";Cgain:"LSBs";TAB(40);"Alpha: ";Alpha
3050 PRINT "Offset Error: ";Coffset:"LSBs";TAB(40);"Beta: ";Beta
3060 PRINT
3070 PRINT "End point Integral Nonlinearity Error ------ File: ";Outfile1$ 
3080 PRINT USING Maxform1;Max1
3090 PRINT USING Minform1;Min1
3100 PRINT
3110 PRINT "Histogram Integral Nonlinearity Error ------ File: ";Outfile2$ 
3120 PRINT USING Maxform1;Max3
3130 PRINT USING Minform1;Min3
3140 PRINT
3150 PRINT "Histogram Differential Nonlinearity Error -- File: ";Outfile3$ 
3160 PRINT USING Maxform1;Bmax
3170 PRINT USING Minform1;Bmin
3180 PRINT
3190 PRINT "Total Error -------------------------------- File: ";Outfile4$ 
3200 Amax=ROUND(Amax,-3) ! Round data
3210 Amin=ROUND(Amin,-3)
3220 Max2=ROUND(Max2,-4)
3230 Min2=ROUND(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 file, default and count values used:"
3280 Vf=ROUND(Vf,-4)
3290 Vl=ROUND(Vl,-4)
3300 PRINT Cf;"->";Cf+1: ";Vf:";v";TAB(40);C1;"->";C1: ";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 occurred!Non_monotonic;";times.
3380 Non_mono(0)=ROUND(Non_mono(0),-4)
3390 PRINT "The last occurrence was;Non_mono(1);";at;"Non_mono(0);";v"
3400 END IF
3410 !
3420 IF Missing_code THEN
3430 PRINT
3440 PRINT "Missing code occurred!Missing_code;";times.
3450 Code(0)=ROUND(Code(0),-4)
3460 PRINT "The last occurrence was;Code(1);";at about;"Code(0);";v"
3470 END IF
3480 !
3490 Eject: IF UPC$(Out$(1,1))="P" THEN PRINT CHR$(12)
3500 PRINTER IS 1
3510 !
3520 !
3530 File: !
3540 ON ERROR GOSUB File_exists
3550 !
3560 Outfiles=Outfiles$ ! Write end point Integral Error
3570 Records=INT(0SAMPLES/16)+1
3580 CREATE BOAT Outfiles$,Records
3590 ASSIGN @Outfiles$ A-29
3600 OUTPUT 00file;0samples;Inl(*)
3610 ASSIGN 00file TO *
3620 ! Write histogram Integral Error
3630 Outfile$=Outfile2$  
3640 Records=INT(0samples/16)+1  
3650 CREATE BOAT Outfile2$,Records  
3660 ASSIGN 00file TO Outfile2$  
3670 OUTPUT 00file;0samples;Inl(*)
3680 ASSIGN 00file TO *
3690 !
3700 Outfile$=Outfile3$
3710 Records=INT((0samples)/16)+1  
3720 CREATE BOAT Outfile3$,Records  
3730 ASSIGN 00file TO Outfile3$
3740 OUTPUT 00file;0samples;Inl(*)
3750 ASSIGN 00file TO *
3760 !
3770 Outfile$=Outfile4$
3780 Records=INT((0samples)/16)+1
3790 CREATE BOAT Outfile4$,Records
3800 ASSIGN 00file TO Outfile4$
3810 OUTPUT 00file;0samples;Abs(*)
3820 ASSIGN 00file 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:"occurred when writing to ":Outfile$
3930 PRINT CHR$(128);CHR$(139)
3940 STOP
3950 END IF
3960 !
3970 !
3980 Exit: !
3990 ASSIGN 01file TO *  
4000 ! Close file
4010 8EEP
4020 PRINT CHR$(139)
4030 OISP "Program terminated."
4040 IWAIT .75
4050 OISP "Loading CRUNCH program."
4060 ILOAD "CRUNCH"
4070 END
for the 9236 CRUNCH ver 3.236

This program takes the data file created by ERROR end
streamlines the data by finding the relative maxima end
minimums and throws the rest of the data away. The output
file is 1/4 the size of the input file (256 points compared
to 1024). The output file can be read by PLOT if a graph
is desired. Version 3.0 puts X axis in terms of AOC output.

MICHAEL PANKRATZ 13 July 1987

OIN In[1,1024],Out[1,256]

PRINTER IS

PRINT CHR$(12);  "Clear screen"
PRINT TAB(27);CHR$(138);"CRUNCH ABSOLUTE ERROR DATA"
PRINT
PRINT
PRINT
INPUT "Enter input INL error filename: ", Infile$  
ASSIGN @file TO Infile$  
OFF ERROR
GOTO Inp1

ON ERROR GOTO No_file
PRINT "Error";ERRN;"occurred when opening ";Infile$
PRINT CHR$(137);CHR$(139)
STOP
END IF

IF ERRN=56 THEN
PRINT "File ";Infile$;" does not exist."
PRINT CHR$(137)
BEEP
WAIT .75
GOTO Inp1
ELSE
PRINT "Error";ERRN;"occurred when opening ";Infile$
PRINT CHR$(128);CHR$(139)
STOP
END IF

INPUT "Enter crushed output filename: ", Outfile$
ENTER @file;Samples  
REOIM In[1,1, Samples-1]
500 Samples2=INT(Samples/4)
510 REDIM Out(I,Samples2-I)
520 ENTER @file:Ini(*)
530 ASSIGN @file TO *
540 !
550 !
560 PRINT "Crunching..." 
570 FOR I=0 TO Samples2-I
580 Out(I,I)=0
590 FOR J=0 TO 3
600 IF ABS(In(I,I*4+J)<ABS(Out(I,I)) THEN Nxt
610 Out(I,I)=In(I,I*4+J) . | Put max value in 2nd array
620 Out(0,I)=I | Make X axis the ADC output
630 Nxt: NEXT J
640 NEXT I
650 !
660 Write_data: | !
670 ON ERROR GOSUB File_exists
680 Records=INT((Samples2+1)/16)+1
690 CREATE BDAT Outfile#;Records | Create new file
700 ASSIGN Outfile# TO Outfile#
710 OUTPUT Outfile#;Samples2;Out(*)) | Write data to file
720 SOTO Exit
730 !
740 File_exists: !
750 IF ERRN=54 THEN
760 PURGE Outfile#
770 RETURN
780 ELSE
790 PRINT CHR$(137)|CHR$(130)
800 PRINT "Error "|ERRN);"occurred when writing to ":Outfile# |
810 PRINT CHR$(128)|CHR$(139)
820 STOP
830 ENO IF
840 !
850 !
860 Exit: !
870 PRINT CHR$(139)
880 ASSIGN @file TO * | Close files
890 ASSIGN @file TO *
900 BEEP
910 OISP "Program terminated."
920 ENO
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
oraa #$80
staa $1039

ldy #PORTA
bset 0,y $20

lda $00

CONVERT staa $1030

ldx #LOOPNO
dex
bne DELAY

ldab ADR3
stab PORTB

bclr 0,y $20
bset 0,y $20

lda $00
ldx #$00
nop
nop

bra CONVERT

Power up A/D.

Point inx to PORTA

Put A/D configuration accumulator A.

Initiate conversion.

Wait for conversion to be completed.

Send pulse to IBM that data is ready.

Waste 10 clock cycles.

B-2
SOURCE FILE: dyrcf8.src

FUNCTION: Program.

DESCRIPTION:

This program does A/D conversions in a loop that causes the HCII'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 $1039
oraa #$c0
staa $1039

ldy #PORTA
bset 0,y $20

ldaa $00

CONVERT staa $1030

DELAY ldx #LOOPNO
dex bset DELAY
bne DELAY

ldab ADR3
stab PORTB

bclr 0,y $20
bset 0,y $20

ldx #$00
ldx #$00
nop
nop

bra CONVERT

Power up A/D.
Point inx to PORTA
Put A/D configuration accumulator A.
Initiate conversion.
Wait for conversion to be completed.
Send pulse to IBM that data is ready.
Waste 10 clock cycles.
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.
ora $#80
staa $1039

ldy #PORTA          Point index 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
CONVERT               accumulator A.
staa $1030            Initiate conversion.

DELAY             
ldx $01

dex DELAY           Wait for conversion to
bne DELAY           be completed.

1dab 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 HCl'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

B-7
Power up A/D.

Point inx to PORTA

Put delay length at locations 01 and 02.

Put A/D configuration accumulator A.

Initiate conversion.

Wait for conversion to be completed.

Send pulse to IBM that data is ready.
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;

HARMONI.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;  

B-10
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;

harmon1.obj : harmon1.c local.h cmath.h complex.h
  msc/AL harmon1;

harmon1.exe : harmon1.obj fft.obj window.obj cmath.lib
  link harmon1 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;

B-13
double samp_freq,
clock_freq,
hist[RESOLUTION];

FILE *out_file;

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

    /* Set up ppi communications. */
    outp(CONTROL, PPI_CONFIG);

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

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

    /* Enter lot number of HCl1. */
    puts("Enter the lot number of the HC11");
    scanf("%s", lot_number);
    printf("The lot number is: %s\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", 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");
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),l,
      out_file);
fwrite((char *)&samp_freq,sizeof(double),l,
      out_file);
fwrite((char *)&num_pts,sizeof(int),l,
      out_file);
fwrite((char *)&intdata,sizeof(int),
      NUM_DFT_POINTS,out_file);
fclose(out_file);

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

B-17
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);
This program reads in the data and information taken in a dynamic histogram test of the hcil 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",date);
    fread(chip_number,sizeof(char),CHIP_NO_LEN,
in_file);
    printf("The chip number is %s\n",chip_number);
    fread(lot_number,sizeof(char),LOT_NO_LEN,in_file);
    printf("The lot number is %s\n",lot_number);
    fread(mode,sizeof(char),MODE_LEN,in_file);
    printf("The mode is %s\n",mode);
    fread((char *)&clock_freq,sizeof(double),1,in_file);
    printf("clock_freq is %f\n",clock_freq);
    fread((char *)&samp_freq,sizeof(double),1,in_file);
    printf("samp_freq = %f\n",samp_freq);
    fread((char *)&num_pts,sizeof(int),1,in_file);
    printf("num_pts = %i\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\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\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);

B-21
for ( i = 0; i < RESOLUTION; i++ )
    fprintf(out_file,"%i %f \n",i,diff[i]);
fclose(in_file);
fclose(out_file);
exit(0);
This program reads in the data and information taken in a dynamic test of the holl ADC. The data is first normalized to a range between 0 and 1, windowed with a Von Hann 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.

**SOURCE FILE:** dyfft.c

**FUNCTION:** main()

**DESCRIPTION:**

**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.

```c
#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.\n");
    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 < 1e-5 )
    trans_data[i].re = 1e-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 array. */

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\nEnter the filename for the ");
puts("output data.");
puts("\nA 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);
}

B-26
/******************************************************
* 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 DFT ( 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.

***END OF DESCRIPTION FILE***

#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);
}
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);
}
FUNCTION: DCOMPLEX cmplx(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

REVIZIONS: None.

*******************************************************************************/

#include "complex.h"

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

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

   return(z);
}
**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)
**
** 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);
}
#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 < le-300 )
        y[i].re = le-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);
This program is used to find the spectrum of an ideal ADC.

**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

**REVIZIONS:** Created from datgen.c.

```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;
}
```c
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

B-50
#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

/* Function 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 cmlpx();
 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:

![Transfer Function Diagram](image)

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

where:

\[ V_{\text{LSB}} = \frac{V_{\text{ref}}}{2^n}, \]

C-1
and two measurable points with the coordinates are:

\[ C_1 \text{ - first A/D transition} \]
\[ = (001, V_{\text{ref}}/16) \text{ or } (001, V_{\text{ref}}/2^{n+1}) \]

\[ C_{FS} \text{ - last A/D transition} \]
\[ = (111, 13V_{\text{ref}}/16) \text{ or } (2^n - 1, V_{\text{ref}} - 3V_{\text{ref}}/2^{n+1}) \]

Transition voltages occur at

\[ V_{\text{tran}} = (V_{\text{ref}}/2^{n+1}) (2i - 1) \text{ for } i = 1, 2, \ldots, 2^n - 1 \]

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

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

manipulating, if true endpoints, we can obtain:

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

\[
= \frac{v_{2^{n+1}}/V_{\text{ref}} - 2^{n+1} + 3}{2^{n+1} - 4}
\]

\[
\frac{C - 2^n + 1}{2(2^n - 1)} = \frac{v_{2^{n+1}}/V_{\text{ref}} - 2^{n+1} + 3}{4(2^{n-1} - 1)}
\]

\[
C - 2^n + 1 = v_{2^{n+1}/2V_{\text{ref}} - 2^{n+1}/2 + 3/2}
\]

C-2
This must equal the non-ideal equation. Therefore,

\[
\frac{a}{V_{LSB}} = \frac{C_{FS} - C_F}{V_{FS} - V_F} \quad \Rightarrow \quad a = V_{LSB} \frac{C_{FS} - C_F}{V_{FS} - V_F}
\]

and

\[
\frac{B}{V_{LSB}} + \frac{1}{2} = \frac{V_{FS}C_F - V_FC_{FS}}{V_{FS} - V_F}
\]

yields:

\[
B = \frac{V_{FS}C_F - V_FC_{FS}}{V_{FS} - V_F} - \frac{1}{2} V_{LSB}
\]

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:

\[
IN(i) = \frac{V_t(i) - \left[ V_t(1) + (i-1)(LSB)\right]}{LSB}
\]

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


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.