A versatile pulse programmer for pulsed
nuclear magnetic resonance spectroscopy

MASTER

Douglas John Adduci

M.S. Thesis submitted to Iowa State University

Ames Laboratory, DOE

Iowa State University

Ames, Iowa 50011

Date Transmitted: May 1979

PREPARED FOR THE U.S. DEPARTMENT OF ENERGY
UNDER CONTRACT NO. W-7405-eng-82

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Available from: National Technical Information Service
U. S. Department of Commerce
P.O. Box 1553
Springfield, VA 22161

Price: Microfiche \$3.00

A versatile pulse programmer for pulsed-
nuclear magnetic resonance spectroscopy

by

Douglas John Adduci

A Thesis Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
MASTER OF SCIENCE

Major: Electrical Engineering

Approved

In Charge of Major Work

For the Major Department

For the Graduate College

Iowa State University
Ames Iowa

-1979

TABLE OF CONTENTS

	Page
INTRODUCTION	1
Background Information	2
THE PULSE SPECTROMETER	5
PULSE SEQUENCES	9
The Free Induction Decay	10
Measurement of T1 and T2	11
The Carr-Purcell Technique	16
Spin Locking	17
Multiple Pulse Experiments	17
Automated T1 and T2 Measurements	19
PULSE PROGRAMMERS	21
SPECIFICATIONS	25
IMPLEMENTATION	30
Microprocessor	33
Sequencer	36
Timebase	44
Slow Clock	49
Pulse Width Control	51
Integrate and Hold	53
PROGRAMMING LANGUAGE	57
OPERATING SYSTEM	62
SUMMARY AND CONCLUSIONS	68
BIBLIOGRAPHY	70

ACKNOWLEDGEMENTS	72
APPENDIX A. EDIT 1.0 USERS GUIDE	73
APPENDIX B. MONITOR AND TRANSLATOR FLOW CHARTS	80

LIST OF FIGURES

	Page
Figure 1. Block diagram of a typical NMR spectrometer	6
Figure 2. Motion of M due to a 90° pulse in the X direction	12
Figure 3. Free induction decay off resonance	12
Figure 4. Measurement of T1 by the $180, \tau, 90$ technique	13
Figure 5. The spin-echo experiment	15
Figure 6. Block diagram of a modular pulse programmer	23
Figure 7. Block diagram of pulse programmer implementation	32
Figure 8. Microcomputer system	34
Figure 9. Sequencer circuit	37
Figure 10. Instruction format	40
Figure 11. Timebase circuit	45
Figure 12. Timebase data word format	46
Figure 13. Slow clock	50
Figure 14. Pulse width control circuit	52
Figure 15. Integrate and hold control	55
Figure 16. Microcode tables	65

LIST OF TABLES

	Page
Table 1. Pulse width range on channels P1-P4	27
Table 2. Pulse programmer input and output signals	56

INTRODUCTION

Nuclear magnetic resonance (NMR) is a tool which has been used by chemists and physicists for the past 25 years. NMR has been used by chemists for nondestructive quantitative and qualitative analysis via chemical shifts and coupling constants of molecules in the liquid state. In 1974, Dr. B.C. Gerstein, while on sabbatical at The California Institute of Technology, studied a variety of NMR experiments with Dr. Robert Vaughan. Realizing the value of these techniques, he requested that the Ames Laboratory instrumentation group fabricate an NMR spectrometer for use in his physical chemistry group. At that time, an exciting new area of high resolution solid state NMR was opening up. An instrument was designed which met or exceeded the experimental needs of that time. Since then, new NMR experiments have been devised which did not obsolete, but quickly surpassed the capabilities of that first instrument. At the same time a physics group, headed by Dr. R.G. Barnes, began requesting pulsed NMR equipment. The need for an instrument with extended capabilities was apparent. Extensive preliminary development was necessary to convince potential users that an instrument with such extended capabilities should be constructed. The project, to build a versatile pulse programmer for pulsed NMR, was ultimately funded through Dr. Gerstein's group. This thesis is a report on that project.

The pages that follow describe the design and development of an extremely versatile logic and timing generator which has specific applications to NMR, but whose uses are by no means restricted to that. The development of a suitable language for operator-machine interaction, and the design of a hardware processor to execute that language will be shown. In addition, the use of commercial microprocessor chips and support software used in language translation and machine control will be demonstrated.

Background Information

The basic principle of NMR, very simply stated, is this. A nucleus of spin I has $2I+1$ energy levels equally spaced with a separation $\Delta E = \hbar H_0 / I$. I is the spin quantum number, H_0 is an applied DC magnetic field, and $\hbar$ is the nuclear magnetic moment. Also, $\hbar = \gamma h I / 2\pi$ where γ is the gyromagnetic moment and h is Planck's constant. From the Bohr relation, $\nu_0 = \Delta E / h = \gamma H_0 / 2\pi$ or $\omega_0 = \gamma H_0$ which is the frequency of radiation that causes a transition from one energy level to another. The idea then is to apply a DC magnetic field H_0 to a spin system and view the absorption of, or alternately the emission of radio frequency energy to obtain chemical information about the system.

NMR has developed along two distinct lines; cw or continuous wave techniques and pulsed techniques. The cw method is often called slow passage because the rf radiation

applied to the sample is slowly swept in frequency while maintaining a constant DC magnetic field. Alternately, the DC field may be swept while holding a constant rf. In either case, the result is the same; the absorption is plotted as a function of the swept variable to obtain the NMR spectrum. The pulsed method suggested by Bloch(1) uses short bursts of rf energy at discrete frequencies to excite the sample system. The time response of the system is then viewed while the rf is off. The Fourier Transform of the time response yields the frequency spectrum of the sample system. For the simple case, the pulsed method with application of FFT gives the same information as the slow passage method. The pulse method, being more efficient, results in a substantial savings in time or, by applying signal averaging techniques, an improvement in signal to noise ratio. Pulse methods are also a direct method for measuring relaxation times of individual lines of a complex molecule. As more scientists begin to recognize the power of pulse techniques, the cw spectrometers continue to be replaced by pulse spectrometers. At the same time, the number and complexity of pulsed experiments continues to grow. Along with the complexity of pulsed experiments comes a need for a truly versatile pulse programmer to generate and control the complex experiment timing. It is the need for such an instrument that has spurred the development of a microprocessor based pulse pro-

grammer for pulsed NMR.

THE PULSE SPECTROMETER

NMR pulse sequences range from the extremely simple to the almost ridiculous. The pulse programmer designer must keep in mind not only the details of the pulse sequence he desires to execute, but also other experimental parameters which could be controlled by the pulse programmer. Events such as oscilloscope triggering, data recorder triggering and clocking, and experiment repetition rates could logically be controlled by the pulse programmer.

Before proceeding, it is necessary to clarify the role of the pulse programmer in relation to the other components of a NMR spectrometer. Figure 1 shows a block diagram of a typical multipulse spectrometer. It is important to make a distinction between the pulse programmer which is a digital timing generator, and the four phase rf gating unit. The four phase rf unit is a slave to the pulse programmer and generates the rf pulses that ultimately irradiate the sample. For some of the more complicated NMR experiments, rf pulses of four different phases must be used. The inputs to the four phase rf unit, X, Y, -X, and -Y, represent rf channels which are 90° out of phase with one another. If X is the reference phase then Y is 90° , -X is 180° , and -Y is 270° out of phase. The rf unit output is the rf composite of the digital inputs. Since there is generally a one to one time relationship between a digital pulse generated by the pulse

Figure 1. Block diagram of a typical NMR spectrometer

programmer and an rf pulse at the output of the rf gating unit, it is convenient to call the pulse programmer outputs X, Y, -X, and -Y also. All timing and phasing relationships for a particular pulse experiment can then be written with respect to the pulse programmer as though it were a part of the rf gating unit.

The reference frequency source is generally a high stability crystal oscillator or a frequency synthesizer. It supplies the rf gating unit with a cw signal at the spectrometer operating frequency. The reference frequency source may also be used to supply the pulse programmer with a reference clock. Deriving both the pulse programmer clock source and the rf gate source from the same reference oscillator establishes a phase coherency between the digital pulses from the pulse programmer and the rf pulses from the rf gating unit. This is a desirable condition when applying coherent signal averaging techniques.

The rf power amp is used to amplify the low level rf pulses from the rf gating unit, typically 1 milliwatt, to a power level of between 100 and 1000 watts. The high power rf pulses are delivered to the sample probe through a probe coupling network. Many such networks have been described in the literature (2,3,4), but the end result is to protect the rf receiver from overload during the rf pulse. The pulse programmer can be used here to supply blanking signals to the

rf power amplifier, useful in reducing off state noise, and gating signals to active probe coupling or receiver gating networks (5,6).

The output of the rf receiver is detected using a phase sensitive detector. The reference phase for the phase sensitive detector is derived from the reference frequency source or the rf gating unit. The detected nuclear signal, following suitable amplification and filtering, is captured with a digital data recorder. Many experiments require that the data be sampled at a precise point in time relative to a given rf pulse. Once again the pulse programmer is used to generate the triggers and data strobes needed to operate the data recorder.

It should be obvious from this discussion that the pulse programmer is really the heart of a pulse spectrometer. Its fingers reach out to almost every component of the spectrometer to provide timing or control. It could be said that the overall versatility of a pulse spectrometer is directly related to the versatility of the pulse programmer.

PULSE SEQUENCES

To talk about pulse sequences and pulse programmer attributes needed to perform pulse sequences, it is not necessary to understand the Bloch equations or quantum mechanics. In terms of executing a particular pulse sequence, it is necessary to know only the information pertaining to that sequence such as pulse widths, rf channels or phases, pulse spacings, data sampling points, et cetera. The details of what the pulse sequence actually does is better left to the chemist. An almost universal language for describing pulse sequences has been adopted by chemists and the like. The elements of the language consist of a pulse width descriptor, a phase descriptor, and a pulse spacing descriptor. The pulse width is described by the angle through which a rf pulse rotates the magnetization M . The angle through which M precesses is given by:

$$\Theta = \gamma H_1 T_p \text{ (radians)}$$

where γ is the gyromagnetic ratio, H_1 is the magnetic field intensity of the applied rf pulse, and T_p is the width of the rf pulse in seconds. For a desired rotation angle, H_1 and T_p may vary as long as their product is constant. Note that describing the pulse width in terms of a rotation angle is a relativistic description. The actual pulse width T_p may vary from one spectrometer to another depending upon the magnitude of H_1 .

The motion of M is given by:

$$dM/dt = \gamma M \times H_1$$

and the direction of precession is about the axis of the H_1 field; i.e. the phase of H_1 . Figure 2 shows the motion of M due to a 90° pulse in the X direction. Thus we have the first two pulse sequence descriptors; the rotation angle and the phase. The third descriptor is simply the time between successive pulses and is usually given by the symbol τ . To describe a sequence of a 180° pulse in the X direction followed by a 90° pulse in the Y direction, we would write:

$$180x, \tau, 90y$$

Although this description of nuclear magnetization and precession is quite terse, sufficient background has been given to examine some commonly used pulse sequences. Each sequence will be examined strictly from the standpoint of the pulse programmer designer to determine the minimum set of pulse programmer requirements needed to execute each sequence.

The Free Induction Decay

The simplest of all pulse sequences is the free induction decay or FID. The FID is produced by applying a single 90° pulse as shown in figure 2. The 90° pulse rotates the magnetization onto the Y axis where its decay or relaxation is viewed immediately following the 90° pulse. In a perfectly homogeneous DC field, the time constant of the

decay would be T_2 . In real life, T_2 is shortened by DC field inhomogeneity. Figure 3 shows a free induction decay off resonance; that is, the frequency of the 90° pulse is slightly different from the resonance frequency given by:

$$\omega_0 = \gamma H_0.$$

Very little instrumentation is required to perform the FID experiment. A single pulse channel with variable pulse width is sufficient to generate the 90° pulse. The phase of the 90° pulse is arbitrary in this case. A trigger is needed to start the data recorder immediately after the pulse, and a clock or timer is needed to control the repetition rate for the experiment. Our FID experiment would be written:

90x, observe, repeat every N seconds.

Measurement of T_1 and T_2

Pulse techniques are very well-suited to measuring T_1 and T_2 relaxation times. When a sample system is at equilibrium, the nuclei are distributed among the energy levels according to the Boltzman distribution. When disturbed by a rf pulse, the nuclei return to equilibrium by a first order relaxation process called T_1 , the spin-lattice relaxation time. The spin-spin relaxation time, T_2 , accounts for the process that causes the nuclear spins to come to equilibrium with each other. The T_1 relaxation measurement is written:

180x, τ , 90x, observe, repeat with new value of τ .

The 180° pulse causes the magnetization to be inverted along

Figure 2. Motion of M due to a 90° pulse in the x direction

Figure 3. Free induction decay off resonance

Figure 4. Measurement of T1 by the 180, τ , 90 technique.
 [(a) M is inverted by a 180° pulse in the X direction.
 (b) After time τ , a 90° pulse rotates M onto the Y axis where the initial value of M is observed.]

the Z axis as shown in figure 4a. During the time τ , the magnetization decays longitudinally along the Z axis back toward its equilibrium position. The 90° pulse (4b) flips the magnetization up onto the Y axis where the initial amplitude, A , of the ensuing FID is observed. The value of T_1 is obtained from a plot of $\ln(A_\infty - A_\tau)$ verses τ .

In a manner similar to the T_1 measurement, the T_2 experiment is written:

$90^\circ X, \tau, 180^\circ X, \tau$, observe, repeat for various values of τ .

The motion of M is depicted in figure 5. The 90° pulse tips M onto the Y axis. During the time τ , the spins start to dephase into the X-Y plane(5a). After time τ , the 180° pulse is applied which rotates the moments about the X axis such that they now move toward each other(5b). At time after the 180° pulse, the moments rephase giving rise to an increase in signal intensity or an echo(5c). Because of this echo effect, the T_2 experiment is often called the spin-echo experiment (7). The experimental information is contained in the amplitude of the echo which must be sampled at precisely τ following the 180° pulse.

Instrumentally, the T_1 and T_2 experiments are almost identical. Both experiments require two pulse channels; one for the 90° pulse and one for the 180° pulse. Both require a timebase to generate the delay τ , a trigger to start the data recorder, and a clock to control the experiment repetition

Figure 5. The spin-echo experiment. [(a) A 90° pulse along X rotates M into the X-Y plane where dephasing occurs. (b) A 180° pulse τ later rotates M around the X axis and causes M to rephase at time 2τ (c) producing an echo.]

rate. Since the repetition clock is needed in all experiments, it will be assumed understood and eliminated from further pulse sequence discussion. Additionally, the spin-echo experiment requires a second delay period τ following the 180° pulse. Depending upon the pulse programmer design, this second delay may come from the same timer that generated the first τ or it may require a second physical timer set to the same value as the first.

The Carr-Purcell Technique

The Carr-Purcell sequence(8) is a simple modification to the spin-echo technique for measuring T_2 . The Carr-Purcell sequence is written:

$90^\circ, \tau, (180^\circ, 2\tau, 180^\circ, 2\tau)$ repeat N times.

The 180° pulses at time $\tau, 3\tau, 5\tau, 7\tau$, et cetera cause echos which must be sampled at time $2\tau, 4\tau, 6\tau, 8\tau$, et cetera. The sequence requires two pulse channels as with the spin-echo experiment, and two timebases to generate the τ and 2τ time delays. A new feature has been introduced in this experiment; that is the capability of the pulse programmer to repeat a segment of a pulse sequence an arbitrary number of times. This looping feature is of major importance to the multiple pulse line narrowing experiments to be discussed.

Spin Locking

Spin locking(9) is a method for reducing the effects of inhomogeneity in the DC magnetic field. The spin locking experiment would be written:

90X, lockY, observe.

The 90° pulse rotates M onto the Y axis as before. Immediately following the 90° pulse, a pulse of extended duration is applied along the Y axis. Since the axis of M and the axis of H₁ are aligned, no force is exerted upon M and M is essentially locked along the Y axis. During the locking pulse, normal relaxation processes are allowed to occur until which time the locking pulse is turned off and the initial amplitude of the resulting FID is sampled. This experiment is no more complicated than the T₁ or spin-echo experiment except that it places some additional requirements on pulse width settability. A typical 90° pulse may vary in length from 1 to 10 microseconds while the spin locking pulse may be as long as several seconds. This places a dynamic range requirement on the pulse width controls of 6 to 7 orders of magnitude.

Multiple Pulse Experiments

NMR studies on solids create a whole new set of problems for the spectroscopist. Dipole interactions between neighboring nuclei cause a line broadening which often masks out

the NMR line of interest. Pulse sequences which are designed to narrow lines in solids have been developed only recently. Two such sequences are the Waugh-4 pulse (10) and the MREV-8 pulse experiment (11). The two experiments are written respectively:

$(90X, 2\tau, 90-X, \tau, 90Y, 2\tau, 90-Y, \tau)$ repeat N times.

and

$90X, \tau, (\tau, 90X, \tau, 90Y, 2\tau, 90-Y, \tau, 90-X, 2\tau, 90-X, \tau, 90Y, 2\tau, 90-Y, \tau, 90X, \tau, \text{observe})$ repeat N times.

Both experiments require four pulse channels, two timebases, and a looping function. The NMR signal is sampled for each execution of the loop. The loop count, N, is therefore dictated by the memory size of the data recorder which may range from 128 to 2048 words or more. Additional timing is required to optimally position the sampling point in one of the 2 sampling windows. The value of τ for these experiments is typically from 2 to 10 microseconds which implies that the minimum 2τ sampling window is 4 microseconds wide.

The MREV-8 pulse experiment has been modified to the extent that it requires a fifth pulse channel and a loop cycle which is 17 pulses long. A 180° pulse is placed at the end of 8 pulse sequence and the loop is extended to contain another 8 pulse cycle. The experiment may be written:

$90X, \tau, ((\text{MREV-8}), 180Y, (\text{MREV-8}), \text{observe})$ repeat N times.

When performing multiple pulse experiments a condition may arise known as the BLAHS or Base-Line Artifacts in High-resolution Spectra (12). Long bursts of rf pulses can induce eddy currents in the sample probe housing. These eddy currents then induce spurious but coherent signals into the sample coil giving rise to the blahs. The blahs, however, can be removed from a spectrum by appropriate data manipulation. If the first pulse of the MREV-8 pulse sequence for example is changed from 90X to 90-X, the sign of the resulting data will be inverted. The blahs will remain constant, unaffected by the data. If on alternate data scans the sign of the data is inverted and the data subtracted from the accumulated data, the blahs will systematically cancel while the data will add coherently. While this technique is most valuable for multiple pulse experiments, the principle can be applied to other experiments as well. It would be desirable, from the standpoint of experiment automation, for the pulse programmer to automatically change the phase of the first pulse on alternate scans. At the same time a flag should be sent to the data averaging system instructing it to add or subtract the next data. This technique will here in be referred to as phase flipping.

Automated T1 and T2 Measurements

Recall from the discussion of T1 and T2 experiments that the value of T1 and T2 is obtained from a log plot of initial

amplitudes verses τ . Students, graduate students in particular, are very receptive toward any kind of device that will do their research for them. It would be desirable then for the pulse programmer, working in conjunction with an appropriate data averaging device, to automatically increment the value of τ while the data averaging device stored the resulting amplitude data and performed the log calculation. This feature will here in be referred to as the auto-timebase.

In addition to generating pulse sequences, the pulse programmer is often required to control the timing of other peripheral equipment. As mentioned earlier, transmitters, receiver gating, and probe coupling circuits are obvious candidates for pulse programmer control. Another category of experiments known as diffusion experiments require the precise application of magnetic field gradient pulses interleaved with the normal rf pulses(13). Control of a field gradient pulser should logically come from the pulse programmer also. In short, the pulse programmer should not be designed to meet only the needs of specific experiments. A better solution would be to design a general purpose instrument with plenty of reserve to satisfy both present and future experimental requirements.

PULSE PROGRAMMERS

A review of the literature has revealed that pulse programmer designs follow basically two philosophies. The first is a hard-wired approach and the second a modular approach. The hard-wired pulse programmer, as the name suggests, is a hardware system that is wired to generate a particular pulse sequence (14,15,16). The pulse sequence is changed by either rewiring the hardware or by substituting a circuit board which is wired to generate the new sequence. The disadvantages of the hard-wired approach are obvious. It would certainly be inconvenient to plug and unplug circuit boards for every sequence change, to say nothing of the design and construction time necessary to generate a new sequence. The only advantage to the hard-wired approach is that a dedicated experiment can be set up with a minimum investment in pulse programmer hardware.

Modular programmers (17,18,19) are based on a system of discrete modules consisting of a main clock module and a series of channel modules. The main clock module contains a master clock oscillator and a counter-timer which sets the experiment repetition rate. A typical channel module contains the pulse width controls for one pulse channel and, depending on the design, up to four separate digital counters. One counter is a series of decade dividers used to scale down the master clock. A second counter is used to set

the number of pulses the channel module will produce. A third counter is used to time the interval between the pulses, and the fourth counter is used to time the start of the next channel module in the series. Figure 6 depicts the basic layout of a modular programmer with four channels. The first channel is started by the main clock module and each successive channel is started by its predecessor. While the modular programmer is a definite improvement over the hard-wired type, it still has some drawbacks; not the least of which is operational in nature. Most of the timing parameters for the modular programmer are entered through thumbwheel and rotary type switches. With a four channel programmer, the operator may be confronted with up to 18 sets of switches, possibly leading to confusion. Modular programmers would appear to be more costly to build than other types due to the duplication of hardware in each channel module and the added expense of packaging each module separately. Modular programmers, while well suited for T1, T2, and Carr-Purcell type measurements, generally require additional hardware modules to control the recycling needed for multipulse experiments. With the addition of still another counting module, the auto-timebase advance feature can be implemented(20). In theory, the modular programmer can be extended to almost any length to provide enough counting modules and pulse channels for a particular experiment. In

Figure 6. Block diagram of a modular pulse programmer

practice, the extended modular system involves rather complicated external switching and interconnections which may be confusing to the inexperienced user.

SPECIFICATIONS

The remainder of this thesis describes a pulse programmer design which circumvents the problems associated with hard-wired and modular programmers. The first task is to define the specifications which will serve as guidelines for the instrument design. These specifications fall into two categories; operational and instrumental.

A primary goal in this project was to simplify operation by eliminating most of the parameter entry switches associated with modular programmers. To that extent, pulse sequence parameters such as pulse channels, timing, and sequencing will be characterized in an easily understandable programming language. The ease with which a pulse sequence can be changed is also of major importance. Tuning the MREV-8 pulse experiment, for example, may require the application of many simple but unique tuning sequences prior to application of the MREV-8 itself. This tuning procedure should be as convenient as possible. It would also be convenient, from an operational standpoint, to have all the signals leaving the pulse programmer available to the operator at a common monitor point. Experience has shown this information to be helpful in setting up specific pulse experiments. Lastly, the pulse programmer should be capable of producing all the major pulse sequences plus providing enough reserve to accommodate new experiments.

Farrar and Becker (21), in chapter 3, describe a set of instrumental specifications for an "All Purpose Unit" usable for high resolution studies in liquids and solids. They state that the pulse widths should have a range from 1 microsecond to 10 milliseconds or longer and that the widths be controllable to a precision of a few percent or better. The time between the pulses should be adjustable from 10 microseconds to 1000 seconds or more, and this time should remain constant to a precision of one part per million or better. The experiment repetition rate should be adjustable between 200 microseconds to about 20 minutes or more. Lastly, the pulse programmer must supply pulses for triggering or starting any sampling or recording devices such as an oscilloscope or signal averaging device.

Actual practice has shown that these specifications tend to be accurate and usable. In most cases, the time between the pulses should be adjusted down to a range of 1 microsecond to 100 seconds, the lower limit being the most useful of the two extremes. In light of all previous discussion concerning pulse programmer requirements, and with application of foresight into the future needs of NMR experiments, the following list of hardware specifications was developed for a versatile NMR pulse programmer.

(a) There will be four continuously variable pulse channels to be designated P1, P2, P3, and P4. Each pulse channel will

have a total pulse width range of 0.5 to 80 microseconds divided into six coarse ranges. Each of the six coarse ranges will be continuously variable over a 4 to 1 ratio. All four pulse channels can be scaled together over a 15% range with a continuously variable scale vernier. Table 1 shows the exact ranges obtainable for each coarse setting with the scale control set to midscale.

Table 1. Pulse width range on channels P1-P4

Coarse position	Range (microseconds)
<hr/>	
1	0.5 - 2
2	1 - 4
3	2 - 8
4	4 - 16
5	10 - 40
6	20 - 80

The decision to limit the maximum pulse width to 80 microseconds is not without basis. For the majority of pulse sequences encountered, either 90° or 180° pulses are used which places the pulse widths in the range of 1 to 20 microseconds. On the other hand, the spin locking experiment requires a locking pulse of up to several seconds in duration. Rather than build a pulse width circuit that will function linearly over six decades, the decision was made to

handle the long pulses in a digital fashion as described in specification C.

(b) There will be a single programmable timebase capable of producing eight different presettable time delays designated D1 through D8. The timebase will control all time related parameters in a pulse sequence. The timebase will have a range from 1 microsecond to 100 milliseconds. A 10MHz TTL square wave will be required as the timebase reference signal.

(c) There will be four level channels designated L1, L2, L3, and L4. These channels are termed levels because they are turned on or off under program control in contrast to the triggered monostable response of the pulse channels. The duration of a level will be controlled digitally using the programmable timebase. The levels are intended to accommodate experiments where pulses longer than 80 microseconds are required.

(d) There will be three trigger channels labeled T1, T2, and T3. The triggers will be short pulses (less than 1 microsecond) to be used for triggering oscilloscopes and data acquisition equipment.

(e) There will be a timer designated the Slow Clock which will control the experiment repetition rate. The slow clock will be set manually with a range of 10 milliseconds to 1000 seconds. There will be provision for starting and stopping

the slow clock either manually or remotely.

(f) There will be a set of controls designated Delay and Integrate used to manually position the sampling window and control the sampling time for multipulse experiments. The sampling sequence will be started under program control.

(g) There will be an auto-timebase advance function to be used with automated relaxation time measurements.

(h) There will be an automatic phase flipping function for use in baseline artifact cancellation schemes. A sign bit will be provided for a remote data averaging device such that the appropriate sign can be applied to the data.

(i) There will be a looping function provided such that all or part of a pulse sequence can be repeated up to 4000 times.

(j) Operator communication with the pulse programmer will be through a 20 milliamp serial line and any ASCII terminal.

This concludes the list of pulse programmer specifications. The next section deals with a discussion of hardware implementation.

IMPLEMENTATION

The design approach chosen here differs from any of the programmers referenced in this paper. At the time this project was conceived, microprocessors were rapidly moving into new instrument designs. A microprocessor based pulse programmer seemed like an effective solution to the parameter entry problem associated with other programmers. The decision to incorporate a microprocessor in an instrument drastically affects the design philosophy. Functions that would normally be built in hardware can be replaced by appropriate software provided that the microprocessor can meet the speed requirements of the system. The advantages of this approach are reduced hardware complexity and enhanced system flexibility. The "Smart" instrument can give the user features that might otherwise be cost restrictive in a strictly hardware design.

The philosophy employed in this design is not unlike that used in modern computer processor designs. The idea was to develop a set of general purpose microprogrammable hardware capable of producing, at any instant, all of the rudiments of NMR pulse sequences. Pulses, triggers, time delays, looping, integrate and hold timing, and the slow clock are functions considered to be a part of the hardware. The specific application and overall versatility of the instrument is determined by the microprocessor software

routines which service the hardware.

A block diagram of the pulse programmer implementation is shown in figure 7. The microprocessor communicates with the programmer hardware through a 16 bit parallel data output port and 5 control lines. The sequencer is the heart of the pulse programmer. It controls the gating of all pulses, levels, and triggers, controls microprogram sequencing, looping, and timebase addressing. Program execution is performed at an asynchronous rate as determined by the timebase. The timebase is a single hardware unit capable of producing 8 different presettable time delays. When started by the slow clock, the timebase produces the appropriate delays, as addressed by the sequencer, until it is stopped by a system clear signal from the sequencer indicating the end of a program. The slow clock is a manually settable counting module which controls the experiment repetition rate. The slow clock is driven from an external 10 MHz source and can be started either manually or remotely. The pulse width control circuit takes the fixed length pulses from the sequencer and produces pulses of continuously variable width at the outputs. Care was taken in the design of the pulse width circuit to insure stability of the pulses. The integrate and hold control is the only circuit designed for a specific application. Its purpose is to generate a variable delay and control signals for an integrate and hold module. Its appli-

Figure 7. Block diagram of pulse programmer implementation

cation is limited primarily to experiments in which the magnetization is repeatedly sampled in a window between pulse sequences, e.g. the MREV-8 experiment. The signal buffers are used to protect the pulse programmer from the user. The signals available to the user at the output monitor are isolated by the buffers thereby reducing the affect of an occasional improper termination, i.e. a short circuit.

The next few sections give a more detailed description of the individual system blocks.

Microprocessor

The microprocessor incorporated in the pulse programmer is a Motorola¹ M6802. There is nothing unique about the M6802 that makes it better suited to this application than any other general purpose chip. The decision to use the M6802 was based entirely upon hardware availability and software support. All programming was accomplished using cross compilers and cross assemblers purchased from Motorola which run at the ISN Computation Center. The microprocessor system, shown in figure 8, was assembled at the board level using boards designed and built by the Ames Laboratory Instrumentation Group.

The CPU board contains the M6802 CPU chip, timing

¹Motorola Semiconductor Products Inc., Box 20912, Phoenix, Arizona 85036

Figure 8. Microcomputer system

crystal, address and control bus buffers, and the power on reset circuitry. The data bus is left unbuffered in this system. The read only memory (ROM) board is used to store the operating system and a program library. The board has capacity for 8K bytes of EPROM only 4K of which are used in this system. The random access memory (RAM) board is used to store program variables and temporary data. Like the ROM board, the RAM board has capacity for 8K bytes but only 2K are installed in the system. The console terminal interface was constructed around the M6850 asynchronous communications interface chip. The board will drive an I/O terminal device through an optically isolated 20 milliamp current loop at baud rates of 110 or 300. The parallel interface was realized using a M6820 parallel interface adapter chip. The M6820 has 16 data lines and 4 control lines available to the user. In this application, the 16 data lines are configured in software to act as outputs for driving the pulse programmer data bus. Of the 4 control lines, one is used as a done flag from the sequencer, one is used to enable the slow clock, a third is used to strobe the parallel data into the sequencer, and the fourth is used as a restart interrupt request line. A restart interrupt may occur either from the operator or a remote data acquisition device and signals the software to reinitialize any variable parameters. The least significant data line doubles as the INVERT bit. The INVERT bit, used

with phase flipping experiments, becomes the sign bit for a remote data acquisition device.

Sequencer

The sequencer is a microprogrammable logic system containing a writable control store. The microcode representing a pulse sequence, as generated by the microprocessor, is loaded into the sequencer micromemories. The sequencer is responsible for execution of the microcode, generation of all pulse level and trigger signals, and management of other system control signals. The sequencer is also responsible for addressing and writing its various micromemories at load time.

Figure 9 shows a signal flow diagram for the sequencer circuit. The circuit operation can be divided into two modes; program execution and program loading. Loading is carried out in the following manner. The microprocessor first clears the ENABLE line causing the interface control logic to issue a system clear signal, SYSCLEAR. This stops the timebase, if it is running, and clears the main memory address register on the sequencer. Also, when ENABLE is low, the loop/jump address multiplexer and timebase address multiplexer are selected such that the main address bus is connected to the loop/jump address bus and the timebase address bus respectively. Now the main memory address register and memory write control can direct the loading of the

Figure 9. Sequencer circuit

microprogram memory, the timebase memory, the loop count memory, and the jump address memory in that order. The microprogram memory holds the executable instructions while the loop count memory and jump address memory hold only information pertinent to controlling program loops. The timebase memory is located on the timebase circuit discussed in the next section. The microprocessor places the data for the appropriate memory on the 16 bit data bus and pulses the write enable line (WE). WE passes through the interface control logic to the memory write control. There the main address bus is decoded and WE is ultimately passed to the write line (R/W) of the selected memory. The trailing edge of WE becomes the clock for the main memory address register causing an address increment after each memory write operation. After the last memory write operation, the microprocessor sets ENABLE which again produces a SYSCLEAR, preparing the sequencer for program execution. Notice that since no provision has been made for random micromemory access, program loading must take place in a sequential fashion starting at address zero. Also, program execution always begins at address zero. The microprogram memory can hold a maximum of 256 instructions and can be jumpered to select either 32, 64, or 128 if all 256 instructions are not needed.

Program execution reflects the fact that NMR pulse sequences tend to be an asynchronous string of pulses, each

pulse representing an event in time. Also, certain groups of pulses may be repeated many times as is the case with multiple pulse experiments. For this reason, the sequencer requires only three types of instructions: a Continue, a Jump, and a Halt instruction. The Continue instruction produces sequential program execution. The Jump instruction provides the backward jumping necessary to generate program loops. The Halt instruction, of course, is used to terminate a sequence of instructions. Figure 10 shows the instruction format for the control words contained in the microprogram memory. Bits 0 through 10 make up the data field while bits 11 through 15 make up the control field. The data field is subdivided into three groups designated pulse channels, level channels, and trigger channels. A bit set in the data field produces a corresponding output when the instruction is executed. For example, if bits 1, 4, and 8 were set in a given instruction, then execution of that instruction would produce an output on pulse channel 2, level channel 1, and trigger channel 1 respectively. Refer again to figure 9. The sequencer is clocked asynchronously by the two phase timebase inputs TB1 and TB2. Upon receipt of TB1, the data portion of the current instruction is gated through the pulse gates and the trigger gates. The outputs of the pulse gates AEN, BEN, CEN, and DEN are enable signals which go to the pulse width control circuit. The output from the trigger gates, TRIG1,

of

Figure 10. Instruction format

TRIG2, and TRIG3, go directly to the output connectors and are available to the user as programmable trigger signals. The level channels (bits 4 through 7 of the control word) are latched in the level latch on the rise edge of TB1. The outputs of the level latch become the programmable levels L1, L2, L3, and L4. As long as a level bit is set in the current instruction, the corresponding level outputs will remain high. This then is the mechanism by which extended pulse lengths or "Levels" are produced. The fall edge of TB1 increments the main memory address register thereby accessing the next instruction in the microprogram memory.

Within the control field of each instruction, bits 11, 12, and 13 form a three bit timebase address. During execution, the timebase address multiplexer is selected such that this three bit address field is passed to the timebase on the address lines DA0, DA1, and DA2. Although the timebase circuit has not yet been discussed, it is sufficient to say that selecting a particular timebase address with the current instruction will produce a subsequent programmed time delay. At the end of that delay time, TB1 will be produced causing execution of the next instruction. It is important to understand that each instruction marks one event in time and that time is measured by selecting one of 8 time delays to follow each event.

The address source for the next instruction is determined by bit 14, the Jump bit. If bit 14 is low, then the instruction is a Continue instruction and the next sequential instruction will be accessed. If bit 14 is high, the instruction is a Jump instruction and the loop and jump circuits shown on the left half of figure 9 come into play. The sequencer is capable of executing up to eight sequential program loops, that is, no nesting of loops is allowed. Each program loop requires three pieces of information to be stored: the starting address of the loop, the loop iteration count, and the ending address of the loop. The starting address and the iteration count are compiled by the microprocessor and loaded into the jump address memory and loop count memory respectively. The ending address of the loop is inherent in the Jump instruction itself.

Execution of a program loop proceeds as follows. The SYSCLEAR signal that follows program loading sets up the initial conditions for the first program loop. The loop/jump memory address multiplexer is selected such that the loop/jump memory address register is connected to the loop/jump address bus. The loop/jump memory address register is cleared thereby accessing the first word in the loop count and jump address memories. The SYSCLEAR signal passes through the loop/jump control circuit to the Load line on the 12 bit loop counter which loads the first iteration count

into the loop counter. As program execution proceeds, the condition of the Jump bit is latched in the loop/jump control on TB1. If the Jump bit is high and the loop count is not zero, then TB2 passes through the loop/jump control to decrement the loop counter by one. At the same time the output of the jump address memory is transferred to the main memory address register thereby forcing the jump. When the loop counter finally does go to zero, the loop/jump memory address register is incremented by one accessing the loop count and jump address data for the next loop. This time when the Jump instruction is executed, the loop counter is zero and the jump is not taken. Rather the next sequential instruction is accessed on TB1 and TB2 is passed to the load line on the loop counter. The loop counter is loaded with the new output of the loop count memory establishing the initial conditions for the next program loop.

Bit 15 set in the current instruction indicates a Halt instruction. A Halt instruction should normally not contain any bits set in the data field. When a Halt instruction is detected by the interface control logic, a SYSCLEAR is produced and the FLAG is set. The FLAG indicates to the microprocessor the end of sequence. SYSCLEAR, recall, clears all address registers on the sequencer and stops the timebase halting the pulse programmer.

Timebase

The timebase is a programmable high speed counting module capable of generating, at any instant, one of eight programmed time delays. Its ability to do so eliminates the need for multiple discrete counters as with modular programmers. The programmable nature of the timebase also provides the mechanism by which the auto-timebase advance function may be implemented.

Figure 11 shows a signal flow diagram of the timebase circuit. Inputs to the timebase are bits D0 through D11 of the 16 bit data bus, address lines DA0 through DA2, a write line TBWE, START, SYSCLEAR, and the 10MHz reference frequency. The output of the timebase is the two phase clock, TB1 and TB2, which drives the sequencer circuit. Under direction of the sequencer, timebase data is loaded into and read from an 8 word by 12 bit register file. At load, time each register is sequentially accessed with address lines DA0 through DA2. The microprocessor places the appropriate data on the data bus and pulses TBWE low storing the data in the register file. At run time, the timebase address specified in each sequencer instruction is placed on the address lines. The selected timebase data word then appears at the outputs of the register file where it is used to produce a subsequent time delay. Figure 12 shows the format for the timebase data

PULSE PROGRAMMER DATA BUS

Figure 11. Timebase circuit

DECADE MULTIPLIER

BIT 11	BIT 10	MULTIPLIER
0	0	$\times 10^{-4}$
0	1	$\times 10^{-5}$
1	0	$\times 10^{-6}$
1	1	$\times 10^{-7}$

Figure 12. Timebase data word format

words. Bits 0 through 9 form a 10 bit binary count yielding a range of 0 to 1023 counts. Bits 10 and 11 are demultiplexed, as shown by the truth table in figure 12, to select one of four decade multipliers ranging from $10E-4$ to $10E-7$.

To build an N modulo counter that will count at a 10MHz rate using standard TTL circuits is not a trivial job. The problem, of course, is purely one of propagation delay around the digital feedback loop. To avoid this problem, the timebase was designed to operate in two states: a count state and a hold state. The hold state lasts for 1 microsecond and allows ample time for loading the 10 bit counter and selecting the decade multiplier. The hold state, being fixed in length and synchronous with the 10MHz reference, can be easily corrected for in software by subtracting 1 microsecond from the normal count period. This implies that the minimum obtainable timebase period is 1 microsecond.

To describe the two state operation, assume that the timebase is initially in the hold state which is the case prior to receiving a START pulse. When the start pulse arrives, the clock sync circuit synchronizes the state control circuit with the 10MHz clock to insure that the first clock pulse processed by the state control is a complete pulse. The state control routes the 10MHz clock pulses to the hold counter. The hold counter is a decade counter

requiring 1 microsecond to count through all ten states. The overflow output of the hold counter, HOLD DECODE, changes the state control from the hold state to the count state. The BCD outputs of the hold counter are decoded with count state 2 and 4 becoming the two phase clock outputs TB1 and TB2 respectively. Each of the two clock phases lasts 100 nanoseconds with 100 nanoseconds between them. TB1 is used internally by the timebase to load the 10 bit counter and multiplier latch and clear the 3 multiplier decades. Clearing the multiplier decades prior to each count state insures constant initial conditions for the count state.

When the state control changes to the count state, the 10MHz clock pulses are routed to the COUNT CLOCK line. Based on the data held in the multiplier latch, the demultiplexer-selector passes to the 10 bit counter, either the count clock or the count clock scaled by the appropriate decade. Overflow of the 10 bit counter is signaled by the COUNT DECODE line causing the state control to return to the hold state, completing the cycle.

The 10 bit counter and multiplier latch double as a pipeline register for the 8 by 12 register file. The data at the output of the register file is latched in the 10 bit counter and multiplexer latch on the trailing edge of TB1. Recall that the trailing edge of TB1 also advances the main memory address register on the sequencer circuit. The new

timebase data is stable at the outputs of the register file only after a time equal to the main memory address register settling time plus the microprogram memory access time plus the register file access time. The pipeline approach allows for a total access time equal to the worst case timebase period of 1 microsecond. It was important to select components in this critical speed path such that the sum of the propagation delays and access times were less than 1 microsecond.

Slow Clock

The slow clock is a counter/timer used to control the experiment repetition rate and also provide several reference frequencies for other outboard equipment. The slow clock period is manually programmed via a two digit thumbwheel switch and a four decade multiplier switch providing a range of 10 milliseconds to 1000 seconds. The slow clock rate is typically not an experimental variable; therefore software programmability did not seem necessary.

Figure 13 shows a signal flow diagram for the slow clock. The slow clock, like the timebase, is driven from the 10MHz reference frequency. The 10MHz reference is divided down to 0.1 Hz by a string of 8 decade dividers. The output of the first decade provides a 1MHz reference for an outboard frequency synthesizer, and the second decade produces a 100KHz reference for a magnetic field stabilization unit.

Figure 13. Slow clock

The outputs of the last four decades provide a 100Hz, 10Hz, 1Hz, or 0.1Hz reference for the two decade BCD counter on the slow clock. The COUNT input to the BCD counter is selected from the last four decades using a four position rotary switch. When the LOAD line goes low, data held in the thumbwheel switch is parallel loaded into the counter. The counter then counts down until zero is reached, at which time the zero detect circuit pulses the ZERO line high. The ZERO pulse passes through gate B to become the LOAD pulse and, if the ENABLE line is high, through gate C to become the START pulse.

The slow clock can be started either from the manual switch labeled RUN, or remotely using the COMRUN line. If both RUN and COMRUN are low, then the output of gate A, RESET, will be high, LOAD will be low, and the last four decades in the reference divider string will be held in the reset state. This insures that when either COMRUN or RUN goes high, one full slow clock period will always pass before the first START pulse occurs.

Pulse Width Control

The purpose of the pulse width control circuit is to produce continuously variable pulse widths at the P1, P2, P3, and P4 outputs. Each pulse channel is triggered by its respective enable signal AEN, BEN, CEN, and DEN originating at the sequencer circuit. Figure 14 is a block diagram of the

Figure 14. Pulse width control circuit

pulse width control circuit. Each of the four monostable circuits can be manually enabled or disabled by the switches AON, BON, CON, and DON. These switches, located on the pulse programmer front panel, allow the operator to quickly manipulate the four pulse channels without rewriting the pulse sequence. The signals AEN, BEN, CEN, and DEN are ANDED with the manual enable switches to form the trigger inputs for the monostables. The monostables used are standard TTL one-shots with special precautions taken to assure pulse width stability and reproducibility. The Vcc supply voltage for the monostables is derived from a separate voltage regulator whose purpose is to provide a stable Vcc free from digital noise. The RC timing network is also supplied from a separate regulator. This voltage, Vpw, is variable allowing the operator to scale all four pulse channels together over a 15 percent range. The variable resistors, R, are precision 10 turn potentiometers and the variable capacitors, C, are actually 6 position rotary switches mounted with fixed value temperature stable ranging capacitors. In this way, the 6 coarse ranges and 4 to 1 pulse width vernier described in the specifications are realized.

Integrate and Hold

The integrate and hold control circuit was included in the pulse programmer for the specific purpose of controlling an external integrate and hold module used for sampling in

multiple pulse experiments. Figure 15 shows a block diagram and associated timing diagram for the circuit. The actual integrate and hold module is assumed to be a separate device, not a part of the pulse programmer proper. The integrate and hold cycle is started by the signal TRIG3 from the sequencer. TRIG3 is produced as a result of executing a command statement containing a T3 parameter. Upon receipt of TRIG3, a variable delay time of 1.8 to 18 microseconds is initiated. This delay allows the operator to precisely position the integrator aperture within a desired sampling window. Following the delay time, a variable integration time period of 0.8 to 8 microseconds is started. Coincident with the start of the integration period, the RESET line to the integrator is released and the INTEGRATE line is activated. At the end of the integration period, the INTEGRATE line is deactivated and a hold delay of 0.5 microseconds is started to allow for integrator settling. After the hold delay, a 0.75 microsecond pulse, RECORD, is issued which is used as a strobe pulse for a data acquisition device. The trailing edge of RECORD sets the RESET line clearing the integrator and completing the integrate and hold cycle.

Table 2 is a summary of the pulse programmer input and output signals which are accessible to the user.

Figure 15. Integrate and hold control

Table 2. Pulse programmer input and output signals

Name	Location	Polarity	Function
P1	Rear BNC	True Low	Output pulses
P2	"	"	
P3	"	"	
P4	"	"	
L1	Rear BNC	True Low	Output levels
L2	"	"	
L3	"	"	
L4	"	"	
T1	Rear BNC	True Low	Output triggers
T2	"	"	
T3	"	"	
10 MHz	Rear BNC	TTL square	Input reference frequency
1 MHz	"	"	Output reference frequency
100 KHz	"	"	Output reference frequency
Comrun	Rear J1-F	True High	Remote run input
Comrst	Rear J1-D	True High	Remote restart input
Invert	Rear J1-E	True High	Output sign bit
Int	Rear J2-E	True High	Integrator control output
Reset	Rear J2-D	True High	Integrator reset output
Record	Rear BNC	True Low	Data strobe output
Monitor	Front BNC	True high	Monitor outputs

PROGRAMMING LANGUAGE

The programming language developed for the pulse programmer was designed to be simple to use and allow full flexibility of the hardware. The elements of the language closely parallel the structure of NMR pulse sequences, hopefully making programming easier to understand. The language allows the user to manipulate all the hardware parameters by using the following symbols:

Dn	n = 1,2,3,....or 8	Delay Times
Tn	n = 1,2, or 3	Trigger Channels
Pn	n = 1,2,3, or 4	Pulse Channels
Ln	n = 1,2,3, or 4	Level Channels
	m = 0 for level off	
	m = 1 for level on	

The delay times are the values assigned to the programmable timebase. A delay time is set by using an assignment statement of the form:

$$Dn = xE-y$$

which is interpreted as $Dn = x \cdot 10^{-y}$ seconds. X must be an integer between 1 and 1023, and Y may be 4, 5, 6, or 7. No imbedded blanks may appear in a delay assignment statement. The smallest resolvable value of Dn is 1 microsecond; therefore Dn can range in value from $10 \cdot 10^{-7}$ seconds to $1023 \cdot 10^{-4}$ seconds. Notice that a time resolution of 100 nanoseconds is obtainable in the range of $10E-7$ to $1023E-7$.

Some examples of delay times are:

D1 = 1 millisecond can be programmed as D1=10E-4.

D3 = 50 microseconds can be programmed as D3=50E-6.

In order to compensate for the 1 microsecond hardware delay inherent in the timebase, any delay time programmed with an exponent of -7 or -6 will be decremented in software by 1 microsecond. It is important to note that the statements D1=1E-5 and D1=10E-6 are therefore not equivalent. The former will result in an effective delay time of 11×10^{-6} seconds while the latter will result in an effective delay time of 10×10^{-6} . While this apparent ambiguity is not very pleasing, its affects can be minimized if one is knowledgeable that the situation exists.

When power is applied, the pulse programmer comes up in the text editor ready to accept input. Programs will be written as a sequence of command statements and assignment statements. All statements in a program must be terminated with a carriage return. A command statement consists of a string of juxtaposed parameters. Blanks may be inserted between parameters to aid in readability. An example of a command statement is:

P1 T3 D2

Each command statement represents one time event. That is, every parameter used in the command statement is initiated at the same instant in time. A delay parameter must be speci-

fied in each command statement and marks the time from the current event to the start of the next event. If no delay parameter is specified, D1 will be used by default. As an example, suppose we wanted to program the following sequence of events:

Issue a pulse on channel P1. After a delay of D1, signal level L2 to go high. After a delay of time D2, start a data recorder by issuing trigger T1. After a delay of time D3 signal level L2 to go low again.

This sequence is shown diagrammatically as follows:

The command statements which would generate this sequence are:

```
P1 D1
L21 D2
T1 D3
L20
```

Note that level L2, which was turned on in the second command statement, will stay high until signaled to go low in the fourth command statement. As a precaution against damaging transmitters and probes, the sequencer will turn off any levels that were inadvertently left on at the end of a se-

quence.

Command statements can be contained within loops. Looping is specified by starting the first command statement in the loop with a left parenthesis and ending the last statement in the loop with a right parenthesis followed by a loop count. The loop count is an integer between 2 and 4095 which indicates the number of times the statements in the loop are to be executed. An example of a program with a loop is:

```
P1 D2
(P2 D1
P3 D1) 500
P1 T1
D1=5E-6
D2=25E-6
```

There may be no blanks between the right parenthesis and the loop count. A program may contain a maximum of eight loops and no nesting of loops is allowed.

There are presently two variable parameters available with the pulse programmer. They are the auto-timebase advance feature and pulse channel invert, or phase flipping. The auto-timebase advance can be used to automatically increment any or all of the delay parameters D1 through D8. The auto-advance feature is called by appending a plus sign to the end of a delay assignment statement followed by an increment value between 1 and 255. For example, if you wanted D4 to be initially 10 microseconds and advance by 5 microseconds after each scan, you would write:

$$D4=10E-6+5$$

D4 will start at 10E-6 on the first scan and advance to 1020E-6 after 202 scans. After the next scan, the mantissa will exceed 1023 so the monitor will issue a restart and D4 will again become 10E-6. As mentioned previously, this feature should be useful in making automated relaxation time measurements.

The pulse channel invert function is used to alternately modify any of up to eight command statements containing pulses P1 through P4. The function is called by entering an I in the command statements you wish to have modified. On alternate scans, the monitor will change any P1's to P2's and P3's to P4's in the command statements containing an I. By connecting the pulse channels to the appropriate RF phases, the invert feature can be used to produce baseline artifact cancellation sequences.

OPERATING SYSTEM

The pulse programmer operating system consists of three parts: a text editor, a translator, and a run time monitor. The text editor¹ allows the user to create or modify a program written in pulse programmer format. The editor also supports a program library which is permanently stored in read only memory. The library feature allows the user to call up a stored program using only a single command. For more information concerning the use of the text editor, see appendix A.

The translator is a subroutine that converts the source program in the editor buffer into microcode executable by the sequencer hardware. The translator checks the program for proper syntax and parameter ranges. If an error is detected by the translator, the line number of the error is printed out followed by a question mark (?). The translator then returns control to the text editor so that the error can be corrected. If the translator was successful, the microcode is written out to the sequencer hardware through the parallel I/O port. The microprocessor enables program execution by setting the ENABLE bit; control is then passed to the run time monitor.

¹M6800 text editor, written by John Geske, Ames Laboratory Computer Services

The monitor watches for one of three events to occur: keyboard input from the user, a restart interrupt, or a done flag from the sequencer. If the operator starts typing on the keyboard, it is assumed that he wants to edit the source program. In that case, the ENABLE bit is cleared, stopping the pulse programmer, and control is returned to the editor. A restart interrupt can come in either from the restart push button on the instruments front panel, or from a remote signal averaging device. In either case, the response is to reinitialize any variable parameters to their initial values as specified in the source program. Reinitialization is done by simply calling the translator which recompiles the original source program thus restoring the initial conditions. The ability to remotely reinitialize variable parameters makes possible automated signal averaging of entire spectra as opposed to single points. For example, an automated measurement of T1 could be performed in two ways: one way would be to signal average individual points on the T1 curve and then plot the curve from the accumulated points. The other method is to acquire the entire T1 curve as quickly as possible and then average successive curves. The latter approach is the most preferable from the standpoint of minimizing experimental errors due to long term equipment instabilities. The ability for a remote data averaging device to reinitialize pulse programmer variables makes automated

measurements of this type easily realizable.

The done flag, when set by the sequencer logic, indicates to the monitor the end of program execution. The monitor then checks to see if any variables need updating. If so, the appropriate modifications are made on a copy of the microcode kept in the microprocessor memory, and the new code is written out to the hardware. The microcode copy maintained in memory is organized as shown in figure 16. The first four tables hold the actual code to be sent to the hardware, and the last two tables hold auto-timebase and phase flip information for the run time monitor. As the translator goes about its business of translating the source program, it builds the appropriate tables in memory. Upon detecting the end of the source program, the translator inserts a halt instruction in the first unused word in the instruction table. The entire contents of the first four tables are then written sequentially to the hardware.

The instruction table holds the 16 bit instructions for the sequencer microprogram memory. This table has a maximum length of 256 words and will vary in length dependent upon the number of words selected on the sequencer board. If all 256 instructions are not needed, the sequencer can be jumpered to select either 128, 64, or 32 instructions. By exercising this option, the microprocessor memory size can be reduced along with the time required to write the microcode

Figure 16. Microcode tables

to the hardware.

The timebase table holds the count and decade multiplier data for the timebase circuit. If a timebase is to be modified by the auto-timebase function, the translator will place the incremental value in the corresponding location in the timebase increment table. After each scan, the monitor examines the timebase increment table for nonzero entries and adds these nonzero values to the corresponding locations in the timebase table. This is done on each scan until a timebase overflow occurs, at which time the translator is called and the source program is recompiled, restoring the initial timebase values.

The loop count table holds the loop counts for up to eight separate program loops. The corresponding entry in the jump address table holds the address of the first instruction of each loop. In the hardware, the loop count and jump address information are utilized in parallel: that is, each time a branch is made to the address specified in the jump address table, the associated loop count is decremented. When the loop count goes to zero, the next jump address and loop count data are accessed to define the next program loop.

The phase flip table holds pointers to the instructions which require the invert function. On alternate scans, the monitor will modify the flagged instructions such that P1's become P2's and P3's become P4's. If the pulse channels P1

through P4 are connected to the RF phases X, -X, Y, and -Y respectively, then the phase flipping function will invert the phases of X and/or Y on alternate scans.

Flow charts for the translator and run time monitor are included in appendix B.

SUMMARY AND CONCLUSIONS

In this report, a description of the sequence of events and the decisions leading to the design of a versatile pulse programmer for pulsed NMR were presented. Background and application information was discussed in order that the reader might better understand the role of the pulse programmer in a NMR spectrometer. Various other design approaches were presented as a basis for comparison. Specifications for this design were proposed, the hardware implementation of the specifications was discussed, and the software operating system was presented.

Currently, there are three pulse programmers of this design in service in the Ames Laboratory. The three units are functionally identical and differ only in the microprocessor implementation. The prototype unit, constructed in 1976, utilized a microprocessor system produced by Pro-Log Corporation.¹ The two most recent units, constructed in 1978, use the microprocessor system designed and built at the Ames Laboratory. Complete circuit diagrams for the pulse programmer and microprocessor system are available from the Ames Laboratory Instrumentation Group.

The pulse programmers are used daily by graduate and

¹Pro-Log Corporation, 2411 Garden Road, Monterey, California 93940

post graduate students in conjunction with their research. Experiments ranging from the simple free induction decay to state of the art multipulse measurements have been successfully performed using the pulse programmer. The ease with which a new pulse sequence may be set up allows the student to test his theories with a minimum of time and trouble. The design has proven to be highly reliable, easy to use, and a very effective solution to the pulse spectrometer control problem.

BIBLIOGRAPHY

1. Bloch, F. "Nuclear Induction." Physical Review 70, No. 7 (October 1946): 460-474.
2. Clark, Gilbert W. and McNeil, John A. "Single Coil Resonant Circuit for Pulsed Nuclear Resonance." Review of Scientific Instruments 44, No. 7 (July 1973): 844-851.
3. Kisman, Kenneth E. and Armstrong, Robin L. "Coupling Scheme and Probe Damper for Pulsed Nuclear Magnetic Resonance Single Coil Probe." Review of Scientific Instruments 45, No. 9 (September 1974): 1159-1163.
4. Hoult, D. I. and Richards, R. E. "An Ultra High Frequency Receiver Protection Scheme." Journal of Magnetic Resonance 22 (1967): 561-563.
5. Grannell, P. K.; Orchard, M. J.; Mansfield, P.; Garroway, A. N. and Stalker, D. C. "A FET Analogue Switch for Pulsed NMR Receivers." Journal of Physics E: Scientific Instruments 6 (1973): 1202-1204.
6. Lowe, I. J. and Tarr, C. E. "A Fast Recovery Probe and Receiver for Pulsed Nuclear Magnetic Resonance Spectroscopy." Journal of Scientific Instruments 1 (1968): 320-322.
7. Hahn, E. L. "Spin Echoes." Physical Review 80, No. 4 (November 1950): 580-594.
8. Carr, H. Y. and Purcell, E. M. "Effects of Diffusion on Free Precession in Nuclear Magnetic Resonance Experiments." Physical Review 94, No. 3 (May 1954): 630-638.
9. Deverell, C.; Morgan, R. E. and Strange, J. H. "Studies of Chemical Exchange by Nuclear Relaxation in the Rotating Frame." Molecular Physics 18, No. 4 (1970): 553-559.
10. Waugh, J. S.; Huber, L. M. and Haeberlen, U. "Approach to High Resolution NMR in Solids." Physical Review Letters 20, No. 5 (January 1968): 180-184.
11. Rhim, W. K.; Elleman, D. D. and Vaughan, R. W. "Enhanced Resolution for Solid State NMR." Journal of Chemical Physics 58, No. 4 (February 1973): 1772-1773.

12. Stejskal, E. O. and Schaefer, Jacob. "Removal of Artifacts From Cross-Polarization NMR Experiments." Journal of Magnetic Resonance 18, No. 3 (June 1975): 560-563.
13. Stejskal, E. O. and Tanner, J. E. "Spin Diffusion Measurements." Journal of Chemical Physics 42, No. 1 (January 1965): 280-292.
14. Franconi, C. and Terenzi, M. "A Programmed Digital IC Pulse Generator for Transient NMR Spectrometry." Review of Scientific Instruments 41, No. 3 (March 1970): 456-461.
15. Favret, A. G. and Meister, R. "Pulse Pattern Generator for Transient NMR Experiments." Review of Scientific Instruments 36, No. 2 (February 1965): 154-157.
16. Ellett, J.D. et al. "Spectrometers for Multiple-Pulse NMR." Advances in Magnetic Resonance 5 (1971): 117-176.
17. Conway, John L. and Cotts, R. M. "Circuit for a Digital Pulse Programmer." Review of Scientific Instruments 48, No. 6 (June 1977): 656-660.
18. Taylor, D. G.; Booth, S. and Allen, P. S. "A Flexible, Modular Pulse Programmer Suitable for Diverse Nuclear Resonance Experiments." Journal of Physics E: Scientific Instruments 7 (1974): 105-110.
19. Lind, A. C. "A Programmable Digital Pulser for NMR." Review of Scientific Instruments 43, No. 12 (December 1972): 1800-1803.
20. Adduci, D. J.; Hornung, P. A. and Torgeson, D. R. "Auto-increment Circuit for a Digital Pulse Programmer." Review of Scientific Instruments 48, No. 6 (June 1977): 661-663.
21. Farrar, Thomas C. and Becker, Edwin D. Pulsed and Fourier Transform NMR. New York: Academic Press, 1971.

ACKNOWLEDGEMENTS

I would like to express my appreciation to my wife Gini and my children Michael and Katie for their many sacrifices. I also wish to thank my parents, Mr. and Mrs. John Adduci, for their support. I would also like to extend my appreciation to Dr. Arthur Pohm, my major professor, for his guidance and to Dr. Bernard Gerstein for making this project possible.

APPENDIX A. EDIT 1.0 USERS GUIDE

EDIT 1.0 is a minimal text editor designed to run in 2K ROM. It is a line editor and has the usual edit commands Collect, Insert, Modify, Delete, and List. Two additional commands, Use and Run, were designed specifically to interface with the pulse programmer translator and program library.

In the normal command mode, EDIT prompts the user for a command with a question mark (?). In the text mode, EDIT prompts for user inputted text with an asterisk (*). Commands must immediately follow the prompt; no blanks are allowed in the command string. If an invalid command is given, the error message 'COMND?' is output. In all cases when an error occurs, EDIT outputs an error message and then returns to the command mode, printing a question mark in the first column of a new line. A description of valid commands and possible errors is given in the section on commands.

There are two restrictions on the text that can be entered into the file:

- (1) All lines must be twenty or less characters in length including the carriage return.
- (2) The maximum number of lines allowed in the file is sixty-four.

Startup

Upon startup of the M6800 microprocessor, it is necessary to initialize the editor. To do this the user must simultaneously strike the CTRL and C keys on the keyboard. EDIT will respond by printing the message "EDIT 1.0" and will then perform a carriage return and line feed and output a question mark in the first column of the new line. EDIT is now in the command mode.

Commands

The following is a complete list and description of the valid commands for EDIT.

Collect

The Collect command allows the user to create a new file or to add lines to the end of the existing file. The form of the command is:

C (CR)

Upon typing C, EDIT executes a carriage return (CR) and line feed (LF) and outputs an asterisk (*) in the first column. The user may now input lines of text into the file; each line terminated by a CR. The Collect command is terminated when a null line is typed; e.g., a CR immediately following the asterisk.

While executing the Collect command, the following errors may occur:

SPACE - There is not enough space left in memory to hold the line of text being entered. The line of text is lost.

SIZE - The line of text being entered is over twenty characters long. The line of text is lost.

Upon the occurrence of an error, the Collect command is terminated and EDIT is again in the command mode.

Delete

The Delete command allows the user to destroy the current file or to delete a single line of the current file.

The form of the Delete command is:

D parameter

The parameter is one or two numeric characters representing a line number in the file. There should be no blanks between the command and the parameter. If the parameter is null then the entire file is deleted. If the parameter is a valid line in the file, that line of text is deleted from the file. The file is then renumbered internally. The following errors may occur when attempting to execute a Delete command:

LINE - An attempt was made to delete a nonexistent line in the file.

PARM - An error occurred in evaluating the parameter.

The Delete command is aborted when an error is encountered; no deletion takes place.

Insert

The Insert command allows the user to insert a line of text into the current file. All lines of text, from the point of insertion to the end of the file, are pushed down in the file to make room for the new line of text. No line is lost when text is inserted in the file. The form of the Insert command is:

I parameter

The value of the parameter must be a valid line in the file. Upon evaluation of the Insert command, EDIT executes a CR and LF and outputs an asterisk (*) in the first column. The user may now input the line of text to be inserted; the line terminates with a CR.

The following errors may occur when attempting to execute an Insert command:

LINE - An invalid line or null parameter was given or the file is already full.

PARM - An error occurred in evaluating the parameter.

The Insert command is aborted when an error is encountered; no insertion takes place.

List

The List command allows the user to list a single line of the current file or list the entire file. The form of the command is:

L parameter

If the parameter is null then the entire file is listed. If

the parameter is a valid line in the file then that line of text is printed. The following errors may occur when executing the List command:

FILE - No current file exists.

LINE - An invalid line was given as a parameter.

Modify

The Modify command allows the user to replace an existing line of text with a new line of text. The form of the command is:

M parameter

If the parameter is a valid line in the file, that line is output and a new line is begun with an asterisk (*) in the first column. The user may now type the new line of text that is to replace the line just output. The new line of text is terminated by a CR.

If the user decides not to modify the line of text just output, a CR immediately following the asterisk will terminate the command with no modification taking place. The following errors may occur when attempting to execute a Modify command:

LINE - An invalid line or null parameter was given as a parameter.

PARM - An error occurred in evaluating the parameter.

SPACE - There is not enough room in memory for the modified line.

On all errors the Modify command is aborted; no modification

takes place.

Run

The Run command allows the user to submit the current file to the translator program. The translator prepares the text and sends it to the pulse programmer hardware for execution. The form of the command is:

R (CR)

Upon execution of the Run command, EDIT transfers control to the translator program. There are two ways of returning to EDIT:

(1) Type any character on the keyboard. This results in the monitor returning control to EDIT and EDIT trying to execute the command. Therefore, to avoid a 'COMND?' error the input character should be a valid command character.

(2) Push the reset button. This action resembles a startup to the processor. To avoid losing the current file, the user may execute any valid command without executing a CTRL-C. A CTRL-C may be executed if the user wishes to destroy the current file. Note in this case, while EDIT is in the command mode, no '?' is output in column one.

The following error may occur when attempting to execute the Run command (not including possible translator errors):

FILE - No current file exists.

Use

The use command allows the user to retrieve a program from the procedure library and place it into the current file. The form of the command is:

Uprog_name

where prog_name is a 1-6 character long name of a program stored in the procedure library. Prog_name must be terminated by a carriage return.

Upon execution of Use, the current file is destroyed and replaced by the file in the procedure library. The following error may occur when attempting to execute the Use command:

NAME - The prog_name given was either over six characters long or no procedure by that name could be found in the library.

The current file is not destroyed if a 'NAME?' error occurs.

APPENDIX B. MONITOR AND TRANSLATOR FLOW CHARTS