Applications of Solid State Deuterium NMR Spectroscopy.

William Lawrence Jarrett Jr

Louisiana State University and Agricultural & Mechanical College

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_disstheses

Recommended Citation
https://digitalcommons.lsu.edu/gradschool_disstheses/4574
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Applications of solid-state deuterium NMR spectroscopy

Jarrett, William Lawrence, Jr., Ph.D.
The Louisiana State University and Agricultural and Mechanical Col., 1988
APPLICATIONS OF SOLID-STATE DEUTERIUM NMR SPECTROSCOPY

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
the Department of Chemistry

by
William Lawrence Jarrett Jr.
B.S., Mississippi State University, 1983
August 1988
Acknowledgements

I wish to thank first and foremost Dr. Leslie G. Butler for allowing me to work with him and for his support and guidance in my research. Special thanks go to Maria Altbach, Margo Jackisch, and Kermin Guo, whose friendship made working in the lab a joy and a pleasure. I would also like to thank Marcus Nauman for his assistance in using the AM-400 spectrometer. In addition, I also thank Dr. Rodney Farlee for allowing Dr. Butler and I to use his facilities at the Dupont Research Center in Wilmington, DE, Dr. Howard Walker of Ethyl Corporation for providing samples of [PPN]+ [2HFe(CO)_4]^+, and Dr. Robert Wittebort of the University of Louisville for the SPOWDER lineshape program as well as valuable assistance in evaluating deuterium lineshapes.

I would also like to thank Dr. Mark McLaughlin and Mickey Cronan for providing samples of the [PPN]+ [2HCr(CO)_5]^− complex.

I especially thank my mother, father, brother, and my sister, as well as Lisa and Trey, for their love and support and for always reminding me about what is really important in life.
Table of Contents

List of Figures ................................................................................................ v

List of Tables ................................................................................................ xi

Abstract ........................................................................................................... xii

1. Introduction .................................................................................................. 1
   1.1 Definition of the Quadrupolar Coupling Constant and Asymmetry Parameter ............................................................... 1
   1.2 Relationships of Deuterium Quadrupolar Coupling Constants and Asymmetry Parameters with Structural and Spectroscopic Parameters ......................................................... 2
   1.3 References ......................................................................................... 7

2. ADLF Spectroscopy ..................................................................................... 9
   2.1 Basic Theory of ADLF Spectroscopy ............................................. 9
   2.2 Construction of the ADLF Spectrometer ......................................... 12
      2.2.1 Proton Detection System ................................................... 13
      2.2.2 CAMAC Crate Modules ................................................... 13
      2.2.3 Magnet, Zero-Shims Coils, and Liquid Nitrogen Dewars .... 14
      2.2.4 Sample Holders ........................................................... 14
      2.2.5 ADLF Probes ................................................................... 29
      2.2.6 Host Computer System ..................................................... 30
   2.3 Operational Description of the ADLF Spectrometer ......................... 30
      2.3.1 Zero-Field Tuning and Irradiation Amplitude Control 30
         2.3.1.1 Selection of Capacitors ............................................. 30
         2.3.1.2 Calibration of the Zero-Field Irradiation Amplitude 31
      2.3.2 Location of Proton Resonance and Determination of 90° Pulse Length ............................................................... 32
      2.3.3 Input and Adjustment of the Spin-Locking Sequence 34
      2.3.4 Initial Sample Run ............................................................. 36
   2.4 Experimental Results ......................................................................... 37
      2.4.1 Para-Substituted Phenyl Acetic Acids ................................ 37
      2.4.2 Naphthalene ............................................................................. 49
      2.4.3 Barium Chlorate Monohydrate and Methyltriphenylphosphonium Bromide ................................................. 55
2.5 References .............................................................................................. 58

3. High-Field Solid-State Deuterium NMR Spectroscopy ......................... 61
  3.1 Introduction ............................................................................................ 61
  3.2 Basic Theory of High-Field Solid-State Deuterium NMR Spectroscopy ......................................................................................... 61
  3.3 Modification of the Bruker AM-400 Spectrometer ................................ 65
    3.3.1 Home-built Probe ................................................................ 65
    3.3.2 Commercial Probe and Amplifier ..................................... 67
  3.4 Experimental Results .......................................................................... 69
    3.4.1 Bis(cyclopentadienyl)zirconium(IV) Dideuteride ......................... 69
      3.4.1.1 Introduction ................................................................ 69
      3.4.1.2 Experimental ............................................................... 70
      3.4.1.3 Results and Discussion ............................................ 73
    3.4.2 Bis(triphenylphosphine)iminium Tetracarbonyl Iron Hydride .......... 76
      3.4.2.1 Introduction ................................................................ 76
      3.4.2.2 Experimental ............................................................... 76
      3.4.2.3 Procedure for Analyzing Solid-State Deuterium Lineshapes and T$_1$ Data ................................................................. 77
      3.4.2.4 Results and Discussion ............................................ 78
    3.4.3 $\mu$-Hydrido-bis(pentacarbonyl)chromium(0) and Pentacarbonylchromium(0) hydride ................................................................. 113
      3.4.3.1 Introduction ................................................................ 114
      3.4.3.2 Experimental ............................................................... 114
      3.4.3.3 Results and Discussion ............................................ 115
  3.5 References ............................................................................................. 124

Appendix 1 .............................................................................................................. 127

Appendix 2 .............................................................................................................. 240

Appendix 3............................................................................................................... 248

Vita .............................................................................................................................. 250
List of Figures

<p>| Figure 1.1 | Plot of calculated deuterium quadrupolar coupling constants as a function of the Na-H bond distance for [Na-H-Na]⁺ | 4 |
| Figure 1.2 | Plot of the change in the deuterium quadrupolar coupling constant as a function of the [Na-H-Na]⁺ bond angle | 5 |
| Figure 1.3 | Plot of calculated values of the asymmetry parameter as a function of the [Na-H-Na]⁺ bond angle | 6 |
| Figure 2.1 | Plot of energy versus asymmetry parameter for an I = 1 spin system in zero magnetic field | 10 |
| Figure 2.2 | ADLF field cycle for a proton-deuterium system | 11 |
| Figure 2.3 | Block Diagram of the ADLF instrument | 15 |
| Figure 2.4 | Block Diagram of the pulse programmer | 16 |
| Figure 2.5 | Diagram of the pulse programmer output register block | 17 |
| Figure 2.6 | Diagram of the pulse programmer control block | 18 |
| Figure 2.7 | Diagram of the pulse programmer control block | 19 |
| Figure 2.8 | Diagram of the pulse programmer buffer block | 20 |
| Figure 2.9 | Diagram of the pulse programmer RAM memory block | 21 |
| Figure 2.10 | Diagram of the pulse programmer control block | 22 |
| Figure 2.11 | Diagram of the zero-field rf switch and volts-to-milliamps circuit of the Zero-Field module | 23 |
| Figure 2.12 | Spin-locking circuitry of the Spin-Locking module | 24 |
| Figure 2.13 | Sample-moving circuitry and RF enable line of the Spin-Locking Module | 25 |
| Figure 2.14 | Schematic of the glass dewar | 26 |
| Figure 2.15 | Schematic of the lower half of the stainless steel dewar | 27 |
| Figure 2.16 | Schematic of the top of the stainless steel dewar | 28 |
| Figure 2.17 | Block diagram of the zero-field irradiation system | 33 |
| Figure 2.18 | ADLF spectrum of phenyl[2,2-^2H]acetic acid | 39 |
| Figure 2.19 | ADLF spectrum of (4-chlorophenyl)[2,2-^2H]acetic acid | 40 |
| Figure 2.20 | Double transition spectrum of (4-chlorophenyl)[2,2-^2H]acetic acid | 41 |
| Figure 2.21 | ADLF spectra of (4-chlorophenyl)[2,2-^2H]acetic acid acquired at 0, 2.5, and 5.0 gauss | 42 |
| Figure 2.22 | ADLF spectrum of (4-bromophenyl)[2,2-^2H]acetic acid | 43 |
| Figure 2.23 | Double transition spectrum of (4-bromophenyl)[2,2-^2H]acetic acid | 44 |
| Figure 2.24 | ADLF spectra of (4-bromophenyl)[2,2-^2H]acetic acid acquired at 0, 2.5, and 5.0 gauss | 45 |
| Figure 2.25 | ADLF spectrum of (4-nitrophenyl)[2,2-^2H]acetic acid | 46 |
| Figure 2.26 | Double transition spectrum of (4-nitrophenyl)[2,2-^2H]acetic acid | 47 |
| Figure 2.27 | ADLF spectra of (4-nitrophenyl)[2,2-^2H]acetic acid acquired at 0 and 5.0 gauss | 48 |
| Figure 2.28 | ADLF spectrum of (4-fluorophenyl)[2,2-^2H]acetic acid | 50 |</p>
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.29</td>
<td>Double transition spectrum of (4-fluorophenyl)[2,2-2H]acetic acid</td>
<td>51</td>
</tr>
<tr>
<td>2.30</td>
<td>ADLF spectrum of $d_7$-naphthalene</td>
<td>54</td>
</tr>
<tr>
<td>2.31</td>
<td>ADLF spectrum of Ba(ClO$_3$) $\cdot$ $^1$H$^2$H$_2$O</td>
<td>56</td>
</tr>
<tr>
<td>2.32</td>
<td>ADLF spectrum of triphenylphosphonium- (methyl-$d_3$) bromide</td>
<td>57</td>
</tr>
<tr>
<td>3.1</td>
<td>Effect of a non-zero asymmetry parameter on the solid-state deuterium NMR lineshape</td>
<td>62</td>
</tr>
<tr>
<td>3.2</td>
<td>Effect of the rate of motion for the C-$^2$H site of a rotating methyl group</td>
<td>64</td>
</tr>
<tr>
<td>3.3</td>
<td>Plots of ln(T$_1$) versus -log($\tau_c$) for the three-site jump and free diffusion models of motion for a methyl group</td>
<td>66</td>
</tr>
<tr>
<td>3.4</td>
<td>Block diagram of the scheme used to implement the high-power Henry amplifier and Cryomagnetics probe</td>
<td>68</td>
</tr>
<tr>
<td>3.5</td>
<td>Solid-state deuterium NMR spectrum of [(η$^5$-C$_5$H$_5$)$_2$Zr($^2$H)$_2$]$_x$ acquired at room temperature</td>
<td>71</td>
</tr>
<tr>
<td>3.6</td>
<td>Plot of Salem's correlation between vibrational force constants and deuterium quadrupolar coupling constants for terminally-bonded hydrides</td>
<td>74</td>
</tr>
<tr>
<td>3.7</td>
<td>Fully-relaxed spectrum of [PPN]+$^2$HFe(CO)$_4$ acquired at 300 K</td>
<td>80</td>
</tr>
<tr>
<td>3.8</td>
<td>Fully-relaxed spectrum of [PPN]+$^2$HFe(CO)$_4$ acquired at 285 K</td>
<td>81</td>
</tr>
<tr>
<td>3.9</td>
<td>Fully-relaxed spectrum of [PPN]+$^2$HFe(CO)$_4$ acquired at 278 K</td>
<td>82</td>
</tr>
</tbody>
</table>
Figure 3.10  Fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 270 K .................................................. 83

Figure 3.11  Fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 240 K .................................................. 84

Figure 3.12  Fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 210 K .................................................. 85

Figure 3.13  Fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 180 K .................................................. 86

Figure 3.14  Plot of $\Delta v$ versus temperature for [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ ........ 87

Figure 3.15  Diagram of proposed model of motion for the deuteron site in [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ .................................................. 89

Figure 3.16  Plot of ln($T_1$) versus log($k$) for a value of $\beta$ of 8.3° .............. 90

Figure 3.17  Plot of ln($T_1$) versus log($k$) for a value of $\beta$ of 13.3° .............. 91

Figure 3.18  Plot of ln($T_1$) versus log($k$) for a value of $\beta$ of 14.9° .............. 92

Figure 3.19  Plot of ln($T_1$) versus log($k$) for a value of $\beta$ of 15.5° .............. 93

Figure 3.20  Plot of ln($T_1$) versus log($k$) for a value of $\beta$ of 18° .............. 94

Figure 3.21  Simulated fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 300 K .................................................. 95

Figure 3.22  Simulated fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 285 K .................................................. 96

Figure 3.23  Simulated fully-relaxed spectrum of [PPN]$^+$ [2$^2$HFe(CO)$_4$]$^-$ acquired at 278 K .................................................. 97
Figure 3.24  Simulated fully-relaxed spectrum of [PPN]+ [2HFe(CO)4]− acquired at 270 K .......................................................... 98

Figure 3.25  Simulated fully-relaxed spectrum of [PPN]+ [2HFe(CO)4]− acquired at 240 K .......................................................... 99

Figure 3.26  Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 300 K .............................................. 101

Figure 3.27  Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 300 K .............................................. 102

Figure 3.28  Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 285 K .............................................. 103

Figure 3.29  Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 285 K .............................................. 104

Figure 3.30  Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 278 K .............................................. 105

Figure 3.31  Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 278 K .............................................. 106

Figure 3.32  Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 270 K .............................................. 107

Figure 3.33  Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 270 K .............................................. 108

Figure 3.34  Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 240 K .............................................. 109

Figure 3.35  Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)4]− acquired at 240 K .............................................. 110

Figure 3.36  Arrhenius plot for [PPN]+ [2HFe(CO)4]− complex ............... 111
Figure 3.37 Experimental spectrum of [K(crypt-222)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(\mu-\text{H})]\(^-\) ............................................ 116

Figure 3.38 Experimental spectrum of [N(Et)\(_4\)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(\mu-\text{H})]\(^-\) ............................................... 117

Figure 3.39 Spectrum of [K(crypt-222)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(\mu-\text{H})]\(^-\)
in which half of data has been zeroed ................................. 118

Figure 3.40 Simulated spectrum for [K(crypt-222)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(\mu-\text{H})]\(^-\) ........................................... 120

Figure 3.41 Experimental spectrum of [PPN\(^+\) \(\text{[}^2\text{HCr(CO)}\(_5\)\text{]}\(^-\)] acquired at 300 K ......................................................... 121

Figure 3.42 Experimental spectrum of [PPN\(^+\) \(\text{[}^2\text{HCr(CO)}\(_5\)\text{]}\(^-\)] acquired at 200 K ......................................................... 122

Figure 3.43 Simulated spectrum of [PPN\(^+\) \(\text{[}^2\text{HCr(CO)}\(_5\)\text{]}\(^-\)] acquired at 300 K ......................................................... 123

Figure A2.1 Example input file for SPOWDER program .......... 243

Figure A2.2 Example output file for SPOWDER program .......... 244

Figure A2.3 Example input file for SPT1 program ................. 247
List of Tables

Table 2.1  Summary of ADLF spectroscopic results .......................... 52
Table 3.1  Parameters used to simulate the spectrum of 
[\(\eta^5\text{-C}_5\text{H}_5\text{Zr(}^2\text{H}_2\text{)}_2\)\text{x}]  ........................................ 72
Table 3.2  Tabulation of data used in plot of vibrational force constants 
versus deuterium quadrupolar coupling constants for 
X-H systems .............................................................. 75
Table 3.3  Experimental spin-lattice relaxation times for 
[PPN]⁺ [\(^2\text{HFe(CO)}_4\)]⁻ ............................................. 79
Table 3.4  Summary of rate data used in Arrhenius plot for 
[PPN]⁺ [\(^2\text{HFe(CO)}_4\)]⁻ ............................................. 112
Abstract

Solid-state deuterium NMR spectroscopic techniques have been applied to a series of organic and organometallic systems. The construction and operation of an ADLF (adiabatic demagnetization in the laboratory frame) spectrometer is described in detail. The instrument has been used to obtain high-field solid-state deuterium NMR spectra for a series of para-substituted phenyl acetic acids which have been deuteriated at the methylene site, as well as d1-napthalene, triphenylphosphonium-(methyl-d3) bromide, and barium chlorate monohydrate (25% deuteriated). The data for the para-substituted phenyl acetic acids show that the two methylene deuterons are inequivalent. This inequivalency is due to the difference in the $2H-C_1-C_2-O$ torsion angle for the deuterium sites; here $C_1$ refers to the methylene carbon, $C_2$ refers to the carboxylic acid carbon, and $O$ refers to the hydroxyl oxygen. In addition, static values of the deuterium quadrupolar coupling constant and asymmetry parameter have been obtained for $(C^2H_3)(C_6H_5)_3PBr$ and Ba(ClO$_3$)$_2$.1H$_2$O, two systems where motional averaging of these two parameter have been previously observed.

Conventional high-field solid-state deuterium NMR spectroscopy has been applied to three organometallic hydride systems. The spectrum of $[(\eta^5-C_5H_5)_2Zr(2H)_2]_x$ is reported; both the terminal and bridging site are observed and have been identified. Variable temperature solid-state deuterium NMR spectra have been acquired for [PPN]$^+ [^2HFe(CO)_4]^-$ . The data have been analyzed by lineshape simulations and interpreted in terms of a libration of the Fe-$^2H$ bond about the C$_3$ molecular axis of the anion. From an Arrhenius plot, the activation energy of the process is calculated to be 18(2) kJoules/mole. Preliminary results of a solid-state deuterium NMR spectroscopic study of two bridging chromium carbonyl complexes and one terminally bonded chromium carbonyl complex are also reported.
1. Introduction

Metal-hydrogen bonding is of great importance in organometallic chemistry. Hydrogen ligands can bind either terminally to a metal atom or as a bridging ligand between two or more metal centers. In addition, hydrogen atoms can also bind to metals either as molecular hydrogen or agostically through a neighboring C-H bond. This variability in bonding gives metal hydride complexes a rich stereochemistry, with many different structures known.

The goal of this research is to use solid-state deuterium NMR spectroscopy to obtain structural information concerning organometallic hydride bonding. The potential of this technique to yield such information arises from the fact that the two parameters experimentally measured, the quadrupolar coupling constant and the asymmetry parameter, are sensitive to the electronic environment surrounding the deuteron.

1.1 Definition of the Quadrupolar Coupling Constant and Asymmetry Parameter

The quadrupolar coupling constant and asymmetry parameter arise from the electrostatic interaction between the nuclear charge and the electron density about the nucleus. For nuclei with nuclear spin angular momentum, I, greater than 1/2, there exists an electric quadrupole moment which can interact with the electric field produced by the surrounding electrons and nuclei. The common form of the Hamiltonian for this interaction is:

$$H_Q = \frac{e^2 q_{zz} Q \hbar}{4 I (2I + 1)} \left( 3 I_z^2 - I^2 - \frac{\eta}{2} [I_+^2 - I_-^2] \right)$$

The $Q$ term is the nuclear quadrupole moment, a constant, and $e$ is the electrostatic unit of charge. The $eq_{zz}$ term is the largest component of the time-averaged electric field.
gradient, which is a 3 X 3 tensor quantity with its trace equal to zero. The angular momentum operators are $I_z, I, I_+, \text{and } I_-$. The asymmetry parameter term, $\eta$, is defined as:

$$
\eta = \frac{|eq_{xx}| - |eq_{yy}|}{|eq_{zz}|}
$$

By convention, the electric field gradient is defined so that $|eq_{zz}| \geq |eq_{xx}| \geq |eq_{yy}|$;\(^5\) Therefore, $\eta$ ranges from 0 to 1.

The quadrupolar coupling constant and asymmetry parameter provide complementary information about the deuterium site. Since the quadrupolar coupling constant is directly proportional to the largest component of the electric field gradient, $eq_{zz}$, it is sensitive to the electron density at the deuterium site. The asymmetry parameter, on the other hand, is a strong function of the distribution of the electron density. By measuring these two parameters, information on the bonding of the deuteron can be derived. The next section lists examples where the quadrupolar coupling constant and asymmetry parameter have been correlated with structural and spectroscopic parameters. The deuterium quadrupolar coupling constants and asymmetry parameters in this study have been obtained using both ADLF (adiabatic demagnetization in the laboratory frame) spectroscopy and conventional high-field solid-state deuterium NMR spectroscopy. A more detailed discussion of these techniques are given in chapters 2 and 3, respectively.

1.2 Relationships of Deuterium Quadrupolar Coupling Constants and Asymmetry Parameters with Structural and Spectroscopic Parameters

Previous studies show that correlations do exist for deuterium quadrupolar coupling constants and asymmetry parameters. For example, the deuterium quadrupolar coupling constants and asymmetry parameters for O-\(^2\text{H}\cdots\text{O}\) hydrogen bonding systems are
functions of the O•••O bond distance and the linearity of the hydrogen bond.\textsuperscript{6-8} As the
O•••O bond distance increases, the deuterium quadrupolar coupling constant increases,
while the asymmetry parameter decreases. This has been verified by experimental
deuterium quadrupolar coupling constants and asymmetry parameters and by calculations
performed on a formaldehyde-water model.\textsuperscript{7} Salem has shown that a linear correlation
exists between deuterium quadrupolar coupling constants and vibrational force constants
for terminally bonded hydrides.\textsuperscript{9} Deuterium quadrupolar coupling constants have been
used to probe the electron density of the methylene carbon bridging two metal centers.\textsuperscript{10}
In addition, it has been proposed that a Karplus-type relationship may exist for the
deuterium quadrupolar coupling constants of C-2H methylene bonds neighboring a
carboxylic acid group.\textsuperscript{11}

Calculations have been done for a model bridging metal hydride system,
[Na–H–Na]\textsuperscript{+}, in order to determine if any trends similar to those for hydrogen bonding
systems exist.\textsuperscript{12} The results for these calculations are shown in Figures 1.1 thru 1.3.
The calculations were done using a Gaussian-82 package and STO-3G basis sets; for
some calculations, additional p and d orbitals have been added to the hydrogen atom and
sodium atoms, respectively. Figure 1.1 illustrates how the calculated deuterium
quadrupolar coupling constant decreases as the Na-H bond distance increases for a linear
Na-H-Na system. This is due to the reduction of the nuclear contribution to the largest
component of the electric field gradient tensor as the Na-H bond distance increases
(equation 1.3):

\[ e_{Q_{zz}} = \sum_n K_n \frac{3z_n^2 - r_n^2}{r_n^5} - e \left( \sum_i \frac{3z_i^2 - r_i^2}{r_i^5} \right) \psi^* \psi \quad \text{eq. (1.3)} \]

Here \( n \) is the summation of all other atoms with nuclear charge \( K_n \), and \( i \) is the
summation of all the electrons in the molecule. This is similar to the trend seen for
Figure 1.2. Plot of calculated deuterium quadrupolar coupling constants as a function of the Na-H bond distance. The Na-H-Na bond angle was fixed at 180°. The values represented by the circles were obtained using standard STO-3G basis sets, while the values represented by the squares were calculated using basis sets which have been modified so that p and d orbitals were added for the hydrogen and sodium atoms, respectively.
Figure 1.2. Plot of the change in the deuterium quadrupolar coupling constant as a function of the [Na-H-Na]$^+$ bond angle. The results of the calculations are presented as the difference between the value of $e^2q_{zz}Q/h$ found at 180° and the values of $e^2q_{zz}Q/h$ determined for different [Na-H-Na]$^+$ bond angles. The Na-H bond distance is fixed at 1.7 Å, and standard STO-3G basis sets have been used.
Figure 1.3. Plot of calculated values of the asymmetry parameter as a function of the Na–H–Na angle. The value of the Na-H bond distance was fixed at 1.7 Å, and standard STO-3G basis sets were used.
symmetric O-H-O systems, where the formation of a hydrogen bond leads to an increase in the O-H bond distance and a reduction in the deuterium quadrupolar coupling constant. Since the formation of a bridging hydrogen bond generally increases the M-H bond distance, the deuterium quadrupolar coupling constant should be lower for a bridging deuteron relative to a terminally-bonded deuteron. Note in Figure 1.1 that the addition of p and d orbitals only slightly affects the calculated values of the deuterium quadrupolar coupling constants and does not alter the overall trend. Figure 1.2 shows how the quadrupolar coupling constant decreases as the Na-H-Na bond angle is varied from 180° to 140°. In addition, the asymmetry parameter increases with the decreasing linearity of the Na-H-Na bond (Figure 1.3). From these calculations and from the trends noted earlier for O-^H-•O systems, the value of the deuterium quadrupolar coupling constant is predicted to decrease as the metal hydride system goes from a terminal to bridging hydrogen bond system.

1.3 References


2. ADLF Spectroscopy

2.1 Basic Theory of ADLF Spectroscopy

In the absence of a magnetic field, the degeneracy of the energy levels of an I = 1 spin system is lifted by the presence of the quadrupolar moment. Figure 2.1 shows how the difference in the energy levels is a function of the quadrupolar coupling constant and the asymmetry parameter. Below Figure 2.1 the relationships between the three possible quadrupolar transitions and the quadrupolar coupling constant and the asymmetry parameter are given. Note that it is possible to determine the quadrupolar coupling constant and the asymmetry parameter by measuring any two of the quadrupolar transitions.

ADLF spectroscopy is a field-cycling technique which has proven valuable in studying quadrupolar nuclei such as $^2$H, $^{14}$N, $^{17}$O, $^{27}$Al, and $^{39}$K. The technique is especially useful in studying non-abundant nuclei such as $^{17}$O, since it detects quadrupolar transitions by monitoring the magnetization of a more abundant spin system, such as protons. Figure 2.2 illustrates the basic field cycle for a sample containing proton and deuterium spins. First, the sample is placed in a region of high magnetic field for a time comparable to the proton spin-lattice relaxation time, $T_1$. The sample is then physically transported to a region of zero magnetic field and irradiated with rf energy of a specific frequency. If the time required to move the sample from high to zero field is short compared to the spin-lattice relaxation time of the proton spin system, the process has occurred adiabatically. After irradiation in zero field for a time shorter than the zero-field spin-lattice relaxation time of the proton spin system, $T_{1d}$, the sample is then returned to the region of applied field. During the movement of the sample to and from regions of high and zero magnetic field, there is a transfer of magnetization between the deuterium spin system and the proton spin system by level crossing. If the zero-field irradiation frequency corresponds to one of the deuterium quadrupolar transitions, the deuterium spin temperature is increased, due to the fact that two of the three deuterium
Figure 2.1. Plot of energy versus asymmetry parameter for an $I = 1$ system in zero magnetic field. The quadrupolar transition frequencies and their relationship to the quadrupolar coupling constant and asymmetry parameter are given below the plot.

$$v_+ = \frac{3}{4} \frac{e^2 q_{zz} Q}{\hbar} (1 + \eta/3)$$

$$v_- = \frac{3}{4} \frac{e^2 q_{zz} Q}{\hbar} (1 - \eta/3)$$

$$v_0 = \frac{1}{2} \eta \frac{e^2 q_{zz} Q}{\hbar}$$

$$v_0 = v_+ - v_-$$

$$\eta = \frac{2 (v_+ - v_-)}{e^2 q_{zz} Q/\hbar}$$
Figure 2.2. ADLF field cycle for a proton-deuterium system. In region A, the proton spins are polarized by the applied field. After moving the sample adiabatically to zero-field, region B, the deuterium spins are irradiated at a specific rf frequency. Then the sample is returned to high-field, and the proton magnetization is measured using a spin-locking sequence. The cycle is repeated for different zero-field rf frequencies.
energy levels have become more equally populated. Thus when level crossing occurs between the deuterium and proton spin systems, magnetization is transferred to the deuterium spin system from the proton spin system, resulting in a decrease in the amount of proton magnetization. After the sample has been returned to a region of applied field, the proton magnetization is measured either by using a 90° pulse and digitization of the free induction decay (FID), or by using an Ostroff-Waugh pulse sequence and digitization of the resulting spin-echoes. The cycle is repeated for each increment of the zero-field irradiation frequency. The resulting spectrum is a plot of proton magnetization versus zero-field irradiation frequency, with the deuterium quadrupolar transitions, $v_0$, $v_-$, and $v_+$, seen as the minima of the spectrum. The $v_+$ transition is generally the most intense, since the $l_0>$ and $l_+>$ energy levels have the largest difference in spin populations. The $v_0$ transition is not normally observed for deuterium due to the signal from dipolar-coupled protons, i.e. the proton dipolar bath. In addition, if two deuterons are within close proximity of each other ($\sim 4$ Å), then it is possible to detect double transitions. These double transitions result from the simultaneous transitions of the dipolar-coupled deuterons upon the adsorption of a single radiofrequency phonon. These double transitions occur at frequencies which are the sums of the quadrupolar transitions. For the ADLF spectra reported here, only the double transitions corresponding the sum of the $v_+$ transitions were observed.

2.2 Construction of the ADLF Spectrometer

The ADLF spectrometer was built using a design by Cheng and Brown. However, modifications have been implemented which allow computer control over data acquisition and zero-field irradiation tuning and amplitude. Extensive use was made of CAMAC devices in the construction of the spectrometer. CAMAC is an international standard for computer instrumentation; modular devices are plugged into a multi-receptacle housing, termed a CAMAC crate, from which they draw power. In addition, the crate also
provides communication lines, called a dataway, on which the devices can send and receive digital information. Interfacing between the host computer and the CAMAC modules was accomplished via a CAMAC device termed the CAMAC crate controller.

2.2.1 Proton Detection System. A Novex transceiver, along with a Novex 400 watt amplifier and preamp as well as a Syntest frequency synthesizer, are used for proton detection. The Novex units are modular devices which are plugged into a Tektronix TM506 power module. Remote operation of the Novex transceiver by the pulse programmer (located in the CAMAC crate) is accomplished by a flat ribbon cable connected between the back of the pulse programmer and the back of the Tektronix power module; this connection allows the pulse programmer to control the phase, length, and amplitude of the rf pulses. Manual operation is accomplished through control knobs located on the front panel of the Novex units. Special features of the Novex system include quadrature detection capability and adjustable receiver deadtime.

2.2.2 CAMAC Crate Modules. A Standard Engineering Optima-850 crate is used to house the CAMAC devices. A Kinetic Systems GPIB-3988 crate controller serves as the interface between the other modules and the host computer, with the crate controller and the computer connected via an IEEE-488 interface bus. Timing of all instrument operations is governed by a Kinetic Systems 3655 timing generator. The proton signal is accumulated by two Transiac 2108 8K by 24 bit signal averagers, which are capable of digitization rates up to 20 MHz and have external clock capability as well. A Kinetic Systems 3072 48 bit output register gives the host computer control over a Comstron 1013 frequency synthesizer used for zero-field irradiation. Another Kinetic Systems 3072 unit is used for computer-control over a system of capacitors, which are used to tune the zero-field coil; this capacitor system was obtained from the University of Illinois on a long-term loan basis and has been described in detail elsewhere. A Standard Engineering Model E256 4 channel 16 bit digital-to-analog converter, along with a Fluke 8840A digital multimeter (connected to the host computer via the IEEE-488 bus) and a
CAMAC module with locally designed circuitry (designated as the Zero-Field module) are required to regulate the amplitude of the zero-field irradiation. A pulse programmer with 1024 steps at 0.1 µs resolution, along with another module (designated as the Spin-Locking module), are used to generate the spin-locking pulse sequence needed for measuring the proton magnetization; The Spin-Locking module also has circuitry for driving the solenoid of the air piston needed to shuttle the sample to and from regions of high and zero magnetic field. Figure 2.3 shows a block diagram of the ADLF spectrometer, while Figures 2.4 - 2.13 give the schematics of the pulse programmer and CAMAC modules.

2.2.3 Magnet, Zero-Field Shim Coils, and Liquid Nitrogen Dewars. A Varian XL-100 electromagnet, equipped with a 0-120 ampere power supply, serves as the source of high magnetic field. When running at a current of approximately ten amps, the electromagnet produces a field of 0.3 Tesla, which corresponds to a proton Larmor frequency of 15 MHz. The gap between the poles of the magnet is 38 mm, large enough to insert a liquid nitrogen dewar for low temperature (77 K) work. Initially, a glass dewar was used; it has been replaced with a stainless steel dewar equipped with a copper interior wall for radio frequency shielding. Figures 2.14 thru 2.16 show the designs of both dewars. A Helmholtz coil is used to create a region of zero magnetic field. The zero-field region was calibrated using a gaussmeter and by acquiring the ADLF spectrum of \( d_2 \)-acetamide, \((\text{CH}_3\text{CON}_2\text{H}_2)\) which has been reported previously.9

2.2.4 Sample Holders. The sample holders are composed of two Teflon pieces. The main body has a diameter of 5/16" and a length of 1.25". The middle has been drilled out to a depth of approximately 1.125" and a width of .25". The cap has a diameter of .25" and a length of 1 1/16". One end of the cap has been tapped for a 6-32 screw, which holds the sample holder on to a fiberglass rod attached to the air piston. The cap is inserted into the main body so that the sample holder has an overall length of 1 15/16". A hole is drilled through the side of the assembled sample holder to allow the insertion of a
Figure 2.3. Block Diagram of the ADLF instrument.
Figure 2.4. Block diagram of the pulse programmer.
Figure 2.5. Diagram of the pulse programmer output register block.
Figure 2.6. Diagram of the pulse programmer control block.
Figure 2.7. Diagram of the pulse programmer control block.
Figure 2.8. Diagram of the pulse programmer buffer block.
Figure 2.9. Diagram of the pulse programmer RAM memory block.
Figure 2.10. Diagram of the pulse programmer control block.
Figure 2.11. Diagram of zero-field rf switch and volts-to-milliams circuit of the Zero-Field module.
Figure 2.12. Spin-Locking circuitry of the Spin-Locking module.

W = 7400
L = 7408

+5 V

W

20 PF

1 KΩ

+5 V

L

0.01 μF

4.7 KΩ

5 KΩ

0.1 μF

100 PF

3.3 KΩ

+5 V

RESET

+5 V

SWITCHES
Figure 2.13. Sample-moving circuitry and RF enable line of the Spin-Locking Module.
Figure 2.14. Schematic of the glass dewar.
Figure 2.15. Lower half of the stainless steel dewar. Note the copper tube used in the lower inside wall for rf shielding. The inner wall is wrapped in Mylar in order to provide insulation. The upper flange has been grooved out for an O-ring.
Figure 2.16. Top cover for the stainless steel dewar. The top is made entirely of brass.
small Kel-F rod to prevent the main body from slipping off the cap. The sample holders can hold approximately 300 to 500 mg of material.

2.2.5 ADLF Probes. Two probes, which also serve as sample transport tubes, have been constructed, one for the glass dewar and one for the stainless steel dewar. Each probe consists of a glass tube with an OD of 13 mm and an ID of 10 mm. At one end of the glass tube a proton detection coil is wound, with the zero-field irradiation coil located 27" from it, measured from center-to-center. Both probes are characterized by their quality factors (Q) and their ringdown times. The quality factor is defined as the energy stored by a circuit divided by the energy which the circuit dissipates. It is a parameter that is related to the effective bandwidth and the power-transmitting efficiency of a probe. The Q can be estimated by dividing the resonant frequency of the probe by the width at half-height of its resonance (as observed using a sweep generator and an oscilloscope). The ringdown time is defined as the time required for the probe ringing to decay so that it is indistinguishable from the baseline. Another parameter used to evaluate the two probes is the inherent noise level of the baseline. For this work, the noise level was determined by measuring the peak-to-peak voltage of the baseline with an oscilloscope and with a setting of -3 db gain on the Novex transceiver.

The detection coil for the glass dewar probe is composed of 12 turns of 18 gauge magnet wire and has a length of 14 mm. A single fixed-value ceramic capacitor in series with the detection coil is used for tuning and impedance matching. The coil is heavily wrapped with Teflon tape in order to provide mechanical stability and to insulate the coil from any nearby metal surfaces. Brass shim stock of .001" thickness is wrapped around the coil and capacitor and is connected to electrical ground in order to shield the detection coil from any external radio frequency noise. The Q of the probe is approximately 20, and the noise level of the baseline at -3db gain was 20 millivolts peak-to-peak. At an rf power level of 137 watts, the 90° pulse length was 3 µs with a ringdown time of 10 µs.

The detection coil of the stainless steel dewar probe is composed of 16 turns of 18 gauge magnet wire and has an overall length of 18 mm. The coil is tuned and impedance
matched with two Polyflon variable capacitors (0.7 - 10 pf range), used in a tapped parallel arrangement;\textsuperscript{10} the capacitors are connected to the coil by an eight inch long piece of RG-58 coaxial cable. The coil is heavily wrapped with Teflon tape for mechanical stability. The initial Q of the probe was 185; however, this large value of Q created a severe probe ringing problem (ringdown time $> 50 \, \mu s$). The Q value of the probe was therefore reduced to 50 by soldering a 12 kΩ resistor in parallel with the coil. At this value of Q and using 360 watts of rf power, the 90° pulse length was 1.3 $\mu s$, with a ringdown time of 18 $\mu s$. The baseline noise at -3db gain was 20 millivolts peak-to-peak.

The zero-field irradiation coils for both probes were similar. The coil for the glass dewar probe was 1" long and composed of 27 turns of 18 gauge magnet wire, giving it a measured inductance (using a grid-dip meter) of 5.6 microHenrys. The coil for the stainless steel probe is 1.3" long and composed of 35 turns of 18 gauge magnet wire, giving it a calculated inductance of 4.7 microHenrys.\textsuperscript{11}

2.2.6 Host Computer System. An IBM C/S 9000 computer, using the Motorola 68000 CPU and equipped with an IEEE-488 interface port, served as the controlling computer for the spectrometer. The software needed for instrumentation is composed of approximately 5000 lines of Pascal and was written in house by Professor L. G. Butler and W. L. Jarrett; the program, along with the linking and compiling steps, is given in Appendix 1.

2.3 Operational Description of the ADLF Spectrometer

2.3.1 Zero-Field Tuning and Irradiation Amplitude Control. In order to maximize the efficiency of the rf radiation applied in zero-field, it is necessary to match the impedance of the coil to the amplifier as closely as possible. A system of computer-controlled capacitors which are in series and parallel to the coil accomplishes this task.

2.3.1.1 Selection of Capacitors. To begin selection of the proper capacitors needed to tune the coil for the frequency range desired, the capacitor system is first set to the
manual mode, the Comstron is set to the lower range of the frequencies being scanned, and the rf output to the zero-field coil is observed with an oscilloscope. For the frequency range needed for deuterium, the series capacitors are not required and are therefore isolated from the active circuit by a short circuit, thereby leaving only the parallel capacitors for tuning. The zero-field rf peak-to-peak current, Ipp, is monitored as parallel capacitance is manually added to or taken out of the tuning circuit through the use of switches located on the front panel of the system. When Ipp is maximized by the appropriate choice of capacitors, the effective inductance of the coil is calculated by the CAP_FIND_TUNING_POINTS subroutine in the master program, which is user-accessed as option one of the capacitor menu. This effective value of inductance is used as a constant by the CAP_SETTING subroutine of the master program to automatically switch on or off the capacitors required to tune the coil at specific frequencies using the equation:

$$f_0 = \left(\frac{2\pi LC}{2}\right)^{1/2} - 1 \quad \text{eq. (2.1)}$$

where $f_0$ is the frequency of the zero-field rf irradiation, $L$ is the inductance of the coil as calculated by the CAP_FIND_TUNING_POINTS subroutine, and $C$ is capacitance required to tune the circuit.

2.3.1.2 Calibration of the Zero-Field Irradiation Amplitude, $H_p$. To determine the magnitude of the applied zero-field rf irradiation, the peak current flowing through the coil, $I_p$, must first be determined. This is done by measuring $V_{pp}$ with an oscilloscope and then using the relationship

$$I_p = 0.5 \left(\frac{V_{pp}}{X_L}\right) \quad \text{eq. (2.2)}$$

where $X_L = 2\pi f_0 L$ is the impedance of a coil with inductance $L$ at a frequency $f_0$. For a solenoid coil, integration of the Biot-Savat law shows that the peak rf field $H_p$ (in units of Telsa) created by $I_p$ (in units of amps) is
where $l$ is the length of the coil in units of meters, $x$ is equal to $l/2$, $r$ is the radius of the coil in units of meters, $n$ is the number of turns of wire of the coil, and $\mu_0 = 4\pi \times 10^{-7}$ Henry/m. In order to monitor $I_p$, the current-carrying cable is passed through a ferrite toroid which has been wrapped with five loops of wire. The peak current generates a proportional voltage in the wire which, after being rectified, is measured by the Fluke 8840A digital multimeter and read by the IBM 9000 computer. A table of induced voltage versus $H_p$ was thus created by recording the measured values of induced voltage for different values of $I_p$.

Computer control over the process of regulating $H_p$ is illustrated in Figure 2.17. The scheme is similar to one used previously. The output of the Comstron is fed through a Hewlett-Packard 10534A mixer, which acts as a variable attenuator. The amount of attenuation is governed by the current applied to the middle port. The current to the mixer is produced by the CAMAC digital-to-analog converter (DAC) and the Zero-Field module. The DAC generates voltages in the range of 0 - 10 volts, which is converted by the Zero-Field module to a corresponding 0 - 10 milliamp current. At the beginning of the ADLF cycle for each frequency, the IBM 9000 computer selects the proper value of capacitors needed for tuning based on the frequency of the zero-field irradiation and sets the initial current to the mixer to 2.5 milliamps. The computer monitors the $H_p$ produced via the Fluke multimeter. If the $H_p$ created is not the desired value, the computer increments or decrements the current to the mixer until a value within $\pm 5\%$ of the desired value is attained.

2.3.2 Location of Proton Resonance and Determination of 90° Pulse Length. To adjust the magnetic field to the resonance condition for protons, a standard sample with a
Figure 2.17. Block diagram illustrating computer control over the zero-field irradiation amplitude.
short spin-lattice relaxation time is placed in the high-field region of the probe, the Novex rf system is put into the repeating single-pulse mode at a rate of approximately one pulse per second, and the oscilloscope is set-up so that a proton signal can be observed after each rf pulse. The standard sample used by our group is bis(cyclopentadienyl)zirconium dichloride, which has a proton spin-lattice relaxation time at 77 K of approximately 0.5 to 1 second. The magnetic field is then adjusted until the proton signal is seen on the oscilloscope. Sometimes it is possible to mistake the $^{19}$F signal from the Teflon tape for the proton signal. The two nuclei are distinguished from each other by removing the sample from the proton detection coil. If a signal is still observed, it is due to the fluorine atoms in the teflon tape wrapped around the coil, and the magnetic field has to be adjusted to a slightly lower value. Once the magnetic field is adjusted so that it is on resonance for protons, the 90° pulse length can be determined. This is accomplished by measuring the 180° pulse length and dividing that value by two. Although this method gives a good approximation of the 90° pulse length, it does not give an exact value since it assumes that the rf pulses are perfectly symmetric; this is generally not true for NMR experiments. However, for the work reported here the amount of error introduced by this method is negligible.

2.3.3 Input and Adjustment of the Spin-Locking Sequence. Once the 90° pulse length has been determined, the spin-locking sequence is determined. The Ostroff-Waugh pulse sequence is $90^\circ_x - \tau - [\theta_y - 2\tau]_n$, where $\theta$ can range from $45^\circ - 90^\circ$; for spectra reported in this work, the value of $\theta$ was $65^\circ$. In order to execute the spin-locking sequence, the following sequence is programmed into the pulse programmer:
Here HALT is an internal programming command which stops the program at the point where it is invoked, $90^\circ_x$ and $\theta_y$ are rf pulse commands sent to the Novex transceiver, and DT, ST, X1, and X2 are external trigger pulses issued by the pulse programmer. The value of $\tau$ is generally set to be slightly greater than the ringdown time of the probe, about 10 to 18 $\mu$s. At the beginning of the ADLF cycle for the frequency being scanned, the computer triggers the pulse programmer twenty times, so that at the end of this command the pulse programmer address counter is at the HALT position.

After a delay approximately equal to the proton spin-lattice relaxation time, the Kinetic Systems timing generator issues three sequential trigger pulses. The first two pulses toggle a 7473 flip-flop IC chip on the Spin-Locking module so that it enables the sample-moving circuitry to shuttle the sample to and from the region of zero magnetic field. The same circuit also toggles on and off the zero-field irradiation via the RF ENABLE line (Figures 2.11 and 2.13). The last trigger pulse is received by the pulse programmer. After the JUNK time delay of 50 to 80 $\mu$s, the pulse programmer issues a $90^\circ_x$ pulse command to the Novex transceiver and a scope trigger, ST, to an oscilloscope in order to monitor the spin-locking signal. The digitizer trigger, DT, and the X1 pulses are sent $\tau - 1$ microseconds after the $90^\circ_x$ rf pulse. The DT trigger pulse enables the Transiac signal averagers to begin acquisition of data. The digitizers are preset so that they acquire one data point for every external X2 clock pulse issued from the pulse programmer. The X1 pulse initiates the spin-locking circuitry on the Spin-Locking module. Upon receiving the X1 pulse from the pulse programmer, the Spin-Locking module begins sending pairs of trigger pulses to the pulse programmer, which are termed RESET and CONTINUE. The RESET pulse sets the program in the pulse programmer back to the beginning, while the CONTINUE pulse triggers the pulse programmer to resume operation. The RESET and CONTINUE pulses are separated by a delay of 1 $\mu$s. After receiving the RESET/CONTINUE pulse pair, the pulse programmer issues a $\theta_y$ pulse. The X2
clocking pulse is then sent to the signal averagers after a delay of \( \tau \) microseconds, so that the data points are taken at the maximum intensity of each spin echo. The number of \textsc{reset/continue} pulses is governed by a DIP switch located in the \textsc{spin-locking} module. A variable resistor, also found in the \textsc{spin-locking} module, controls the time between \textsc{reset/continue} pulse pairs; The delay between pulse pairs is initially set to be equal to the sum \( 2 \tau + \theta_y \). All the spectra reported here were acquired using 128 spin echoes.

After writing the pulse sequence to the pulse programmer, the spin-locking echoes for the standard sample are observed with an oscilloscope. The echoes should decay exponentially. Sometimes the spin-locking echoes do not decay in a smooth exponential fashion. This can be corrected by adjusting the Syntest frequency synthesizer either up or down by \( \sim 100 \) kHz. After adjusting the frequency in order to obtain a series of smoothly decaying spin-echoes, the time between \textsc{reset/continue} pulse pairs is adjusted (via the variable resistor) in order to insure that the maximum intensity of the spin-echoes occur at a time \( \tau \) after the \( \theta_y \) pulse. This is accomplished by adjusting the variable resistor until maximum integral values of signal strength are obtained for the standard sample using the procedure which measures the proton spin-lattice relaxation time of a sample; this procedure is discussed in more detail in the section 2.3.4.

\textbf{2.3.4 Initial Sample Run.} After having optimized the experimental parameters with the standard, samples of interest are now run. First, the proton spin-lattice relaxation is determined for the sample using the \textsc{zero\_spin\_lattice} subroutine, which is option one in the main zero-field NMR menu of the master program. This procedure uses a saturation-recovery method in which sample is initially saturated with 20 computer-triggered spin-locking sequences. After a delay of one second, the proton magnetization is measured. The routine is then repeated for a delay which is twice the value of the preceding delay, up to a limit of 1024 seconds.

After obtaining the proton spin-lattice relaxation time for high magnetic field, the zero-
field spin lattice relaxation time for the sample is determined. This is done by the subroutine ZERO_ZERO_SPIN_LATTICE, which is option two in the zero-field NMR menu. The procedure starts by issuing 20 computer-triggered pulses to the pulse programmer. After a delay approximately equal to the high-field spin-lattice relaxation time, the sample is shuttled to the region of zero-field and is kept there for 0.5 seconds. The sample is then shuttled back to high-field and the proton magnetization is measured. The procedure is repeated in 0.5 second increments up to a maximum zero-field delay of 5 seconds.

After determining the proton spin-lattice relaxation times in high and zero magnetic field, the zero-field irradiation amplitude, $H_p$, is checked for the frequency region being swept. This can be done automatically by the CAP_TEST procedure, which is option two of the capacitor menu. The procedure steps through the desired frequency range in 10 kHz steps and measures the $H_p$ obtained for each step. If $H_p$ is too low, the output of the Comstron is increased until the desired value of $H_p$ is obtained. Another way to check is to use option three of the capacitor menu, which sets the capacitors and the Comstron for an input frequency and $H_p$. By checking the lower and upper limits of the range being scanned, one can make sure the Comstron is set for the proper output. After finishing the preliminary steps, a low resolution scan is performed first, usually using 1 or 2 kHz increments and a $H_p$ of 44 milligauss or more, in order to quickly locate the transitions; a high resolution scan then follows, with increments of 0.5 kHz and a typical $H_p$ of 29 milligauss or less in order to reduce the effects of power broadening.2

2.4 Experimental Results

2.4.1 Para-Substituted Phenyl Acetic Acids. The ADLF spectrum for phenyl[2,2-$^{2}$H] acetic acid is shown in Figure 2.18. There are two sets of $v_-$ and $v_+$ transitions, indicating that two inequivalent deuterium sites are present. Similar spectra are seen for (4-chlorophenyl) and (4-bromophenyl)[2,2-$^{2}$H] acetic acids (Figures 2.19 and 2.22,
respectively), while a 3 peak spectrum is observed for (4-nitrophenyl)[2,2-2H] acetic acid. Figures 2.21, 2.24, and 2.27 illustrate the use of Zeeman perturbation to identify \( v_- \) and \( v_+ \) transitions. Deuterium quadrupolar transitions are shifted by the application of small magnetic fields in a characteristic fashion; The \( v_- \) transition is asymmetrically broadened to lower frequency, while the \( v_+ \) transition is shifted to higher frequency. Thus it is possible to distinguish between \( v_+ \) and \( v_- \) transitions even in cases where the peaks overlap. An example of such a situation is the single quantum ADLF spectrum for (4-nitrophenyl)[2,2-2H]acetic acid. In zero magnetic field there are only three peaks visible. When a 5 gauss magnetic field is applied, however, the central peak is broadened in both frequency directions, indicating that there are overlapping quadrupolar transitions present.

The ADLF spectra for these compounds show the presence of two inequivalent deuterium sites for the methylene carbon. The existence of double transition spectra for (4-chlorophenyl), (4-bromophenyl), and (4-nitrophenyl)[2,2-23H]acetic acids (Figures 2.20, 2.23, and 2.26, respectively) confirm that the inequivalent deuterium sites are located on the same methylene carbon and are not due to the presence of different crystallographic forms of the materials. The inequivalency of the methylene carbon sites can be traced to the hydroxyl oxygen atom of the neighboring carboxylic acid group. Single crystal X-ray studies for the (4-chlorophenyl)-, (4-bromophenyl)-, and (4-nitrophenyl)acetic acids reveal that in the solid state one of the deuterium sites of the methylene carbon is closer to the hydroxyl oxygen than the other. The oxygen atom thus has a greater influence on the electronic environment of one methylene deuteron with respect to the other. This unequal perturbation is evident in the differing quadrupolar coupling constants and asymmetry parameters for the two sites. This type of neighboring oxygen effect has been used to explain the unusual asymmetry in the spectrum of thymine-methyl-\( d_3 \). In addition, the effect of a neighboring oxygen atom on the quadrupolar coupling constants and asymmetry parameters of methylene deuterons has
Figure 2.18. ADLF spectrum of phenyl[2,2-2H] acetic acid (~96%; C²H₂ = 100%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 350 and 3 seconds, respectively. The amplitude of the zero-field irradiation was 21 milligauss.
Figure 2.19. ADLF spectrum of (4-chlorophenyl)[2,2-2H]acetic acid (~93%; C\text{\textsubscript{2}H\textsubscript{2}} = 100%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 360 and 3 seconds, respectively. The zero-field irradiation amplitude was 34.4 milligauss.
Figure 2.20. Double transition 77 K ADLF spectrum of (4-chlorophenyl)[2,2-$^{2}$H]acetic acid. The high- and zero-field delays were 360 and 4 seconds, respectively. The zero-field irradiation amplitude was 1.9 gauss.
Figure 2.21. ADLF 77 K spectra of (4-chlorophenyl)[2,2-\textsuperscript{2H}] acetic acid which have been acquired at applied magnetic fields of a) 0 gauss, b) 2.5 gauss, and c) 5.0 gauss.
Figure 2.22. ADLF spectrum of (4-bromophenyl)[2,2-^2H]acetic acid (~50%; C^2H_2 = 100%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 300 and 2 seconds, respectively. The zero-field irradiation amplitude was 34 milligauss.
Figure 2.23. Double transition 77 K ADLF spectrum of (4-bromophenyl)[2,2-$^2$H]acetic acid. The high- and zero-field delays were 300 and 2 seconds, respectively. The zero-field irradiation amplitude was 3 gauss.
Figure 2.24. ADLF spectra of (4-bromophenyl)[2,2-$^2$H]acetic acid acquired at applied magnetic fields of a) 0 gauss, b) 2.5 gauss, and c) 5.0 gauss.
Figure 2.25. ADLF spectrum of (4-nitrophenyl)[2,2-²H]acetic acid (~80%; C²H₂ = 100%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 180 and 3 seconds, respectively. The zero-field irradiation amplitude was 21 milligauss.
Figure 2.26. Double transition 77 K ADLF spectrum of (4-nitrophenyl)[2,2-^2H]acetic acid. The high- and zero-field delays were 180 and 3.5 seconds, respectively. The zero-field irradiation amplitude was 1.9 gauss.
Figure 2.27. ADLF spectra of (4-nitrophenyl)[2,2-\textsuperscript{2}H]acetic acid acquired at magnetic fields of a) 0 gauss and b) 5.0 gauss.
been simulated using an acetic acid dimer model.\textsuperscript{18}

Normally, $v_+$ and $v_-$ transitions observed in the single quantum spectrum can be identified by the use of Zeeman-perturbed and double transition spectra. In the case of (4-fluorophenyl)[2,2-\textsuperscript{2}H]acetic acid, however, the use of these techniques could not identify the quadrupolar transitions. The single quantum spectrum (Figure 2.28) is composed of a complex pattern of overlapping peaks; moreover, the double transition spectrum (Figure 2.29) has three peaks instead of the one expected for a single pair of deuterons which are dipolar coupled. Multiplet spectral structures due to $^2\text{H}-^2\text{H}$ coupling are often seen in the ADLF spectra of highly deuteriated samples;\textsuperscript{9} however, these additional peaks are not as intense as the $v_-$ and $v_+$ transitions, which is the case for (4-fluorophenyl)[2,2-\textsuperscript{2}H]acetic acid. A more plausible explanation is that there may be more than one crystallographic form present in the sample.\textsuperscript{15} The latter explanation is supported by the double transition spectrum, which indicates that more than one pair of deuterons are present. Although the crystal structure of the compound at room temperature did not indicate the presence of more than one crystallographic form, it does not rule out the possibility of a phase transition occurring at lower temperatures. Table 2.1 summarizes all the ADLF spectroscopic results.

\textbf{2.4.2 Napthalene.} ADLF spectroscopy has been done for a napthalene sample which has been deuteriated at the alpha position. The sample was prepared via a Grignard reaction with 1-bromonapthalene and deuterium oxide. Mass spectrometry determined the extent of deuteriation to be 79 \%, (C-$^2\text{H}$; 100\%) and solution-state deuterium NMR spectroscopy indicated that the ratio of the alpha C-$^2\text{H}$ sites versus the beta C-$^2\text{H}$ sites was 24/1. Figure 2.30 shows the ADLF spectrum of the sample; the experimental parameters used to obtain the spectrum are given below the figure. Two sets of $v_-$ and $v_+$ transitions are evident for the sample, due to the presence of two sets of crystallographically inequivalent alpha sites in the unit cell for napthalene.\textsuperscript{19} This is consistent with the lack of double transitions observed for this sample, indicating that the
Figure 2.28. ADLF spectrum of (4-fluorophenyl)[2,2-\text{H}]acetic acid (~95%; C\text{H}_2 = 100\%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 270 and 3 seconds, respectively. The zero-field irradiation amplitude was 21 milligauss.
Figure 2.29. Double transition 77 K ADLF spectrum of (4-fluorophenyl)[2,2-\textsuperscript{2}H]acetic acid. The high- and zero-field delays were 270 and 4 seconds, respectively. The zero-field irradiation amplitude was 3 gauss.
Table 2.1 Summary of ADLF Spectroscopic Results for Studied Compounds

<table>
<thead>
<tr>
<th>Compound</th>
<th>Site 1</th>
<th>Site 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phenyl[2,2-\textsubscript{2}H\textsubscript{2}]acetic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\nu_+)</td>
<td>130.5(8)</td>
<td>126.5(5)</td>
</tr>
<tr>
<td>(\nu_-)</td>
<td>128.0(8)</td>
<td>123.5(5)</td>
</tr>
<tr>
<td>(\epsilon^2\eta Q/h), kHz</td>
<td>172.3(7)</td>
<td>166.7(7)</td>
</tr>
<tr>
<td>(\eta)</td>
<td>0.036(12)</td>
<td>0.029(11)</td>
</tr>
<tr>
<td>(4-Bromophenyl)[2,2-\textsubscript{2}H\textsubscript{2}]acetic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\nu_+)</td>
<td>132.0(7)</td>
<td>127.5(7)</td>
</tr>
<tr>
<td>(\nu_-)</td>
<td>129.0(7)</td>
<td>125.5(7)</td>
</tr>
<tr>
<td>(\epsilon^2\eta Q/h), kHz</td>
<td>174.0(9)</td>
<td>168.7(9)</td>
</tr>
<tr>
<td>(\eta)</td>
<td>0.034(17)</td>
<td>0.024(17)</td>
</tr>
<tr>
<td>(4-Chlorophenyl)[2,2-\textsubscript{2}H\textsubscript{2}]acetic acid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\nu_+)</td>
<td>132.0(7)</td>
<td>127.0(7)</td>
</tr>
<tr>
<td>(\nu_-)</td>
<td>129.0(7)</td>
<td>125.0(7)</td>
</tr>
<tr>
<td>(\epsilon^2\eta Q/h), kHz</td>
<td>174.0(9)</td>
<td>168.2(9)</td>
</tr>
<tr>
<td>(\eta)</td>
<td>0.034(16)</td>
<td>0.024(17)</td>
</tr>
<tr>
<td>(4-Fluorophenyl)[2,2-\textsubscript{2}H\textsubscript{2}]acetic acid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiplet transitions observed from 123.0 to 130.5
(4-Nitrophenyl)[2,2-\textsuperscript{2}H\textsubscript{2}]acetic acid

<table>
<thead>
<tr>
<th></th>
<th>\textit{v}_+</th>
<th>\textit{v}_-</th>
<th>\textit{e}^2\textit{q}_{zz}\textit{Q}/h, kHz</th>
<th>\eta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>130.5(5)</td>
<td>126.5(5)</td>
<td>171.5(6)</td>
<td>0.047(12)</td>
</tr>
</tbody>
</table>

Naphthalene

<table>
<thead>
<tr>
<th></th>
<th>\textit{v}_+(1)</th>
<th>\textit{v}_+(2)</th>
<th>\textit{v}_-(1)</th>
<th>\textit{v}_-(2)</th>
<th>\textit{e}^2\textit{q}_{zz}\textit{Q}/h, kHz (averaged)</th>
<th>\eta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>137.5(6)</td>
<td>140.5(4)</td>
<td>132.0(6)</td>
<td>135.0(10)</td>
<td>181.5(17)</td>
<td>0.058(16)</td>
</tr>
</tbody>
</table>

Barium chlorate monohydrate (25% deuteriated)

<table>
<thead>
<tr>
<th></th>
<th>\textit{v}_+</th>
<th>\textit{v}_-</th>
<th>\textit{e}^2\textit{q}_{zz}\textit{Q}/h, kHz</th>
<th>\eta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>192.0(10)</td>
<td>179.0(7)</td>
<td>247.3(11)</td>
<td>0.105(2)</td>
</tr>
</tbody>
</table>

Triphenylphosphonium-(methyl-d\textsubscript{3}) bromide

<table>
<thead>
<tr>
<th></th>
<th>\textit{v}_+</th>
<th>\textit{v}_-</th>
<th>\textit{e}^2\textit{q}_{zz}\textit{Q}/h, kHz</th>
<th>\eta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>132.0(9)</td>
<td>130.5(5)</td>
<td>175.0(9)</td>
<td>0.017(5)</td>
</tr>
</tbody>
</table>
Figure 2.30. ADLF spectrum of \(d_1\)-napthalene which has been deuteriated at one of the alpha positions (79%; \(C^-H = 100\%)). The spectrum was acquired at 77 K and at an applied field of 0 gauss. The high- and zero-field delays were 300 and 3 seconds, respectively. The zero-field irradiation amplitude was 21 milligauss.
two deuterium sites are not in close proximity of each other. The quadrupolar coupling constant and asymmetry parameter, determined by using the averaged values of the \( v_- \) and \( v_+ \) transitions, are 181.3 kHz and 0.060, respectively. These values are typical for an aromatic \( C^2H \) bond.\(^20\) However, they do differ from the reported values of 176 kHz and 0.053 determined previously from a high magnetic field study of a sample of \( d_{10} \)-naphthalene.\(^20b\)

2.4.3 Barium Chlorate Monohydrate and Methyltriphenylphosphonium Bromide.

Barium chlorate monohydrate and methyltriphenylphosphonium bromide have been studied in order to determine if ADLF spectroscopy at 77 K could yield the static quadrupolar coupling constants and asymmetry parameters for these compounds. The water molecule in solid barium chlorate monohydrate, \( \text{Ba(ClO}_3\text{)}\text{2} \cdot \text{H}_2\text{O} \), is known to undergo rapid 180° flips about its \( C_2 \) axis.\(^21\) Single-crystal deuterium NMR spectroscopy work shows that the quadrupolar coupling constant increases from 121.5 kHz to 243.5 kHz as the temperature decreases from 25° C to -130° C; the asymmetry parameter decreases over the same temperature span from 0.976 to 0.074.\(^22\) Rapid rotation about the \( C_3 \) axis of the methyl group of \( (\text{C}_2\text{H}_3)(\text{C}_6\text{H}_5)_3\text{PBr} \) reduces the quadrupolar coupling constant from a value of 170 kHz expected for an aliphatic \( C^2H \) bond to 47 kHz, as determined by room temperature high-field solid-state deuterium NMR spectroscopy.\(^23\) Thus one would expect the ADLF spectra at 77 K to yield larger values of the quadrupolar coupling constants for these two systems.

A sample of \( \text{Ba(ClO}_3\text{)}\text{2} \cdot \text{H}_2\text{O} \) was prepared by refluxing approximately one gram of the protonated form in a 75/25 mixture of \( \text{H}_2\text{O}/\text{H}_2\text{O} \) overnight and crystallizing the material from the solution. \( (\text{C}_2\text{H}_3)(\text{C}_6\text{H}_5)_3\text{PBr} \) was obtained from Aldrich; before use, the sample was dried in an oven overnight and then quickly loaded into a sample holder and into liquid nitrogen in order to minimize absorption of water from the atmosphere. The proton spin-lattice relaxation times for both of these compounds are \( \geq 750 \) seconds and thus only few spectral acquisitions were attempted.
Figure 2.31. ADLF spectrum of Ba(ClO$_3$)$_2$ • $^{2}$H$^1$HO (~25%; $^{2}$H$_2$O = 100%) acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 750 and 2 seconds, respectively. The zero-field irradiation amplitude was 44 milligauss.
Figure 2.32. ADLF spectrum of triphenylphosphonium-(methyl-$d_3$) bromide acquired at 77 K and an applied field of 0 gauss. The high- and zero-field delays were 1020 and 2 seconds, respectively. The zero-field irradiation amplitude was 44 milligauss.
The best spectrum for Ba(ClO₃)₂ • ¹H²HO is shown in Figure 2.31. The quadrupolar coupling constant and asymmetry parameter are 247 kHz and 0.105, respectively. The values of the quadrupolar coupling constant and asymmetry parameter are similar to values predicted for the static hydrate (~270 kHz, ~0.1).²² Figure 2.32 gives the spectrum for (C²H₃)(C₆H₅)₃PBr at 77 K. There is a great deal of fine structure in the spectrum, probably due to ²H - ²H coupling.⁹ If the two most intense peaks are assigned as the v₋ and v₊ transitions, one obtains a quadrupolar coupling constant of 175 kHz and an asymmetry parameter of 0.017, which are reasonable values for a static aliphatic C-²H bond. In order to obtain structural information from solid-state deuterium NMR spectroscopy, the static or non-motionally averaged value of the deuterium quadrupolar coupling constant is required. From the studies of Ba(ClO₃)₂ • ¹H²HO and (C²H₃)(C₆H₅)₃PBr, it appears that low-temperature ADLF spectroscopy may be useful in obtaining such values.

2.5 References


3. High-Field Solid-State Deuterium NMR Spectroscopy

3.1 Introduction

High-field solid-state deuterium NMR spectroscopy has been used to study the
dynamics of small molecules and polymers for a number of years. However, the
technique has only recently been used to study organometallic hydride systems. Herein are reported the results of using high-field solid-state deuterium NMR
spectroscopy to obtain dynamic and structural information on three organometallic hydride systems.

3.2 Basic Theory of High-Field Solid-State Deuterium NMR Spectroscopy

The high-field deuterium NMR spectrum for a solid is composed of signals whose positions are dependent upon the orientation of the applied magnetic field, $H_0$, with respect to the principal axis of the electric field gradient. This relationship is described by equations 3.1 to 3.4:

\[
v = v_o \pm v_Q f(\theta, \phi) \quad \text{eq. (3.1)}
\]

\[
2\pi v_o = \gamma_2 \frac{H_o}{2} \quad \text{eq. (3.2)}
\]

\[
v_Q = \frac{3}{4} \frac{e^2 q_{zz} Q}{\hbar} \quad \text{eq. (3.3)}
\]

\[
f(\theta, \phi) = (3\cos^2 \theta - 1 + \eta \sin^2 \theta \cos 2\phi)/2 \quad \text{eq. (3.4)}
\]

where $\theta$ and $\phi$ are the polar angles describing the orientation of $H_0$ with respect to the principal axes system of the electric field gradient. For a polycrystalline sample, the spectrum is composed of doublets corresponding to all possible orientations. For the case where the asymmetry parameter is zero, which is true for most terminally-bonded $X-^2H$ systems, the spectrum will resemble a Pake doublet for two coupled $I = 1/2$ spin systems. As the asymmetry parameter increases, however, the lineshape of the spectrum will alter considerably. Figure 3.1 illustrates the effect of a nonzero asymmetry.
Figure 3.1. The effect of a non-zero asymmetry parameter on the deuterium lineshape.
parameter on the spectrum.

Solid-state deuterium NMR spectroscopy has proven to be a useful probe in studying motional dynamics in solids. The deuterium lineshape is sensitive to the rate and type of motion occurring at the deuterium site. Three different motional regions can be defined: (1) The slow exchange region, where the correlation time of the motion, $\tau_c$, is much greater than the inverse of the quadrupolar coupling constant; (2) The intermediate exchange region, where $\tau_c$ is approximately equal to the inverse of the quadrupolar coupling constant; (3) The fast exchange region, where $\tau_c$ is much less than the inverse of the quadrupolar coupling constant. In the slow region, the spectrum is composed of overlapping spectra corresponding to the two or more sites which the deuteron may occupy. These overlapping spectra will coalesce in the intermediate exchange region, and in the fast exchange region the spectrum will have a lineshape with a quadrupolar coupling constant and asymmetry parameter which is averaged over the sites involved in the motion. Figure 3.2 shows how the deuterium lineshape for a rotating methyl group changes as a function of the rate of motion. Computer simulation of the lineshapes in the slow and fast exchange regions is straightforward. However, simulations for deuterons undergoing an intermediate rate of exchange are complicated by anisotropy in the spin-spin relaxation time, $T_2$. Certain orientations will have a short spin-spin relaxation time, which results in a loss of signal intensity during the quadrupolar echo pulse sequence and a corresponding distortion of the deuterium lineshape. Therefore, in order to properly simulate lineshapes which are in the intermediate exchange region, a computer program that can calculate the lineshape for each orientation and thus compensate for spin-spin relaxation anisotropy is required. In addition, the computer program must be able to take into account the effect of finite pulsewidths on the deuterium spectrum. Simulations reported in this work were done using the computer program SPOWDER, which was kindly provided for us by Dr. R. J. Wittebort. Appendix 2 gives a brief description of SPOWDER along with an example
Figure 3.2. Effect of the rate on the deuterium lineshape for a methyl group. The top trace refers to the slow exchange region, the middle trace refers to the intermediate exchange region, and the bottom trace refers to the fast exchange region.
of program input.

The deuterium spin-lattice relaxation time is also a function of the rate and type of motional process the deuteron is undergoing.\textsuperscript{4,10} The behavior of deuterium spin-lattice relaxation times is anisotropic with respect to the orientation of the principle axis of the electric field gradient tensor. The magnitude of the anisotropy is dependent upon $\tau_c$.

Figure 3.3 gives the plots of $\ln(T_1)$ versus $-\log(\tau_c)$ for the $\theta = 0^\circ$ and $\theta = 90^\circ$ peaks of the three site jump and free diffusion models of the rotation of a C-$^2$H bond in a methyl group. Note in Figure 3.3 how the curves differ for the two different modes of motion. By obtaining spin-lattice relaxation times for different $\tau_c$ values (via variable temperature experiments), it is sometimes possible to distinguish between different motional models.\textsuperscript{10} In addition, the activation energy of the motional process can sometimes be determined from Arrhenius plots of $\ln(k)$, where $k$ is the rate of motion, versus inverse temperature.\textsuperscript{6b,6d}

3.3 Modification of the Bruker AM-400 Spectrometer

In order to obtain high-field solid-state deuterium NMR spectra, it was necessary to modify the Bruker AM-400 solution NMR spectrometer. Initially, spectra were acquired using a locally constructed probe in place of the normal solution probes. More recently, a scheme has been devised which allows the Bruker instrument to use a commerical high-power solids probe and high-power rf amplifier to obtain solid-state spectra.

3.3.1 Locally Constructed Probe. A solids probe has been constructed using the body of an old solution probe. The coil of the probe is composed of 13 turns of 18 gauge magnet wire. The coil has a diameter of 5 mm and a length of 13 mm, giving it a calculated inductance of one microHenry. The coil is tuned by two Johanson variable capacitors (0 - 20 picofarads) using a tapped-parallel circuit.\textsuperscript{11} The quality factor of the probe, after being reduced by soldering a 69 k$\Omega$ resistor in parallel with the coil, is estimated to be approximately 100. The 90$^\circ$ pulse length is 5.3 $\mu$s, as determined from a
Figure 3.3. Plots of $\ln(T_1)$ versus $-\log(\tau_c)$ for the three-site jump and free diffusion models of motion for a methyl group. Symbols: solid line, free diffusion for the $\theta = 90^\circ$ peak; circles, free diffusion for the $\theta = 0^\circ$ peak; squares, three-site jump for the $\theta = 90^\circ$ peak; triangles, three-site jump for the $\theta = 0^\circ$ peak.
sample of $^2$H$_2$O, using an rf power of 240 watts. The pulse length could have been reduced if the full 380 watts of rf power the Bruker spectrometer produces at the resonant frequency for deuterium could have been used. However, at the higher rf power the dielectric of the capacitors begins to break down, thereby creating an arcing problem which causes the effective 90° pulse length to vary randomly. The probe is equipped with a glass dewar and a place to insert a thermocouple, thereby permitting variable temperature work.

3.3.2 Commercial Probe and Amplifier. The 5.3 μs 90° pulse lengths are too long to be useful for general solid-state deuterium work. This is because the rf excitation spectrum is a function of the 90° pulse length. At a 90° pulse length of 5.3 μs, the intensity of the rf excitation decays to 50% of its center frequency value at a spectral width of 115 kHz. Since solid-state deuterium lineshapes are typically 120 to 250 kHz in width, a long 90° pulse length will excite the deuterium spectrum unevenly and cause distortion in the lineshapes. In order to obtain shorter pulses, a scheme has been used which incorporates a Cryomagnetic Systems solids probe and a Henry 2006A high-power rf amplifier into the Bruker spectrometer's rf pulsing and receiving chain. Figure 3.4 illustrates how the system is set-up. The output of the Bruker transmitter is attenuated to a value of 35 watts and is used to drive the Henry amplifier. At an input drive of 35 watts, the Henry unit will produce a peak output of ~950 watts, which decays to a final value of 400 watts at a pulse length of 50 μs. However, for the time scale involved for typical values of 90° and 180° pulse lengths (2 to 4 μs), the output can be considered to be uniform. Seven pairs of SK3100 crossed diodes are placed between the output of the amplifier and the probe. In order to protect the Bruker receiver from the high rf power, a quarter wave cable and 5 pairs of SK3100 crossed diodes connected to electrical ground are placed between the probe and the Bruker preamp. The quarter wave cable has the property that input impedance at one end of the cable times the input impedance at the other end of the cable is equal to the square of the characteristic
Figure 3.4. Scheme used to implement the high-power Henry amplifier and Cryomagnetics probe. Seven pairs of crossed diodes are placed in series between the Henry amplifier and probe, while five pairs of crossed diodes to ground are placed directly in front of the Bruker preamp.
impedance of the cable; for RG-58 co-axial cable the characteristic impedance is 50 Ω. The crossed diodes act as an rf switch; they will pass high-power rf but not low-power rf. When the transmitter is on, the crossed diodes connected in front of the Bruker receiver act as a short to ground, thereby presenting a near zero impedance at one end of the quarter wave cable. The other end of the quarter wave cable then acts as a near infinity impedance and blocks the rf power of the Henry amplifier from the receiver. When the transmitter is off, the crossed diodes to ground no longer conduct and the quarter wave cable now detects only the input impedances of the Bruker receiver and the probe, both of which are 50 Ω. The crossed diodes in series between the transmitter and the probe prevent the signal from going back into the output of the transmitter.

The Cryomagnetics probe has a quality factor of approximately 153 and a ringdown time of 20 μs. The 90° pulse lengths using this arrangement are 1.7 to 1.8 μs, depending on how well the output of the Henry amplifier is matched to the impedance of the probe. Appendix 3 gives operational details on how to use the Henry amplifier.

3.4 Experimental Results

3.4.1 Bis(cyclopentadienyl)zirconium(IV) Dideuteride

3.4.1.1 Introduction. Solid-state deuterium NMR spectroscopy has been performed on [(η⁵-C₅H₅)₂Zr(²H)₂]ₓ and has identified the two different deuterons which are present.³ This study is a novel case in which solid-state deuterium NMR spectroscopy has been able to provide structural information concerning a bridging hydride.

Bis(cyclopentadienyl)zirconium(IV) dihydride is related to a family of bis(cyclopentadienyl) zirconium alkyl hydrides used for activating olefins in organic synthesis.¹⁴ (Cp)₂ML'L' complexes are generally found as monomeric, bent sandwich structures with a cis L-L' configuration.¹⁵ However, sixteen electron zirconium complexes tend to oligomerize. For example, [(η⁵-CH₃C₅H₄)₂Zr(H)(μ-H)]ₓ forms a dimer in the solid state.¹⁶ [(η⁵-C₅H₅)₂Zr(H)₂]ₓ forms either a dimer or a polymer in
the solution state, and its solid state structure is poorly defined. IR spectroscopy of the complex shows M-H stretches due to terminal and bridging hydride sites.

3.4.1.2 Experimental. The title complex studied was prepared using a literature method. Initially, the sample was contaminated with aluminum powder traced to the LiAlH₄ reagent used. This impurity was removed by recrystallization from THF prior to use. Preliminary spectra taken with a Bruker AM-400 spectrometer and a locally constructed probe indicate that the aluminum impurity (~ 2%) had little effect on the spectrum.

Solid-state deuterium NMR spectra were acquired on a Bruker CXP-300 at the Dupont Research Center in Wilmington, Delaware, at a resonant frequency for deuterium of 46.1 MHz. With a solenoid coil and a 90° pulse length of 3.6 μs, good results were obtained using a simple quadrupolar echo sequence. However, a significant increase of the intensity of the wings were obtained using a composite pulse sequence; the spectra reported here were obtained using the composite sequence. Figure 3.5 shows the experimental and simulated spectrum for [(C₅H₅)₂Zr(²H)₂]ₓ acquired at 300 K. Table 3.1 gives the parameters used in the simulation, which assumed two deuterium sites for the fit. The deuterium spin-lattice relaxation was approximately 10 seconds; presumably, the rapid rotation of the cyclopentadienyl rings contributes to a dipolar relaxation mechanism.

In a solid-state deuterium NMR experiment, the shapes of overlapping lineshapes are influenced by motional processes and differential spin-lattice relaxation times. Spectra acquired at 185 K were similar to those obtained at room temperature, indicating that there is little motional averaging occurring, and that the values reported in Table 3.1 are static values. Spectra obtained at different delays gave the same relative areas of the two sites, indicating that the difference in the site intensities are due to relative abundances and not due to differential spin-lattice relaxation times. However, the relative areas of the two sites did vary with sample preparation. This may be attributed to the variability of
Figure 3.5. Solid-state deuterium NMR spectrum of \([\eta^5-C_5H_5)_2ZrH_2]_x\) acquired at room temperature. The top trace is the experimental spectrum, while the bottom trace is the best fit of two deuterium powder patterns to the observed spectrum.
Table 3.1. \([(\eta^5 \cdot \text{C}_5\text{H}_5)\text{Zr}(^2\text{H})_2]_x\) Deuterium NMR Spectral Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Terminal Site</th>
<th>Bridging Site</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e^2q_{zz}Q/h$, kHz</td>
<td>46.7(5)</td>
<td>32.7(20)</td>
</tr>
<tr>
<td>$\eta$</td>
<td>0.0</td>
<td>&lt;0.1</td>
</tr>
<tr>
<td>Relative area, %</td>
<td>77</td>
<td>23</td>
</tr>
<tr>
<td>Gaussian broadening, kHz</td>
<td>1.5</td>
<td>5</td>
</tr>
<tr>
<td>chemical shift,(^a) ppm</td>
<td>0(10)</td>
<td>0(30)</td>
</tr>
</tbody>
</table>

\(^a\)Relative to MeSi\(_4\)
polymeric formation due to the weakly bridging hydrogen bond. Early transition metal complexes are known to insert into cyclopentadienyl rings and facilitate the exchange between C-H and M-H hydrogens. However, the quadrupolar coupling constant of a cyclopentadienyl C-^2H bond has a motionally averaged value of ~95 kHz$^2$, which is substantially larger than the observed values. In addition, no C-^2H bonds were observed in the deuterium solution NMR spectrum for one of the samples.

### 3.4.1.3 Results and Discussion

To begin the process of identifying the two sites, the deuterium quadrupolar coupling constant for a terminally bound Zr-^2H site must be estimated. This is done by using Salem's correlation between vibrational force constants and deuterium quadrupolar coupling constants for X-H systems, which is shown in Figure 3.6$^2$: the values used in Figure 3.6 are listed in Table 3.2. The vibrational force constant can be obtained from the IR stretching frequency. Using a frequency of 1520 cm$^{-1}$, the quadrupolar coupling constant for the terminal site is estimated to be ~ 40 kilohertz.

After determining an approximation of the deuterium quadrupolar coupling constant for the terminal site, the bridging site can be addressed. A deuteron bridging two metal centers will have a smaller quadrupolar coupling constant than a terminally-bound deuteron for the same metal center. The reduction in the value of the deuterium quadrupolar coupling constant is due to three factors: (1) The formation of a bridging metal hydride bond is accompanied by an increase in the metal - hydrogen bond distance. For example, $d$(Mn-H) = 1.601 Å for HMn(CO)$^5$ and $d$(Cr-H) = 1.675-1.750 Å for [HCr(CO))$_{10}$].$^2$ (2) The positive nuclear contribution to the electric field gradient is larger in magnitude than the negative electronic term (see equation 1.3, section 1.2). Li-^2H$^2$ and ^2H$_2$O$^2$ have positive quadrupolar coupling constants. Since there are no large deviations from the line in Figure 3.6, one can assume that all X-^2H systems have positive deuterium quadrupolar coupling constants. (3) The nuclear term of the electric field gradient is a stronger function of the internuclear distance than the electronic term.
Figure 3.6. Salem's correlation between vibrational force constants and deuterium quadrupolar coupling constants for terminally-bonded hydrides. The filled circles represent diatomic and small molecules, the squares represent metal hydrides, and the open circle represents the results for the zirconium complex. Table 3.2 tabulates the data used for this figure (except the Zr datum point). The slope of the line is $2.6 \times 10^3$ dynes/cm-kHz, and the y intercept is $3.4 \times 10^4$ dynes/cm.
Table 3.2. Deuterium quadrupolar coupling constants and vibrational force constants for X-H systems.

<table>
<thead>
<tr>
<th>X</th>
<th>$e^2q_{zz}Q/h$(kHz)</th>
<th>$k(10^5$ dynes/cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>225$^a$</td>
<td>5.75$^a$</td>
</tr>
<tr>
<td>Li</td>
<td>33$^a$</td>
<td>1.03$^a$</td>
</tr>
<tr>
<td>Be</td>
<td>69.5$^b$</td>
<td>2.26$^c$</td>
</tr>
<tr>
<td>B</td>
<td>105$^b$</td>
<td>3.04$^c$</td>
</tr>
<tr>
<td>C</td>
<td>170$^b$</td>
<td>4.48$^c$</td>
</tr>
<tr>
<td>N</td>
<td>230$^b$</td>
<td>6.03$^c$</td>
</tr>
<tr>
<td>O</td>
<td>314$^b$</td>
<td>7.79$^c$</td>
</tr>
<tr>
<td>F</td>
<td>340$^a$</td>
<td>9.65$^a$</td>
</tr>
<tr>
<td>Si</td>
<td>91$^d$</td>
<td>2.48$^c$</td>
</tr>
<tr>
<td>P</td>
<td>115$^d$</td>
<td>3.25$^c$</td>
</tr>
<tr>
<td>S</td>
<td>155$^d$</td>
<td>4.3$^d$</td>
</tr>
<tr>
<td>Cl</td>
<td>190$^b$</td>
<td>5.16$^a$</td>
</tr>
<tr>
<td>Mn</td>
<td>68.07$^e$</td>
<td>1.80$^f$</td>
</tr>
<tr>
<td>Mo</td>
<td>528</td>
<td>2.06$^g$</td>
</tr>
<tr>
<td>W</td>
<td>548</td>
<td>2.11$^g$</td>
</tr>
</tbody>
</table>

$^e$Ireland, P. S.; Olson, L. W.; Brown T. L J. Amer. Chem. Soc. 1975, 97, 3548.
Previous studies of $\text{O}^2\text{H}$ systems show that the deuterium quadrupolar coupling constant decreases upon formation of a hydrogen bond. Thus in going from a M-$^2\text{H}$ system to a M-$^2\text{H}$-M system, a similar decrease in the deuterium quadrupolar coupling constant should also occur.

$[(\text{C}_5\text{H}_5)_2\text{Zr}^2\text{H}_2]_x$ shows two different overlapping powder patterns. The one with the largest quadrupolar coupling constant, 46.7 (5) kHz, has been assigned to the terminal site, while the one with the smaller value, 32.7 (20), has been assigned to the bridging site. Moreover, the relative amount of terminal versus bridging deuterium sites have also been determined. This work represents the first report of solid-state deuterium NMR spectroscopy for a bridging metal hydride site.

3.4.2 Bis(triphenylphosphine)iminium Tetracarbonyl Iron Hydride

3.4.2.1 Introduction. The anionic tetracarbonyl metal hydrides $\text{HM(CO)}_4^-$ ($M = \text{Fe, Ru, Os}$) have been of interest as possible model compounds for the homogeneous catalytic properties of metal carbonyls. The solid-state and solution structure of the iron complex has been determined by single-crystal X-ray methods and by conductivity measurements and IR spectroscopy, respectively. In addition, the acid and base properties of the iron triad have been investigated. Herein, the results of a solid-state deuterium NMR spectroscopic study for the bis(triphenylphosphine)iminium ([PPN]$^+$) salt of the tetracarbonyl iron hydride are reported.

3.4.2.2 Experimental. The [PPN]$^+$ $[2\text{HFe(CO)}_4]$ complex was provided by Dr. Howard W. Walker from the Ethyl Technical Center in Baton Rouge and prepared according to a literature method. Samples were sealed under vacuum in 5 mm NMR tubes ~1.5 - 1.8 cm in length for use in the solid-state NMR probe. Solid-state deuterium NMR spectra were acquired at a resonant frequency for deuterium of 61.4 MHz using a Bruker AM-400 solution spectrometer which has been modified for solid-state work with a high-power rf amplifier and probe, as described in section 3.3.2. Variable-temperature
inversion-recovery spectra were obtained using a modified version of the quadrupolar echo pulse sequence in which a 180° pulse, along with a variable delay, is applied before the first 90° pulse. The number of relaxation delays varied from 5 to 8, and the number of scans varied from 2500 to 8600, depending on the spin-lattice relaxation time for the sample at the temperature at which the data was being acquired. The delay between 90° pulses was 35 μs. Acquisition of the echo is initiated 27 μs after the second 90° pulse, with the signal left-shifted 18 μs before application of the Fourier transformation. An exponential linebroadening factor of 750 Hz was applied to all the data. A simple two-step phase cycling routine, where the phase of the first 90° pulse is alternated between 0° and 180°, was used in order to cancel the effects of probe ringing and to eliminate quadrature phase errors. The 90° pulse length was 1.8 μs, insuring that there is less than a -0.15 db difference in rf excitation energy between the edges and the center frequency of the reported spectra. Spin-lattice relaxation times for the θ = 90° and θ = 0° peaks were determined from a three parameter fit of the equation

\[ I(t) = I_0 \{ 1 - 2A \exp[-t/T_1] \} \]

where I(t) are the peak intensities at various values of t. An overall recycle time of ~5 times the spin-lattice relaxation time was used for acquiring the data except at 180 K, where the recycle time was ~2 times the spin-lattice relaxation. Simulations were done using the program SPOWDER. In all the reported simulated spectra, the effects of a finite 90° pulse and the application of an exponential linebroadening factor have been included.

3.4.2.3 Procedure for Analyzing Solid-State Deuterium Lineshapes and T1 Data. Solid-state deuterium NMR lineshapes and spin-lattice relaxation times are sensitive to the rate and type of motion occurring at the deuterium site. The procedure used to analyze deuterium spectra is thus straightforward. Solid-state spectra and spin-lattice relaxation times are acquired at different temperatures in order to determine if any motional averaging of the lineshape is occurring. If the spectra do show signs of motional
averaging, a model of the motion is then hypothesized based on the known or proposed structure of the molecular system. From this model ln(T1) versus log(k) tables for the θ = 90° and θ = 0° peaks of the spectrum are generated using the computer program SPT1 (Appendix 2). The validity of the model can often be confirmed by observing whether the same relationship between the two θ peaks is observed for the calculated and experimental values. From the T1 data an initial rate of motion is estimated and used in the lineshape simulations. The rate is adjusted until the best match between the simulations and the experimental spectra is obtained. The rates determined from the lineshape analysis are then used in an Arrhenius plot of ln(k) versus 1/T, from which the activation energy (Ea) can be determined from the slope of the line.

3.4.2.4 Results and Discussion. The spin-lattice relaxation data for the θ = 0° and θ = 90° peaks for the [PPN]+[2HFe(CO)4]− complex are presented in Table 3.3. The fully-relaxed spectra for the iron complex obtained at temperatures of 300, 285, 278, 270, 240, 210, and 180 K are shown in Figures 3.7 thru 3.13, respectively; the number of scans, the recycle delay, and the temperature at which the spectrum was acquired are given below each figure. The spectra shown in Figures 3.7 thru 3.13 have the typical pattern seen for deuterium sites with axial symmetry (i.e. η = 0). The splitting between the θ = 90° peaks (designated as Δv) increases as the temperature decreases; the relationship between Δv and temperature is shown in Figure 3.14. The dependence of Δv with respect to temperature indicates that the deuterium site is undergoing some type of motional averaging. However, the deuterium lineshapes for each temperature did not show any distortion which occurs when the rate of motion is comparable to the value of the deuterium quadrupolar coupling constant. Thus the rate of motion for the deuterium site is in the fast-exchange limit.7

At 210 K Δv reaches a value of 57 kHz, which corresponds to a deuterium quadrupolar coupling constant of 76 kHz. This is believed to be the static or non-motionally averaged value for the deuterium quadrupolar coupling constant for the iron
Table 3.3. Experimental Spin-lattice Relaxation Times for $[\text{PPN}^+\, [\text{HFe(CO)}_4]^-$

<table>
<thead>
<tr>
<th>Temperature</th>
<th>$T_{1, 90^\circ}$ (seconds)</th>
<th>$\Delta^a$</th>
<th>$T_{1, 0^\circ}$ (seconds)</th>
<th>$\Delta^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>300 K</td>
<td>0.101(7)</td>
<td>0.84</td>
<td>0.178(23)</td>
<td>0.85</td>
</tr>
<tr>
<td>285 K</td>
<td>0.126(6)</td>
<td>0.85</td>
<td>0.216(24)</td>
<td>0.86</td>
</tr>
<tr>
<td>278 K</td>
<td>0.144(7)</td>
<td>0.88</td>
<td>0.207(77)</td>
<td>0.94</td>
</tr>
<tr>
<td>270 K</td>
<td>0.188(3)</td>
<td>0.88</td>
<td>0.232(8)</td>
<td>0.91</td>
</tr>
<tr>
<td>240 K</td>
<td>0.525(5)</td>
<td>0.81</td>
<td>0.618(32)</td>
<td>0.77</td>
</tr>
<tr>
<td>210 K</td>
<td>1.48(1)</td>
<td>0.80</td>
<td>1.71(2)</td>
<td>0.80</td>
</tr>
<tr>
<td>180 K</td>
<td>1.52(3)</td>
<td>0.76</td>
<td>1.52(4)</td>
<td>0.71</td>
</tr>
</tbody>
</table>

$\Delta^a$ refers to the $A$ parameter in the equation $I(t) = I_0 [1 - 2A \exp(-t/T_1)]$.
Figure 3.7. Fully-relaxed experimental spectrum for \([\text{PPN}]^+ [\text{HFe(CO)}_4]^\) obtained at 300 K. The number of scans is 5900, and the recycle delay is 1.5 seconds. The value of \(\Delta v\) is given above the spectrum.
Figure 3.8. Fully-relaxed experimental spectrum for \([\text{PPN}]^+ [^2\text{Fe(CO)}_4^-] \) obtained at 285 K. The number of scans is 8600, and the recycle delay is 1.2 seconds. The value of \(\Delta v\) is given above the spectrum.
Figure 3.9. Fully-relaxed experimental spectrum for [PPN]$^+\text{[}^{2}\text{HFe(CO)}_4\text{]}^{-}$ obtained at 278 K. The number of scans is 5000, and the recycle delay is 1.5 seconds. The value of $\Delta v$ is given above the spectrum.
Figure 3.10. Fully-relaxed experimental spectrum for \([\text{PPN}]^+ [^{2}\text{HFe(CO)}_4]^\text{-}\) obtained at 270 K. The number of scans is 5200, and the recycle delay is 2.5 seconds. The value of \(\Delta v\) is given above the spectrum.
Figure 3.11. Fully-relaxed experimental spectrum for [PPN]$^+\ [^{2}HFe(CO)_4]^-$ obtained at 240 K. The number of scans is 4000, and the recycle delay is 3 seconds. The value of $\Delta v$ is given above the spectrum.
Figure 3.12. Fully-relaxed experimental spectrum for [PPN]$^+$ [HFe(CO)$_4$]$^-$ obtained at 210 K. The number of scans is 2500, and the recycle delay is 5 seconds. The value of $\Delta v$ is given above the spectrum.
Figure 3.13. Fully-relaxed experimental spectrum for [PPN]$^+\ [^{2}HFe(CO)_{4}]^{-}$ obtained at 180 K. The number of scans is 3000, and the recycle delay is 3 seconds. The value of $\Delta v$ is given above the spectrum.
Figure 3.14. Plot of Δν versus temperature for [PPN]$^+$ [2HFe(CO)$_4$]$^-$. 
complex. This conclusion is supported by the predicted value of ~70 kHz using an estimated value of 1906 cm\(^{-1}\) for the Fe-H stretch and Salem's correlation between deuterium quadrupolar coupling constants and vibrational force constants for terminally-bonded hydride systems.\(^{23}\)

The crystal structure for [PPN]\(^+\)\([^2\text{HFe(CO)}_4]^-\) has been reported by Smith and Bau to be a distorted trigonal bipyramid, with the hydrogen ligand occupying an axial position.\(^{32}\) A reasonable model of motion for the deuterium site which is consistent with the crystal structure and the solid-state deuterium NMR lineshape is a librational motion, in which the deuteron "wobbles" in an area described by a cone about the molecular C\(_3\) axis of the anion with a cone angle \(\beta\). This type of motion can be simulated by using a simple 4-site jump model, illustrated in Figure 3.15; one site is set to be colinear with the molecular C\(_3\) axis, while the three other sites can be described by the polar coordinates \((\beta, 0^\circ)\), \((\beta, 120^\circ)\), and \((\beta, 240^\circ)\), in accord with the symmetry of the anion. The value of \(\beta\) can be determined from the solid-state deuterium NMR lineshape, since the value of \(\Delta v\) for an axially symmetric site is given by:\(^7\)

\[
\Delta v = \Delta v'(3 \cos^2 \beta - 1)/2
\]  eq. (3.5)

Here \(\Delta v'\) is related to the static or non-motionally averaged value of the deuterium quadrupolar coupling constant.

Figures 3.16 thru 3.20 show plots of ln\(T_1\) versus log(\(k\)) for the different values of \(\beta\) generated with the SPT1 program. These plots show that the \(T_1\) minimum shifts to higher values as \(\beta\) decreases, thereby demonstrating the sensitivity the spin-lattice relaxation time has with respect to the type of motion. The \(T_1\) curves for all the values of \(\beta\) show that at rates less than the rate where the spin-lattice relaxation time is a minimum, the value of \(T_1\) for the \(\theta = 90^\circ\) and \(\theta = 0^\circ\) peaks are equal. At rates greater than the rate where the spin-lattice relaxation time is a minimum the \(T_1\) for the \(\theta = 0^\circ\) peak is ~2 times greater than the \(T_1\) at the \(\theta = 90^\circ\) peak.

Figures 3.21 thru 3.25 show the simulations using the program SPOWDER for the
Figure 3.15. Proposed four-site jump model of motion for the deuteron site in [PPN]+[²HFe(CO)₄]⁻. The jump rate between all sites is assumed to be equal, and the equilibrium population of all the sites are assumed to be equal. The locations of the sites relative to the C₃ axis of the anion are given in polar coordinates.
Figure 3.16. Plot of $\ln(T_1)$ versus $\log(k)$ using the four-site jump model shown in Figure 3.15. The value of $\beta$ is 8.3°; the solid line refers to the $\theta = 90^\circ$ peak, and the circles refer to the $\theta = 0^\circ$ peak.
Figure 3.17. Plot of $\ln(T_1)$ versus $\log(k)$ using the four-site jump model shown in Figure 3.15. The value of $\beta$ is 13.3°; the solid line refers to the $\theta = 90^\circ$ peak, and the circles refer to the $\theta = 0^\circ$ peak.
Figure 3.18. Plot of $\ln(T_1)$ versus $\log(k)$ using the four-site jump model shown in Figure 3.15. The value of $\beta$ is $14.9^\circ$; the solid line refers to the $\theta = 90^\circ$ peak, and the circles refer to the $\theta = 0^\circ$ peak.
Figure 3.19. Plot of $\ln(T_1)$ versus $\log(k)$ using the four-site jump model shown in Figure 3.15. The value of $\beta$ is $15.5^\circ$; the solid line refers to the $\theta = 90^\circ$ peak, and the circles refer to the $\theta = 0^\circ$ peak.
Figure 3.20. Plot of $\ln(T_1)$ versus $\log(k)$ using the four-site jump model shown in Figure 3.15. The value of $\beta$ is 18°; the solid line refers to the $\theta = 90°$ peak, and the circles refer to the $\theta = 0°$ peak.
Figure 3.21. Simulated fully-relaxed spectrum of [PPN]$^+$ [HFe(CO)$_4$]$^-$ acquired at 300 K. The value of $\beta$ is 18°, and the rate used is $3.2 \times 10^8$ rads/s.
Figure 3.22. Simulated fully-relaxed spectrum of [PPN]⁺ [²HFe(CO)₄]⁻ acquired at 285 K. The value of β is 15.5°; and the rate used is 2.5 X 10⁸ rads/s.
Figure 3.23. Simulated fully-relaxed spectrum of $[\text{PPN}]^+ [^2\text{HFe(CO)}_4]^- \text{acquired at 278 K. The value of } \beta \text{ is } 14.9^\circ, \text{ and the rate used is } 2.2 \times 10^8 \text{ rads/s.}$
Figure 3.24. Simulated fully-relaxed spectrum of [PPN]+ [2HFe(CO)₄]⁻ acquired at 270 K. The value of $\beta$ is 13.3°, and the rate used is $2.0 \times 10^8$ rads/s.
Figure 3.25. Simulated fully-relaxed spectrum of [PPN]$^+[\text{HFe(CO)}_4]^-$ acquired at 240 K. The value of $\beta$ is 8.3°, and the rate used is $5.4 \times 10^7$ rads/s.
fully-relaxed spectra acquired at temperatures of 300, 285, 278, 270, and 240 K. The value of $\beta$ and the rate used for the simulation is given below each figure; the static value of the deuterium quadrupolar coupling constant was assumed to be 76 kHz for all the simulations. The match in $\Delta v$ between experimental and simulated spectra were within 1 kHz agreement. Initially, an attempt was made to fit the experimental data to a model with a single value of $\beta$; however, simulations using such a model could not vary $\Delta v$ smoothly without introducing a great deal of lineshape distortion which was not evident in the experimental spectra. In addition, spin-lattice relaxation times calculated using a single $\beta$ model were in disagreement with experimental values by at least a factor of ten for some of the temperatures at which spectra were acquired.

Figures 3.26, 3.28, 3.30, 3.32, and 2.34 show the experimental inversion-recovery spectra for $[\text{PPN}]^+ [^2\text{HFe(CO)}_4]^-$ acquired at 300, 285, 278, 270, and 240 K, respectively; the number of scans, the recycle delay, and the temperature at which the spectra were taken are given below each each figure. Figures 3.27, 3.29, 3.31, 3.33, and 3.35 are the simulations for the inversion-recovery spectra, with the value of $\beta$ and the rate used given below each figure; the values of $k$ and $\beta$ are summarized in Table 3.4. Figure 3.36 shows the Arrhenius plot of $\ln(k)$ versus $1/T$ using the rates determined by lineshape analysis. The slope of the line yields an activation energy of 18(2) kJoules/mole and a preexponential factor of $5 \times 10^{11}$ rads/s. Similar values for the activation energy have been found for the rotation of tightly packed methyl groups$^{6a,6b}$ and for the 180° flips of $^2\text{H}_2\text{O}$ in hydrated salts.$^{35}$

For the spectra acquired at 210 and 180 K, $\beta$ could not be determined from the deuterium lineshape, due to the fact that for $\beta$ values less than $\sim 7^\circ$ the change in $\Delta v$ is less than the resolution of the experiment ($\pm 2$ kHz). Values of $\beta$ predicted using spin-lattice relaxation data gave rates which were much higher than what would be predicted using the Arrhenius plot given in Figure 3.36. A possible explanation for this deviation is that at lower temperatures the mechanism for spin-lattice relaxation is no longer primarily due
Figure 3.26. Experimental inversion-recovery spectra for \([\text{PPN}]^+ [^2\text{HFe(CO)}_4]^−\) obtained at 300 K. The number scans acquired for each spectrum is 5900, and the recycle time is 1.5 seconds. The variable delay is shown beside each spectrum.
Figure 3.27. Simulated inversion-recovery spectra for [PPN]+ [$^2H_{Fe(CO)_4}$]− acquired at 300 K. The value of $\beta$ is 18°, and the rate used is 3.2 X 10^8 rads/s. The variable delays are given beside each spectrum.
Figure 3.28. Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)₄]⁻ obtained at 285 K. The number scans acquired for each spectrum is 8600, and the recycle time is 1.2 seconds. The variable delay is shown beside each spectrum.
Figure 3.29. Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)₄]⁻ acquired at 285 K. The value of β is 15.5°, and the rate used is 2.5 $\times$ 10⁸ rads/s. The variable delays are given beside each spectrum.
Figure 3.30. Experimental inversion-recovery spectra for [PPN]+ [2HFe(CO)₄]⁻ obtained at 278 K. The number scans acquired for each spectrum is 5000, and the recycle time is 1.5 seconds. The variable delay is shown beside each spectrum.
Figure 3.31. Simulated inversion-recovery spectra for $[PPN]^+\ [^{2}HFe(CO)_{4}]^{-}$ acquired at 278 K. The value of $\beta$ is 14.9°, and the rate used is $2.2 \times 10^8$ rads/s. The variable delays are given beside each spectrum.
Figure 3.32. Experimental inversion-recovery spectra for [PPN]+ [²HFe(CO)₄]⁻ obtained at 270 K. The number scans acquired for each spectrum is 5200, and the recycle time is 2.5 seconds. The variable delay is shown beside each spectrum.
Figure 3.33. Simulated inversion-recovery spectra for [PPN]+ [2HFe(CO)₄]⁺ acquired at 270 K. The value of β is 13.3°, and the rate used is 2.0 X 10⁸ rads/s. The variable delays are given beside each spectrum.
Figure 3.34. Experimental inversion-recovery spectra for [PPN][^2HFe(CO)4] obtained at 240 K. The number scans acquired for each spectrum is 4000, and the recycle time is 3 seconds. The variable delay is shown beside each spectrum.
Figure 3.35. Simulated inversion-recovery spectra for [PPN]$^+$ [2HFe(CO)$_4$]$^-$ acquired at 240 K. The value of \( \beta \) is 8.3\( ^\circ \), and the rate used is \( 5.4 \times 10^7 \) rads/s. The variable delays are given beside each spectrum.
Figure 3.36. Plot of $\ln(k)$ versus $1/T$ for [PPN]$^+$ $[^{2}^{2}\text{HFe(CO)}_{4}]^{-}$. The slope of the line is $-2.18 \text{ K} \pm 0.26$, and the R factor is 0.98.
Table 3.4. Summary of Rate Data for [PPN]+ [2HFe(CO)₄]+.

<table>
<thead>
<tr>
<th>1/T X 1000 (K⁻¹)</th>
<th>k X 10⁷ (rads/s)</th>
<th>ln(k)</th>
<th>θ (degrees)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.333</td>
<td>32(1)</td>
<td>19.6(2)</td>
<td>18°</td>
</tr>
<tr>
<td>3.509</td>
<td>25(1)</td>
<td>19.3(2)</td>
<td>15.5°</td>
</tr>
<tr>
<td>3.597</td>
<td>22(1)</td>
<td>19.2(2)</td>
<td>14.9°</td>
</tr>
<tr>
<td>3.703</td>
<td>20(1)</td>
<td>19.1(2)</td>
<td>13.3°</td>
</tr>
<tr>
<td>4.167</td>
<td>5.4(10)</td>
<td>17.8(2)</td>
<td>8.3°</td>
</tr>
</tbody>
</table>
to the motion of the deuteron but to another pathway, such as cross-relaxation with the protons of the molecule.36

For the proposed model of motion for the deuterium site, the question arises whether the source of the motion is due to motion of the entire anion, or if the motion is limited to the Fe-2H bond. From the root-mean-square amplitude of thermal vibration for the carbonyl oxygen *trans* to the hydride ligand,30 the maximum value of $\beta$ due to vibration of the anion at room temperature is $\sim 9^\circ$. This is approximately one-half the the value of $\beta$ predicted from the solid-state deuterium NMR lineshape at 300 K. Thus, the motion of the deuterium site is primarily due to libration of the Fe-2H bond, with vibrational motion of the entire anion possibly becoming significant at low (< 240 K) temperatures. Another source of the deuteron motion could be the formation of an acyl intermediate of the iron complex. This is ruled out by the fact that the closest approach of the hydride ligand to a carbonyl carbon, using a value of $\beta$ of 18°, is $\sim 1.6$ Å, which is too long for C-H bond formation.

From solid-state deuterium NMR spectra and spin-lattice relaxation times, it is evident that the $^2$H site in [PPN]$^+$ [$^2$HFe(CO)$_4$]$^-$ is undergoing some type of motional averaging. A proposed model of motion which is consistent with the known solid-state structure is a libration of the Fe-H bond, with the rate and magnitude of the libration decreasing as the temperature decreases. This is supported by the fact that the experimental spectra can be successfully simulated using the proposed model. Although it is evident how deuterium quadrupolar coupling constants can be motionally averaged by organometallic systems such as iron complexes with deuteriated cyclopentadienyl rings or $\eta^2$-molecular hydrogen complexes, it is not common to think of a terminally-bonded metal hydride site to be capable of such motional averaging in the solid-state. This study shows how solid-state deuterium NMR spectroscopy has the potential to investigate motions in organometallic hydrides.
3.4.3 μ-Hydrido-bis(pentacarbonyl)chromium(0) and Pentacarbonylchromium(0) hydride

3.4.3.1 Introduction. The hydridometal carbonyl anions [HM\(_2\)(CO)\(_{10}\)]\(^-\) (M = Cr, Mo, W) have attracted a great deal of interest because they contain a single unsupported three center/two electron bond.\(^{25}\) X-ray and neutron analysis of these systems show that the solid-state molecular configuration is highly dependent upon lattice packing forces, due to the highly flexible nature of the M-H-M linkage.\(^{36}\) The M-H-M bond angle varies from 123° to 160°, and the bridging hydrogen atom can be displaced from the M-M bond axis up to a distance of 0.9 Å.\(^{25}\)

In this section, preliminary results of solid-state deuterium NMR spectroscopy for two bridging metal hydrides, [K(C\(_{18}\)H\(_{36}\)N\(_2\)O\(_6\))]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(μ-\(^2\)H)]\(^-\) and [N(C\(_2\)H\(_5\))\(_4\)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(μ-\(^2\)H)]\(^-\), and the analogous terminal chromium hydride, [PPN]\(^+\) [\(^2\)HCr(CO)\(_5\)]\(^-\), are reported. Both of the bridging hydride complexes have been characterized by neutron diffraction. For [K(C\(_{18}\)H\(_{36}\)N\(_2\)O\(_6\))]\(^+\) salt (designated as [K-(crypt-222)]\(^+\)) the M-H-M bond angle is 145.1° and the Cr-Cr bond distance is 3.300 Å.\(^{36}\) The average Cr-H bond distance for the complex is 1.729 Å. The tetraethylammonium salt (designated as [N(Et)\(_4\)]\(^+\)) has a similar Cr-Cr bond distance, 3.386 Å, and average Cr-H bond distance, 1.722 Å, but a larger M-H-M bond angle, 158.9°.\(^{37}\) The Cr-H bond distances for both of these complexes is longer than previously reported values for terminal Cr-H bonds (1.53 - 1.60 Å).\(^{38}\) Thus according to the trend predicted in section 1.2 from bridging metal hydride systems, the solid-state deuterium spectra for the two bridging chromium carbonyl hydrides should have a smaller deuterium quadrupolar coupling constant than the terminally-bonded [PPN]\(^+\) [\(^2\)HCr(CO)\(_5\)]\(^-\) species.

3.4.3.2 Experimental. The sample of [K(crypt-222)]\(^+\) [Cr\(_2\)(CO)\(_{10}\)(μ-\(^2\)H)]\(^-\) was kindly provided by Dr. Jeffery L. Petersen at the University of West Virginia. The [N(Et)\(_4\)]\(^+\) salt was prepared according to a literature procedure.\(^{39}\) The [PPN]\(^+\)
The [2HCr(CO)5]− complex was prepared by Michael J. Cronan Jr. of Dr. McLaughlin's research group. The samples were sealed under vacuum in 5 mm NMR tubes ~1.5-1.8 cm in length. Solid-state deuterium NMR spectra were collected on a Bruker AM-400 solution spectrometer which has been modified for solid-state work as outlined in section 3.3.2. The spectra for both of the bridging complexes were obtained at 270 K, while spectra for [PPN]+ [2HCr(CO)5]− was acquired at 300 and 200 K. A quadrupolar echo pulse sequence was used to acquire the data. The 90° pulse length was 1.8 μs, and the delay between pulses was 35 μs. Acquisition of the echo was initiated 27 μs after the second 90° pulse; the signal was left-shifted 18 μs before application of the Fourier transform. An exponential linebroadening factor was applied to all data in order to enhance the signal-to-noise. A two step phase-cycling procedure, where the phase of the first 90° pulse is alternated between 0° and 180°, was used in order to cancel the effects of probe ringing and to eliminate quadrature phase errors.12

3.4.3.3 Results and Discussion. The spectrum for [K(crypt-222)]+ [Cr2(CO)10(μ−2H)]− is shown in Figure 3.37, while the spectrum for [N(Et)4]+ [Cr2(CO)10(μ−2H)]− is shown in Figure 3.38; the recycle time, the amount of linebroadening, and the number of scans is given below each figure. Only a few scans were acquired for the samples due to their long spin-lattice relaxation times (~300 seconds for the [K(crypt-222)]+ salt and ~180 seconds for the [N(Et)4]+ salt). The signal-to-noise (S/N) for both spectra is not large enough for extensive lineshape evaluation. In order to increase S/N for the spectrum of the [K(crypt-222)]+ salt, the receiver phase of the NMR spectrometer was adjusted so that signal was acquired in only one of the two receiver channels. The data obtained in the other channel, which is only noise, was then zeroed using a Pascal program before the application of the Fourier transformation. This procedure has the effect of doubling the effective signal-to-noise, since the contribution of noise has been halved. However, one must take care in analyzing spectra which has been treated in such a manner, since the procedure forces the
Figure 3.37. The spectrum for [K(crypt-222)]^+ [Cr_2(CO)_10(\mu-^2H)]^- acquired at 270 K. The recycle time was 300 seconds, and the number of scans was 240. An exponential linebroadening factor of 1500 Hz had been applied to the spectrum.
Figure 3.38. The spectrum for $[\text{N(Et)}_4]^+ [\text{Cr}_2(\text{CO})_{10}(\mu-^2\text{H})]^- \text{ acquired at } 270 \text{ K.}$ The recycle time was 180 seconds, and the number of scans was 404. An exponential linebroadening factor of 3000 Hz had been applied to the spectrum.
spectrum to be symmetric about the center frequency. The spectrum of \([\text{K(crypt-222)})^+ [\text{Cr}_2(\text{CO})_{10}(\mu-^2\text{H})]^-\) in which the noise in one receiver channel has been zeroed is shown in Figure 3.39. The lineshape is characteristic of a deuterium site which is undergoing a slow (~10^4 s\(^{-1}\)) two-site jump whose polar coordinates are (0\(^\circ\), \(\beta\)) and (180\(^\circ\), \(\beta\)); here \(\beta\) is the angle the eqzz tensor (which lies along the Cr-2H bond axis) has with respect to the local molecular axis of symmetry at the chromium site.\(^7\)\(^8\) Preliminary lineshape simulations, using a value of \(\beta\) of 25\(^\circ\), indicate that the deuterium quadrupolar coupling constant is ~80 kHz (Figure 3.40).

Spectra for the \([\text{PPN}]^+ [^2\text{HCr(\text{CO})_5}]^-\) complex acquired at 300 and 200 K are shown in Figures 3.41 and 3.42, respectively. The spectrum obtained at 200 K was acquired with only a few scans due to the long (~90 seconds) deuterium spin-lattice relaxation. The 300 K spectrum shows signs of a two-site jump, a type of motion unexpected for a terminal Cr-2H bond. One possible explanation is that during the preparation of the material appreciable amounts of the bridging dimer species was also formed. At this writing the integrity of the sample is being checked by routine analysis. The 200 K spectrum has too low signal-to-noise for lineshape evaluation; however, after the spectral data was massaged in the same manner as the spectrum for \([\text{K(crypt-222)})^+ [\text{Cr}_2(\text{CO})_{10}(\mu-^2\text{H})]^-\), the resulting spectrum for the terminally-bonded species also showed signs of a two-site jump, indicating that the motional process has not been frozen out at 200 K. Preliminary lineshape simulations, using a value of \(\beta = 25^\circ\), indicate that the deuterium quadrupolar coupling constant is ~60 kHz (Figure 3.43).

The results of this study are intriguing in two ways. First, the question arises of what kind of solid-state structure can allow a terminally-bonded Cr-2H bond to have a two-site jumping motion. Solid-state deuterium NMR spectra can be influenced by the crystallographic form of the material.\(^6\)\(^a\) The answer may lie in determining the crystal structure of \([\text{PPN}]^+ [^2\text{HCr(\text{CO})_4}]^-\). Another question is why the deuterium quadrupolar coupling constant of the bridging species is larger than the deuterium quadrupolar...
Figure 3.39. The spectrum for \([K(\text{crypt-222})]^+ [\text{Cr}_2(\text{CO})_{10}(\mu-\text{H})]^-\) where the data acquired in one of the receiver channels has been zeroed.
Figure 3.40. Simulation for the spectrum of \([\text{K(crypt-222)}]^+ [\text{Cr}_2(\text{CO})_{10}(\mu-^2\text{H})]^-\) acquired at 270 K. A two-site jump model, with the sites located at polar coordinates \((0^\circ, 25^\circ)\) and \((180^\circ, 25^\circ)\), was used in the simulation. The value of the deuterium quadrupolar coupling constant was 80 kHz, and the rate was \(4.1 \times 10^5 \text{ rads/s}\).
Figure 3.41. The spectrum for \([\text{PPN}^+] [^{2}\text{HCr(CO)}_5]^-\) acquired at 300 K. The recycle time was 8 seconds, and the number of scans was 3206. An exponential linebroadening factor of 750 Hz had been applied to the spectrum.
Figure 3.42. The spectrum for $[\text{PPN}]^+ [^{2}\text{HCr(CO)}_5]^+$ acquired at 200 K. The recycle time was 90 seconds, and the number of scans was 552. An exponential linebroadening factor of 750 Hz had been applied to the spectrum.
Figure 3.43. Simulation for the spectrum of [PPN]$^+$ [H$_2$Cr(CO)$_5$]$^-$ acquired at 300 K. A two-site jump model, with the sites located at polar coordinates (0°, 25°) and (180°, 25°), was used in the simulation. The value of the deuterium quadrupolar coupling constant was 60 kHz, and the rate was 1.3 X 10$^5$ rads/s.
coupling constant of the terminally bonded species, in contrast to what was predicted in section 1.2 and the solid-state deuterium NMR spectral results for the 

$$[(\eta^5-C_5H_5)_2\text{Zr}^2\text{H}_2]_x$$

complex. This question may be answered by single crystal NMR spectroscopy on the bridging chromium hydride, since this technique yields both the deuterium quadrupolar coupling constant and the asymmetry parameter, and perhaps by further computational work. Nevertheless, this study is another example of how solid-state deuterium NMR spectroscopy can be used to study motional dynamics of organometallic hydrides.

3.5 References

   
   


   
   
   


11. Reference 10, Section 2.

12. Reference 13, Section 2.


22. Reference 10, Section 1.

23. Reference 9, Section 1.


references therein.


28. References 6-8, Section 1.


33. The Fe-H stretch is obscured by interference due to the PPN cation. M. Darensbourg, personal communication.

34. a) Reference 21, Section 2.


Appendix 1

A1.1 Introduction. All the programs listed in this appendix were run on an IBM 9000 computer using version 2.0 of the CSOS operating system. The main program for performing the ADLF experiment is ZERO.EXE. Most of the subroutines used in ZERO are defined in a series of other files, termed library files. Thus ZERO must be linked to these files before it can be used. The library files are ARRAYS, SAVEFID, CAMLIB, CAPLIB, DMMLIB, PULSE, GRLIB, and GRADRS. ARRAYS defines the arrays which are used by the other library files and the main program. SAVEFID is used to store high-field NMR data in a format which can be read by a commercial data processing software package provided by IBM. CAMLIB has the procedures used to issue instrument control and data acquisition commands to the CAMAC devices. CAPLIB has the procedures needed to tune the zero-field circuit and the zero-field irradiation amplitude. DMMLIB contains the routines needed to communicate with the Fluke multimeter, and PULSE has the procedures which concern the pulse programmer. GRADRS and GRLIB are part of a commercially obtained FORTRAN graphics package used to plot ADLF data on the IBM 9000 monitor and plotter.

A1.2 Compilation Step. Under the CSOS operating system, compilation is accomplished in two steps. First the source code of the program (designated with a PAS extension) is accepted by the front-end compiler (PASCAL.EXE) which checks the program's syntax and creates an intermediate file, which is designated with an "I" extension (i.e. ARRAYS.I). This intermediate file is then passed to the code generator (PASS2.EXE) which produces the object code from the intermediate code, writes it to a file with an "OBJ" extension, and deletes the intermediate code file. Compilation is done in one step by using the subroutine P1.SUB. Thus, for example, to compile ARRAYS.PAS, one types in the command "P1 1.ARRAYS", where the number one in front of ARRAYS refers to the disk drive where the program is located. P1.SUB has
A1.3 Linking Step. After ZERO and the library files have been compiled, they are linked together and an executable file (ZERO.EXE) is created using the ALINK command:

```
ALINK l:ZERO, 1:A, 1:S, 1:CMLB, 1:CPLB, 1:D, 1:P, 1:GRD, 1:GR2, CPOSIL, CFOSIL, CFTNLIB, +M=0
```

The object files have been renamed in order for the entire sequence to fit on one line of code. CPOSIL, CFOSIL, and CFTNLIB are system library files used by the Pascal and FORTRAN programs, and +M=0 is an ALINK option which allocates 50% of the largest available contiguous block of memory for the main program. The linking sequence is contained in the subroutine Z1.SUB, which allows one to link the main program to the library files by typing "Z1". After compilation and linking, the program is executed by typing "ZERO".
UNIT ARRAYS; {UPDATE 5/27/87} {DEFINES ARRAYS NEEDED FOR NMRLIB}

{**************************************************************************************************}

INTERFACE

USES {$U CPOSIL.LIB} system_library, system_calls, IEEE488;

type

  Y_ARRAY  = ARRAY [1..8192] OF REAL;
  RPOINTER = ^Y_ARRAY;
  INT_ARRAY = ARRAY [1..8192] OF INTEGER;
  IPOINTER = ^INT_ARRAY;
  LINT_ARRAY = ARRAY [1..8192] OF LONGINT;
  LPOINTER = ^LINT_ARRAY;
  DOUT_ARRAY = ARRAY [1..10] OF INTEGER;
  DOUT_POINTER = ^DOUT_ARRAY;
  STR   = packed array[1..40] of char;
  STRING60 = STRING[60]; (**Used for JOBRECORD**)
  STRING20 = STRING[20]; (*
  BLOCKFILE = FILE; (*
  DATARRAY = ARRAY [0..16384] OF LONGINT; (*
  SHIMARRAY = ARRAY [1..16] OF INTEGER; (*)
  DATA_PTR = ^DATARRAY; (*)

JOBRECORD = RECORD

  SOURCE   : LONGINT; {-9000}
  STATUS   : LONGINT; {1}
  TITLE    : STRING60; {62 BYTES}
  FILENAME : STRING20; {22 BYTES; VOLUME:FILENAME.EXT}
  ACCESS   : INTEGER; {0}
  OPERATOR : STRING20; {USER ID}
  SIZE     : INTEGER; {N: # OF LONGINT DATA}
  SWPWIDTH : REAL; {IN HERTZ}
  FLTRWDTH : REAL; {IN HERTZ}
  PULSWDTH : REAL; {PULSE WIDTH IN uS}
  RCVDELAY : REAL; {DELAY BTWN PULSE AND RCVR ON}
  ADCDELAY : REAL; {DELAY BTWN RCV ON AND ADC ON}
  RCVTIME  : REAL; {RECEIVER ON TIME: ACQUISITION TIME}
  RLXDELAY : REAL; {POST-SCAN RELAXATION DELAY}
  RCVGAIN  : LONGINT; {1}
  DIGRES   : INTEGER; {DIGITIZER RESOLUTION}
  QUAD     : BOOLEAN; {TRUE: QUAD ON/OFF}
  SCNSREQD : LONGINT; {SCANS REQUIRED}
SCNSDONE : LONGINT; {SCANS ACTUALLY DONE}
NORMCONST : INTEGER; {0 : FOR FFT}
LBTOTAL : REAL; {0}
SPECFREQ : REAL; {15351000 Hz FOR DEUTERIUM AT 100 MHz}
NUCLEUS : STRING8;
TEMPERATURE : REAL; {300}
LOCKGAIN : LONGINT; {0}
LOCKPWR : INTEGER; {0}
MAGSTAT : INTEGER; {0}
SHIMCOIL : SHIMARRAY; {0}
LEFT : INTEGER; {0}
RIGHT : INTEGER; {0}
TOP : INTEGER; {0}
BOTTOM : INTEGER; {0}
DATA : DATA_PTR; {ACCESS DATA AS DATA^[I]}
REFOFF : REAL; {REFERENCE FREQ OFFSET - FOR X AXIS}
SOLVENT : STRING20; {SOLID}
YEAR,MONTH : INTEGER; {FOR GET TIME}
DAY, HOUR : INTEGER; {FOR GET TIME}
MINUTE : INTEGER; {FOR GET TIME}
SECOND : INTEGER; {FOR GET TIME}
OBSOFF : REAL; {OBSERVE CHANNEL OFFSET}
DECOFF : REAL; {DECOUPLER CHANNEL OFFSET}
LIMIT : ARRAY [1 .. 2] OF INTEGER; {**LEFT AND RIGHT USER LIMITS**}
DUMMY : ARRAY [1 .. 1024] OF BYTE; {**PADDING FOR BLOCKREADS**}
end; {**OF JOBRECORD**}

{******************************************************************************}
IMPLEMENTATION

end.
UNIT CAMLIB;  {UPDATE 5/21/87}
{*******************************************************************************}

INTERFACE
USES {$U CPOSIL.LIB) system_library,system_calls,alpha,IEEE488,cserr,
    {$U 1:A.OBJ} ARRAYS;

procedure OPEN_SCRNO; { CAMAC logic unit number on system }
procedure KEY_PRESSED; { Get the CAMAC logic unit number on system }
procedure GET_LUN2(var LUN2:INTEGER); { CAMAC logic unit number on system }
procedure TIMEOUT_ENABLE; { Will enable lun2 timeout period for 5 seconds }
procedure PCK_DOUT_ARRAY(BUFSZ:INTEGER; PACK:DOUT_POINTER);
    {♦Will take DOUT data and pack it}
procedure UNPCK_DOUT_ARRAY(BUFSZ:INTEGER; UNPACK:DOUT_POINTER);
    {♦Will take DOUT data and unpack it}
procedure UNPCK_DMA_ARRAY(SA_ARRAY:LPOINTER; LENGTH:INTEGER);
    {♦Unpack DMA data from SA's}
procedure IFC_CAMAC; {♦Clear CAMAC interface*}
procedure OPEN_CAMAC; {♦Open and initialized CAMAC}
procedure CLS_LUN2; { Close the CAMAC logic unit number on system }
procedure ONE_BYTE_SETUP; {♦Sets up CAMAC crate controller for one byte}
procedure TWO_BYTE_SETUP; {♦" " " for two bytes}
procedure THREE_BYTE_SETUP; {♦" " " for three bytes}
procedure SEND_ARRAY(ARRAY_SIZE,N,A,F:INTEGER;
    ARRAY_POINTER : IPOINTER);
procedure SAMPLE_POSITION_RESET;
procedure
procedure TIMER_CH_SETUP(CH1,CH2,CH3,CH4,CH5,CH6,CH7,CH8:LONGINT);
    {♦Timer}
procedure
procedure TIMER_CS(RECYCLE,CHANNELS_ACTIVE,FREQ_POWER_OF_TEN:INTEGER);
procedure TIMER_EXE; {♦Starts up timing generator}
procedure TIMER_STOP; {♦Stops the timing generator}
procedure DAC_SETUP(CHANNEL:INTEGER;DAC_OUT:REAL); {♦Bipolar, 4-channel}
procedure CAPS_SETUP(SERIES_PARALLEL,CAP:LONGINT); {♦Series, parallel}
procedure SA_SETUP(SLOT,RECORD_LENGTH,PRETRIG,INTERVAL:INTEGER);
    {♦Signal Ave}
procedure SA_SET_LAM_WHEN_DONE(SLOT:INTEGER); {♦LAM set in crate
contr}  
procedure SA_SWEEP(SLOT:INTEGER;SCANS:LONGINT);  {Number of 
averages}  
procedure SA_RESET(SLOT:INTEGER);  {Reset}  
procedure SA_DONE(SLOT:INTEGER;var DONE:BOOLEAN);  {Check contr; Is SA 
done?}  
procedure SA_READ(SLOT,LENGTH:INTEGER);  {DMA read of signal averager}  
procedure SA_SREAD(SA_ARRAY:LPOINTER; SLOT,LENGTH:INTEGER);  
{Slow read of signal averagers}  
procedure NAF_GET_COMMAND;  {Will ask for the NAF Command Sequence}  
procedure NAF_DATA_GET;  {Will ask for 3_bytes data}  
procedure NAF_SINGLE;  {Will send one NAF command to the Crate}  
function NAF_KEYBOARD :BOOLEAN;  {Any keys pressed at keyboard?}  
procedure NAF_WRITE_DATA;  {Will send one NAF and 3_bytes data to the Crate}  
procedure NAF_3READ_DATA;  {Will send one NAF and read 3_bytes data}  
procedure NAF_MREAD_DATA;  {Will send one NAF and read 3_bytes data in 
loop}  
procedure FREQ_STROBE;  {Will stobe Frequency Data into Synthesizer}  
procedure FREQ_SET(FREQUENCY:REAL);  {Set frequency synthesizer (units 
kHz)}  
procedure FREQ_CLEAR;  {Clear frequency synthesizer for local operation}  
procedure TIMES_TO_S CREEN;  {Get time of day and print it on screen}  
procedure SATURATE(NUMBER_OF_REPEATS:INTEGER);  {Fire pulse 
programmer}  
procedure NAF_C_DISPLAY;  {Writes options for NAF_CAMAC control on 
screen}  
procedure NAF_CAMAC;  {Case statement of NAF options}  
procedure CAMAC_DISPLAY;  {Writes options for CAMAC modules on screen}  
procedure CAMAC_MENU;  {Options for direct control of CAMAC modules}  
{******************************************************************************}  

IMPLEMENTATION

const  
C = ':';

var  
DOUT : DOUT_ARRAY;  
DIN : DOUT_ARRAY;  
FPK : ARRAY [1..3] OF INTEGER;
DMA_ARRAY : ARRAY [1 .. 12288] OF INTEGER;
B_IOPTR  : LONGINT;
BUFSZ,BUFOF,BFFOF : LONGINT;
TRANMODE,LOTRM  : LONGINT;
HITRM,OPT,DIR : LONGINT;
RECNUM  : LONGINT;
ERRCODE : INTEGER;
NAF_N,NAF_A,NAF_F : INTEGER;
NAF_DH,NAF_DM,NAF_DL : INTEGER;
DEV2  : STRING8;
LUN2  : INTEGER;

procedure OPEN_SCRN0;  {THIS WILL GET A LUN for #SCRN0}
var
   name : STRING8;
begin
   RECNUM := -1;
   name := '#SCRN0';
   %sysopen(15,name,0,0,0,nil,RECNUM,ERRCODE);
   if ERRCODE <> 0 then begin
      %prterr(%cserr);
      KEY_PRESSED;
   end;
end;

procedure KEY_PRESSED;
var
   ASCII,SCANCODE,K1,K2JERRCODE,ERR : integer;
begin
   writelnO Press any key to continue.
   repeat
      %getkey(ASCII,SCANCODE,K1,K2,ERRCODE);
   until (ERRCODE = 0);
end;

procedure GET_LUN2(*var LUN2:INTEGER *);  {THIS WILL GET A LUN for CAMAC}
begin
   LUN2 := 122;
   writelnO The LUN for #BUSB? is ',LUN2);
end;
procedure TIMEOUT_ENABLE; {THIS WILL SEND A %sysfunc CALL TO} begin {3988 TO ENABLE TIMEOUT PERIOD OF 5 SECOND} FPK[1] := 30; FPK[2] := 100; FPK[3] := 0; %sysfunc(LUN2,@FPK,ERRCODE); if ERRCODE <> 0 then begin writeln(' Error in TIMEOUT_ENABLE on 3988, ERRCODE = ',ERRCODE); %prterr(%cserr); KEY_PRESSED; end; end;

procedure PCK_DOUT_ARRAY(*BUFSZ:INTEGER; PACK:DOUT_POINTER*); var {the data into an integer array for %swrite} I,J,K : INTEGER; begin I := 1; J := 1; K := BUFSZ+1; PACK[K] := 0; while ( J < K ) do begin PACK[I] := PACK[J]*256 + PACK[J+1]; I := I + 1; J := J + 2; end; end;

procedure UNPCK_DOUT_ARRAY(*BUFSZ:INTEGER; UNPACK:DOUT_POINTER*); var TEMP1,TEMP2 : INTEGER; I,J : INTEGER; DIN_TEMP : ARRAY [1 .. 10] OF INTEGER; begin I := 1; J := 1; while ( I <= BUFSZ ) do begin TEMP1 := UNPACK[J] and (256*(64+32+16+8+4+2+1));
TEMPI := TEMPI div 256;
IF ( UNPACKA[J] < 0 ) THEN TEMPI := TEMPI + 128;
TEMPI := UNPACKA[J] and (128+64+32+16+8+4+2+1);
IF ( I = BUFSZ ) THEN TEMPI := 0;
DIN_TEMP[I] := TEMPI; {ASSIGN HIGH BYTE TO DIN_TEMP[I]}
DIN_TEMP[I+1] := TEMPI; {ASSIGN LOW BYTE TO DIN_TEMP[I+1]}
I := I + 2;
J := J + 1;
end;
I := 1;
while ( I <= BUFSZ ) do begin
   UNPACKA[I] := DIN_TEMP[I];
   I := I + 1;
end;
end;

procedure UNPCK_DMA_ARRAY(* SA_ARRAY:LPTR; LENGTH:INTEGER*);
   {This procedure will take DMA data and unpack}
var
   {the data from %sread into an LONGINT array}
   I,J : INTEGER; {Note: Integers stored in 2's complement}
   C_256 : LONGINT;
   C_65536 : LONGINT;
   SIZE : INTEGER;
begin
   C_256 := 256;
   C_65536 := 65536;
case LENGTH of
      0: SIZE := 12288;
      1: SIZE := 6144;
      2: SIZE := 3072;
      3: SIZE := 1536;
      4: SIZE := 768;
      5: SIZE := 384;
      6: SIZE := 192;
      7: SIZE := 96;
end;
I := 1;
J := 1;
repeat
   IF ( DMA_ARRAY[I] >= 0 ) THEN
SA_ARRAY\[J\] := DMA_ARRAY[I] * C_256
ELSE
SA_ARRAY\[J\] := (DMA_ARRAY[I] AND 32767) * C_256 + 8388608;
I := I + 1;
IF (DMA_ARRAY[I] >= 0) THEN
SA_ARRAY\[J\] := SA_ARRAY\[J\] + (DMA_ARRAY[I] DIV 256)
ELSE
SA_ARRAY\[J\] := SA_ARRAY\[J\] + ((DMA_ARRAY[I] AND 32512) DIV 256) + 128;
J := J + 1;
SA_ARRAY\[J\] := (DMA_ARRAY[I] AND 255) * C_65536;
I := I + 1;
IF (DMA_ARRAY[I] >= 0) THEN
SA_ARRAY\[J\] := SA_ARRAY\[J\] + DMA_ARRAY[I]
ELSE
SA_ARRAY\[J\] := SA_ARRAY\[J\] + (DMA_ARRAY[I] AND 32767) + 32768;
I := I + 1;
J := J + 1;
until (I > SIZE);  \{12288 FOR FULL SIZED DATA ARRAY\}
end;

procedure IFC_CAMAC; \{*Will sent one IFC %sysfunc Sequence to 3988*\}
begin
FPK[1] := 10;
FPK[2] := 0;
FPK[3] := 0;
%sysfunc(LUN2,@FPK,ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in IFC command is ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure OPEN_CAMAC; \{INITIALIZES AND CLEARS GPIB\}
begin
GET_LUN2(LUN2);
DEV2 := '#BUSB?';
DIR := 2; OPT := 0;
TRANMODE := 0; B_IOPTR := 0;
%sysopen(LUN2,DEV2,DIR,OPT,TRANMODE,NIL,B_IOPTR,ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in sysopen for #BUSB? is ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
TIMEOUT_ENABLE;
IFC_CAMAC;
end;

procedure CLS_LUN2; {CLOSE THE SYSTEM I/O TO CAMAC CRATE}
begin
  %sysclose(LUN2,nil,ERRCODE);
  if ERRCODE <> 0 then begin
    writeln(' Error in CLS_LUN2 %sysclose, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
end;

procedure ONE_BYTE_SETUP; {SETS UP GPIB FOR AN 8 BIT TRANSFER}
begin
  DOUT[1] := 30;
  DOUT[2] := 0;
  DOUT[3] := 17;
  DOUT[4] := 0;
  DOUT[5] := 2;
  DOUT[6] := 0;
  BUFSZ := 6;
  PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

  %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
  if ERRCODE <> 0 then begin
    writeln('Error in ONE_BYTE_SETUP swrite, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
end;
procedure TWO_BYTE_SETUP; {THIS SETS UP A 16 BIT TRANSFER}
begin
  DOUT[1] := 30;
  DOUT[2] := 0;
  DOUT[3] := 17;
  DOUT[4] := 0;
  DOUT[5] := 1;
  DOUT[6] := 0;
  BUFSZ := 6;
  PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE) ;
if ERRCODE <> 0 then begin
  writeln(' Error in TWO_BYTE_SETUP swrite, ERRCODE = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure THREE_BYTE_SETUP; {THIS SETS UP A 24 BIT TRANSFER}
begin
  DOUT[1] := 30;
  DOUT[2] := 0;
  DOUT[3] := 17;
  DOUT[4] := 0;
  DOUT[5] := 0;
  DOUT[6] := 0;
  BUFSZ := 6;
  PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE) ;
if ERRCODE <> 0 then begin
  writeln(' Error in THREE_BYTE_SETUP swrite, ERRCODE = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
procedure SEND_ARRAY(*ARRAY_SIZE,N,A,F :INTEGER;
ARRAY_POINTER : IPOINTER*);
var
  I : INTEGER;
begin
  for I := 1 TO ARRAY_SIZE do begin
    TWO_BYTE_SETUP;
    DOUT[1] := N;
    DOUT[2] := A;
    DOUT[4] := trunc(ARRAY.Pointer[I] / 256);
    BUFSZ := 5;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
    if ERRCODE <> 0 then begin
      writeln(' Error in SEND_ARRAY swrite, ERRCODE = ',ERRCODE);
      %prterr(%cserr);
      KEY_PRESSED;
    end;
  end;
end;

procedure SAMPLE_POSITION_RESET;
const {THIS WILL RESET SAMPLE POSITION}
  SLOT = 6;
begin
  ONE_BYTE_SETUP;
  DOUT[1] := SLOT;
  DOUT[2] := 0;
  DOUT[3] := 8;
  DOUT[4] := 0;
  BUFSZ := 4;
  PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
```plaintext
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE)
;
if ERRCODE <> 0 then begin
    writeln(' Error in SAMPLE_POSITION_RESET swrite, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
end;
end;

procedure
TIMER_CH_SETUP(*CH1,CH2,CH3,CH4,CH5,CH6,CH7,CH8:LONGINT*);
const {THIS WILL SETUP TIMER CHANNELS}
    SLOT = 1;
begin
    TWO_BYTE_SETUP;
    DOUT[1] := SLOT;
    DOUT[2] := 0;
    DOUT[3] := 16;
    DOUT[4] := TRUNC (CH1 / 256);
    BUFSZ := 5;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
    %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE)
;
if ERRCODE <> 0 then begin
    writeln(' Error in TIMER_CH_SETUP swrite 1, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
end;
    TWO_BYTE_SETUP;
    DOUT[1] := SLOT;
    DOUT[2] := 1;
    DOUT[3] := 16;
    DOUT[4] := TRUNC ((CH1+CH2) / 256);
    DOUT[5] := (CH1+CH2) - (DOUT[4] * 256);
    BUFSZ := 5;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
```
141

\%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE)
;
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 2, ERRCODE = ',ERRCODE);
  \%prterr(%cserr);
  KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 2;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1+CH2+CH3) / 256);
DOUT[5] := (CH1+CH2+CH3) - (DOUT[4] * 256);
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

\%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE)
;
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 3, ERRCODE = ',ERRCODE);
  \%prterr(%cserr);
  KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 3;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1+CH2+CH3+CH4) / 256);
DOUT[5] := (CH1+CH2+CH3+CH4) - (DOUT[4]*256);
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

\%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE)
;
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 4, ERRCODE = ',ERRCODE);
  \%prterr(%cserr);
KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 4;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1+CH2+CH3+CH4+CH5) / 256);
DOUT[5] := (CH1+CH2+CH3+CH4+CH5) - (DOUT[4]*256);
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ, @DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;

%swrite(LUN2, @DOUT[1], BUFOF, BUFSZ, LOTRM, HITRM, RECNUM, ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 5, ERRCODE = ', ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 5;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1+CH2+CH3+CH4+CH5+CH6) / 256);
DOUT[5] := (CH1+CH2+CH3+CH4+CH5+CH6) - (DOUT[4]*256);
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ, @DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2, @DOUT[1], BUFOF, BUFSZ, LOTRM, HITRM, RECNUM, ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 6, ERRCODE = ', ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1+CH2+CH3+CH4+CH5+CH6+CH7) / 256);
DOUT[5] := (CH1+CH2+CH3+CH4+CH5+CH6+CH7) - (DOUT[4]*256);
procedure
TIMER_CS(*RECYCLE, CHANNELS_ACTIVE, FREQ_POWER_OF_TEN: INTEGER *);
const {THIS WILL SETUP CONTROL/STATUS REGISTER}
  SLOT = 1;
var
  CONTROL_CODE : INTEGER;
begin
  CONTROL_CODE := RECYCLE*64 + (CHANNELS_ACTIVE - 1)*8 +
  FREQ_POWER_OF_TEN;

BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ, @DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2, @DOUT[1], BUFOF, BUFSZ, LOTRM, HITRM, RECNUM,
   ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 7, ERRCODE = ', ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
TWO_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 7;
DOUT[3] := 16;
DOUT[4] := TRUNC ((CH1 + CH2 + CH3 + CH4 + CH5 + CH6 + CH7 + CH8) / 256);
DOUT[5] := (CH1 + CH2 + CH3 + CH4 + CH5 + CH6 + CH7 + CH8) - (DOUT[4]*256);
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ, @DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2, @DOUT[1], BUFOF, BUFSZ, LOTRM, HITRM, RECNUM,
   ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_CH_SETUP swrite 8, ERRCODE = ', ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;
procedure TIMER_EXE;  {THIS EXECUTES THE TIMING GENERATOR PULSE SEQUENCE}  
const
    SLOT = 1;  
begin
    DOUT[1] := SLOT;  
    DOUT[2] := 0;  
    BUFSZ := 3;  
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);  
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUOF := 0;  
    %swrite(LUN2,@DOUT[1],BUOF,BUFSZ,LOTRM,HITRM,RECNUM, 
            ERRCODE);  
    if ERRCODE <> 0 then begin  
        writeln(' Error in TIMER_EXE swrite, ERRCODE = ',ERRCODE);  
        %prterr(%cserr);  
        KEY_PRESSED;  
    end;  
end;

procedure TIMER_STOP;  {*This procedure will stop the timing generator*}  
const {*by writing zeros to cycle control register*}
SLOT = 1;
begin
THREE_BYTE_SETUP;
DOUT[1] := SLOT;
DOUT[2] := 0;
DOUT[3] := 17;
DOUT[4] := 0;
DOUT[5] := 0;
DOUT[6] := 0;
BUFSZ := 6;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in TIMER_STOP swrite, ERRCODE = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure DAC_SETUP(*CHANNEL:INTEGER,DAC_OUT:REAL*);
const {THIS WILL SETUP DIGITAL TO ANALOG CONVERTER}
  SLOT = 22;
var
  TEMP : LONGINT;
begin
  if (DAC_OUT < 0.0) then DAC_OUT := 20.0 + DAC_OUT;
  TEMP := trunc(DAC_OUT * 3276.7);
  TWO_BYTE_SETUP;
  DOUT[1] := SLOT;
  DOUT[2] := CHANNEL;
  DOUT[3] := 16;
  DOUT[4] := TRUNC(TEMP / 256);
  BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
  %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
      ERRCODE);
if ERRCODE <> 0 then begin
    writeln(' Error in DAC_SETUP swrite, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
end;
end;
end;

procedure CAPS_SETUP(*SERIES_PARALLEL,CAP:LONGINT*);
const {THIS WILL SETUP CAPACITOR ARRAY}
    SLOT = 16; {SERIES_PARALLEL = 0 FOR SERIES CAPACITANCE}
begin { " " = 1 FOR PARALLELED CAPACITANCE}
    TWO_BYTE_SETUP; {SHORT IS BIT 12 OF SERIES CAPACITANCE}
    DOUT[1] := SLOT;
    DOUT[3] := 16;
    DOUT[4] := TRUNC(CAP / 256);
    BUFSZ := 5;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
    %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
        ERRCODE);
    if ERRCODE <> 0 then begin
        writeln(' Error in CAPS_SETUP swrite, ERRCODE = ',ERRCODE);
        %prterr(%cserr);
        KEY_PRESSED;
    end;
end;

procedure SA_SETUP(*SLOT,RECORD_LENGTH,PRETRIG,INTERVAL:INTEGER*);
var
    CONTROL_CODE : LONGINT;
begin
    CONTROL_CODE := RECORD_LENGTH*64 + PRETRIG*8 + INTERVAL;
    TWO_BYTE_SETUP;
    DOUT[1] := SLOT;
    DOUT[2] := 0;
    DOUT[3] := 16;
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ,DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
    ERRCODE);
if ERRCODE <> 0 then begin
    writeln(' Error in SA.SETUP swrite, ERRCODE = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
end;
end;

procedure SA_SET_LAM_WHEN_DONE(*SLOT:INTEGER*);
begin
    ONE_BYTE_SETUP; {Enable LAM to be active}
    DOUT[1] := SLOT; {when SCAN COMPLETE}
    DOUT[2] := 0;
    DOUT[3] := 26;
    DOUT[4] := 0;
    BUFSZ := 4;
PCK_DOUT_ARRAY(BUFSZ,DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
    ERRCODE);
if ERRCODE <> 0 then begin
    writeln(' Error in SA_SET_LAM_WHEN_DONE[',SLOT,'] swrite = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
end;
end;

procedure SA_SWEEP(*SLOT:INTEGER, SCANS:LONGINT*);
var
    INVERT_SCANS : LONGINT;
begin
    INVERT_SCANS := 65536 - SCANS;
    TWO_BYTE_SETUP;
    DOUT[1] := SLOT;
    DOUT[2] := 0;
DOUT[3] := 17;
BUFSZ := 5;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_SWEEP[,SLOT,] swrite = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure SA_RESET(*SLOT:INTEGER*);
begin
  DOUT[1] := SLOT;
  DOUT[2] := 0;
  DOUT[3] := 9;
  BUFSZ := 3;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_RESET[,SLOT,] swrite = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure SA_DONE(*SLOT:INTEGER;var DONE:BOOLEAN*);
begin
  DOUT[1] :=30;
  DOUT[2] :=12;
  DOUT[3] := 1;
  BUFSZ := 3;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@DOUT,BUFOF,BUFSIZE,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_DONE[\',SLOT,'] swrite = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
BUFSIZE := 3;
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%sread(LUN2,@DIN,BUFOF,BUFSIZE,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_DONE[\',SLOT,'] sread = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
UNPCK_DOUT_ARRAY(BUFSIZE,@DIN);
DONE := FALSE;
if ( SLOT < 9 ) then begin
  if ( (DIN[3] and round(exp((SLOT-1)*ln(2)))) > 0 ) then DONE := TRUE;
end;
if ( SLOT < 17 ) and (SLOT > 8) then begin
  if ( (DIN[2] and round(exp((SLOT-9)*ln(2)))) > 0 ) then DONE := TRUE;
end;
if ( SLOT > 16 ) then begin
  if ( (DIN[1] and round(exp((SLOT-17)*ln(2)))) > 0) then DONE := TRUE;
end;
end;

procedure SA_SREAD(* SA_ARRAY:LPOINTER; SLOT,LENGTH:INTEGER*);
var
  SIZE : LONGINT;
  TEMP : LONGINT;
  IJ : INTEGER;
begin
  case LENGTH of
    0: SIZE := 8192;
    1: SIZE := 4096;
    2: SIZE := 2048;
    3: SIZE := 1024;
    4: SIZE := 512;
5: SIZE := 256;
6: SIZE := 128;
7: SIZE := 64;
end;
THREE_BYTE_SETUP;  {*Send this command twice*}
DOUT[1] := SLOT;
DOUT[2] := 1;
DOUT[3] := 16;
DOUT[4] := 0;
DOUT[5] := 0;
DOUT[6] := 0;
BUFSZ := 6;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
      ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_SREAD[',SLOT,'] swrite 1 = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
{  Second time for N A(1) F(16)
DOUT[1] := SLOT;
DOUT[2] := 1;
DOUT[3] := 16;
DOUT[4] := 0;
DOUT[5] := 0;
DOUT[6] := 0;
BUFSZ := 6;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
      ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_SREAD[',SLOT,'] swrite 2 = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
for I := 1 to SIZE do begin
  DOUT[1] := SLOT;  {Query 2108}
DOUT[2] := 0;
DOUT[3] := 2;
BUFSZ := 3;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_SREAD[',SLOT,'] swrite 3 = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
for J := 1 TO 10 do
  DIN[J] := 0;
BUFSZ := 3; LOTRM := 13; HITRM := 13; RECNUM := 0;
BUFOF := 0; ERRCODE := 0;
%sread(LUN2,@DIN[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in SA_SREAD sread at 3*pt = ',I,' ERRCODE = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
UNPCK_DOUT_ARRAY(BUFSZ,@DIN);
TEMP := DIN[1]; {BEWARE OF DATA OVERFLOWS CONVERTING}
  TEMP := TEMP*256 + DIN[2]; {TO 24-BIT NUMBERS IN LONGINT FORMAT}
  TEMP := TEMP*256 + DIN[3];
  SA_ARRAY^[I] := TEMP;
end;
end;

procedure NAF_GET_COMMAND; {Will ask for the NAF Command Sequence}
begin
  write (' N, A, F =');
  readln (INPUT, NAF_N, NAF_A, NAF_F);
end;

procedure NAF_DATA_GET; {Will ask for 3_bytes data}
begin
write (' DH, DM, DL =');
readln (INPUT, NAF_DH, NAF_DM, NAF_DL);
end;

procedure NAF_SINGLE; {Will send one NAF command to the Crate}
begin
NAF_GET_COMMAND;
DOUT[1] := NAF_N;
DOUT[2] := NAF_A;
BUFSZ := 3;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in NAF_SINGLE swrite = ',ERRCODE);
  perror(%cserr);
  KEY_PRESSED;
end;
end;

function NAF_KEYBOARD (*:boolean*);
var
  ASCII,SCANCODE,KBF1,KBF2 : INTEGER;
begin
%getkey(ASCII,SCANCODE,KBF1,KBF2,ERRCODE);
if ( ERRCODE = 0 ) then NAF_KEYBOARD := true
else NAF_KEYBOARD := false;
end;

procedure NAF_WRITE_DATA; {Will send one NAF and 3-bytes data to the Crate}
begin
THREE_BYTE_SETUP;
NAF_GET_COMMAND;
NAF_DATA_GET;
DOUT[1] := NAF_N;
DOUT[2] := NAF_A;
DOUT[4] := NAF_DH;
BUFSZ := 6;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in NAF_WRITE_DATA swrite = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure NAF_3READ_DATA; {Will send one NAF and read 3_bytes data}
var
  TEMP : LONGINT;
begin
  THREE_BYTE_SETUP;
  NAF_GET_COMMAND;
  DOUT[1] := NAF_N;
  DOUT[2] := NAF_A;
  BUFSZ := 3;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in NAF_3READ_DATA swrite = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
BUFSZ := 3;
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%sread(LUN2,@DIN[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
writeln('Error in NAF_3READ_DATA sread = ',ERRCODE);
%prterr(%cserr);
KEY_PRESSED;
end;
writeln(' DH, DM = ',DIN[1], ' DL, X = ',DIN[2]);
writeln(' call UNPCK_DOUT_ARRAY');
UNPCK_DOUT_ARRAY(BUFSZ,@DIN);
writeln(' DH = ',DIN[1], ' DM = ',DIN[2], ' DL = ',DIN[3]);
TEMP := DIN[1];            \{BEWARE OF DATA OVERFLOWS CONVERTING\}
TEMP := TEMP*256 + DIN[2]; \{TO 24-BIT NUMBERS IN LONGINT FORMAT\}
TEMP := TEMP*256 + DIN[3];
writeln(' or 24-byte total = ',TEMP);
writeln(' Press any letter to go on.');
repeat
   until (NAF_KEYBOARD = true);
end;

procedure NAF_MREAD_DATA; \{Will send one NAF and read 3_bytes data\}
var
   TEMP : LONGINT;
   CHANNEL_NUMBER : INTEGER;
begin
   NAF_GET_COMMAND;
   CHANNEL_NUMBER := 0;           \{Starting channel number\}
repeat
   THREE_BYTE_SETUP;
   DOUT[1] := NAF_N;
   DOUT[2] := NAF_A;
   BUFSZ := 3;
   PCK_DOUT_ARRAY(BUFSZ,@DOUT);
   LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
   %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
ERRCODE);
   if ERRCODE <> 0 then begin
      writeln('Error in NAF_MREAD_DATA swrite = ',ERRCODE);
      %prterr(%cserr);
      KEY_PRESSED;
   end;
end;
BUFSZ := 3;
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%sread(LUN2,@DIN[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in NAF_3READ_DATA sread = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
UNPCK_DOUT_ARRAY(BUFSZ,@DIN);

TEMP := DIN[1];  {BEWARE OF DATA OVERFLOWS CONVERTING}
TEMP := TEMP*256 + DIN[2];  {TO 24-BIT NUMBERS IN LONGINT FORMAT}
TEMP := TEMP*256 + DIN[3];
writeln (' CHANNEL ',CHANNEL_NUMBER,' = ' ,TEMP);
CHANNEL_NUMBER := CHANNEL_NUMBER + 1;
until (NAF_KEYBOARD = true);
end;

procedure FREQ_STROBE;
const
  SLOT = 17;
var
  I,Z : INTEGER;
begi
  ONE_BYTE_SETUP;
  DOUT[1] := SLOT;  {*******PLACE VALUE OF N****}
  DOUT[2] := 1;
  DOUT[3] := 16;
  DOUT[4] := 3;
  BUFSZ := 4;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
  if ERRCODE <> 0 then begin
    writeln('Error in FREQ_STROBE swrite 1 = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
for I := 1 to 2 do begin  {Start Strobe Loop}
  ONE_BYTE_SETUP;
  DOUT[1] := SLOT;  {*****PLACE VALUE OF N***}
  DOUT[2] := 1;
  DOUT[3] := 16;  {Must pause at this state for 1 microsec}
  DOUT[4] := 3;
  BUFSZ := 4;  {*NOT NECESSARY TO RE-ASSIGN*}
  PCK_DOUT_ARRAY(BUFSZ,@DOUT);
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
  if ERRCODE <> 0 then begin
    writeln('Error in FREQ_STROBE swrite 2 = ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
end;  {End first part of strobe loop}

procedure FREQ_SET(*FREQUENCY:REAL*);  {Units are kHz}
const
  SLOT = 17;
var
  FREQ_TEMP : LONGINT;
  FREQ_RANGE : INTEGER;

FREQ_DIGIT : ARRAY [1 .. 5] OF INTEGER;
I : INTEGER;
begin
if ((FREQUENCY >= 0.0001) and (FREQUENCY < 1.0)) then begin
    FREQ_RANGE := 5*2;
    FREQ_TEMP := TRUNC( FREQUENCY*10000.0);
end;
if ((FREQUENCY >= 1.0) and (FREQUENCY < 10.0)) then begin
    FREQ_RANGE := 4*2;
    FREQ_TEMP := TRUNC( FREQUENCY*10000.0);
end;
if ((FREQUENCY >= 10.0) and (FREQUENCY < 100.0)) then begin
    FREQ_RANGE := 3*2;
    FREQ_TEMP := TRUNC( FREQUENCY*1000.0);
end;
if ((FREQUENCY >= 100.0) and (FREQUENCY < 1000.0)) then begin
    FREQ_RANGE := 2*2;
    FREQ_TEMP := TRUNC( FREQUENCY*100.0);
end;
if ((FREQUENCY >= 1000.0) and (FREQUENCY < 10000.0)) then begin
    FREQ_RANGE := 1*2;
    FREQ_TEMP := TRUNC( FREQUENCY*10.0);
end;
for I := 5 downto 1 do begin
    FREQ_DIGIT[I] := TRUNC(FREQ_TEMP/PWROFTEN(I-1));
    FREQ_TEMP := TRUNC(FREQ_TEMP - FREQ_DIGIT[I]*PWROFTEN(I-1));
end;
THREE_BYTE_SETUP;
DOUT[1] := SLOT;  ****PLACE VALUE OF N****
DOUT[2] := 0;  {A(0)}
DOUT[3] := 16;  {F(16)}
DOUT[4] := FREQ_RANGE*16 + FREQ_DIGIT[5];
BUFSZ := 6;
PCK_DOUT_ARRAY(BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
   ERRCODE);
if ERRCODE <> 0 then begin
158

writelnCError in FREQ_SET swrite 1 = ERRCODE);
%prterr(%cserr);
KEY_PRESSED;
end;
FREQ_STROBE;
for I := 1 to 5000 do begin {Start Delay Loop}
FREQ_JEMP := FREQJTEMP;
end; {EndDelay Loop}
end;

procedure FREQ_CLEAR;
{*Will issue commands to frequency synthesizer*}
const
SLOT =17;
begin
THREE_BYTE_SETUP;
DOUT[ 1] := SLOT;
{*****PLACE VALUE OF N****}
DOUT[2] := 0; {A(0)}
DOUT[3] := 16; {F(16)}
DOUT[4] := 0;
DOUT[5) := 0;
DOUT[6} := 0;
BUFSZ := 6;
PCK_DOUT_ARRA Y (BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%s write(LUN2,@DOUT[ 1],BUFOF,BUFSZ.LOTRM,HITRM,RECNUM,
ERRCODE);
if ERRCODE <> 0 then begin
writelnCError in FREQ_CLEAR swrite 1 = ERRCODE);
%prterr(%cserr);
KEY_PRESSED;
end;
ONE_BYTE_SETUP;
DOUTfl] := SLOT; {*******PLACE VALUE OF N****}
DOUT[2] := 1;
DOUT[3] := 16;
DOUT[4] := 0;
BUFSZ := 4;
PCK_DOUT_ARRAY (BUFSZ,@DOUT);
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@DOUT[l],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,


ERRCODE);
if ERRCODE <> 0 then begin
  writeln('Error in FREQ_CLEAR swrite 2 = ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
end;

procedure TIMES_TO_SCREEN; {*Cooper IBM book, p144*}
var
  YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRAC : INTEGER;
begin
  %gettime(YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRAC);
  writeln('Date is ',MONTH:2,'-',DAY:2,'-',YEAR:2);
  writeln('Time is ',HOUR:2,':',MINUTE:2,':',SECOND:2);
end;

procedure SATURATE(*NUMBER_OF_REPEATS:INTEGER*);
var
  J : INTEGER;
begin
  {***STARTING UP SATURATE*****}
  if NUMBER_OF_REPEATS = 1 then begin
    ONE_BYTE_SETUP;
    DOUT[1] := 3;
    DOUT[2] := 4;
    DOUT[3] := 16;
    DOUT[4] := 0;
    BUFSZ := 4;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
    %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
             ERRCODE);
    if ERRCODE <> 0 then begin
      writeln('Error in SATURATE[1] swrite = ',ERRCODE);
      %prterr(%cserr);
      KEY_PRESSED;
    end;
  end;
else for J := 1 to NUMBER_OF_REPEATS do begin
    ONE_BYTE_SETUP;
    DOUT[1] := 3;
    DOUT[2] := 4;
    DOUT[3] := 16;
    DOUT[4] := 0;
    BUFSZ := 4;
    PCK_DOUT_ARRAY(BUFSZ,@DOUT);
    LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
    %swrite(LUN2,@DOUT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
            ERRCODE);
    if ERRCODE <> 0 then begin
        writeln('Error in SATURATE[N] swrite = ',ERRCODE);
        %prterr(%cserr);
        KEY_PRESSED;
    end;
end;
end;

procedure CAMAC_DISPLAY; {Writes options for CAMAC modules on screen}
begin
    %cls;
    %gotoxy(0,1);
    writeln('    CAMAC Modules (Direct Control)
    1: Timer Generator Control Status Register');
    writeln('    2: Timer Generator Channel Setup');
    writeln('    3: Timer Generator Execute');
    writeln('    4: Timer Generator Stop');
    writeln('    5: Saturate');
    writeln('    6: Signal Averagers Control Status Register');
    writeln('    7: Signal Averager Number of Sweeps');
    writeln('    8: Frequency Set');
    writeln('    9: Frequency Clear');
    writeln('   10: Frequency Strobe');
    writeln('   11: Capacitor Setup');
    writeln('   12: DAC setup');
    writeln('24 Exit to MAINMENU');
end;
procedure CAMAC_MENU;  {Options for direct control of CAMAC modules} 

var
  command : INTEGER;
  END_CAMAC_MENU : BOOLEAN;
  RECYCLE, CHANNELS_ACTIVE, FREQ_POWER_OF_TEN : INTEGER;
  CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8 : INTEGER;
  SLOT, N, M : INTEGER;
  RECORD_LENGTH : INTEGER;
  N_LONG : LONGINT;
  X : REAL;

begin

  %cls;
  END_CAMAC_MENU := FALSE;

repeat

  CAMAC_DISPLAY;
  readln (command);

  case command of
  1: {Timer Generator Control Status Register}
    begin
      write('Recycle = 1, single cycle =0: ? ');
      readln(RECYCLE);
      write('Number of channels active? ');
      readln(CHANNELS_ACTIVE);
      write('Power of ten count rate (Ex. 1=10Hz, 3=1kHz)? ');
      readln(FREQ_POWER_OF_TEN);
      TIMER_CS(RECYCLE, CHANNELS_ACTIVE, FREQ_POWER_OF_TEN);
    end;
  2: {Timer Generator Channel Setup}
    begin
      writeln('Enter the number of counts for all eight channels');
      readln(CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8);
      TIMER_CH_SETUP(CH1, CH2, CH3, CH4, CH5, CH6, CH7, CH8);
    end;
  3: {Timer Generator Execute}
    TIMER_EXE;
  4: {Timer Generator Stop}
    TIMER_STOP;
  5: {Saturate}
    begin
write(' Fire pulse programmer N times; Enter N ? ');  
readln(N);  
SATURATE(N);  
end;

6: {Signal Averagers Control Status Register}  
begin  
  writeln('Record Length: 0=8K, 1=4K, 2=2K, 3=1K ');  
  writeln(' 4=512, 5=256, 6=128, 7=64 ');  
  writeln('Sampling Rate: 0=20MHz, 1=10MHz, 2=5MHz, 3=2MHz ');  
  writeln(' 4=1MHz, 5=500kHz, 6=200kHz, 7=Ext Clock');  
  write(' Enter Slot, Record Length, and rate parameter ?');  
  readln(SLOT,RECORD_LENGTH,N);  
  SA_SETUP(SLOT,RECORD_LENGTH,0,N);  
end;

7: {Signal Averager Number of Sweeps}  
begin  
  write(' Enter Slot, Enter scans to take (max is 65536) ? ');  
  readln(SLOT,N_LONG);  
  SA_SWEEP(SLOT,N_LONG);  
end;

8: {Frequency Set}  
begin  
  write(' Enter frequency (units are kHz, real format) ? ');  
  readln(X);  
  FREQ_SET(X);  
end;

9: {Frequency Clear}  
FREQ_CLEAR;

10: {Frequency Strobe}  
FREQ_STROBE;

11: {Capacitor Setup}  
begin  
  writeln(' Series=0, Parallel =1; Switch Number ? ');  
  readln(N,M);  
  CAPS_SETUP(N,M);  
end;

12: {DAC setup}  
begin  
  write(' Enter Channel (0 to 3); Volts (-10.0 to +10.0) ? ');  
  readln(N,X);  
  DAC_SETUP(N,X);  
end;
end;
24: END_CAMAC_MENU := TRUE;
end;
until (END_CAMAC_MENU = TRUE);
end;

procedure NAF_C_DISPLAY;  { Writes options for NAF_CAMAC control on screen } begin
  %cls;
  %gotoxy(0,1);
  writeln(' NAFCAMAC TEST PROGRAM ');
  writeln;
  writeln(' 1 Send One NAF Command');
  writeln(' 2 Send One NAF and 3-bytes of Data');
  writeln(' 3 Send One NAF and read 3-bytes of DATA');
  writeln(' 4 Send One NAF and read 3-bytes of DATA - THEN REPEATS ');
  writeln(' (Note;=: End Option 4 by pressing any key ');
  writeln(' 5 Send One IFC Command');
  writeln(' 24 Exit to MAINMENU');
end;

procedure NAF_CAMAC;
var
  command : INTEGER;
  END_NAF_MENU : BOOLEAN;
begin
  %cls;
  END_NAF_MENU := FALSE;
  repeat
    NAF_C_DISPLAY;
    readln (command);
    case command of
      1: NAF_SINGLE;
      2: NAF_WRITE_DATA;
      3: NAF_3READ_DATA;
      4: NAF_MREAD_DATA;
      5: IFC_CAMAC;
      24: END_NAF_MENU := TRUE;
    end;
until (END_NAF_MENU = TRUE);
end;
END.
UNIT CAPLIB; {UPDATE 10/29/86}
{*****************************************************}

INTERFACE
USES {$U CPOSIL.LIB} system_library, system_calls,
{[$U 1:A.OBJ} ARRAYS,
{[$U 1:D.OBJ} DMMLIB,
{[$U 1:CMLB.OBJ} CAMLIB;

procedure CAP_PAUSE(WAIT:INTEGER);
procedure CAP_SETTING(FREQ:REAL);
procedure CAP_POP_SAMPLE_TO_ZERO_FIELD;
{Function CAP_VOLTS_TO_GAUSS(var VOLTS:REAL) : REAL;}
procedure CAP_TUNE(var H1_WANTED,H1_OBTAINED,FREQ:REAL);
procedure CAP_FIND_TUNING_POINTS;
procedure CAP_TEST;
procedure CAP_DISPLAY_MENU;
procedure CAP_MAIN;
{*****************************************************}

IMPLEMENTATION

const
PI = 3.141595;
MU = 1.25664E-2; {Gauss/Henry}
N = 23; {turns}
LENGTH = 2.3; {cm}
DIAMETER = 2.3; {cm}
SCALE_FACTOR = 1.0;
CS_INCREMENT = 18.0E-12; CS_MAX = 41999.0E-12;
CP_INCREMENT = 325.0E-12; CP_MAX = 3.99E-6;
Series_Parallel_Limit = 1000.0;

var
CS_INTEGER, CP_INTEGER : LONGINT;
PR : TEXT;

procedure CAP_PAUSE(*WAIT:INTEGER*);
var
YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC : INTEGER;
START, STOP : INTEGER;
begin
%_gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
START := SECOND;
if START < (60 - WAIT) then begin
  repeat
    %_gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
    STOP := SECOND;
    until (STOP - START) >= WAIT;
  end
else begin
  START := START - 60;
  repeat
    %_gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
    if (SECOND + WAIT) >= 60 then STOP := SECOND - 60 else STOP := SECOND;
    until (STOP - START) >= WAIT;
  end;
end;

procedure CAP_SETTING(*FREQ:REAL*);
var
  L, CT, CP, CS : REAL;
begin
{** GOOD ONLY FOR LOW FREQUENCIES ***************}
  CS := 0.0; CP := 0.0;
  if (FREQ < 15.0) then L := 6.0E-6;
  if (((FREQ >= 15.0) and (FREQ < 40.0)) = TRUE) then L := 8.5E-6;
  if (((FREQ >= 40.0) and (FREQ < 60.0)) = TRUE) then L := 8.5E-6;
  if (((FREQ >= 60.0) and (FREQ < 80.0)) = TRUE) then L := 8.5E-6;
  if (((FREQ >= 80.0) and (FREQ < 100.0)) = TRUE) then L := 8.5E-6;
  if (((FREQ >= 100.0) and (FREQ < 1200.0)) = TRUE) then begin
    L := 5.6E-6; CS := 1.7E-8;
  end;
  if (((FREQ >= 1200.0) and (FREQ < 1400.0)) = TRUE) then begin
    L := 4.4E-6; CS := 1.2E-8;
  end;
  if (((FREQ >= 1400.0) and (FREQ < 1600.0)) = TRUE) then begin
    L := 4.3E-6; CS := 8.7E-9;
  end;
  if (((FREQ >= 1600.0) and (FREQ < 1800.0)) = TRUE) then begin
    L := 5.5E-6; CS := 4.4E-9;
  end;
{** GOOD ONLY FOR HIGH FREQUENCIES ***************}
  if (((FREQ >= 1000.0) and (FREQ < 1200.0)) = TRUE) then begin
    L := 5.6E-6; CS := 1.7E-8; end;
  if (((FREQ >= 1200.0) and (FREQ < 1400.0)) = TRUE) then begin
    L := 4.4E-6; CS := 1.2E-8; end;
  if (((FREQ >= 1400.0) and (FREQ < 1600.0)) = TRUE) then begin
    L := 4.3E-6; CS := 8.7E-9; end;
  if (((FREQ >= 1600.0) and (FREQ < 1800.0)) = TRUE) then begin
    L := 5.5E-6; CS := 4.4E-9; end;
if ( ( (FREQ >= 1800.0) and (FREQ < 2000.0) ) = TRUE ) then begin
  L := 7.1E-6;  CS := 3.6E-9; end;
if ( ( (FREQ >= 2000.0) and (FREQ < 2200.0) ) = TRUE ) then begin
  L := 4.2E-6;  CS := 2.9E-9; end;
if ( ( (FREQ >= 2200.0) and (FREQ < 2400.0) ) = TRUE ) then begin
  L := 6.2E-6;  CS := 1.8E-9; end;
if ( ( (FREQ >= 2400.0) and (FREQ < 2600.0) ) = TRUE ) then begin
  L := 5.8E-6;  CS := 1.5E-9; end;
if ( ( (FREQ >= 2600.0) and (FREQ < 2800.0) ) = TRUE ) then begin
  L := 5.6E-6;  CS := 1.2E-9; end;
if ( ( (FREQ >= 2800.0) and (FREQ < 3000.0) ) = TRUE ) then begin
  L := 4.3E-6;  CS := 1.5E-9; end;
if (FREQ >= 3000.0) then begin
  L := 7.5E-6;  CS := 6.8E-10; end;
{** END OF TUNING DATA LIST ***************}
CT := 1.0/(L*SQR(2.0*PI*FREQ*1000.0));
if ( FREQ >= Series_Parallel_Limit) then 
  CP := 1.0/(1.0/CT - 1.0/CS)
else 
  CP := CT;
{** CHECK FOR CAPACITOR RANGE LIMITS AND ASSIGN SWITCH SETTINGS **}
  CS_INTEGER := round(CS / CS_INCREMENT);
  if ( CS > CS_MAX ) then CS_INTEGER := 2047;  {All ON}
{** Turn SHORT on for low frequency operation **}
  if ( FREQ <= Series_Parallel_Limit) then CS_INTEGER := 2048;
  CP_INTEGER := round(CP / CP_INCREMENT);
  if ( CP > CP_MAX ) then CP_INTEGER := 8191;  {All ON}
{** WRITE SWITCH SETTINGS **}
  CAPS_SETUP(0,CS_INTEGER);
  CAPS_SETUP(1,CP_INTEGER);
end;

procedure CAP_POP_SAMPLE_TO_ZERO_FIELD;
begin
  SAMPLE_POSITION_RESET; CAP_PAUSE(2);
  TIMER_CH_SETUP(1,0,0,0,0,0,0,0); CAP_PAUSE(1);
  TIMER_CS(0,1,1); CAP_PAUSE(1);
  TIMER_EXE;
end;
{
function CAP_VOLTS_TO_GAUSS(*var VOLTS:REAL : REAL*);
var
I_PEAK : REAL;
H1_PEAK : REAL;
begin
I_PEAK := SCALE_FACTOR * VOLTS;
H1_PEAK := MU * I_PEAK * N / LENGTH;
CAP_VOLTS_TO_GAUSS := H1_PEAK;
end;

procedure CAP_TUNE(*var H1_WANTED,H1_OBTAINED,FREQ: REAL*);
const
Mixer_Limit = 5.0;
var
Mixer_Drive : REAL;
H1_ERROR : REAL;
X : REAL;
DONE : BOOLEAN;
Mixer_Increment : REAL;
H1_ERR_LIMIT : REAL;
LOOPS : INTEGER;
begin
CAP_SETTING(FREQ);
Mixer_Drive := 0.25; Mixer_Increment := 0.25; H1_ERR_LIMIT := 2.0;
DAC_SETUP(0,Mixer_Drive);
CAP_POP_SAMPLE_TO_ZERO_FIELD;
DONE := FALSE; writeln;
while ( (Mixer_Drive <= Mixer_limit) and (DONE = FALSE) ) do
begin
DAC_SETUP(0,Mixer_Drive);
DMM_VDCSLOW(X);
{  H1_OBTAINED := CAP_VOLTS_TO_GAUSS(X); }
H1_OBTAINED := X;
writeln (Mixer_Drive:7;3, H1_OBTAINED:9:4);
H1_ERROR := 100.0 * ( H1_WANTED - H1_OBTAINED ) / H1_WANTED;
if ( H1_ERROR <= H1_ERR_LIMIT ) then DONE := TRUE
else Mixer_Drive := Mixer_Drive + Mixer_Increment;
end;
Mixer_Increment := 0.01; H1_ERR_LIMIT := 1.0;
DONE := FALSE; LOOPS := 0;
while ( (Mixer_Drive <= Mixer_Limit) and (DONE = FALSE) ) do
begin
DMM_VDCSLOW(X);
{  H1_OBTAINED := CAP_VOLTS_TO_GAUSS(X); }
H1_OBTAINED := X;
writeln (Mixer_Drive:7:3, H1_OBTAINED:9:4);
H1_ERROR := 100.0 * ( H1_WANTED - H1_OBTAINED ) / H1_WANTED;
if ( abs(H1_ERROR) <= H1_ERR_LIMIT ) then DONE := TRUE;
if DONE = FALSE then begin
if H1_ERROR > 0 then
   Mixer_Drive := Mixer_Drive + Mixer_Increment;
if H1_ERROR < 0 then
   Mixer_Drive := Mixer_Drive - Mixer_Increment;
DAC_SETUP(0,Mixer_Drive); LOOPS := LOOPS + 1;
if (LOOPS >= 50) then DONE := TRUE; {*Just quit and go on*}
end;
end;
SAMPLE_POSITION_RESET;
end;

procedure CAP_FIND_TUNING_POINTS;
var
L,CT,CP,CS : REAL;
H1_PEAK,FREQ,X : REAL;
CS_INT, CP_INT : INTEGER;
SHORT,DONE : BOOLEAN;
A : CHAR;
A_STRING : STRING80;
begin
DONE := FALSE;
%cls;
writeln('Turn on PRINTER');
writeln('Set Auto/Man switch to MAN; Put all caps switches down');
writeln('Is SHORT up? (up=Parallel LC; down=Series-Parallel LC');
write ('Enter SHORT setting [U or D ]');
readln(A); if (A = 'U') then SHORT := TRUE else SHORT := FALSE; {*SHORT=UP=TRUE means only parallel capacitors are used*}
repeat
  writeln;
  write('Enter Frequency [kHz, real format] [0.0 to stop] ');
  readln(FREQ);
  if (FREQ = 0.0) then DONE := TRUE
  else
    begin {begin big loop}
      FREQ_SET(FREQ);
      A_STRING := 'S2T0'; DMM_COMMAND(A_STRING);
      if (SHORT = TRUE) then CS := 0.0;
      if (SHORT = FALSE) then begin
        writeln('Enter series (top row) capacitors that are on. End with 0');
        CS := 9.0E-12;
        repeat
          write(' '); read(CS_INT);
          case CS_INT of
            1: CS := CS + 18.0E-12;
            2: CS := CS + 39.0E-12;
            3: CS := CS + 78.0E-12;
            4: CS := CS + 156.0E-12;
            5: CS := CS + 330.0E-12;
            6: CS := CS + 670.0E-12;
            7: CS := CS + 1450.0E-12;
            8: CS := CS + 2900.0E-12;
            9: CS := CS + 5800.0E-12;
            10: CS := CS + 11600.0E-12;
            11: CS := CS + 22000.0E-12;
          end;
        until (CS_INT = 0);
      end;
      writeln('Enter parallel (bottom row) capacitors that are on. End with 0');
      CP := 78.0E-12;
      repeat
        write(' '); read(CP_INT);
        case CP_INT of
          1: CP := CP + 325.0E-12;
          2: CP := CP + 750.0E-12;
          3: CP := CP + 1500.0E-12;
          4: CP := CP + 3000.0E-12;
          5: CP := CP + 6000.0E-12;
        end;
6: CP := CP + 12000.0E-12;
7: CP := CP + 24000.0E-12;
8: CP := CP + 0.05E-6;
9: CP := CP + 0.125E-6;
10: CP := CP + 0.250E-6;
11: CP := CP + 0.50E-6;
12: CP := CP + 1.0E-6;
13: CP := CP + 2.0E-6;
end;
until (CP_INT = 0);

{** Now calculate how much capacitance is in the circuit **}
if (SHORT = TRUE) then CT := CP
else CT := CS*CP/(CS + CP);
L := 1.0/( CT * SQR( 2.0 * PI * (FREQ * 1.0E3) ) );
DMM_VDCSLOW(X);
 writeln;
 writeln(FREQ:7:2,'kHz C P = ' ,CP:7,'  CS=',CS:7,' L= ',L:11,
 ' VOLTS= ',X:11);
 writeln(PR,FREQ:7:2,'kHz CP=',CP:7,' CS= ',CS:7,' L=',L:11,
 ' VOLTS= ',X:11);
end; {begin big loop}
until (DONE = TRUE);
end;

procedure CAP_TEST;
var
 FREQ,VOLTS : REAL;
 FREQ_START,FREQ_STOP : REAL;
 H1_WANTED,H1_OBTAINED : REAL;
begin
 %cls;
 writeln(' STARTING TUNE TEST');
 write(' Enter Starting and Stopping Frequencies (Increment = 10kHz). ');
 readln (FREQ_START,FREQ_STOP);
 write(' Enter desired H1 (volts)? ');
 readln(H1_WANTED);
 writeln(PR,' H1_WANTED = ' ,H1_WANTED:12:5);
 FREQ := FREQ_START;
 repeat
  FREQ_SET(FREQ);
  CAP_TUNE(H1_WANTED,H1_OBTAINED,FREQ);
writeln(FREQ:7:1,'kHz H1 = ',H1_OBTAINED:12:5);
write(PR,FREQ:7:1,'kHz H1 = ',H1_OBTAINED:12:5);
write(PR,' CS = ',CS_INTEGER);
writeln(PR,' CP = ',CP_INTEGER);
FREQ := FREQ + 10.0;
until(FREQ > FREQ_STOP);
DAC_SETUP(0,0.0);
end;

procedure CAP_DISPLAY_MENU;
begin
   %cls;
   writeln(' Capacitor Tuning Command Menu');
   writeln;
   writeln(' 1 Find tuning points in new circuit.');
   writeln(' 2 Run standard test over specified frequency range.');
   writeln(' 3 Set the capacitors for a given frequency and H1.');
   writeln(' 24 Exit to MAINMENU');
end;

procedure CAP_MAIN;
var
   capnd,ioerr : INTEGER;
   capexit  : BOOLEAN;
   FREQ  : REAL;
   H1_WANTED : REAL;
   H1_OBTAINED : REAL;
begin
   capexit := FALSE;
   repeat
      CAP_DISPLAY_MENU;
      readln(capnd);
      case capnd of
         1: CAP_FIND_TUNING_POINTS;
         2: CAP_TEST;
         3: begin
            write('Enter Frequency and H1_wanted. ');
            readln (FREQ,H1_WANTED);
         end;
      end;
end;
\[ FREQ\_SET(FREQ); \]
\[ CAP\_TUNE(H1\_WANTED, H1\_OBTAINED, FREQ); \]
\[ \text{end}; \]
\[ 24: \text{capexit} := \text{TRUE}; \]
\[ \text{end}; \]
\[ \text{until (capexit = TRUE);} \]
\[ \text{end}; \]

END.
UNIT DMMLIB; {UPDATE 5/21/87}

*******************************************************************************

INTERFACE
USES {$U CPOSIL.LIB} system_library,system_calls,IEEE488;

procedure DMM_PAUSE;
procedure DMM_STRING_TO_INTEGER;
procedure DMM_INT_TO_STRING;
procedure DMM_REAL_FROM_STRING(var X:REAL);
procedure GET_LUN3(var LUN3:INTEGER); {ASSIGN LUN3 TO DMM FOR SYSTEM USE}
procedure DMM_OPEN; {INITIALIZES AND CLEARS DMM}
procedure CLS_DMM; {CLOSE DMM}
procedure DMM_VDCSLOW(var VOLTS:REAL);
procedure DMM_VDCFAST(var VOLTS:REAL);
procedure DMM_COMMAND(var A_STRING:STRING80);
procedure DMM_DISPLAY_MENU;
procedure DMM_MAIN;

*******************************************************************************

IMPLEMENTATION

type DUMMY_ARRAY = array [1..10] of integer;

var
  DMM_STRING : STRING80;
  DMM_INT : ARRAY [1..80] OF INTEGER;
  FPK : DUMMY_ARRAY;
  DEV3 : STRING8;
  B_IOPTR : LONGINT;
  BUFSZ : LONGINT;
  TRANMODE,LOTRM : LONGINT;
  HITRM,OPT,DIR,BUFOF : LONGINT;
  RECNUM : LONGINT;
  LUN3,ERRCODE,II : INTEGER;
  DMM_ERROR : BOOLEAN;

procedure DMM_PAUSE;
var
  A : CHAR;
begin
  writeln;
  writeln( 'Ready to go on? Press any key');
  readln(A);
end;

procedure DMM_STRING_TO_INTEGER;
var
  I, J : INTEGER;
  LNAME : INTEGER;
begin
  LNAME := LENGTH(DMM_STRING);
  I := 1;
  J := 1;
  repeat
    if ( J <= LNAME ) then begin
      DMM_INT[I] := 256*ord(DMM_STRING[J]); {Pack in High Byte}
      J := J + 1;
    end;
    if ( J <= LNAME ) then begin
      DMM_INT[I] := DMM_INT[I] + ord(DMM_STRING[J]); {Pack in Low Byte}
      J := J + 1;
      I := I + 1;
    end;
  until (J > LNAME);
  BUFSZ := LNAME;
end;

procedure DMM_INT_TO_STRING;
begin
  DMM_ERROR := FALSE;
  DMM_STRING[1] := chr( (DMM_INT[1] div 256) );
if( ( (ord(DMM\_STRING[10])=ord('2')) and (ord(DMM\_STRING[11])=ord('1'))) ) =
Tn
    DMM\_ERROR := TRUE;
if ( DMM\_ERROR = FALSE ) then begin
    if ( chr(DMM\_INT[6]) = ',' ) then begin
        DMM\_STRING[12] := ',';
        DMM\_STRING[16] := chr( (DMM\_INT[8] mod 256) );
    end;
end;
end;

procedure DMM\_REAL\_FROM\_STRING(*var X:REAL*);
var
    X1,X2,X3,X4,X5,X6 : REAL;
    SIGN,EXP\_SIGN : CHAR;
    TEN\_TO\_THE\_X : INTEGER;
begin
    if ( DMM\_ERROR = FALSE ) then begin
        SIGN := DMM\_STRING[1];
        if ( '.' = DMM\_STRING[3] ) then begin
            X1 := ( ord(DMM\_STRING[2]) - 48 ) ;
            X2 := ( ord(DMM\_STRING[4]) - 48 ) /10.0;
            X3 := ( ord(DMM\_STRING[5]) - 48 ) /100.0;
            X4 := ( ord(DMM\_STRING[6]) - 48 ) /1000.0;
            X5 := ( ord(DMM\_STRING[7]) - 48 ) /10000.0;
            X6 := ( ord(DMM\_STRING[8]) - 48 ) /100000.0;
        end;
        if ( '.' = DMM\_STRING[4] ) then begin
            X1 := ( ord(DMM\_STRING[2]) - 48 ) *10.0;
            X2 := ( ord(DMM\_STRING[3]) - 48 ) ;
            X3 := ( ord(DMM\_STRING[5]) - 48 ) /10.0;
            X4 := ( ord(DMM\_STRING[6]) - 48 ) /100.0;
        end;
    end;
X5 := ( ord(DMM_STRING[7]) - 48 ) /1000.0;
X6 := ( ord(DMM_STRING[8]) - 48 ) /10000.0;
end;
if ( '=' = DMM_STRING[5]) then begin
  X1 := ( ord(DMM_STRING[2]) - 48 ) *100.0;
  X2 := ( ord(DMM_STRING[3]) - 48 ) *10.0;
  X3 := ( ord(DMM_STRING[4]) - 48 ) ;
  X4 := ( ord(DMM_STRING[5]) - 48 ) ;
  X5 := ( ord(DMM_STRING[7]) - 48 ) /10.0;
  X6 := ( ord(DMM_STRING[8]) - 48 ) /100.0;
end;
if ( '=' = DMM_STRING[6]) then begin
  X1 := ( ord(DMM_STRING[2]) - 48 ) *1000.0;
  X2 := ( ord(DMM_STRING[3]) - 48 ) *100.0;
  X3 := ( ord(DMM_STRING[4]) - 48 ) *10.0;
  X4 := ( ord(DMM_STRING[5]) - 48 ) ;
  X5 := ( ord(DMM_STRING[7]) - 48 ) /10.0;
  X6 := ( ord(DMM_STRING[8]) - 48 ) /100.0;
end;
X := X1 + X2 + X3 + X4 + X5 + X6;
if ( SIGN = '-' ) then X := -1.0 * X;
EXP_SIGN := DMM_STRING[10];
TEN_TO_THE_X := ( ord(DMM_STRING[11]) - 48 ) ;
if (EXP_SIGN = '+') then
  X := X * PWROFTEN(TEN_TO_THE_X);
if (EXP_SIGN = '-') then
  X := X / PWROFTEN(TEN_TO_THE_X);
end;
if ( DMM_ERROR = TRUE ) then X := 0.0;
end;

procedure GET_LUN3(* var LUN3:INTEGER *); {ASSIGN LUN3 TO DMM FOR
SYSTEM USE}
begin
  LUN3 := 133;
  writeln(' The LUN for #BUSC? is ',LUN3);
end;

procedure DMM_OPEN; {INITIALIZES AND CLEARS DMM}
var  {TIMEOUT SET TO 5 SECONDS}
    I : INTEGER;
begin
    GET_LUN3(LUN3);
    DEV3 := '#BUSC?';
    B_IOPTR := 0;
    TRANMODE := 0; LOTRM := 13; HITRM := 13; RECNUM := 0; OPT := 0;
    DIR := 2;
    %sysopen(LUN3,DEV3,DIR,OPT,TRANMODE,NIL,B_IOPTR,ERRCODE);
    if ERRCODE <> 0 then begin
        writeln('Error in sysopen for #BUSC? is ',ERRCODE);
        writeln('Type any number to continue.');
        readln(I);
    end;
    FPK[1] := 30;
    FPK[1] := 100;
    FPK[1] := 0;
    %sysfunc(LUN3,@FPK,ERRCODE);
    if ERRCODE <> 0 then begin
        writeln('Error in timeout(5sec) for #BUSC? is ',ERRCODE);
        writeln('Type any number to continue.');
        readln(I);
    end;
    DMM_STRING := '*';
    DMM_STRING_TO_INTEGER;
    BUFOF := 0;
    %swrite(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
            ERRCODE);
    if ERRCODE <> 0 then begin
        writeln('Error in syswrite for #BUSC? is ',ERRCODE);
        writeln('Type any number to continue.');
        readln(I);
    end;
end;

procedure CLS_DMM;  {CLOSE DMM}
var
    I : INTEGER;
begin
    %sysclose(LUN3,NIL,ERRCODE);
    if ERRCODE <> 0 then begin

writeln('Error in %sysclose for #BUSC? is ',ERRCODE);
writeln('Type any number to continue.');
readln(I);
end;
end;

procedure DMM_VDCSLOW(*var VOLTS:REAL*);
var delay_counter : integer;
begin
  delay_counter := 0;
  DMM_STRING := 'F1R0S0T0?';  { VDC, Auto, Slow rate }
  DMM_STRING_TO_INTEGER;
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM, ERRCODE);
  if ERRCODE <> 0 then writeln('Error in swrite is ',ERRCODE);
  BUFSZ := 11; BUFOF := 0; ERRCODE := 0;
  %sread(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM, ERRCODE);
  if ERRCODE <> 0 then writeln('Error in sread is ',ERRCODE);
  DMM_INT_TO_STRING;
  DMM_REAL_FROM_STRING(VOLTS);
end;

procedure DMM_VDCFAST(*var VOLTS:REAL*);
var
  delay_counter : integer;
begin
  delay_counter := 0;
  DMM_STRING := 'F1R0S1T0?';  { VDC, Auto, Medium rate }
  DMM_STRING_TO_INTEGER;
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM, ERRCODE);
  if ERRCODE <> 0 then writeln('Error in swrite is ',ERRCODE);
  BUFSZ := 11; BUFOF := 0; ERRCODE := 0;
  %sread(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM, ERRCODE);
  if ERRCODE <> 0 then writeln('Error in sread is ',ERRCODE);
  DMM_INT_TO_STRING;
procedure DMM_COMMAND(*var A_STRING:STRING80*);
    var
        I : INTEGER;
    begin
        DMM_STRING := A_STRING;
        DMM_STRING_TO_INTEGER;
        LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
        %swrite(LUN3,@DMM_INT[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
                ERRCODE);
    end;

procedure DMM_DISPLAY_MENU;
    begin
        %cls;
        writeln(' Digital Multimeter Command Menu');
        writeln;
        writeln(' 1 Open IEEE-488 bus to DMM');
        writeln(' 2 Get voltage reading; Slow reading rate.');
        writeln(' 3 Get voltage reading; Fast reading rate.' );
        writeln(' 4 Send any command string to DMM');
        writeln(' 24 Exit to MAINMENU');
    end;

procedure DMM_MAIN;
    var
        exit : boolean;
        command : integer;
        VOLTS : REAL;
        A_STRING : STRING80;
    begin
        exit := FALSE;
        repeat
            DMM_DISPLAY_MENU;
            readln(command);
            case command of
                1: DMM_OPEN;
                2: {Get a voltage reading}
begin
  DMM_VDCSLOW(VOLTS);
  writeln(' Volts = ',VOLTS, ' Error code = ',DMM_ERROR);
  DMM_PAUSE;
end;

3: {Get a voltage reading}
  begin
    DMM_VDCFAST(VOLTS);
    writeln(' Volts = ',VOLTS, ' Error code = ',DMM_ERROR);
    DMM_PAUSE;
  end;

4: begin
  writeln('Enter a command string');
  readln(A_STRING);
  DMM_COMMAND(A_STRING);
end;

24: exit := TRUE;
end;
until (exit = TRUE);
end;
end.
UNIT PULSE;  {UPDATE 6/18/87}
{************************************************************************************}

INTERFACE
USES {SU 0:CPOSIL.LIB} system_library, system_calls, IEEE488, cserr,
    {SU 1:A.OBJ} ARRAYS,
    {SU 1:CMLB.OBJ} CAMLIB,
    {SU 1:S.OBJ} SAVEFID;

procedure PULSE_PACK_ARRAY;
procedure PULSE_UNPACK_ARRAY;
procedure LIST_COMMANDS;
procedure BIT_STORE_COMMANDS;
procedure SET_DWELL_COMMANDS;
procedure NEW_BITS_COMMANDS;
procedure LOAD_COMMANDS;
procedure PP_SL_WRITE;   {Will send pulse sequence to slot-3}
procedure PP_SL_READ;   {Will read pulse sequence from slot-3}
procedure MAKE_COMMANDS;
procedure DECODE_COMMANDS;
function PP_SAFETY_TEST: BOOLEAN; {FALSE IF BAD; FALSE IF GOOD}
procedure PULSE_SAVE;
procedure PULSE_READ(var NAME, HEADER: STRING80);
procedure SPIN_LOCK;
procedure PULSE_MENU;
procedure PULSE_MAIN;
{************************************************************************************}

IMPLEMENTATION

var
    PP_I_O   : ARRAY [1 .. 10] OF INTEGER;
    BIT      : ARRAY [1 .. 16] OF INTEGER;
    COMMANDS : ARRAY [1 .. 1024] OF INTEGER;
    COMM_POINTER: IPOINTER;
    STATE_DWELL : ARRAY [1 .. 1024] OF LONGINT;
    DWELL    : REAL;
    PNTER    : INTEGER;
    ANSWER   : CHAR;
    BUFSZ,LUN2 : INTEGER;
ERRCODE : INTEGER;
LOTRM,HITRM : INTEGER;
RECNUM,BUFOF : LONGINT;

procedure PULSE_PACK_ARRAY;
var
  I,J,K,L : INTEGER;
  TEMP : LONGINT;
begin
  I := 1;
  J := 1;
  K := BUFSZ + 1;
  for L := K to 10 do
    PP_I_0[L] := 0;
  while ( J < K ) do begin
    TEMP := PP_I_0[J]*256 + PP_I_0[J+1];
    PP_I_0[I] := 0 or TEMP;
    I := I+ 1;
    J := J + 2;
  end;
end;

procedure PULSE_UNPACK_ARRAY;
var
  TEMPI,TEMP2 : INTEGER;
  I,J : INTEGER;
  DIN_TEMP : ARRAY [1..10] OF INTEGER;
begin
  I := 1;
  J := 1;
  while ( I <= BUFSZ ) do begin
    TEMPI := PP_I_0[J] and (256*(64+32+16+8+4+2+1));
    TEMPI := TEMPI div 256;
    IF ( PP_I_0[J] < 0 ) THEN TEMPI := TEMPI + 128;
    TEMP2 := PP_I_0[J] and (128+64+32+16+8+4+2+1);
    IF ( I = BUFSZ ) THEN TEMP2 := 0;
    DIN_TEMP[I] := TEMPI; {ASSIGN HIGH BYTE TO DIN_TEMP[I]}
    DIN_TEMP[I+1] := TEMP2; {ASSIGN LOW BYTE TO DIN_TEMP[I+1]}
    I := I + 2;
  end;
end;
J := J + 1;
end;
I := 1;
while ( I <= BUFSZ ) do begin
PP_I\_O[I] := DIN\_TEMP[I];
I := I + 1;
end;
end;

procedure LIST\_COMMANDS;
var
    I,J,K : INTEGER;
begin
%cls;
%gotoxy(25,0);
write('ESC - if displayed state selection is ready to be used');
%gotoxy(3,1);
writeln(STATE\_DWELL[1], ' - TOTAL NUMBER OF STATES');
%gotoxy(3,2);
DWELL := (STATE\_DWELL[PNTER] and $0FFF0000) / (10 * $00010000); 
writeln(DWELL:6:2, ' DWELL IN CURRENT STATE (us)'); 
%gotoxy(3,4); 
writeln('CURRENT STATE PARAMETERS');
%gotoxy(3,5); 
write(BIT[1], ' A0 - NOVEX rf amplitude LSB'); 
%gotoxy(3,6); 
write(BIT[2], ' A1 - NOVEX'); 
%gotoxy(3,7); 
write(BIT[3], ' A2 - NOVEX'); 
%gotoxy(3,8); 
write(BIT[4], ' A3 - NOVEX rf amplitude MSB'); 
%gotoxy(3,9); 
write(BIT[5], ' 90\textdegree - NOVEX rf phase'); 
%gotoxy(3,10); 
write(BIT[6], ' 180\textdegree - NOVEX rf phase'); 
%gotoxy(3,11); 
write(BIT[7], ' PULSE ON - NOVEX'); 
%gotoxy(3,12); 
write(BIT[8], ' DIGITIZER TRIGGER'); 
%gotoxy(3,13); 

write(BIT[9],' SCOPE TRIGGER');
%gotoxy(3,14);
write(BIT[10],' X1');
%gotoxy(3,15);
write(BIT[11],' X2');
%gotoxy(3,16);
write(BIT[12],' HALT PULSE PROGRAMMER OUTPUT');
end;

procedure BIT_STORE_COMMANDS;
var
  I : INTEGER;
begin
  BIT[1] := STATE_DWELL[PINTER] and $001;
  BIT[2] := STATE_DWELL[PINTER] and $002;
  BIT[3] := STATE_DWELL[PINTER] and $004;
  BIT[4] := STATE_DWELL[PINTER] and $008;
  BIT[5] := STATE_DWELL[PINTER] and $010;
  BIT[7] := STATE_DWELL[PINTER] and $040;
  BIT[8] := STATE_DWELL[PINTER] and $080;
  BIT[9] := STATE_DWELL[PINTER] and $100;
  BIT[10] := STATE_DWELL[PINTER] and $200;
  BIT[12] := STATE_DWELL[PINTER] and $800;
  for I := 1 to 12 do begin
    if BIT[I] > 0 then BIT[I] := 1;
  end;
end;

procedure SET_DWELL_COMMANDS;
var
  I : LONGINT;
begin
  %gotoxy(3,18);
  writeln('Enter dwell time in microseconds');
  writeln('(to finish pulse sequence with this state, enter 0.0)');
  readln(DWELL);
if (DWELL>0.0) THEN
  begin
    DWELL := DWELL * 10;
    I := TRUNC(DWELL) * $00010000;
    STATE_DWELL[PINTER] := (STATE_DWELL[PINTER] and $0000FFFF) + I;
  end
else
  STATE_DWELL[PINTER] := (STATE_DWELL[PINTER] and $0000FFFF) + $8FFF0000;
end;

procedure NEW_BITS_COMMANDS;
  label 999;
  var
    I,J : INTEGER;
  begin
    repeat
      %gotoxy(2,5);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[1]);
      if I = 27 then GOTO 999;
      %gotoxy(2,6);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[2]);
      if I = 27 then GOTO 999;
      %gotoxy(2,7);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[3]);
      if I = 27 then GOTO 999;
      %gotoxy(2,8);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[4]);
      if I = 27 then GOTO 999;
      %gotoxy(2,9);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[5]);
      if I = 27 then GOTO 999;
      %gotoxy(2,10);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[6]);
      if I = 27 then GOTO 999;
      %gotoxy(2,11);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[7]);
      if I = 27 then GOTO 999;
      %gotoxy(2,12);
      READ_KEY(I,J);
      if I = 32 then READLN(BIT[8]);
      if I = 27 then GOTO 999;
    end;
%gotoxy(2,11);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[7]);
  if I = 27 then GOTO 999;
%gotoxy(2,12);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[8]);
  if I = 27 then GOTO 999;
%gotoxy(2,13);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[9]);
  if I = 27 then GOTO 999;
%gotoxy(2,14);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[10]);
  if I = 27 then GOTO 999;
%gotoxy(2,15);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[11]);
  if I = 27 then GOTO 999;
%gotoxy(2,16);
READ_KEY(I,J);
  if I = 32 then READLN(BIT[12]);
  if I = 27 then GOTO 999;
%gotoxy(3,17);
writeln('Do you want to make anymore changes? - Y / ');
readln(ANSWER);
  if (ANSWER='Y') then LIST_COMMANDS;
until(ANSWER<> 'Y');
999: IF (PNTER < 1024) THEN PNTER := PNTER + 1;
   STATE_DWELL[1] := PNTER;
   STATE_DWELL[PNTER] := STATE_DWELL[PNTER] and $0FF0000;
   STATE_DWELL[PNTER] := STATE_DWELL[PNTER] + BIT[1]*$001 +
                           BIT[2]*$002 + BIT[3]*$004 + BIT[4]*$008 + BIT[5]*$010 + BIT[6]*$020 +
                           BIT[7]*$040 + BIT[8]*$080 + BIT[9]*$100 + BIT[10]*$200 + BIT[11]*$400 +
                           BIT[12]*$800;
   SET_DWELL_COMMANDS;
%cls;
end;
procedure LOAD_COMMANDS;
var
 I,J      : INTEGER;
PNT_COM   : INTEGER;
PNT_S_D   : INTEGER;
DWELL_INDEX : INTEGER;
MAX_STATES : INTEGER;
begin
  MAX_STATES := STATE_DWELL[1];
  PNT_COM := 1;
  COMMANDS[1] := $0080; {HALT IS BIT-12, HALT ON = $0080}
  FOR PNT_S_D := 2 TO MAX_STATES DO BEGIN
    DWELL_INDEX := (STATE_DWELL[PNT_S_D] div $00010000);
    IF (STATE_DWELL[PNT_S_D] < 0) THEN DWELL_INDEX := 1024 -
       PNT_COM;
    IF ((PNT_COM + DWELL_INDEX) > 1024) THEN DWELL_INDEX := 1024 -
       PNT_COM;
    FOR J := 1 TO DWELL_INDEX DO BEGIN
      COMMANDS[PNT_COM] := STATE_DWELL[PNT_S_D] and $0000FFFF;
      PNT_COM := PNT_COM + 1;
    END;
  END;
  J := PNT_COM;
  for PNT_COM := J to 1024 do
    COMMANDS[PNT_COM] := $0000;
end;

procedure PP_SL_WRITE; {Will send pulse sequence to slot-3}
var
 TEMP1,TEMP2,TEMP3 : INTEGER;
begin
  GET_LUN2(LUN2);
  ONE_BYTE_SETUP;
  PP_L_O[1] := 3; {THIS WILL SETUP PULSE PROGRAMMER FOR INPUT}
  PP_L_O[2] := 3; {N(3), A(3), F(16) : RESET THE ADDRESS COUNTER TO}
  PP_L_O[3] := 16; {ZERO AND HALT}
  PP_L_O[4] := 0;
  BUFSZ := 4;
  PULSE_PACK_ARRAY;

LOTROM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@PP_I_0[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in %swrite in PP_SL_WRITE is ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
ONE_BYTE_SETUP;
PP_I_0[1] := 3; {THIS WILL SETUP PULSE PROGRAMMER FOR INPUT}
PP_I_0[2] := 1; {N(3), A(1), F(16): ACTIVATE THE CAMAC MODE}
PP_I_0[3] := 16; {FOR READING OR WRITING TO PULSE PROGRAMMER MEMORY}
PP_I_0[4] := 0; {Q LAMP = ON}
BUFSZ := 4;
PULSE_PACK_ARRAY;
LOTROM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0;
%swrite(LUN2,@PP_I_0[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in %swrite in PP_SL_WRITE is ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
end;
COMM_POINTER := @COMMANDS[1];
SEND_ARRAY(1024,3,0,16,COMM_POINTER);
ONE_BYTE_SETUP;
PP_I_0[1] := 3; {Reset the address counter}
PP_I_0[2] := 3;
PP_I_0[3] := 16;
PP_I_0[4] := 0;
BUFSZ := 4;
PULSE_PACK_ARRAY;
LOTROM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@PP_I_0[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
if ERRCODE <> 0 then begin
  writeln(' Error in %swrite in PP_SL_WRITE is ',ERRCODE);
  %prterr(%cserr);
  KEY_PRESSED;
procedure PP_SL_READ;  {Will read pulse sequence from slot-3}
var
  I, TEMP1, TEMP2 : INTEGER;
begin
  GET_LUN2(LUN2);
  ONE_BYTE_SETUP;
  PP_I_O[1] := 3;  {THIS WILL SETUP PULSE PROGRAMMER FOR OUTPUT}
  PP_I_O[2] := 3;  {N(3), A(3), F(16) : RESET THE ADDRESS COUNTER TO}
  PP_I_O[3] := 16;  {ZERO AND HALT}
  PP_I_O[4] := 0;
  BUFSZ := 4;
  PULSE_PACK_ARRAY; BUFOF := 0;
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN2,@PP_I_O[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
  if ERRCODE <> 0 then begin
    writeln(' Error in %swrite in PP_SL_READ is ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
  ONE_BYTE_SETUP;
  PP_I_O[1] := 3;  {THIS WILL SETUP PULSE PROGRAMMER FOR OUTPUT.}
  PP_I_O[3] := 16;  {FOR READING OR WRITING TO PULSE PROGRAMMER
       MEMORY}
  PP_I_O[4] := 0;  {Q LAMP = ON}
  BUFSZ := 4;
  PULSE_PACK_ARRAY;
  LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
  %swrite(LUN2,@PP_I_O[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
  if ERRCODE <> 0 then begin
    writeln(' Error in %swrite in PP_SL_READ is ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
  for I := 1 to 1024 do begin
TWO_BYTE_SETUP;
PP_I_O[1] := 3;  {Read the pulse programmer with auto-increment}
PP_I_O[2] := 0;  {after every read command}
BUFSZ := 3;
PULSE_PACK_ARRAY;
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@PP_I_O[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
if ERRCODE <> 0 then begin
   writeln(' Error in %swrite in PP_SL_READ is ',ERRCODE);
   %prterr(%cserr);
   KEY_PRESSED;
end;
BUFSZ := 2; BUFOF := 0;
%sread(LUN2,@PP_I_O[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
if ERRCODE <> 0 then begin
   writeln(' Error in %sread in PP_SL_READ is ',ERRCODE);
   %prterr(%cserr);
   KEY_PRESSED;
end;
COMMANDS[I] := PP_I_O[1];
end;

ONE_BYTE_SETUP;
PP_I_O[1] := 3;  {Reset the address counter}
PP_I_O[2] := 3;
BUFSZ := 4;
PULSE_PACK_ARRAY;
LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
%swrite(LUN2,@PP_I_O[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,
       ERRCODE);
if ERRCODE <> 0 then begin
   writeln(' Error in %swrite in PP_SL_READ is ',ERRCODE);
   %prterr(%cserr);
   KEY_PRESSED;
end;
end;
procedure MAKE_COMMANDS;
begin
STATE_DWELL[1] := 2;
STATE_DWELL[3] := $00000000;
PNTER := 2;
repeat {until no more states need to be added}
BIT_STORE_COMMANDS;
LIST_COMMANDS;
NEW_BITS_COMMANDS;
until (STATE_DWELL[PNTER] < 0);
LOAD_COMMANDS;
end;

procedure DECODE_COMMANDS;
var
I,J,K,L : INTEGER;
START, STOP : INTEGER;
begin
%cls;
writeln (' Enter start and stop range for decoding (min=1, max=1024)');
writeln (' Note: Separate integers by a space.');
readln (START, STOP);
I := START;
repeat {until I = stop value}
%cls;
writeln;
writeln(' Amplitude Phase Pulse ');
writeln(' I A3 A2 A1 A0 90^ 180^ 1=on DT ST X1 X2 HALT');
K := 3;
while ( (I<=STOP) and (K<=23) ) do begin
BIT[1] := COMMANDS[I] and $001;
BIT[3] := COMMANDS[I] and $004;
BIT[4] := COMMANDS[I] and $008;
BIT[8] := COMMANDS[I] and $080;
BIT[9] := COMMANDS[I] and $100;
BIT[10] := COMMANDS[I] and $200;
BIT[12] := COMMANDS[I] and $800;
for L := 1 to 12 do begin
  if BIT[L] > 0 then BIT[L] := 1;
end;
gotoxy(1,K);
writeln(I);
gotoxy(6,K);
writeln(BIT[4]);
gotoxy(9,K);
writeln(BIT[3]);
gotoxy(12,K);
writeln(BIT[2]);
gotoxy(15,K);
writeln(BIT[1]);
gotoxy(20,K);
writeln(BIT[5]);
gotoxy(26,K);
writeln(BIT[6]);
gotoxy(31,K);
writeln(BIT[7]);
gotoxy(38,K);
writeln(BIT[8]);
gotoxy(42,K);
writeln(BIT[9]);
gotoxy(48,K);
writeln(BIT[10]);
gotoxy(53,K);
writeln(BIT[11]);
gotoxy(58,K);
writeln(BIT[12]);
I := I + 1;
K := K + 1;
end;
writeln('Do you want more commands printed? - Y/');
readln(ANSWER);
until(ANSWER<> 'Y');
end;
function PP_SAFETY_TEST (*: BOOLEAN*);

var
  I, TEMP1, TEMP2 : INTEGER;

begin
  GET_LUN2(LUN2);
  PP_SAFETY_TEST := FALSE; {GOOD NEWS}
  for I := 1 to 1024 do begin
    if ((COMMANDS[I] and 64) and (COMMANDS[I] and 2048)) > 0 then begin
      ONE_BYTE_SETUP;
      PP_I_0[1] := 3; {THIS WILL SETUP PULSE PROGRAMMER FOR INPUT}
      PP_I_0[2] := 3; {N(3), A(3), F(16) : RESET THE ADDRESS COUNTER T}
      PP_I_0[3] := 16; {ZERO AND HALT}
      PP_I_0[4] := 0;
      BUFSZ := 4;
      PULSE_PACK_ARRAY;
      LOTRM := 13; HITRM := 13; RECNUM := 0; BUFOF := 0; ERRCODE := 0;
      %swrite(LUN2,@PP_I_0[1],BUFOF,BUFSZ,LOTRM,HITRM,RECNUM,ERRCODE);
      if ERRCODE <> 0 then begin
        writeln(' Error in %swrite in PP_SAFTY_TEST is ',ERRCODE);
        %prterr(%cserr);
        KEY_PRESSED;
      end;
    end;
    TIMER_STOP;
    %cls;
    writeln;
    writeln("***************************************************1 ');
    writeln(" WARNING WARNING WARNING WARNING WARNING ');
    writeln;
    writeln(" BAD PULSE SEQUENCE IN PULSE PROGRAMMER ');
    writeln;
    writeln(" WARNING WARNING WARNING WARNING WARNING ');
    writeln("***************************************************1 ');
    PP_SAFETY_TEST := TRUE; {BAD NEWS}
  end;
end;
procedure PULSE_SAVE;
var
  HEADER, NAME: STRING80;
  F1: TEXT;
  I: INTEGER;
begin
  %cls;
  writeln('Enter a name for the data file (Ex:1:TEST.DAT)');
  readln(NAME);
  writeln('Enter a one line title for the pulse sequence');
  readln(HEADER);
  REWRITE(F1, NAME);
  writeln(F1, HEADER);
  for I := 1 to 1024 do begin
    writeln(F1, COMMANDS[I]);
  end;
  CLOSE(F1);
end;

procedure PULSE_READ(*var NAME, HEADER: STRING80*);
var
  F1: TEXT;
  I: INTEGER;
begin
  RESET(F1, NAME);
  readln(F1, HEADER);
  writeln(HEADER);
  for I := 1 to 1024 do begin
    readln(F1, COMMANDS[I]);
  end;
  CLOSE(F1);
end;

procedure SPIN_LOCK;
const
  POWER  = $0004; {*A2*}
  PHASE_90 = $0010; {*RF PHASE IS 90 DEGREES*}
  PULSE   = $0040;
  DT      = $0080; {*Digitizer trigger*}
  ST      = $0100; {*Scope trigger*}
X1 = $0200;
X2 = $0400;
HALT = $0800;
CLEAR = $0000;
SPARE = 1.0;

var
  TAU, JUNK : REAL;
  FIRST_PULSE, ECHO_PULSE : REAL;
  I : INTEGER;
  TIME_UNIT : LONGINT;
begin
  TIME_UNIT := $00010000;
  for I := 1 to 1024 do
    STATE_DWELL[I] := CLEAR;
    %cls;
    writeln('Procedure SPIN_LOCK');
    write('Enter duration of first pulse (us)');
    readln(FIRST_PULSE);
    write('Enter duration of an echo pulse (us)');
    readln(ECHO_PULSE);
    write('Enter tau (us)');
    readln(TAU);
    JUNK := 102.4 - (FIRST_PULSE + ECHO_PULSE + 2*TAU + SPARE);
  STATE_DWELL[1] := 12;
  STATE_DWELL[3] := trunc(TAU*10)*TIME_UNIT + CLEAR;
  STATE_DWELL[4] := TIME_UNIT + X2;
  STATE_DWELL[7] := trunc(JUNK*10)*TIME_UNIT + CLEAR;
  STATE_DWELL[8] := trunc(FIRST_PULSE*10)*TIME_UNIT + PULSE + POWER + ST;
  STATE_DWELL[9] := trunc((TAU-2)*10)*TIME_UNIT + CLEAR;
  STATE_DWELL[10] := TIME_UNIT + DT + X1;
  LOAD_COMMANDS;
end;
procedure PULSE_MENU; { Writes options for pulse programming on screen }
begin
%cls;
%gotoxy(0,1);
writeln(' PULSE PROGRAMMING MENU ');
writeln;
writeln(' 1 Create pulse sequence and load pulse programmer');
writeln(' 2 Decode pulse programmer coding');
writeln(' 3 Write coding to the pulse programmer ');
writeln(' 4 Read existing coding in the pulse programmer');
writeln(' 5 Save Pulse Sequence on disk');
writeln(' 6 Create SPIN LOCKING pulse sequence');
writeln(' 7 Read Pulse Sequence from disk');
writeln(' 24 Exit to MAINMENU');
end;

procedure PULSE_MAIN;
var
  NAME,HEADER : STRING80;
  exit : BOOLEAN;
  CHOICE : INTEGER;
begin
  exit := FALSE;
  repeat
    PULSE_MENU;
    readln (CHOICE);
    case CHOICE of
      1: MAKE_COMMANDS;
      2: DECODE_COMMANDS;
      3: PP_SL_WRITE;
      4: PP_SL_READ;
      5: PULSE_SAVE;
      6: SPIN_LOCK;
      7: begin
        %cls;
        writeln('Enter a file name (Ex 1:TEST.DAT) ');
        readln(NAME);
        PULSE_READ(NAME,HEADER);
        end;
    end;
  until (CHOICE = 24);
end;

END.
unit SAVEFID; \{ Puts the data in a format which can be read \}
\{ by the IBM software \}

\{ *********************************************************************** \}
INTERFACE
\{ \$U CPOSIL.LIB \} system_library,system_calls,
\{ \$U 1:A.OBJ \} ARRAYS;

procedure READ_KEY (var ASCII,SCANCODE : INTEGER);
procedure SAVE_FID(NUMBER_QF_SCANS: LONGINT; DELAY:REAL;
\quad SA_RATE_CODE: INTEGER;
\quad var REAL_FID,IMG_FID: LINT_ARRAY);

\{ *********************************************************************** \}
IMPLEMENTATION

const
\quad \text{B} = '\',\;
var
\quad FIDRECRD : JOBRECORD;
\quad DATA_R_I : DATARRAY;
\quad I_BLOCKS : INTEGER;
\quad TEMPFID : BLOCKFILE;
\quad REALFID : BLOCKFILE;
\quad I,J,K : INTEGER;
\quad FRACTION : INTEGER;

procedure READ_KEY (* var ASCII,SCANCODE : INTEGER *);
\quad VAR
\quad \quad KBF1,KBF2 : INTEGER;
\quad \quad ERRCODE : INTEGER;
\quad \begin{verbatim}
BEGIN
REPEAT
\%getkey(ASCII,SCANCODE,KBF1,KBF2,ERRCODE);
UNTIL(ERRCODE = 0);
END;
\end{verbatim}

procedure SAVE_FID;(*NUMBER_OF_SCANS: LONGINT; DELAY: REAL;
SA_RATE_CODE: INTEGER;
REAL_FID, IMG_FID: LINT_ARRAY*)

label 100,999,99;

var
X : INTEGER;

begin
  FIDRECRD.SOURCE := -9000;
  FIDRECRD.STATUS := 1;
  FIDRECRD.ACCESS := 0;
  FIDRECRD.SIZE := 16384;
  FIDRECRD.LOCKGAIN := 0;
  FIDRECRD.LOCKPWR := 0;
  FIDRECRD.MAGSTAT := 0;
  FIDRECRD.LEFT := 0;
  FIDRECRD.RIGHT := 0;
  FIDRECRD.TOP := 0;
  FIDRECRD.BOTTOM := 0;
  FIDRECRD.TITLE := 'A';
  FIDRECRD.FILENAME := '1:A.FID';
  FIDRECRD.OPERATOR := 'ME';

  case SA_RATE_CODE of
    0: FIDRECRD.SWPWIDTH := 20000000.0;
    1: FIDRECRD.SWPWIDTH := 10000000.0;
    2: FIDRECRD.SWPWIDTH := 5000000.0;
    3: FIDRECRD.SWPWIDTH := 2000000.0;
    4: FIDRECRD.SWPWIDTH := 1000000.0;
    5: FIDRECRD.SWPWIDTH := 500000.0;
    6: FIDRECRD.SWPWIDTH := 200000.0;
  otherwise:
    FIDRECRD.SWPWIDTH := 200000.0;
  end;

  FIDRECRD.FLTRWDTH := 500000.0;
  FIDRECRD.PULSWDTH := 1.3;
  FIDRECRD.RCVDELAY := 0.0;
  FIDRECRD.ADCDELAY := 0.0;
  FIDRECRD.RCVTIME := 0.004096;
  FIDRECRD.RLXDELAY := DELAY;
  FIDRECRD.RCVGAIN := 1;
  FIDRECRD.DIGRES := 8;
  FIDRECRD.QUAD := TRUE;
FIDRECRD.SCNSREQL := NUMBER_OF_SCANS;
FIDRECRD.SCNSDONE := FIDRECRD.SCNSREQL;
FIDRECRD.LBTOTAL := 0.0;
FIDRECRD.SPECFREQ := 15351000.0;
FIDRECRD.NUCLEUS := 'DEUTERIUM';
FIDRECRD.TEMPERATURE := 300.0;

for X := 1 to 16 do begin
  FIDRECRD.SHIMCOIL[X] := 0;
end;

FIDRECRD.REFOFF := 0.0;
FIDRECRD.SOLVENT := 'SOLID';
%gettime(FIDRECRD.YEAR,FIDRECRD.MONTH,FIDRECRD.DAY,
   FIDRECRD.HOUR,FIDRECRD.MINUTE,FIDRECRD.SECOND,FRACTION);
FIDRECRD.YEAR := 1;
FIDRECRD.MONTH := 1;
FIDRECRD.DAY := 1;
FIDRECRD.HOUR := 1;
FIDRECRD.MINUTE := 1;
FIDRECRD.SECOND := 1;
FIDRECRD.OBSOFF := 0.0;
FIDRECRD.DECOFF := 0.0;
FIDRECRD.LIMIT[1] := 1;
%cls;
writeln('This procedure will save data on a format which');
writeln('can be read by the IBM software');
writeln;
writeln('To change the value of a parameter, press the space bar');
writeln('ONCE!!! and then type in the new value of the parameter.);
writeln('If you press the space bar more than once, an IORESULT will occur');
writeln('and your data will not be written to disk.');
writeln;
writeln('To go to the next parameter, press the BACKSPACE key');
writeln;
writeln('Press ESC to continue');
I := 1;
repeat
  READ_KEY(I,J);
until(I = 27);
I := 1;
%cls;
%gotoxy(40,0);
write('ESC - to exit page');
%gotoxy(11,5);
write('Title: ',B,B,B,B,FIDRECRD.TITLE,',[Default value]);
%gotoxy(11,7);
writeln('Filename: ',B,B,B,FIDRECRD.FILENAME);
% gotoxy(11,9);
write('Operator: ',B,B,B,B,FIDRECRD.OPERATOR);
%gotoxy(11,10);
write('Nucleus: ',B,B,B,B,FIDRECRD.NUCLEUS);
%gotoxy(11,11);
write('Temperature: ',FIDRECRD.TEMPERATURE);
%gotoxy(11,12);
write('Solvent: ',B,B,B,B,FIDRECRD.SOLVENT);

repeat
%gotoxy(24,6);
repeat
READ_KEY(I,J);
if I = 32 then begin
readln(FIDRECRD.TITLE);
%gotoxy(24,6);
end;
if I = 27 then GOTO 999;
until(I = 8);
%gotoxy(24,8);
repeat
READ_KEY(I,J);
if I = 32 then begin
readln(FIDRECRD.FILENAME);
%gotoxy(24,8);
end;
if I = 27 then GOTO 999;
until(I = 8);
%gotoxy(24,9);
repeat
READ_KEY(I,J);
if I = 32 then begin
readln(FIDRECRD.OPERATOR);
%gotoxy(24,9);
end;
if I = 27 then GOTO 999;
until(I = 8);
%gotoxy(24,10);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.NUCLEUS);
  %gotoxy(24,10);
end;
if I = 27 then GOTO 999;
until(I = 8);
%gotoxy(24,11);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.TEMPERATURE);
  %gotoxy(24,11);
end;
if I = 27 then GOTO 999;
until(I = 8);
%gotoxy(24,12);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.SOLVENT);
  %gotoxy(24,12);
end;
if I = 27 then GOTO 999;
until(I = 8);
until(K = 86);
999:%cls;
%gotoxy(40,0);
write('ESC-to exit page2 ');
%gotoxy(11,5);
write('Sweepwidth in Hz: ',B,B,B,B,B,B,B,B,B,B,B,FIDRECRD.SWPWIDTH);
%gotoxy(11,6);
write('Filterwidth in Hz: ',B,B,B,B,B,B,B,B,B,B,B,FIDRECRD.FLTRWDTH);
%gotoxy(11,7);
write('Pulsewidth in uS: ',FIDRECRD.PULSWIDTH);
%gotoxy(11,8);
write('Receiver delay in uS: ',FIDRECRD.RCVDELAY);
%gotoxy(11,9);
write('ADC delay in uS: ',FIDRECRD.ADCDELAY);
%gotoxy(11,10);
write('Receiver on time [.004096S]: ',FIDRECRD.RCVTIME);
%gotoxy(11,11);
write('Relaxation delay in S: ',FIDRECRD.RLXDELAY);
%gotoxy(11,12);
write('Receiver gain: ',FIDRECRD.RCVGAIN);
%gotoxy(11,13);
write('Digitizer resolution: ',FIDRECRD.DIGRES);
%gotoxy(11,14);
write('Scans done: ',FIDRECRD.SCNSREQD);
FIDRECRD.SPECFREQ := FIDRECRD.SPECFREQ/1.0E6;
%gotoxy(11,15);
write('Spectral frequency in MHz: ',FIDRECRD.SPECFREQ);
FIDRECRD.SPECFREQ := FIDRECRD.SPECFREQ*1.0E6;

repeat
%gotoxy(40,5);
repeat
READ_KEY(I,J);
  if I = 32 then begin
    readln(FIDRECRD.SWPWIDTH);
    %gotoxy(40,5);
  end;
  if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,6);
repeat
READ_KEY(I,J);
  if I = 32 then begin
    readln(FIDRECRD.FLTRWDTH);
    %gotoxy(40,6);
  end;
  if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,7);
repeat
   READ_KEY(I,J);
   if I = 32 then begin
      readln(FIDRECRD.PULSWIDTH);
      %gotoxy(40,7);
      end;
   if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,8);
repeat
   READ_KEY(I,J);
   if I = 32 then begin
      readln(FIDRECRD.RCVDELAY);
      %gotoxy(40,8);
      end;
   if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,9);
repeat
   READ_KEY(I,J);
   if I = 32 then begin
      readln(FIDRECRD.ADCDELAY);
      %gotoxy(40,9);
      end;
   if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,10);
repeat
   READ_KEY(I,J);
   if I = 32 then begin
      readln(FIDRECRD.RCVTIME);
      %gotoxy(40,10);
      end;
   if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,11);
repeat
   READ_KEY(I,J);
   if I = 32 then begin

readln(FIDRECRD.RLXDELAY);
%gotoxy(40,11);
end;
if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,12);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.RCVGAIN);
  %gotoxy(40,12);
end;
if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,13);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.DIGRES);
  %gotoxy(40,13);
end;
if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,14);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.SCNSREQD);
  %gotoxy(40,14);
end;
if I = 27 then GOTO 99;
until(I = 8);
%gotoxy(40,15);
repeat
READ_KEY(I,J);
if I = 32 then begin
  readln(FIDRECRD.SPECFREQ);
  FIDRECRD.SPECFREQ := FIDRECRD.SPECFREQ*1.0E6;
  %gotoxy(40,15);
end;
if I = 27 then GOTO 99;
until(I = 8);
until(K = 86);
99:%cls;
writeln;
writeln('To write to disk, press the BACKSPACE key');
writeln;
writeln('Press ESC to escape without writing to disk');
writeln;
I := 1;
repeat
  READ_KEY(I,J);
  if I = 27 then GOTO 100;
until(I = 8);
I := 1;
REWRITE(REALFID,FIDRECRD.FILENAME);
I_BLOCKS := BLOCKWRITE(REALFID,FIDRECRD,2);
writeln('The number of blocks written is ',I_BLOCKS);
I_BLOCKS := BLOCKWRITE(REALFID,REAL_FID,64);
writeln('The number of blocks written is ',I_BLOCKS);
I_BLOCKS := BLOCKWRITE(REALFID,IMG_FID,64);
writeln('The number of blocks written is *,I_BLOCKS);
CLOSE(REALFID);
100:end;
end.
program ZERO(INPUT,OUTPUT); { UPDATE: 24 SEP 1987}
USES {$U CPOSIL.LIB} system_library,SYSTEM_CALLS,IEEE488,cserr,
        PRINT,GENERIC_PRINTER,
        {$U 1:A.OBJ} ARRAYS,
        {$U 1:S.OBJ} SAVEFID,
        {$U 1:CMLB.OBJ} CAMLIB,
        {$U 1:CPLB.OBJ} CAPLIB,
        {$U 1:D.OBJ} DMMLIB,
        {$U 1:P.OBJ} PULSE;
        {* 1:ARRAYS.OBJ ARRAYS,
         1:SAVEFID.OBJ SAVEFID,
         1:CAMLIB.OBJ CAMLIB,
         1:CAPLIB.OBJ CAPLIB,
         1:DMMLIB.OBJ DMMLIB,
         1:PULSE.OBJ PULSE;*}

        type Data_array = array [1..500] of real;
        S_string = packed array [1..40] of char;
        Pointer = ^Data_array;

        var
        FID_REAL  : LINT_ARRAY;
        FID_REAL_PTR : LPOINTER;
        FID_IMG   : LINT_ARRAY;
        FID_IMG_PTR : LPOINTER;
        NOISE_REAL : LINT_ARRAY;
        NOISE_R_PTR : LPOINTER;
        NOISE_IMG : LINT_ARRAY;
        NOISE_I_PTR : LPOINTER;
        PLOTS_DATA : Data_array;
        PLOTN_DATA : Data_array;
        PLOTSN_DATA : Data_array;
        main_exit : BOOLEAN;
        main_choice,ioerr : INTEGER;
        S,N,D,A : CHAR;
        SCANS : LONGINT;
        SA_RATE_CODE : INTEGER;
        A11,SCNCDE,K1,K2,ERR : INTEGER;
        SA_RECORD_LENGTH : INTEGER;
        DELAY : REAL;
ZERO_DELAY : REAL;
ZERO_TITLE : STR;
VIB_DELAY : REAL;
PR : TEXT;
DATA_ALL : ARRAY[1..10, 1..500] OF REAL;
X_POINTS : INTEGER;
FREQ : REAL;
FREQ_START : REAL;
FREQ_STOP : REAL;
FREQ_INC : REAL;
NUMBER_OF_INCREMENT : INTEGER;
H1_VOLTS : REAL;
H1_OBTAINED : REAL;
INT_TOTAL : REAL;
INT_ONE : REAL;
INT_TWO : REAL;
INT_THREE : REAL;
INT_FOUR : REAL;
INT_FIVE : REAL;
INT_SIX : REAL;
INT_SEVEN : REAL;
INT_EIGHT : REAL;
CHK_INT_ONE : REAL;
CHK_INT_TWO : REAL;
TITLE : STRING80;
DEV1 : STRING8;
LUN1 : INTEGER;
SA_SLOW_READ : BOOLEAN;

procedure GRINIT(var LUN,EERR : longint); External;

procedure GRPLOT(var N : longint; var X,Y : Data_array;
    var LINE,ICNTRL,IERR : longint); External;

procedure GRSMBL(var N : longint; var X,Y : Data_array;
    var ISYM : longint; var XMAG : real;
    var IERR : longint); External;

procedure GRLABL(var Label_string : S_tring; Length_of_String : integer;
    var LOC,IERR : longint); External;
procedure GRCLOS(var IER R : longint); External;

procedure CHECK_ERROR(var IERR : longint);
begin
  if IERR <> 0 then begin
    writeln('Error ',IERR:6,' in graphics driver call');
    %prterr(%cserr);
    HALT;
  end;
end;

procedure PLOT_XY(Plt_pointer: Pointer; SELECTOR: char);
var
  Plt_array : Data_array;
  Point_array: Data_array;
  un1,icntrl,ierr,isym,line,n,mag,loc : longint;
  strng : S_string;
  xmag : real;
  I : integer;
begin
  for I := 1 to 500 do begin
    Plt_array[I] := 0;
    Point_array[I] := 0;
  end;
  line := 1; isym := 6; icntrl := 0; xmag := 0.05; un1 := 20;
  n := 256;
  for I := 1 to n do begin
    Point_array[I] := I;
    Plt_array[I] := Plt_pointer^I;
  end;
  GRINIT(un1,ierr);
  CHECK_ERROR(ierr);
  GRPLOT(n,Point_array,Plt_array,line,icntrl,ierr);
  CHECK_ERROR(ierr);
  GRSMBL(n,Point_array,Plt_array,isym,xmag,ierr);
  CHECK_ERROR(ierr);
  strng := ' POINTS PLOTTED ' ;
  loc := 1;
  GRLABL(strng,40,loc,ierr);
  CHECK_ERROR(ierr);
strng := 'RELATIVE INTENSITY';
loc := 2;
GRLABL(strng,40,loc,ierr);
CHECK_ERROR(ierr);
case SELECTOR of
'A': strng := 'FID_REAL AND FID_IMG';
'S': strng := 'FID_REAL';
'N': strng := 'NOISE_FID_REAL';
'D': strng := 'FID_REAL - NOISE_FID_REAL';
end;
loc := 0;
GRLABL(strng,40,loc,ierr);
CHECK_ERROR(ierr);
GRCLOS(ierr);
CHECK_ERROR(ierr);
end;

procedure LOAD_PLOT_ARRAYS;
var
I : integer;
begin
for I := 1 to 128 do begin {*The number of echoes*}
PLOTS_DATA[I] := FID_REAL[I];
PLOTS_DATA[I+128] := FID_IMG[I];
end;
end;

procedure CHANGE_PRINTER_COLOR;
var
VOLUMN : STRING8;
BIO : LONGINT;
IPacket : array [1 .. 10] of integer;
Packet : packed array [1 .. 3] of char;
ercode,ioerr: integer;
lun,pen,printype : integer;
BUFOF : LONGINT;
begin
writeln('What type of printer you are using?
');
writeln(' 1=integral printer, 2=5182 color printer, 3=No Change. ');
read(printype);
case prctype of
  1: begin
    lun := 144;
    VOLUMN := '#PR';
    BIO := 0; errcode := 0;
    writeln('What color would you like the printer to use?');
    writeln('0=red,1=green,2=blue,3=black');
    readln(pen);
    %sysopen(lun, VOLUMN, 0, 0, 0, NIL, BIO, errcode);
    if errcode <> 0 then writeln(' errcode in opening = ', errcode);
    IPacket[1] := 6;
    IPacket[3] := 0;
    %sysfunc(lun, @IPacket[1], errcode);
    if errcode <> 0 then writeln(' errcode in %sysfunc = ', errcode);
    %sysclose(lun, NIL, errcode);
    if errcode <> 0 then writeln(' errcode in closing = ', errcode);
    writeln(PR, ' Set new ribbon position. ');
  end;
  2: begin
    reset(PR, '#PR');
    if IORESULT <> 0 then begin
      ioerr := IORESULT;
      writeln(' IORESULT in reset #PR is ', ioerr);
      rewrite(PR, '#PR');
      if IORESULT <> 0 then begin
        ioerr := IORESULT;
        writeln(' IORESULT in rewrite #PR is ', ioerr);
      end;
    end;
    lun := %getlun(@PR);
    writeln('What color would you like the printer to use?');
    writeln('0=yellow,1=red,2=blue,3=black');
    readln(pen);
    %setfiltr(PR, 1);
    Packet[1] := CHR(27);
    if (pen = 0) then Packet[2] := CHR(121);
    if (pen = 1) then Packet[2] := CHR(109);
    if (pen = 2) then Packet[2] := CHR(99);
    if (pen = 3) then Packet[2] := CHR(98);
Packet[3] := CHR(13);
BUFOF := 0;
%swrite(lun,@Packet,BUFOF,3,13,13,0,errcode);
%setfilpr(PR,0);
writeln(PR,' Set new ribbon position. ');
CLOSE(PR);
end;
3: writeln;
end;
{$I+}
end;

procedure PAUSE(WAIT:INTEGER); { waiting for X of second }
var
  YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC : INTEGER;
  START, STOP : INTEGER;
begin
  %gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
  START := SECOND;
  if START < (60 - WAIT) then begin
    repeat
      %gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
      STOP := SECOND;
    until (STOP - START) >= WAIT;
  end
  else begin
    START := START - 60;
    repeat
      %gettime(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,FRAC);
      if (SECOND + WAIT) >= 60 then STOP := SECOND - 60 else STOP := SECOND;
    until (STOP - START) >= WAIT;
  end;
end;

procedure FINAL_ZERO_TIMER_STOP;
begin
  SAMPLE_POSITION_RESET; PAUSE(2);
  TIMER_CH_SETUP(1,2,1,0,0,0,0,0); PAUSE(2);
  TIMER_CS(0,3,0); PAUSE(2);
  TIMER_EXE; PAUSE(4); IFC_CAMAC;
procedure ZERO_RF_CHECK;
var
  X : REAL;
begin
  DMM_VDCSLOW(X);
  if X < 0.001 then FREQ_STROBE;
end;

procedure ZERO_PLOT_ANY(var INTEGRAL_SELECTED : INTEGER);
var
  un1,icntrl,ierr,isym,line,n,mag,loc : longint;
  x,y : Data_array;
  strng : String;
  xmag : real;
  I : INTEGER;
  NUMBER_OF_POINTS : INTEGER;
begin
  line := 1; isym := 1; icntrl := 0; xmag := 0.5; un1 := 20;
  NUMBER_OF_POINTS := 0;
  for I := 1 to 500 do begin
    if DATA_ALL[1,I] > 0.0 then NUMBER_OF_POINTS := NUMBER_OF_POINTS +1;
    if DATA_ALL[1,I] > 0.0 then x[I] := DATA_ALL[1,I];
  end;
  n := NUMBER_OF_POINTS;
  for I := 1 to NUMBER_OF_POINTS do
    y[I] := DATA_ALL[1+INTEGRAL_SELECTED,I];
    GRINIT(un1,ierr);
    CHECK_ERROR(ierr);
    GRPLOT(n,x,y,line,icntrl,ierr);
    CHECK_ERROR(ierr);
    GRSMBL(n,x,y,isym,xmag,ierr);
    CHECK_ERROR(ierr);
    strng := 'SEARCH FREQUENCY (KHZ) '; loc := 1;
    GRLABL(strng,40,loc,ierr);
    CHECK_ERROR(ierr);
    strng := 'RECOVERED MAGNITIZATION (ARB. UNITS) ';

loc := 2;
GRLABL(strng,40,loc,ierr);
CHECK_ERROR(ierr);
case INTEGRAL_SELECTED of
  1: loc := 0;
  2: loc := 0;
  3: loc := 0;
  4: loc := 0;
  5: loc := 0;
  6: loc := 0;
  7: loc := 0;
  8: loc := 0;
end;
loc := 0;
GRLABL(strng,40,loc,ierr);
CHECK_ERROR(ierr);
GRCLOS(ierr);
CHECK_ERROR(ierr);
%gotoxy(0,0);
case INTEGRAL_SELECTED of
  1: writeln('Integral ONE');
  2: writeln('Integral TWO');
  3: writeln('Integral THREE');
  4: writeln('Integral FOUR');
  5: writeln('Integral FIVE');
  6: writeln('Integral SIX');
  7: writeln('Integral SEVEN');
  8: writeln('Integral EIGHT');
end;

procedure ZERO_DATA_ALL;
var
  I : INTEGER;
begin
  for I := 1 to 500 do begin
    DATA_ALL[1,I] := 0.0;
    DATA_ALL[2,I] := 0.0;
    DATA_ALL[3,I] := 0.0;
    DATA_ALL[4,I] := 0.0;
  end;
end;
DATA_ALL[5,I] := 0.0;
DATA_ALL[6,I] := 0.0;
DATA_ALL[7,I] := 0.0;
DATA_ALL[8,I] := 0.0;
DATA_ALL[9,I] := 0.0;
DATA_ALL[10,I] := 0.0;
FID_REAL[I] := 0;
FID_IMG[I] := 0;
NOISE_REAL[I] := 0;
NOISE_IMG[I] := 0;
PLOTS_DATA[I] := 0;
PLOTN_DATA[I] := 0;
PLOTSN_DATA[I] := 0;
end;
end;

procedure ZERO_INTEGRATE; {*This will integrate Signal Averager data*}
const
NPOINTS = 128; { *Total number of points in each data array*}
var
I,J,HF_NPOINTS : INTEGER;
SUM_REAL,SUM_IMG : REAL;
DELTA_REAL,DELTA_IMG : REAL;
FIRST_REAL,FIRST_IMG : REAL;
SECOND_REAL,SECOND_IMG : REAL;
begin
INT_ONE := 0.0; INT_TWO := 0.0; INT_THREE := 0.0;
INT_FOUR := 0.0; INT_FIVE := 0.0; INT_SIX := 0.0;
INT_SEVEN := 0.0; INT_EIGHT := 0.0;
SUM_REAL := 0.0; SUM_IMG := 0.0;
DELTA_REAL := 0.0; DELTA_IMG := 0.0;
FIRST_REAL := 0.0; FIRST_IMG := 0.0;
SECOND_REAL := 0.0; SECOND_IMG := 0.0;
{ HF_NPOINTS := NPOINTS div 2; }
HF_NPOINTS := 100;
for I := (1 + HF_NPOINTS) to (NPOINTS-3) do begin
  SECOND_REAL := FID_REAL[I] + SECOND_REAL;
  SECOND_IMG := FID_IMG[I] + SECOND_IMG;
end;
SECOND_REAL := SECOND_REAL / (NPOINTS - HF_NPOINTS-3);
SECOND_IMG := SECOND_IMG / (NPOINTS - HF_NPOINTS-3);
for I := 1 to NPOINTS do begin
    FIRST_REAL  := FID_REAL[I] + FIRST_REAL;
    FIRST_IMG   := FID_IMG[I] + FIRST_IMG;
    DELTA_REAL  := (FID_REAL[I] - SECOND_REAL) + DELTA_REAL;
    DELTA_IMG   := (FID_IMG[I] - SECOND_IMG) + DELTA_IMG;
end;
INT_ONE := DELTA_REAL;
INT_TWO := DELTA_IMG;
INT_THREE := ABS(DELTA_REAL) + ABS(DELTA_IMG);
INT_FOUR := SQRT(SQR(DELTA_REAL) + SQR(DELTA_IMG));
INT_FIVE := ABS(FIRST_REAL) + ABS(FIRST_IMG);
INT_SIX := SQRT(SQR(FIRST_REAL) + SQR(FIRST_IMG));
INT_SEVEN := FIRST_REAL;
INT_EIGHT := FIRST_IMG;
INT_ONE := INT_ONE/SCANS;  INT_TWO := INT_TWO/SCANS;
INT_THREE := INT_THREE/SCANS;  INT_FOUR := INT_FOUR/SCANS;
INT_FIVE := INT_FIVE/SCANS;  INT_SIX := INT_SIX/SCANS;
INT_SEVEN := INT_SEVEN/SCANS;  INT_EIGHT := INT_EIGHT/SCANS;
DATA_ALL[1,X_POINTS] := FREQ;  {Save current frequency}
DATA_ALL[2,X_POINTS] := INT_ONE + DATA_ALL[2,X_POINTS];
DATA_ALL[3,X_POINTS] := INT_TWO + DATA_ALL[3,X_POINTS];
DATA_ALL[4,X_POINTS] := INT_THREE + DATA_ALL[4,X_POINTS];
DATA_ALL[5,X_POINTS] := INT_FOUR + DATA_ALL[5,X_POINTS];
DATA_ALL[6,X_POINTS] := INT_FIVE + DATA_ALL[6,X_POINTS];
DATA_ALL[7,X_POINTS] := INT_SIX + DATA_ALL[7,X_POINTS];
DATA_ALL[8,X_POINTS] := INT_SEVEN + DATA_ALL[8,X_POINTS];
DATA_ALL[9,X_POINTS] := INT_EIGHT + DATA_ALL[9,X_POINTS];
DATA_ALL[10,X_POINTS] := INT_ONE + INT_TWO + INT_THREE + 
                          INT_FOUR + DATA_ALL[10,X_POINTS];
writeln('Frequency = ',FREQ:7:2);
writeln('Integral(one) DR = INT_ONE);
writeln('Integral(two) DI = INT_TWO);
writeln('Integral(three) DR + DI = INT_THREE);
writeln('Integral(four) SQRT(DR^2+DI^2) = INT_FOUR);
writeln('Integral(five) SR + SI = INT_FIVE);
writeln('Integral(six) SQRT(SR^2+SI^2) = INT_SIX);
writeln('Integral(seven) SR = INT_SEVEN);
writeln('Integral(eight) SI = INT_EIGHT);
procedure ZERO_TIMES_TO_PRINTER;  {*Cooper IBM book, p144*}
var
  YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRAC : INTEGER;
begin
  %gettime(YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, FRAC);
  writeln(PR, 'Date is ', MONTH:2, '-', DAY:2, '-', YEAR:2);
  writeln(PR, 'Time is ', HOUR:2, ':', MINUTE:2, ':', SECOND:2);
end;

procedure ZERO_T_PARAMETERS;
var
  ANSWER: CHAR;
begin
  write('Relaxation Delay(s) [real number]?');
  readln(Delay);
  writeln('Current ZERO_DELAY and VIB_DELAY are ', ZERO_DELAY:5:2,
          ', VIB_DELAY:5:2, ' secs');
  write('Do you want to change these values? [Y/N (default)]?');
  readln(ANSWER);
  if ((ANSWER = 'Y') or (ANSWER = 'y')) then begin
    writeln('Values can range from 0.0 seconds up.');
    write('Enter ZERO_DELAY '); readln(ZERO_DELAY);
    write('Enter VIB_DELAY '); readln(VIB_DELAY);
  end;
  CHK_INT_ONE := 1E11; CHK_INT_TWO := 1E11;
  write('Do you wish to change values for checking integrals 1 & 2 [Y/N (default)]?');
  readln(ANSWER);
  if ((ANSWER = 'Y') or (ANSWER = 'y')) then begin
    write('Enter checking value for integral one'); readln(CHK_INT_ONE);
    write('Enter checking value for integral two'); readln(CHK_INT_TWO);
  end;
  SA_SLOW_READ := TRUE;
  write('Use DMA method to read signal averager data? [Y/N (default)]?');
  readln(ANSWER);
  if ((ANSWER = 'Y') or (ANSWER = 'y')) then begin
    writeln('Procedure SA_READ[Slot] will be used.');
  end;

procedure SET_TWO_FREQ; {*This will set frequency synthesizer to*}  
{"two frequencies*}  
label 999,100;  
var  
CH1,CH2,CH3,CH4 : LONGINT;  
FREQ_ONE,FREQ_TWO : real;  
DONE_8 : BOOLEAN;  
DONE_10 : BOOLEAN;  
FOREVER : BOOLEAN;  
ASCII,SCANCODE,KBF1,KBF2,ERRCODE : INTEGER;  
begin  
%cls;  
ZERO_DATA_ALL;  
writeln('This program will set the freq synthesizer for 2 frequencies');  
writeln('To exit this procedure hit ECS');  
writeln('What is the sample being run');  
readln(TITLE);  
writeln(' What is the first frequency?');  
readln(FREQ_ONE);  
writeln(' What is the second frequency?');  
readln(FREQ_TWO);  
writeln('What value of HI do you wish for coil?[typically 0.025V]');  
readln(H1_VOLTS);  
writeln(PR,' SET_TWO_FREQ PROCEDURE ');
writeln(PR,' TITLE);
ZERO_DELAY := 2.0; VIB_DELAY := 1.5;
ZERO_T_PARAMETERS;
writeln(PR,' Frequency will equal ',FREQ_ONE,' and ',FREQ_TWO);
FREQ := FREQ_ONE; X_POINTS := 1; ZERO_DATA_ALL;
FOREVER := FALSE;
repeat
  FREQ_SET(FREQ); FREQ_SET(FREQ);
  100: CAP_TUNE(H1_VOLTS,H1_OBTAINED,FREQ);
      PAUSE(2); SATURATE(20);
      TIMER_CS(1,3,1); CH1 := round(10*DELAY);
      CH2 := round(10.0*ZERO_DELAY); CH3 := round(10.0*VIB_DELAY);
      TIMER_CH_SETUP(CH1,CH2,CH3,0,0,0,0,0); TIMER_EXE;
      SA_SETUP(8,SA_RECORD_LENGTH,0,7);
      SA_SETUP(10,SA_RECORD_LENGTH,0,7);
      SA_SWEEP(8,SCANS); SA_SWEEP(10,SCANS);
      SA_RESET(8); SA_RESET(10); PAUSE(2);
      SA_RESET(8); SA_RESET(10); PAUSE(2);
      SA_SET_LAM_WHEN_DONE(8);
      SA_SET_LAM_WHEN_DONE(10);
      gotoxy(40,52);
  writeln('Sample being run is ',TITLE);
  writeln('Frequency = ',FREQ:7:2);
  writeln('Integral(one) DR = ',INT_ONE);
  writeln('Integral(two) DI = ',INT_TWO);
  writeln('Integral(three) IDRI + IDII = ',INT_THREE);
  writeln('Integral(four) SQRT(DRA^2+DIA^2) = ',INT_FOUR);
  writeln('Integral(five) ISRI + ISII = ',INT_FIVE);
  writeln('Integral(six) SRA^2 + SIA^2 = ',INT_SIX);
  writeln('Integral(seven) SR = ',INT_SEVEN);
  writeln('Integral(eight) SI = ',INT_EIGHT);
DONE_8 := FALSE;
DONE_10 := FALSE;
repeat
  SA_DONE(8,DONE_8);
  SA_DONE(10,DONE_10);
  ZERO_RF_CHECK;
  %getkey(ASCII,SCancode,KBF1,KBF2,ERRCODE);
  if ASCII = 27 then goto 999;
  if ASCII = 1 then %crtpage(0);
if ASCII = 2 then %crtpage(1)
until( (DONE_8 = TRUE) or (DONE_10 = TRUE));
%crtpage(0);   %cls;
TIMER_STOP; writeln('Issued TIMER_STOP');
writeln('SIGNALS: DONE_8 = ', DONE_8, ', DONE_10 = ',DONE_10);
if SA_SLOW_READ = TRUE then begin
    writeln('Issuing SA_SREAD[8]');
    SA_SREAD(FID_REAL_PTR,8,SA_RECORD_LENGTH);
    writeln('Issuing SA_SREAD[10]');
    SA_SREAD(FID_IMG_PTR,10,SA_RECORD_LENGTH);
end
else begin
    SA_READ(8,SA_RECORD_LENGTH);
    UNPCK_DMA_ARRAY(FID_REAL_PTR,SA_RECORD_LENGTH);
    SA_READ(10,SA_RECORD_LENGTH);
    UNPCK_DMA_ARRAY(FID_IMG_PTR,SA_RECORD_LENGTH);
end;
SAMPLE_POSITION_RESET;
ZERO_INTEGRATE;   X_POINTS := X_POINTS + 1;
LOAD_PLOT_ARRAYS;
%crtpage(1);
PLOT_XY(@PLOTS_DATA,'A');
%crtpage(0);
write(PR,'Freq ',FREQ:7:2);
write(PR,' ',INT_ONE:9:2);
write(PR,' ',INT_TWO:9:2);
write(PR,' ',INT_FOUR:12:2);
write(PR,' ',INT_FIVE:15:2);
write(PR,' ',H1_OBTAINED:9:4,'V');
writeln(PR);
TIMER_STOP;
if FREQ = FREQ_ONE then FREQ := FREQ_TWO
else FREQ := FREQ_ONE;
until FOREVER;
999: TIMER_STOP;
%crtpage(0);
FINAL.ZERO_TIMER_STOP;
DAC_SETUP(0,0,0);
IFC_CAMAC;
end;
procedure SAVE_TO_DISK;
label 50;
var
  F1 : TEXT;
  I,ioerr : INTEGER;
  counter : integer;
begin
  counter := 1 + TRUNC( (FREQ_STOP - FREQ_START) / FREQ_INC );
  REWRITE(F1,'1:TEMP.DAT');
  if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in reset 1:TEMP.DAT is \',ioerr);
    goto 50;
  end;
  for I := 1 to counter do begin
    writeln(F1,DATA_ALL[1,I],',',DATA_ALL[2,I],',',
            DATA_ALL[3,I],',',DATA_ALL[4,I]);
    writeln(F1,DATA_ALL[5,I],',',DATA_ALL[6,I],',',
            DATA_ALL[7,I],',',DATA_ALL[8,I]);
    writeln(F1,DATA_ALL[9,I],',',DATA_ALL[10,I]);
  end;
  CLOSE(F1);
50: end; {$I+}

procedure ZERO_FIELD_CYCLING; { *This is main program for zero-field*}
label 999,100;
var
  CH1,CH2,CH3,CH4 : LONGINT;
  DONE_8     : BOOLEAN;
  DONE_10    : BOOLEAN;
  ASCII,SCANCODE,KBF1,KBF2,ERRCODE,Z : INTEGER;
  LOOPS,LOOPS_DONE : INTEGER;
  INTEGRAL_SELECTED : INTEGER;
  m,n,k     : integer;
begin
  %cls;
  ZERO_DATA_ALL;
  LOOPS := 0; m := 0; n := 2;
  while LOOPS >= 0 do begin
LOOPS := LOOPS + 1; m := m + 1;
k := m div n;
if k = 0 then begin
  SAVE_TO_DISK;
n := n + 2;
end;
X_POINTS := 0; FREQ := FREQ_START; FREQ_SET(FREQ);
FREQ_SET(FREQ);
DAC_SETUP(0,0.0);
while FREQ <= FREQ_STOP do begin
  X_POINTS := X_POINTS + 1;
  100: CAP_TUNE(H1_VOLTS,H1_OBTAINED,FREQ);
  PAUSE(2); SATURATE(20);
  TIMER_CS(1,3,1); CH1 := round(10*DELAY);
  CH2 := round(10.0*ZERO_DELAY); CH3 := round(10.0*VIB_DELAY);
  TIMER_CH_SETUP(CH1,CH2,CH3,0,0,0,0,0); TIMER_EXE;
  SA_SETUP(8,SA_RECORD_LENGTH,0,7);
  SA_SETUP(10,SA_RECORD_LENGTH,0,7);
  SA_SWEEP(8,SCANS); SA_SWEEP(10,SCANS);
  SA_RESET(8); SA_RESET(10);
  SA_SET_LAM_WHEN_DONE(8); SA_SET_LAM_WHEN_DONE(10);
  if ((LOOPS>l) OR (LOOPS=-1) OR (FREQ>FREQ_START)) then begin
    INTEGRAL_SELECTED := 4;
%crtpage(l);
ZERO_PLOT_ANY(INTEGRAL_SELECTED);
%gotoxy(40,52);
writeln('Loops = ',LOOPS);
%gotoxy(40,53);
writeln('Frequency = ',FREQ:7:2);
writeln('Integral(one) DR = ',INT_ONE);
writeln('Integral(two) DI = ',INT_TWO);
writeln('Integral(three) |DR| + |DI| = ',INT_THREE);
writeln('Integral(four) SQRT(DR^2+DI^2) = ',INT_FOUR);
writeln('Integral(five) |SR| + |SI| = ',INT_FIVE);
writeln('Integral(six) SR^2 + SI^2 = ',INT_SIX);
writeln('Integral(seven) SR = ',INT_SEVEN);
writeln('Integral(eight) SI = ',INT_EIGHT);
writeln('Sample being run is ',TITLE);
writeln(Data is saved every 2 loops in file 1:TEMP.DAT);
  end;
repeat
SA_DONE(8,DONE_8); SA_DONE(10,DONE_10);
ZERO_RF_CHECK;
%getkey(ASCII,SCANCODE,KBF1,KBF2,ERRCODE);
if ASCII = 11 then goto 999;
if ASCII = 1 then %crtpage(0);
if ASCII = 2 then %crtpage(1);
if ASCII = 27 then begin
  if LOOPS > 0 then LOOPS_DONE := LOOPS;
  LOOPS := -1;
end;
until ((DONE_8 = TRUE) or (DONE_10 = TRUE));
%crtpage(0); %cls;
TIMER_STOP; writeln('Issued TIMER_STOP');
writeln('SIGNALS: DONE_8 = ', DONE_8, ', DONE_10 = ',DONE_10);
if SA_SLOW_READ = TRUE then begin
  writeln('Issuing SA_SREAD[8]');
  SA_SREAD(FJD_REAL_PTR,8,SA_RECORD_LENGTH);
  writeln('Issuing SA_SREAD[10]');
  SA_SREAD(FID_IMG_PTR,10,SA_RECORD_LENGTH);
end
else begin
  SA_READ(8,SA_RECORD_LENGTH);
  UNPCK_DMA_ARRAY(FID_REAL_PTR,SA_RECORD_LENGTH);
  SA_READ(10,SA_RECORD_LENGTH);
  UNPCK_DMA_ARRAY(FID_IMG_PTR,SA_RECORD_LENGTH);
end;
SAMPLE_POSITION_RESET;
ZERO_INTEGRATE;
LOAD_PLOTARRAYS;
%crtpage(1);
PLOT_XY(@PLOTS_DATA,'A');
%crtpage(0);
if CHK_INT_ONE < abs(INT_ONE) then goto 100;
if CHK_INT_TWO < abs(INT_TWO) then goto 100;
if ((LOOPS = 1) OR (LOOPS = -1)) then begin
  write(PR,'Freq ',FREQ:7:2);
  write(PR,' ',INT_ONE:9:2);
  write(PR,' ',INT_TWO:9:2);
  write(PR,' ',INT_FOUR:12:2);
  write(PR,' ',INT_FIVE:15:2);
write(PR,' \text{H1\_OBTAINED:9:4}',V');
writeln(PR); end;  
\{\text{END IF}\}
FREQ := FREQ + FREQ\_INC; FREQ\_SET(FREQ); FREQ\_SET(FREQ);
end;  
\{\text{** End of the Cycle **}\}
end;  
\{\text{** End of LOOPS Cycle **}\}
999: \text{T\_L\_STOP} ;
\text{\%crtpage}(0);
\text{FINAL\_ZERO\_T\_STOP} ;
\text{DAC\_SETUP}(0,0.0);
\text{IFC\_CAMAC} ;
\text{writeln}(PR,'\text{Number of Loops = ',LOOPS\_DONE});
end;

\text{procedure} \text{ZERO\_SPIN\_LATTICE} ;
\text{label} 100,999;
\text{var}
CH1,CH2,CH3,CH4 : \text{LONGINT} ;
DONE\_8,DONE\_10 : \text{BOOLEAN} ;
ASCII,SCANCODE,KBF1,KBF2,ERRCODE : \text{INTEGER} ;
delay\_counter,increment : \text{integer} ;
T1\_MAX : \text{REAL} ;
\text{begin}
\%cls;
\text{writeln(' This procedure will measure T1. ')} ;
\text{writeln(' For the first part, disconnect the Samp. Post. coax.' ) ;}
\text{writeln(' NOTE: You must supply the name of the sample you are running');}
\text{writeln(' Name of the sample you are running?') ;}
\text{readln(TITLE) ;}
\text{writeln(PR,' T1 in High Field for ',TITLE)} ;
\text{writeln(PR,' Note: Experiment is done without rf irradiation. ')} ;
\text{writeln(PR,' The integrals printed are INT\_1,INT\_2,INT\_4, and INT\_5') ;}
\text{writeln(' What is the maximum relaxation delay? ')} ;
\text{ZERO\_DELAY} := 0.0 ;
\text{VIB\_DELAY} := 0.0 ;
\text{ZERO\_T\_PARAMETERS} ;
\text{T1\_MAX} := DELAY ;
\text{X\_POINTS} := 0; \text{ZERO\_DATA\_ALL} ;
\text{SAMPLE\_POSITION\_RESET} ;
FREQ := 0.0;
DELAY := 1.0;
%cls;
while DELAY <= T1_MAX do begin
   X_POINTS := X_POINTS + 1;
100: PAUSE(2); SATURATE(20);
   TIMER_CS(1,3,1); CH1 := round(10*DELAY); CH2 := 1; CH3 := 1;
   TIMER_CH_SETUP(CH1,CH2,CH3,0,0,0,0,0); TIMER_EXE;
   FREQ_SET(300.0); FREQ_SET(300.0); DAC_SETUP(0,0,0);
   SA_SETUP(8,SA_RECORD_LENGTH,0,7);
   SA_SETUP(10,SA_RECORD_LENGTH,0,7);
   SA_SWEEP(8,SCANS); SA_SWEEP(10,SCANS);
   SA_RESET(8); SA_RESET(10); PAUSE(2);
   SA_RESET(8); SA_RESET(10); PAUSE(2);
   SA_SET_LAM_WHEN_DONE(8);
   SA_SET_LAM_WHEN_DONE(10);
   %crtpage(0); %cls;
   %gotoxy(40,52);
   writeln('Sample being run is ',TITLE);
   %gotoxy(40,53);
   writeln('Delay = ',DELAY);
   writeln('Integral(one) DR = ',INT_ONE);
   writeln('Integral(two) DI = ',INT_TWO);
   writeln('Integral(three) IDRI + IDII = ',INT_THREE);
   writeln('Integral(four) SR + SI = ',INT_FOUR);
   writeln('Integral(five) DR + DI = ',INT_FIVE);
   writeln('Integral(six) SR + SI = ',INT_SIX);
   writeln('Integral(seven) SR = ',INT_SEVEN);
   writeln('Integral(eight) SI = ',INT_EIGHT);
   DONE_8 := FALSE;
   DONE_10 := FALSE;
   repeat
      SA_DONE(8,DONE_8);
      SA_DONE(10,DONE_10);
      %getkey(ASCII,SCancode,KBF1,KBF2,ERCCODE);
      if ASCII = 27 then goto 999;
      if ASCII = 1 then %crtpage(0);
      if ASCII = 2 then %crtpage(1);
   until (DONE_8 = TRUE) or (DONE_10 = TRUE);
   %crtpage(0); %cls;
   TIMER_STOP; writeln('Issued TIMER_STOP');
writeln('SIGNALS: DONE_8 = ', DONE_8, ', DONE_10 = ', DONE_10);
if SA_SLOW_READ = TRUE then begin
  writeln('Issuing SA_SREAD[8]');
  SA_SREAD(FID_REAL_PTR,8,SA_RECORD_LENGTH);
  writeln('Issuing SA_SREAD[10]');
  SA_SREAD(FID_IMG_PTR,10,SA_RECORD_LENGTH);
end else begin
  SA_READ(8,SA_RECORD_LENGTH);
  UNPCK_DMA_ARRAY(FID_REAL_PTR,SA_RECORD_LENGTH);
  writeln('Issued SA_READ[8]');
  SA_READ(10,SA_RECORD_LENGTH);
  UNPCK_DMA_ARRAY(FID_IMG_PTR,SA_RECORD_LENGTH);
  writeln('Issued SA_READ[10]');
end;
SAMPLE_POSITION_RESET;
ZERO_INTEGRATE;
if CHK_INT_ONE < abs(INT_ONE) then goto 100;
if CHK_INT_TWO < abs(INT_TWO) then goto 100;
LOAD_PLOTARRAYS;
%crtpage(1);
PLOT_XY(@PLOTS_DATA,'A');
%crtpage(0);
write(PR,'Delay ',DELAY:7:2);
write(PR,' ',INT_ONE:9:2);
write(PR,' ',INT_TWO:9:2);
write(PR,' ',INT_FOUR:12:2);
write(PR,' ',INT_FIVE:15:2);
writeln(PR);
DELAY := 2 * DELAY;
end; {** End of the Cycle **}
999: TIMER_STOP;
%crtpage(0);
FINAL_ZERO_TIMER_STOP;
DAC_SETUP(0,0.0);
IFC_CAMAC;
end;

procedure ZERO_ZERO_FIELD_SPIN_LATTICE;
label 100,999;
var
  CH1,CH2,CH3,CH4: LONGINT;
DONE_8,DONE_10: BOOLEAN;
ASCII,SCANCODE,KBF1,KBF2,ERRCODE: INTEGER;
TAU_ZERO_FIELD: REAL;
begin
  %cls;
  writeln('This procedure will measure T1D. ');
  writeln('Connect the Samp. Post. coax.');
  writeln;
  writeln('NOTE: You must supply the name of the sample you are running');
  writeln('Name of the sample you are running?');
  readln(TITLE);
  writeln(PR,' T1 in Zero Field for ',TITLE);
  writeln(PR,' Note: Experiment is done without rf irradiation. ');
  writeln(PR,' The integrals printed are INT_1,INT_2,INT_4, and INT_5');
  ZERO_DELAY := 0.5;
  VIB_DELAY := 1.5;
  ZERO_T_PARAMETERS;
  X_POINTS := 0; ZERO_DATA_ALL;
  SAMPLE_POSITION_RESET;
  TAU_ZERO_FIELD := 0.5;
  while TAU_ZERO_FIELD <= 5.0 do begin
    X_POINTS := X_POINTS + 1;
    100: PAUSE(2); SATURATE(20);
    TIMER_CS(1,3,1); CH1 := round(10*DELAY);
    CH2 := round(10.0*TAU_ZERO_FIELD); CH3 := round(10.0*VIB_DELAY);
    TIMER_CH_SETUP(CH1,CH2,CH3,0,0,0,0,0); TIMER_EXE;
    FREQ_SET(300.0); FREQ_SET(300.0); DAC_SETUP(0,0.0);
    SA_SETUP(8,SA_RECORD_LENGTH,0,7);
    SA_SETUP(10,SA_RECORD_LENGTH,0,7);
    SA_SWEEP(8,SCANS); SA_SWEEP(10,SCANS);
    SA_RESET(8); SA_RESET(10); PAUSE(2);
    SA_RESET(8); SA_RESET(10); PAUSE(2);
    SA_SET_LAM_WHEN_DONE(8);
    SA_SET_LAM_WHEN_DONE(10);
    %crtpage(0); %cls;
gotoxy(40,52);
  writeln('Sample being run is ',TITLE);
  %gotoxy(40,53);
writeln('Delay = ',DELAY);
writeln('Integral(one) DR = ',INT_ONE);
writeln('Integral(two) DI = ',INT_TWO);
writeln('Integral(three) |DRI + |DII = ',INT_THREE);
writeln('Integral(four) SQRT(DR^2+DI^2) = ',INT_FOUR);
writeln('Integral(five) |SR + |SII = ',INT_FIVE);
writeln('Integral(six) SR^2 + SI^2 = ',INT_SIX);
writeln('Integral(seven) SR = ',INT_SEVEN);
writeln('Integral(eight) SI = ',INT_EIGHT);

DONE_8 := FALSE;
DONE_10 := FALSE;
repeat
    SA_DONE(8,DONE_8);
    SA_DONE(10,DONE_10);
    %getkey(ASCII,SCANCODE,KBF1,KBF2,ERRCODE);
    if ASCII = 27 then goto 999;
    if ASCII = 1 then %crtpage(0);
    if ASCII = 2 then %crtpage(1);
until ( DONE_8 = TRUE ) or ( DONE_10 = TRUE );
%crtpage(0); %cls;
TIMER_STOP; writeln('Issued TIMER_STOP');
writeln('SIGNALS: DONE_8 = ',DONE_8,' , DONE_10 = ',DONE_10);
if SA_SLOW_READ = TRUE then begin
    writeln('Issuing SA_SREAD[8]');
    SA_SREAD(FID_REAL_PTR,8,SA_RECORD_LENGTH);
    writeln('Issuing SA_SREAD[10]');
    SA_SREAD(FID_IMG_PTR,10,SA_RECORD_LENGTH);
end
else begin
    SA_READ(8,SA_RECORD_LENGTH);
    UNPCK_DMA_ARRAY(FID_REAL_PTR,SA_RECORD_LENGTH);
    writeln('Issued SA_READ[8]');
    SA_READ(10,SA_RECORD_LENGTH);
    UNPCK_DMA_ARRAY(FID_IMG_PTR,SA_RECORD_LENGTH);
    writeln('Issued SA_READ[10]');
end;
SAMPLE_POSITION_RESET;
ZERO_INTEGRATE;
if CHK_INT_ONE < abs(INT_ONE) then goto 100;
if CHK_INT_TWO < abs(INT_TWO) then goto 100;
LOAD_PLOT_ARRAYS;
%crtpage(1);
PLOT_XY(@PLOTS_DATA,'A');
%crtpage(0);
write(PR,'Delay ',TAU_ZERO_FIELD:7:2);
write(PR,' ',INT_ONE:9:2);
write(PR,' ',INT_TWO:9:2);
write(PR,' ',INT_FOUR:12:2);
write(PR,' ',INT_FIVE:15:2);
writeln(PR);
TAU_ZERO_FIELD := TAU_ZERO_FIELD + 0.5;
end; {** End of the Cycle **}
999: TIMER_STOP;
%crtpage(0);
FINAL_ZERO_TIMER_STOP;
DAC_SETUP(0,0.0);
IFC_CAMAC;
end;

procedure ZERO_SAVE_DATA;
label 50;
var
  F1 : TEXT;
  SNAME : STRING80;
  I,ioerr : INTEGER;
begin
  %cls;
  writeln('*** Insert Disk ***');
  writeln('Enter Filename:[Ex. IrTEST.DAT]');
  readln(SNAME);
  X_POINTS := 1 + TRUNC( (FREQ_STOP - FREQ_START) / FREQ_INC );
  {$i-}
  REWRITE(F1,SNAME);
  if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in reset ','SNAME,' is ',ioerr);
    goto 50;
  end;
  for I := 1 to X_POINTS do begin
    writeln(F1,DATA_ALL[1,I], ' ',DATA_ALL[2,I], ' ');
DATA_ALL[3,I],',DATA_ALL[4,I]);
writeln(Fl,DATA_ALL[5,I],',DATA_ALL[6,I],',
DATA_ALL[7,I],',DATA_ALL[8,I]);
writeln(Fl,DATA_ALL[9,I],',DATA_ALL[10,I]);
writeln(DATA_ALL[1,I],',DATA_ALL[2,I],',
DATA_ALL[3,I],',DATA_ALL[4,I]);
end;
CLOSE(Fl);
50: end; {$I+}

procedure ZERO_READ_DATA;
label 50;
var
  F1 : TEXT;
  SNAME : STRING80;
  I,ioerr : INTEGER;
begin
  %cls;
  ZERO_DATA_ALL;
  writeln('Enter Filename:[Ex. 1:TEST.DAT]');
  readln(SNAME);
  reset(Fl,SNAME);
  if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in reset ',SNAME:12,' is ioerr');
    goto 50;
  end;
  I := 1;
  readln(F1,DATA_ALL[1,I],DATA_ALL[2,I],DATA_ALL[3,I],DATA_ALL[4,I]);
  readln(F1,DATA_ALL[5,I],DATA_ALL[6,I],DATA_ALL[7,I],DATA_ALL[8,I]);
  readln(F1,DATA_ALL[9,I],DATA_ALL[10,I]);
  writeln(DATA_ALL[1,I]);
  while not EOF(Fl) do begin
    I := I + 1;
    readln(F1,DATA_ALL[1,I],DATA_ALL[2,I],DATA_ALL[3,I],DATA_ALL[4,I]);
    readln(F1,DATA_ALL[5,I],DATA_ALL[6,I],DATA_ALL[7,I],DATA_ALL[8,I]);
    readln(F1,DATA_ALL[9,I],DATA_ALL[10,I]);
    writeln(DATA_ALL[1,I]);
procedure ZERO_FREQUENCY_PARAMETERS;
var
  TIME : REAL;
begin
  writeln('Name of the sample you are running?');
  readln(TITLE);
  writeln('rf Irradiation level [in Volts]?');
  readln(H1_VOLTS);
  writeln('Starting frequency in kHz[real number]?');
  readln(FREQ_START);
  writeln('Stopping frequency in kHz[real number]?');
  readln(FREQ_STOP);
  writeln('Frequency increment [real number]?');
  readln(FREQ_INC);
  NUMBER_OF_INCREMENTS := round((FREQ_STOP-FREQ_START)/FREQ_INC);
  if (NUMBER_OF_INCREMENTS <> 0) then begin
    TIME := ( (DELAY + 15) * NUMBER_OF_INCREMENTS )/ 3600.0;
    writeln(PR,'**Sample being run is ',TITLE,'**');
    writeln(PR,'Time required per loop = ',TIME:8:2,' hours');
    writeln(PR,'Starting frequency in kHz ',FREQ_START:7:2);
    writeln(PR,'Stopping frequency in kHz ',FREQ_STOP:7:2);
    writeln(PR,'Frequency increment ',FREQ_INC:7:2);
    writeln(PR,'Time required per loop= ',TIME:8:2,' hours');
    writeln(PR,'rf Irradiation level [in Volts] = ',H1_VOLTS:9:4);
  end;
end;

procedure PRINT_ALL_INTEGRAL_VALUES;
var
  freq_range,I : integer;
begin
  freq_range := 0;
  for I := 1 to 500 do
if DATA_ALL[1,I] > 0.0 then freq_range := freq_range +1;
writeln(PR,'The sample is ',TITLE);
writeln(PR,'Integrals 1 and 2 are the summed values from slot 8 and 10');
writeln(PR,'Integral 3 is ABS(1) + ABS(2)');
writeln(PR,'Integral 4 is SQRT(SQR[1] + SQR[2]));
writeln(PR);
for I := 1 to freq_range do begin
  write(PR,'Freq = ',DATA_ALL[1,I]:7:2);
  write(PR,' ',DATA_ALL[2,I]:9:2);
  write(PR,DATA_ALL[3,I]:9:2);
  write(PR,DATA_ALL[4,I]:12:2);
  writeln(PR,DATA_ALL[5,I]:15:2);
end;
end;

procedure ZERO_MENU; { Writes options for CAMAC control on screen }
begin
  %cls;
  %gotoxy(0,1);
  writeln (' ZERO FIELD NMR ');
  writeln (' UPDATE: 24 SEPT 1987 ');
  writeln (' 1  Find spin lattice relaxation time in high field: T1 ');
  writeln (' 2 Find spin lattice relaxation time in zero field: T1D');
  writeln (' 3 Starts a zero-field spectrum');
  writeln (' 4 Plot all data on screen (One Integral at a time) ');
  writeln (' 5 Save all data on disk ');
  writeln (' 6 Read data from disk ');
  writeln (' 7 Two frequency option ');
  writeln (' 8 Print total integral values from last run');
  writeln (' 24 Exit to MAIN MENU');
end;

procedure ZERO_MAIN;
label 80;
var
  INTEGRAL_SELECTED,LUN2,ioerr,zero_choice : INTEGER;
  zero_exit : BOOLEAN;
begin
  GET_LUN2(LUN2);
zero_exit := FALSE;
repeat
  ZERO_MENU;
  readln (zero_choice);
  case zero_choice of
    1: ZERO_SPIN_LATTICE;
    2: ZERO_ZERO_FIELD_SPIN_LATTICE;
    3: begin
      ZERO_DELAY := 2.0;
      VIB_DELAY := 1.5;
      ZERO_T_PARAMETERS;
      ZERO_FREQUENCY_PARAMETERS;
      if (NUMBER_OF_INCREMENT<>0) then
        ZERO_FIELD_CYCLING;
      end;
    4: begin %cls;
      writeln('The plot is done on SCR#1; To return to the main');
      writeln('screen, type A; To return to the plot type B;');
      writeln('To exit type C.');
      writeln;
      writeln('Enter number of integral to plot.');
      readln(INTEGRAL_SELECTED);
      %crtpage(1);
      %cls;
      ZERO_PLOT_ANY(INTEGRAL_SELECTED);
      repeat
        %getkey(AII,SCNCDE,K1,K2,ERR);
        if ((ALL = 65) or (ALL = 97)) then %crtpage(0);
        if ((ALL = 66) or (ALL = 98)) then %crtpage(1);
      until ((ALL = 67) or (ALL = 99));
      %crtpage(0);
      %gotoxy(40,52);
      write('Print out this integral plot?[Y/N/S] ');
      repeat
        %getkey(AII,SCNCDE,K1,K2,ERR);
        if ((ALL = 89) or (ALL = 121)) then begin
          %crtpage(1);
          %setprmode(PR,2);
        end
     end
      CLOSE(PR);
if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in %GPRTSCR is ',ioerr);
end;
repeat
    %getkey(All,SCNCDE,K1,K2,ERR);
until ((All = 65) or (All = 97));
%crtpage(0);
reset(PR,'#PR');
if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in reset #PR is ',ioerr);
end;
able($1+)
goto 80;
eend;
if ((All = 83) or (All = 115)) then begin
    %crtpage(1);
    CLS_LUN2;
    CLS_DMM;
    HALT;
    end;
until ((All = 78) or (All = 110)); { REPEAT }
end;
5: ZERO_SAVE_DATA;
6: ZERO_READ_DATA;
7: SET_TWO_FREQ;
8: PRINT_ALL_INTEGRAL_VALUES;
24: zero_exit := TRUE;
end;
80: until (zero_exit = TRUE);
end;

procedure READ_OLD_SA_DATA;
begin
    %cls;
    write('Enter number of scans? ');
    readln(SCANS);
    write('Enter DELAY? ');
    readln(DELAY);
writeln('Digitization Rate ?');
writeln('0=20MHz, 1=10MHz, 2=5MHz, 3=2MHz, 4=1MHz, 5=500kHz,');
writeln(' 6=200kHz, 7=Ext.');
write(' Enter rate code? ');
readln(SA_RATE_CODE);
SA_READ(8,SA_RECORD_LENGTH);
UNPCK_DMA_ARRAY(FID_REAL_PTR,SA_RECORD_LENGTH);
SA_READ(10,SA_RECORD_LENGTH);
UNPCK_DMA_ARRAY(FID_IMG_PTR,SA_RECORD_LENGTH);
end;

procedure GET_LUN1(var LUN1 : INTEGER);
var
  DEV1 : TEXT;
begin
  LUN1 := 111;
  writeln(' The LUN for #BUSA? is ',LUN1);
end;

procedure INIT_IEEE_DEVICES;
var
  FPT : ARRAY [0..2] OF INTEGER;
  B_IOPTR : LONGINT;
  TRANMODE,DIR,OPT,ERRCODE : INTEGER;
begin
  GET_LUN1(LUN1);
  B_IOPTR := 0;
  TRANMODE := 0; OPT := 0; DIR := 2;
  DEV1 := '#BUSA?';
  %sysopen(LUN1,DEV1,DIR,OPT,TRANMODE,NIL,B_IOPTR,ERRCODE);
  if ERRCODE <> 0 then begin
    writeln(' Error in sysopen for #BUSA? is ',ERRCODE);
    %prterr(%cserr);
    KEY_PRESSED;
  end;
  FPT[0] := 29;
  FPT[1] := 0;
  FPT[2] := 0;
  %sysfunc(LUN1,@FPT,ERR);
  if ERR <> 0 then begin
writeln(' Error on IEEE-488 timeout disable is ',ERR);
%prterr(%cserr);
KEY_PRESSED;
end;
FPT[0] := 34;
FPT[1] := 1;
FPT[2] := 0;
%sysfunc(LUN1,@FPT,ERR);
if ERR <> 0 then begin
  writeln(' Error on IEEE_488 SET_DMA_LIMIT is ',ERR);
  %prterr(%cserr);
  KEY_PRESSED;
end;
OPEN_CAMAC;
DMM_OPEN;
end;

procedure MAINMENU ; {Writes options for CAMAC control on screen}
begin
  %cls;
  %gotoxy(0,1);
  writeln(' ZEROFIELD NMR PROGRAM ');
  writeln(' 24 SEPT 1987 ');
  writeln;
  writeln(' 1 Clear the screen, then hit any key to go on. ');
  writeln(' 2 CAMAC menu ');
  writeln(' 3 NAF menu ');
  writeln(' 4 Pulse Programmer menu ');
  writeln(' 5 Digital Multimeter menu ');
  writeln(' 6 Zero Field NMR menu ');
  writeln(' 7 Read old Signal Averager Data ');
  writeln(' 8 Save FID data on disk in IBM-NMR format ');
  writeln(' 9 Capacitor Menu ');
  writeln(' 23 Toggle to the graphics screen (Press Ctrl-H to return to menu; ');
  writeln(' 24 Exit to MONITOR ');
end;
{**********THIS IS THE MAIN PROGRAM******************}

begin
  OPEN_SCRN0;
  CHANGE_PRINTER_COLOR;
reset(PR,'#PR');
if IORESULT <> 0 then begin
  ioerr := IORESULT;
  writeln(' IORESULT in reset #PR is ',ioerr);
  rewrite(PR,'#PR');
  if IORESULT <> 0 then begin
    ioerr := IORESULT;
    writeln(' IORESULT in rewrite #PR is ',ioerr);
    end;
  end;
end;

{${}-}
FED_REAL_PTR := @FID_REAL[1];
FID_IMG_PTR := @FID_IMG[1];
NOISE_R_PTR := @NOISE_REAL[1];
NOISE_I_PTR := @NOISE_IMG[1];
INIT_IEEE_DEVICES;
KEY_PRESSED;
main_exit := FALSE;
repeat
  MAINMENU;
  readln (main_choice);
  case main_choice of
    1: {Clear the screen for a bit}
      begin
        %cls;
        repeat until (NAF_KEYBOARD = TRUE);
        end;
    2: CAMAC_MENU;
    3: NAF_CAMAC;
    4: PULSE_MAIN;
    5: DMM_MAIN;
    6: ZERO_MAIN;
    7: READ_OLD_SA_DATA;
    8: SAVE_FID(SCANS,DELAY,SA_RATE_CODE,FID_REAL,FID_IMG);
    9: CAP_MAIN;
    23: begin
      %crtpage(1);
      repeat
        %getkey(A11,SCNCDE,K1,K2,ERR);
until(AII = 8);
%crtpage(0);
end;

24: main_exit := TRUE;
end;

until (main_exit = TRUE);
end.
Appendix 2

A2.1 Introduction. SPOWDER is a FORTRAN program which simulates deuterium NMR lineshapes for different rates and models of motion. The program was obtained from Dr. R. J. Wittebort, University of Louisville, and has been modified slightly in order to run on the LSU X-ray facility MicroVax. In addition to deuterium NMR lineshapes, the program will also simulate inversion-recovery spectra and calculate spin-lattice relaxation times for the $\theta = 90^\circ$ and $\theta = 0^\circ$ peaks. Two modified versions of SPOWDER, POWDER and SPT1, have also been used. SPT1 creates tables of rate versus $T_1$ for the $\theta = 90^\circ$ and $\theta = 0^\circ$ peaks, while POWDER generates additional output files for relaxation calculations.

A2.2 SPOWDER. SPOWDER requires an input file (titled INPUT.DAT) which defines the type of calculations to be done and the parameters needed for the simulation. The type of calculation is indicated by a keyword of 6 characters, having an A(3) I(3) format. The keywords are defined in sections A2.2.1-A2.2.5.

A.2.2.1 TAU00N. TAU indicates that a calculation in which the spacing between $90^\circ$ pulses is varied is to be done. The list of taus, expressed in $\mu$s, is given below the keyword. The program requires that TAU, along with a minimum value of 1 for N, be in INPUT.DAT.

A.2.2.2 KIN00N. KIN indicates that a calculation where the jump rate between sites is incremented N times is requested. The size of the increment is given below the keyword. The jump rate is incremented using the formula:

$$X = \log_{10}(\text{OLD\_RATE}) + (N - 1) \times \text{INCREMENT}$$

$$\text{NEW\_RATE} = 10^X$$

The number of output data files created by this option is $2N + 1$.

A.2.2.3 REL00N. REL indicates that a relaxation calculation or a quadrupolar order calculation is to be done. The values of the variable delay list for the relaxation
calculation is given below the keyword, along with a line of input indicating the pulse length in µs of some tip angle θ and a number indicating whether a Zeeman relaxation calculation (= 1) or a quadrupolar order calculation (= 2) is desired. The output data files will have a "Z" or "Q" extension added to the file name when this option is performed. In the case of relaxation calculations, the program also generates a series of output files with an "F" extension. These files are identical to the "Z" and "Q" files except that the effect of a finite pulse width is included in the simulation as well. In addition, a file without a letter extension and having the simulation for the fully relaxed spectrum is created. The number of output files generated by this option is 2N + 1.

A2.2.4 NAM000. NAM indicates the name of the output files for the simulation. The file name is given below the keyword.

A2.2.5 END000. END000 indicates that no more options are requested.

SPOWDER can perform these options simultaneously, with the different types of calculations done in a nested loop fashion. The order in which the options are done are KIN, TAU, and REL. The total number of options which can be done is given by the relationship:

NRATES*NTAUS*(2NRDELAYS + 1) ≤ 22

Here NRATES, NTAUS, and NRDELAYS are the number of rate increments, the number of taus, and the number of relaxation delays, respectively.

A2.2.6 Spectral Parameters. After receiving the requested options, the parameters required for the simulation are read next. The parameters and their definitions are given below.

SPLIT, DIFF, LB, HW, SF, NPTS, NAVG

SPLIT = quadrupolar coupling constant in hertz.

DIFF = η * SPLIT * 0.75 (expressed in hertz).

LB = line broadening in hertz.

HW = half-width of the spectra in hertz (spectral width + 2).
SF = spectrometer frequency in MHz.

NPTS = number of points in the simulation to be calculated (maximum is 200).

NAVG = number of Euler betas relating the laboratory frame to the crystal frame to calculate. Good results are usually obtainable with 70-180 betas.

NSITE = number of jump sites in the model.

GAMMA, BETA, ALPHA = the Euler angles (expressed in degrees) relating the crystal frame to the principle axis for each jump site. NSITE sets of angles are required. Note: GAMMA = polar phi (φ), BETA = polar (θ).

RS(NR, NK) = jump rate from site NK to site NR in rads/sec. Only the upper triangle of the rate matrix is needed.

PEQ = the equilibrium population of the sites 1 thru NSITE.

TITLE = comment which is written into the output file.

Figure A2.1 shows a typical input file; Figure A2.2 shows an output file created by SPOWDER using the corresponding input file.

A2.2.7 Compilation and Execution of SPOWDER. The program is compiled using the command sequence FORTRAN SPOWDER. An execution file is created by the command LINK:

```
LINK SPOWDER,MATRIX,ALEAST
```

MATRIX and ALEAST are object files which have subroutines used by SPOWDER for matrix diagonalization and least-squares analysis of the relaxation data. The program is executed by the command RUN SPOWDER or submitted to batch on the X-ray facility MicroVax by typing the command "SPOWDER", which is a special command defined by the LOGIN.COM file.

As a final note, the SPOWDER program generates the lineshape for the $0 \rightarrow 1$ transition; thus before plotting the data, it must first be symmetrized. Symmetrization and plotting are carried out by the program PLOTDATA, which uses the PLOT10 library.
TAU001
35.
REL007
3.6,180,1
.005
.030
.05
.1
.20
.500
1.5
NAM000
BETA7
END000
76000,0.0,750,100000,61.4,200,150
4
0.0,0.0,0.0
0.,14.9,0.0
120.,14.9,0.0
240.,14.9,0.0
180000000
180000000
180000000
180000000
180000000
180000000
0.25,0.25,0.25,0.25
FOUR SITE BETA=14.9 5/6/88

Figure A2.1. Input file for the SPOWDER program.
BRIEF SUMMARY OF SIMULATION PARAMETERS

COUPLING CONSTANT =  76000.000
ETA=      0.000
LINEBROADENING  750.000
HALFWIDTH OF PLOT  100000.000
SPEC FREQ (MHZ)  61.4 TIME DELAY = 1.50000
PULSE SPACING (USEC) 0.3500E-04
4 JUMP SITES IN THE SIMULATION

EULER ANGLES PAS-CF
0.0000  0.0000  0.0000
0.0000  14.9000  0.0000
0.0000  14.9000  120.0000
0.0000  14.9000  240.0000

JUMP RATES LOWER DIAGONAL, ELEMENTS OF FIRST 5 ROWS
.00E+00 .18E+09 .18E+09 .18E+09 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00
.00E+00 .00E+00 .18E+09 .18E+09 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00
.00E+00 .00E+00 .00E+00 .18E+09 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00
.00E+00 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00 .00E+00

EQUILIBRIUM POPULATIONS
0.2500  0.2500  0.2500  0.2500  0.0000  0.0000  0.0000  0.0000  0.0000
200

61.40000  100000.0
0.000000E+00  0.000000E+00  0.000000E+00  0.228930E-04  0.265915E-04
0.317018E-04  0.392491E-04  0.515036E-04  0.747289E-04  0.132489E-03
0.701301E-02  0.693243E-03  0.659506E-03  0.673489E-03  0.745073E-03
0.681633E-03  0.731946E-03  0.781108E-03  0.836210E-03  0.879489E-03
0.741614E-03  0.909059E-03  0.757350E-03  0.838437E-03  0.892885E-03
0.959106E-03  0.821040E-03  0.864109E-03  0.971265E-03  0.809629E-03
0.896922E-03  0.902333E-03  0.105898E-02  0.875586E-03  0.105007E-02
0.939517E-03  0.115304E-02  0.998672E-03  0.104725E-02  0.120062E-02
0.114668E-02  0.125627E-02  0.111375E-02  0.121495E-02  0.134992E-02
0.126265E-02  0.142885E-02  0.145843E-02  0.140284E-02  0.146542E-02
0.168488E-02  0.175145E-02  0.180940E-02  0.189034E-02  0.200394E-02
0.233653E-02  0.255308E-02  0.284171E-02  0.329015E-02  0.397611E-02
0.367603E-02  0.123353E-02  0.579843E-03  0.345013E-03  0.233464E-03
0.131808E-03  0.105529E-03  0.869228E-04  0.731088E-04  0.625797E-04
0.477230E-04  0.423321E-04  0.378735E-04  0.341311E-04  0.309550E-04
0.258804E-04  0.238283E-04  0.220285E-04  0.204385E-04  0.190243E-04
0.166312E-04  0.156107E-04  0.146874E-04  0.138483E-04  0.130828E-04
0.117431E-04  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00

FOUR SITE BETA=14.9 5/6/88

Figure A2.2. Output file created by the program SPOWDER. Some of the zero data points have been deleted for convenience.
for plotting calls. Another program, PREPARE, creates files which can be read by
PLOTDATA from the output files. The output files have to be renamed before they can
be read by PREPARE. The files are renamed using the command file RNM.COM, while
PREPARE and PLOTDATA are executed using the command file PLT.COM. The
sequence for plotting the output of SPOWDER is thus:

@RNM OUTPUT # (# refers to the number of files)
@PLT OUTPUT#

The command file RNM.COM renames files by placing a number extension on the file
name (i.e. OUTPUT.DAT;5 → OUTPUT5.DAT;1).

A2.3 POWDER. POWDER is a modification of the SPOWDER program where in
addition to the normal output files three additional files are created,
RATEMATRIX.OUT, T1OUTPUT.DAT, and T1CALC.DAT. RATEMATRIX.OUT
lists the rate matrix used for the calculations. T1OUTPUT.DAT gives the T_1 values for
various values of betas used in the powder average. T1CALC.OUT summarizes the
parameters used for the relaxation calculation. The input file for POWDER is identical to
SPOWDER. The program is submitted to batch on the X-ray facility MicroVax by typing
the command "POWDER", which is special command defined in the LOGIN.COM file.
POWDER cannot perform multiple rate or tau calculations.

A2.4 SPT1. SPT1 is a modification of SPOWDER program, where tables of spin-
lattice relaxation times versus rate for the θ = 90° and θ = 0° peaks of the deuterium
spectrum are created instead deuterium lineshapes. The program requires an input file
(T1INPUT.DAT) which has the necessary parameters for the calculation. The input file
is similar to those used by SPOWDER and POWDER; however, no keywords are
necessary. The major change in input file format is in the first line containing the
calculation parameters:

SPLIT, SF, NAVG, RATE, RATEINC

RATE = initial rate of the model.
RATEINC = rate increment. The rate is incremented in the same manner as in KIN.

The program creates two output files, T1RUN90.DAT and T1RUN0.DAT, which have the incremented values of $T_1$ and rate. Each file has 50 data points. The value of NAVG has to be a minimum of 1. Figure A2.3 shows a typical input file for the program.
Figure A2.3. Input file for SPT1 program.
Appendix 3

A3.1 Introduction. The Henry Radio 2006A amplifier is a commercial unit which has been modified at the factory for NMR applications. It is a narrow-banded device tuned to a frequency of 61.4 MHz and having a maximum output of 1 kilowatt. The amplifier is based on the design for a 160 meter amplifier published in recent editions of the Radio Amatuer's Handbook.

A3.2 Description of Front Panel Controls. Two meters, along with four push buttons, are located on the left-side of the front panel. The top meter measures the plate current of the tube. The bottom meter gives either the grid current or the high-voltage from the power supply, depending on whether the IG or HV button underneath the multimeter is pushed in. The other two buttons, OP and STBY, will light the red or yellow lights located on the top center of the front panel when they are engaged. Normally, when the STBY light is on, the amplifier will not amplify the input drive. However, for the 2006A series the amplifier will amplify the input drive when either light is on; thus the output of the amplifier must be always be connected to a dummy load as a safety precaution whenever the amplifier is on but not being used.

The on/off switch and the SSB/FM control are located on the right-hand side of the front panel. The FM mode is used for FM or AM work, while the SSB (single-side band) mode is normally used for voice transmissions. The SSB mode is used for NMR applications. The OUTPUT TUNE and LOAD control are located in the center of the front panel. These two controls are used to adjust the tuning and impedance matching of the amplifier output until they match the NMR probe.

A3.3 Checkout Procedure. An oscilloscope and two 20 db attenuators are needed to set-up the amplifier for NMR work. The output of the Bruker AM-400 solution spectrometer is measured first. A 10 db attenuator is attached to the output of the Bruker amplifier and the peak-to-peak voltage (V_{pp}) is monitored with the oscilloscope. The
input of the oscilloscope is set for 50 Ω impedance and is protected from the high-power rf by a 20 db attenuator. At 10 db attenuation, the output of the Bruker transmitter should be ~119 Vpp, which corresponds to ~35 watts.

The output of the Bruker amplifier is then used to drive the Henry amplifier. During routine operation, the plate and grid currents should be ~0 milliamps and the high-voltage should be 2400 volts (in the SSB mode). The output of the Henry amplifier is monitored with the oscilloscope, using a input impedance setting of 50 Ω. The oscilloscope is protected from the high-power rf of the Henry amplifier by two 20 db attenuators connected in series to the input of the oscilloscope. The Vpp is maximized by simultaneously adjusting the OUTPUT TUNE and LOAD controls until the optimum settings are found. With a drive of 35 watts and OUTPUT TUNE and LOAD settings of ~5 and ~100, respectively, the Henry amplifier will produce a peak output of ~616 Vpp, corresponding to ~950 watts. Before connecting the probe to the Bruker preamp, the amount of pulse bleed-through past the quarter wave cable is checked with the oscilloscope. Normally, the preamp should see only ~14 Vpp (~0.5 watts) of pulse bleed-through during NMR operation.
BIRTH DATE: July 16, 1961

EDUCATION:
1983-1988 Ph. D., Inorganic Chemistry, Louisiana State University, Baton Rouge, LA
  Research Director: Professor Leslie G. Butler
  3.93/4.0 GPA, Minor in Physical Chemistry
  Doctoral Thesis: "Applications of Solid-State Deuterium NMR Spectroscopy"
  Date of graduation: August, 1988

1979-1983 B. S., Chemistry (3.97/4.0 GPA)
  Mississippi State University, Starkville, MS (Summa Cum Laude)

PUBLICATIONS:


ORGANIZATIONS:
American Chemical Society
Phi Kappa Phi
Phi Lambda Upsilon (President, Alph Mu Chapter, 1985-1987)
Graduate Student Council (Secretary, 1985-1986)

PRESENTATIONS:


FUTURE PLANS
Will be accepting a postdoctoral position with Dr. Lon Mathias at the University of Southern Mississippi in Hattiesburg, Mississippi.
DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: William Lawrence Jarrett, Jr.

Major Field: Chemistry

Title of Dissertation: Applications of Solid-State Deuterium NMR Spectroscopy

Approved:

[Signatures]

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

[Signatures]

Date of Examination:

June 23, 1988