Spectrometer Manager: A Versatile Control Software for Pulse EPR Spectrometers

BORIS EPEL,[1]* IGOR GROMOV,[2] STEFAN STOLL,[2] ARTHUR SCHWEIGER,[2] DANIELLA GOLDFARB[3]

[1] *Max-Planck Institute of Bioinorganic Chemistry, Mülheim an der Ruhr, D-45470, Germany*
[2] *Laboratory of Physical Chemistry, ETH Hönggerberg, Zürich, CH-8093, Switzerland*
[3] *Department of Chemical Physics, Weizmann Institute of Science, Rehovot, 76100, Israel*

ABSTRACT: A versatile control software for pulse EPR spectrometers is introduced. Common and task-specific problems are discussed and their solutions are described. The software provides the full spectrum of possibilities needed to perform arbitrary multidimensional pulse experiments. It allows for an easy interfacing of commonly used hardware components and enables straightforward modifications of the spectrometer. Good performance, configurability, and a number of unique features turn this software into an excellent tool for the operation of modern EPR spectrometers and upgrading old ones. © 2005 Wiley Periodicals, Inc. Concepts Magn Reson Part B (Magn Reson Engineering) 26B: 36–45, 2005

KEY WORDS: pulse EPR; control software; pulse-programming language

INTRODUCTION

The rapid development of modern state-of-the-art pulse EPR spectroscopy constantly creates new challenges for the hardware of the spectrometers. Because commercial spectrometers cannot usually adapt to them immediately, the design of home-built EPR spectrometers with unique features still has high priority in many research groups worldwide. Such spectrometers can be built from independent stand-alone devices, which are connected, configured, and managed in different ways. This means that the designer of a new spectrometer has to solve the problem of communication with separate components and of experiment automation, and will have to develop custom control and acquisition software. Similar problems can arise when an existing experimental setup lacks some problem-specific components. Although such components can often be found as separate devices, the existing spectrometer software cannot control them, and the experimentalists have to introduce additional control software.

Many research groups active in the field of EPR have developed various control programs independently (a recent example is the program written by A.V. Astashkin from the University of Arizona), which are usually compatible only with a specific instrument in a particular configuration. Because the structure of these programs is typically tailored to particular devices, even a minor change in the configuration of the spectrometer becomes complicated. Data format incompatibility, differences in the user interface and in functionality, and limited possibilities are common problems of these programs. More general approaches were used in other areas of research, especially in magnetic resonance imaging (MRI). Although some MRI programs (*1*) are general enough and can in principle be applied to EPR experiments,

Received 7 December 2004; revised 10 March 2005; accepted 14 March 2005

*On leave from MRS laboratory, Kazan State University, Kazan 420008, Russian Federation.

Correspondence to: Boris Epel; E-mail: epel@mpi-muelheim.mpg.de

Concepts in Magnetic Resonance Part B (Magnetic Resonance Engineering), Vol. 26B(1) 36–45 (2005)

Published online in Wiley InterScience (www.interscience.wiley.com). DOI 10.1002/cmr.b.20037

they are too complicated and oriented toward specialized systems and "intelligent" device controllers. This is in contrast to a typical pulse EPR console, which consists of general-purpose devices and a single personal computer, performing all tasks from device control to data management.

The following is the suggested set of principal requirements for a pulse EPR spectrometer control software:

- Uniformity—unified user interface for all experiments and instruments
- Versatility—a fast and easy adaptation of the program to the spectrometer hardware configurations
- Convenience—real-time tuning and monitoring capabilities for all device settings and experimental parameters, quick application of new experimental schemes, easy to learn graphical user interface and good visual appearance
- Fast performance and robustness

In this article we present a software that matches these requirements. The Borland C++ Builder^{TM} 6.0 (*2*) was used for its implementation.

FEATURES

The Spectrometer Manager for pulse EPR (SpecMan) is a general spectrometer control software, an all-in-one solution for pulse sequence generation, remote device management, and acquisition of the corresponding signals. It has the following features:

- A pulse (sequence) programming language (PPL)
- User-defined pulse commands, variables, and arrays
- Arithmetic operations on variables, arrays, and constants
- Conditional statements, loops, and parallel commands execution
- A configurable PPL interpreter that turns PPL commands into predefined groups of pulses generated by pulse programmers. An optimization of the pulse sequence for an efficient device programming at minimum time
- Loop engine for up to three-dimensional (3D) experiments. Transient signals can be acquired as a fourth dimension
- All PPL variables and device settings can be used as independent parameters for the experiment
- All information recorded by devices (including transient signals) can be stored
- Random (nonsequential) execution of the experiment
- Real-time signal processing (integration and baseline correction)
- A user-friendly interface
- All relevant parameters are visible or one click away
- Single-document type of interface, two view ports, file browser
- Hardware and pulse configurations wizards
- Real-time monitoring of acquired signals
- Duty cycle and maximum pulse-length protections. Safe ranges for device settings can be specified to protect hardware from being damaged
- An easy-to-support kernel-driver program structure. Drivers for devices that are commonly used in pulse EPR. Flexible adjustment of the program to the spectrometer hardware setup

Minimum system requirements: IBM PC compatible computer with a 400 MHz processor, 64 MB RAM, Microsoft Windows^{TM} (95/98/9x/2000/XP).

CONCEPT AND STRUCTURE

SpecMan is based on the following assumptions: (i) a general experiment can be represented as mutually independent data points or transients recorded as a function of independent discrete parameters, and (ii) the pulse sequence can be described as a function of independent parameters. Because the parameters of the program are represented by arrays, their indices can also act as independent parameters. We call them "axes." A single axis can index an arbitrary amount of associated dependent parameters. The special case of an axis where all parameters are constant is described by one element only. The lengths of the other axes are not limited. The experiment is completed when all possible combinations of indices are scanned, producing an *N*-dimensional matrix of experimental points, where *N* is the number of independent parameters.

The structure of the program is shown in Fig. 1. The different blocks of the figure represent functional modules, and arrows show the data flow inside the program. The part of the program that scans independent indices during the experiment is called the loop engine. It can maintain a number of axes: X, Y, and Z for 3D experiments; T for transient data; and P for the initial setup of parameters. A special "sum" option allows one to sum data points over an axis. The order

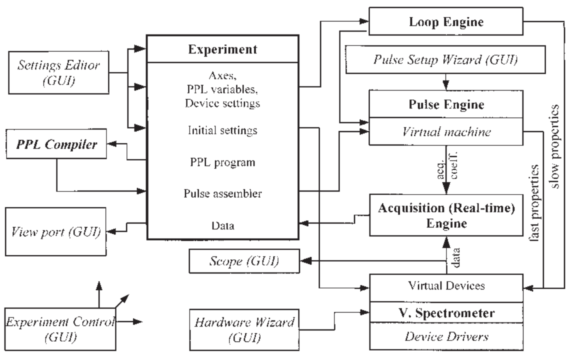


Figure 1 Structure of SpecMan. The hardware-dependent part is shown in the shadowed box. Major program modules are shown in bold. Components of the graphical user interface are denoted by GUI.

of the X, Y, Z, and sum indices can be freely chosen. Each axis can be scanned sequentially or randomly, depending on the settings. For example, consider a two-pulse echo experiment, recorded as a function of the magnetic field with a two-step phase cycle. There are two independent parameters (field and time τ between the pulses) and one additional index, the phase cycle. The different phase cycle traces are not included in the data, but each point of the final data set is the sum of points corresponding to different phase cycle indices with different weight factors ($+1$ and -1 in this case). If the user's intention is to record the dependence of the echo-detected EPR line shape on τ, the index order will be (i) phase cycle, (ii) τ, and (iii) magnetic field. Otherwise, an experiment that measures the two-pulse echo decay as a function of the magnetic field may have the index order (i) magnetic field, (ii) phase cycle, and (iii) τ.

The virtual devices represent the connected hardware components. The mechanism of interaction between virtual and real devices is built into the virtual spectrometer program module. The device-dependent parts of the virtual spectrometer (device drivers) contain an implementation of the single device functionality and are able to execute standard requests such as to set and acquire device settings (properties) and to initialize a device. The full configuration of the spectrometer contains the description of the connected virtual devices (name, assigned driver, initial settings) and is loaded from a text (*.cfg) file. A special hardware wizard helps to create and modify this file. The configuration file also contains various restrictions on device settings to protect the spectrometer from being damaged. Different setups can be realized by modifying this configuration file only. The settings editor can define any of the device properties as parameter of an experiment.

Two kinds of device properties have been classified to cover all possible device functionalities. The first type is created for device settings that are constant during execution of a single experimental point (slow property in our terminology), such as microwave power, magnetic field, or temperature. These properties are readable (acquired data) and writable. Writable properties can be controlled directly through the graphical user interface (GUI).

The second type of properties include pulse sequence events and is designed for devices that are active during the execution of the experiment (fast properties). Each of the fast properties describes a single output of an arbitrary pulse generator with two logical states. The pulse engine automatically generates pulse event sequences according to the user-written program in PPL. The heart of the pulse engine is the virtual machine, the interpreter of the PPL assembler commands. It has eight-bit commands, a 16-bit index of commands and variables, a double floating-point precision format of data, and one operational register. The vocabulary of the virtual machine has 28 commands, including arithmetic operations; register, indexed, and double-indexed data transfer; and logical operations. Up to 254 user-defined (two reserved) pulse commands and 32,768 variables (including array elements) are supported. The overall length of a PPL program is limited to 32,768 assembler commands. The PPL compiler is built using GNU Bison and Lexer packages (*3*). The description of the user-defined pulse commands is loaded from the pulse configuration file (*.cfp). The pulse setup wizard assists in the creation of the pulse configuration. Using the settings editor any PPL variable can be defined as an independent experimental parameter.

The information on each point is stored in the experiment data block. It contains the experiment axes description, including associated variables, the pulse program, initial device settings, and, after the end of experiment, the acquired data. The data from the devices can be processed in real time in the acquisition engine before it is stored in the memory. The whole experimental procedure is loaded from and stored to experiment (*.exp), data (*.d01), or template (*.tpl) files. The GUI provides full control over preparation and flow of the experiment.

Control Sequence

Each point of an experiment is produced according to the control sequence, which includes (i) programming of slow properties, (ii) execution of the sequence, and (iii) data transfer from acquisition devices. The sequence of actions during the first and last stages has

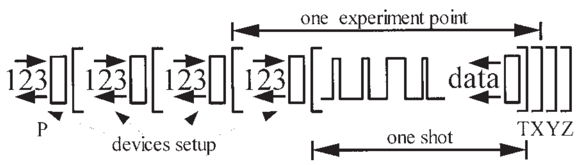


Figure 2 Sequence of events in the loop engine. The brackets represent the cycles along the axes points.

no influence on the result, whereas the behavior of the devices during the second stage is ordered in time and no control is possible during execution. The time diagram of an experiment is presented in Fig. 2. The setup of the device properties occurs before each experimental point. When the properties corresponding to all axes are programmed, the pulse sequence is triggered and the data are acquired. At the end of data acquisition, the next experimental point can be programmed. Additional device setups (not shown in the figure) can be done after every point as well as before and after the experiment. It can be used to delay the next point and to switch on devices during run-time only.

Pulse Programming Language (PPL)

The syntax of PPL is partially inherited from the spectrometer control software used at the Weizmann Institute and at ETH (*4*). PPL is designed to describe the time sequence of events. The main difference between PPL and conventional programming languages is that each PPL command starts at a specific time defined by a hidden variable (internal time counter) and has a specific duration. The user cannot directly access this counter, but it can be changed by the execution of the PPL commands. Each PPL command affects multiple pulse channels (see the pulse generation section for a more detailed description). Because all pulse commands have user-defined actions, the PPL is hardware independent, and programs written in PPL can be executed on any spectrometer.

Figure 3(a) gives an example of a PPL program for a two-pulse echo experiment. It starts with the declaration of the variables and the acquisition stream "a." Two microwave (MW) pulses (**mwpulse**) are separated by delays (**wait**) and are followed by a **detect** command. The **detect** command has two functions: (i) it generates the trigger for the data acquisition and (ii) specifies the acquisition stream and its coefficient, which is required for phase cycling (+1 for the "a" stream in this case). Although the program looks simple, for a typical pulse EPR application, the user-defined **mwpulse** command will generate a series of pulses fed to the MW switches, the protection switch, the phase shifters, and the amplifier gate, each with its own specific timings (see the pulse generation section).

a

```
time t90, t180, tau
signal a

mwpulse t90
 wait  tau
mwpulse t180
 wait  tau
detect '+a'
```

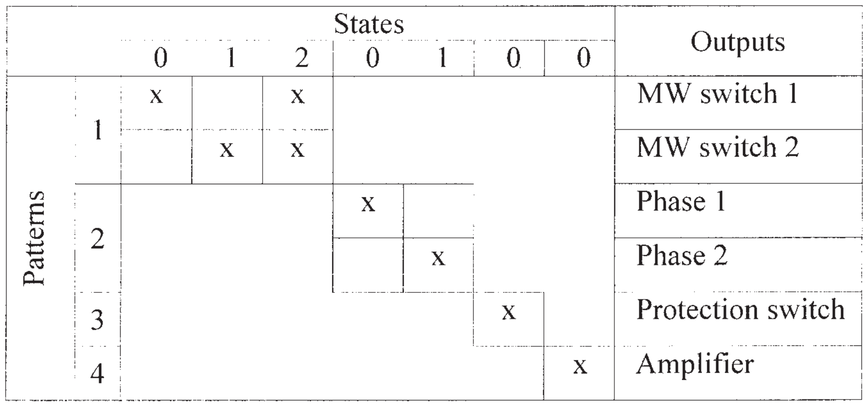
b

```
time t90, t180, t1, t2, tau
int ph      %% phase index
%% quadrature detection
signal a,b

%% detection coefficients
dpha = ['+a','-a','+a','-a']
dphb = ['+b','-b','+b','-b']
%% phase cycling pulse pattern
ph180 = [0, 0, 1, 1]
ph90  = [0, 1, 0, 1]

mwpulse t90, 0
  wait tau
mwpulse t90, 0
  wait t1
mwpulse t180, 1, ph180(ph)
  wait t2
mwpulse t90, 0, ph90(ph)
  wait tau
detect dpha(ph),dphb(ph)
```

Figure 3 Examples of PPL programs. (a) Two-pulse echo sequence; (b) HYSCORE sequence with four-step phase cycle.



		States							Outputs
		0	1	2	0	1	0	0	
Patterns	1	x		x					MW switch 1
			x	x					MW switch 2
	2				x				Phase 1
						x			Phase 2
	3						x		Protection switch
	4							x	Amplifier

Figure 4 Definition of **mwpulse** command patterns for a pulse EPR spectrometer with two channels and two phase shifters. Crossed cells indicate the presence of corresponding pulses in the output pattern.

From a programming point of view, the pulse program is a subroutine executed at every experimental point with the declared variables as arguments. Together with other experimental parameters, these variables are visible in the loop engine and have to be declared using one of the variable type specifiers: **time**, **int**, **real**, or **bool**. Other data types, like arrays, are defined only within the subroutine and cannot be controlled by the loop engine. The PPL output returns the names of acquisition streams and detection coefficients, which are used later by the acquisition engine. Multiple acquisitions in the pulse program are not yet supported. The assignment of devices to PPL streams occurs in the loop engine. Additional language features include conditional statements (**if** . . . **else** . . . **end**), looping (**repeat** . . . **end**), and parallel execution of pulse subsequences (**parallel** . . . **end**).

Pulse Generation

In general, an EPR spectrometer requires complex control pulse patterns. A number of service pulses with different timings are required. Thus, a PPL command has to be able to produce multiple pulses. For example, consider a typical realization of the **mwpulse** command on an EPR spectrometer with two MW channels. The command has to affect six devices: two MW switches, two phase-shifters, the receiver protection switch, and the trigger (gate) for the MW amplifier. Consequently, the pulse programmer device has to have at least six outputs. Overall, there are 64 different combinations of output states that can be generated by these outputs, but only six of them have practical importance because the amplifier and the protect switch pulses are always required and only one phase shifter at a time is used. Thus, it is more reasonable not to use a general index to all states but rather only four indices for specifically grouped outputs (patterns) as is shown in Fig. 4. This means that a pulse command would have four parameters in addition to the pulse duration. But the one-state pattern indices for protection switch and amplifier can be omitted. Hence, in addition to the duration, the **mwpulse** command will have indices for the MW switch and phase patterns only. For example, an "**mwpulse** 50 ns, 0, 1" statement will produce a pulse on the first MW channel with the second phase state, and with protection switch and amplifier pulses. Alternation of pattern indices is extensively used for phase cycling as shown in Fig. 3(b).

Each pulse output has an independent set of parameters. The lead and trail parameters define the shifts of the real events relative to those generated by PPL in order to compensate for hardware delays. Maximum duty cycle and maximum pulse-length restrictions protect the spectrometer from damaging. An additional parameter specifies the algorithm of the pulse generation. Six different algorithms are used. The "exact" algorithm produces a pulse with the length specified in the pulse command. "Up," "down," and "up/down" algorithms serve to produce triggers with corresponding active slopes. The length of the trigger pulse is a parameter of the output as well. The "potential" algorithm ensures that the outputs have a certain state at points in time defined by the pulse command. It is used, for example, to operate phase shifters. The "trigger" and "potential" algorithms minimize the transfer time of the pulse sequence to the hardware. The "space" algorithm generates nonoptimized events.

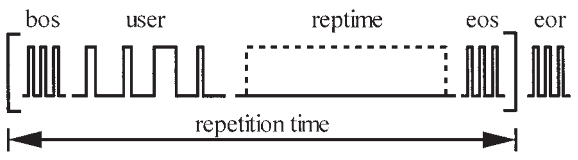


Figure 5 General scheme of a pulse sequence.

Pulse Sequence

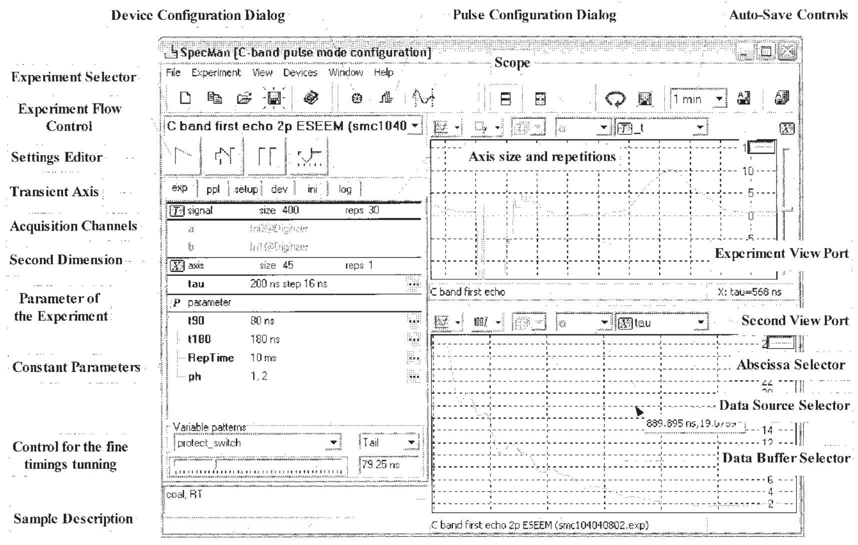
The whole pulse sequence generated by the pulse engine is presented on Fig. 5. It consists of a user-programmed command sequence and automatically inserted service commands. This allows one to exclude experiment-independent commands from the pulse program. The sequence starts with the "beginning of sequence" (**bos**) command. **Bos** is followed by the **user** sequence produced by the pulse program. The "repetition time sequence" (**reptime**) command can be inserted before or after the user sequence, depending on the settings. The length of the **reptime** command is calculated according to the repetition time of the experiment. Finally, "end of sequence" (**eos**) finishes the loop of the main sequence. After the repetition loop, the "end of run" (**eor**) command is inserted. The special **scope** command provides the scope trigger for signals monitoring on an oscilloscope. It is automatically inserted at the beginning or the end of a specified time interval in the pulse sequence. All commands are defined in the pulse configuration.

Graphical User Interface

The SpecMan GUI design has a single-document style. Only one experiment can be configured and executed at a time. The layout of the main interface components is presented in Fig. 6. The left part of the interface contains the experiment selector, the experiment flow control buttons, and the settings editor (from top to bottom). The experiment selector allows browsing of spectra stored in the memory. The settings of stored experiments can be copied to the active experiment. The setting editor has four panels: experiment axes editor, PPL program text editor, and initial and run-time device setup. Two additional panels contain the configuration files browser and a message log window. The shadowed horizontal bars of the axes editor represent axes. Each axis (except P) has a size and a certain number of repetitions. The order of bars (from top to bottom) represents the order in which axis indices will be modified (from inner to outer ones). The designation of the axis (T, X, Y, Z) shows the order in which experimental data will be saved to the file. The white bars below each axis represent the associated parameters (i.e., tau) and acquisition streams (a, b). The right part of the interface presents two 1D view ports for experimental data. Each view port has selectors for the data source (acquisition buffer or main storage array), for the acquisition stream, and for the abscissa dimension of the plot. Scroll bars on the right allow browsing data along the other dimensions.

Real-time Signal Processing

If a transient recorder is available, SpecMan can acquire the shape of a spin echo. However, for most applications, only the integral of the echo signal is of interest. SpecMan can integrate in real-time transient



SpecMan [C-band pulse mode configuration]

C band first echo 2p ESEEM (smc1040

T signal	size 400	reps 30
a	tr0@Digitizer	
b	tr1@Digitizer	
X axis	size 45	reps 1
tau	200 ns step 16 ns	
P parameter		
t90	80 ns	
t180	180 ns	
RepTime	10 ms	
ph	1, 2	

protect_switch Tail 79.25 ns

coal, RT

C band first echo X: tau=568 ns

889.895 ns,19.0755

C band first echo 2p ESEEM (smc104040802.exp)

Figure 6 The front end of SpecMan.

signals in a user-specified range. In addition, zero- and first-order baseline corrections are implemented.

Experimental Data Protection

There are situations where the normal flow of an experiment can be disturbed (e.g., due to deterioration of the sample, hardware failure, or a sudden change in temperature). For such cases, SpecMan has a multi-level protection for already recorded data. The data acquisition streams of SpecMan are buffered. This means that the data from devices are first stored in an intermediate buffer and are added to the main storage array only at the end of the scan. When the user observes an experimental flow disturbance, the experiment can be stopped and preserved in the main storage array in the state it was before the last scan, or the last scan can be suppressed, allowing the user to continue with the experiment. The content of intermediate buffer and main storage array of any stream can be monitored on-screen. An auto-save option is included, where SpecMan saves the current state of the main storage arrays at defined time intervals. This is a convenient feature, which is not available in commercial pulse EPR spectrometers. It avoids losing data, which have been accumulated over a long time.

Device Drivers

Here we provide a list of devices that are fully compatible with the current version of SpecMan. Pulse programmers (highest time resolution is shown in brackets): DG535 (Stanford Research, 10 ps), RS690 (Interface Technology, 4 ns), PulseBlaster (SpinCore, 6.6 ns, shortest pulse 52.8 ns), PulseBlaster ESR (SpinCore, 3.3 ns, shortest pulse 3.3 ns, shortest delay 16.5 ns), and AWG1000-DOUTS (Chase Scientific, 1 ns). Fast digitizers/averagers: DP and AP series (Acqiris, dwell time up to 0.5 ns). ADC boards: E-series and Basic Multifunction DAQ boards (National Instruments, 200 kS/s). Field controllers B-H15, ER032M (Bruker). RF synthesizers and generators: PTS310 (Programmed Test Sources, Inc., connected through 96-pin digital in/out port), arbitrary waveform generator LW420 (LeCroy), SMX and SMT series (Rhode and Schwartz), EGS4420 (Agilent), AWG 1000 (Chase Scientific), and MW generator SMR40 (Rhode and Schwartz).

RESULTS

In this section, three spectrometers that are controlled by the SpecMan shell are described. The first one is a pulsed EPR/ENDOR W-band spectrometer (*5*) at the Weizmann Institute of Science. The console is based on an X-band setup (*4*) with some modifications. It consists of a 400 MHz computer, a RS690 pulse generator, a National Instruments 6014 analog-to-digital converter connected to a boxcar integrator, a National Instruments PCI-GPIB (peripheral component interface-general purpose interface bus) communication card, and a PCI-96-DIO digital input-output card connected to the two-channel frequency synthesizer PTS310. Figure 7 shows a two-pulse echo decay of a Cu-doped L-histidine single crystal measured on this spectrometer using SpecMan. The pulse program listed in Fig. 3(a) was used to obtain this decay.

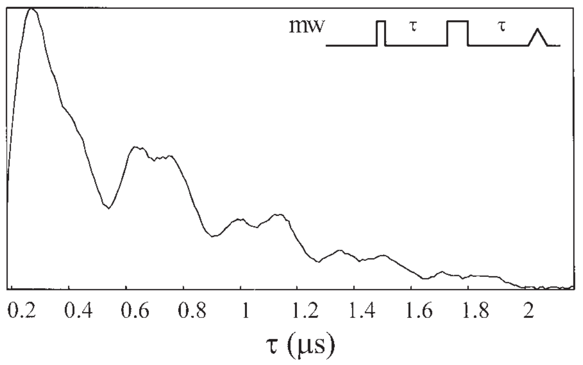


Figure 7 W-band two-pulse echo envelope of a single crystal of Cu-doped L-histidine measured at 8 K.

The second setup is located at the Max-Planck Institute of Bioinorganic Chemistry and is designed for performing ENDOR and ELDOR experiments on systems with large hyperfine couplings, as found for example in compounds with manganese ions. It is installed on a Bruker ESP380 spectrometer. The standard RF synthesizer of the Bruker ESP380 spectrometer cannot generate frequencies above 150 MHz. In addition, the present spectrometer does not have a second MW channel. Thus, external RF and MW generators had to be used. A proxy console was built on a SpinCore PulseBlaster-100^{TM} pulse generator, an Acqiris DP235 fast digitizer, Rohde and Schwartz SMR40 and Agilent EGS4420 generators, and an NI PCI-GPIB card. To perform ELDOR experiments, the SMR40 generator was connected to a channel of the ESP380 spectrometer. The MW pulse sequence was programmed using the ESP380 console. The generator frequency was controlled via a GPIB. The spin-echo signal was recorded on the proxy console, which was triggered by the zero-time pulse of the ESP380. Figure 8(a) shows ELDOR-detected NMR

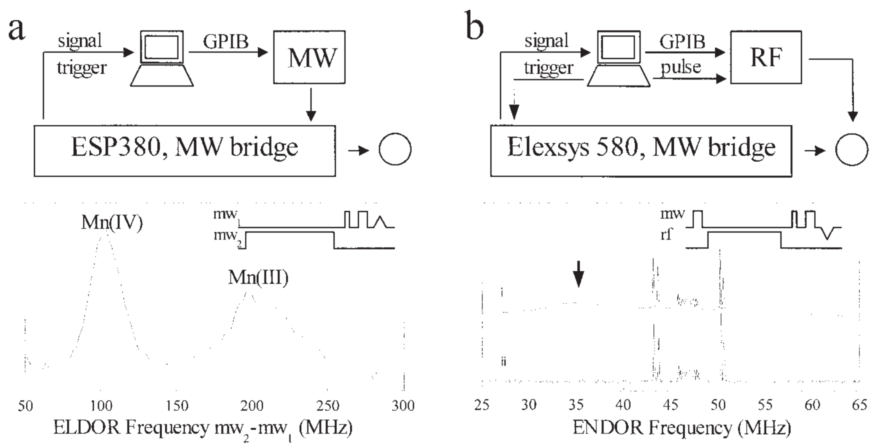


Figure 8 Block diagrams of proxy setups, created to extend the possibilities of a Bruker ESP380 X-band spectrometer and a Q-band spectrometer equipped with an E580 console together with examples of recorded spectra. The corresponding pulse sequences are shown as insets. Left: X-band ELDOR-detected NMR spectrum of ^{55}Mn in BIPY recorded at 3.8 K. The first MW pulse length was 9 μs. Features attributed to hyperfine transitions of Mn in different electron states are marked. Right: Q-band proton Davies-ENDOR spectra of a Cu(II)-doped α-glycine single crystal. The spectra were recorded at 15 K with an RF pulse length of 25 μs and an unmatched RF coil. The incident RF power was about 450 W. Spectrum (i) is recorded with sequential acquisition and the spectrum (ii) is recorded with random acquisition. The arrow indicates the artefact caused by heating.

(*6*) spectra of [$Mn^{III}Mn^{IV}(m\text{-}O)_2bipy_4$]$ClO_4$ (BIPY complex) (B. Epel and L. Kulik, unpublished results). For the ENDOR setup, the output of the EGS4420 was directly connected to the RF amplifier and was gated using the pulse from the PulseBlaster. It also generated trigger pulses for the spectrometer. The zero-time pulse of the spectrometer was used to trigger the acquisition.

A similar setup, which consists of a RS690 pulse generator, an AP240 fast averager, and an LW420 arbitrary waveform generator, is used at the Swiss Federal Institute of Technology (ETH) to perform random swept ENDOR experiments on a Q-band EPR/ENDOR spectrometer equipped with a Bruker E580 console. Proton Davies-ENDOR spectra of a Cu(II)-doped α-glycine single crystal obtained with this setup are shown in Fig. 8(b).

One of the unique features of SpecMan is its ability to execute an experiment along one dimension in random order. It has been demonstrated (*7*) that random acquisition removes the correlation between adjacent experimental points. Especially for ENDOR spectra with broad lines and weak signals, this type of acquisition diminishes unwanted effects caused by RF-induced warming. This is illustrated in Fig. 8(b), where the hump around 35 MHz, generated by a sequential acquisition, is eliminated completely by random acquisition.

Performance

The two main functions of the program, computation and communication with devices, will inevitably need some extra experimental time. The efficiency of the software-hardware complex can be estimated using the overhead parameter defined as: OH = (real experiment length/calculated experiment length − 1) × 100%. Here we present a summary of the overhead values measured on various hardware configurations for two of the most popular pulse EPR experiments, HYSCORE (*8*) and Davies ENDOR (*9*). The HYSCORE experiment requires four MW pulses and one detect trigger pulse. A total of 128 points were acquired in each of the two dimensions with 10 shots per point, a four-step phase cycle, two detection channels (in quadrature), and a repetition time of 1 ms. The Davies-ENDOR experiment requires three MW, one RF, and one detect trigger pulse. A time trace with 500 points was measured on a single detection channel with 10 shots per point and a repetition time of 1 ms. Other settings, such as length of pulses and integration window, do not affect the overhead. Less complicated experiments generally show smaller overhead in similar configurations. An increase of the repetition time or the number of shots typically reduces the overhead as well. Use of different versions of the Windows operating system also has some impact on the performance.

Table 1 Overhead of the HYSCORE and the Davies-ENDOR Experiments on Different Hardware Configurations

Configuration	OH, HYSCORE	OH, Davies ENDOR
SpecMan, dummy configuration. All required pulses are generated, AMD Duron 600 MHz/1.2 GHz processor*	27%/12%	22%/10%
SpecMan, Interface Technology RS690 pulse generator (GPIB) and Acqiris AP240 averager (PCI), LeCroy LW420 (GPIB)	828%	1200%
SpecMan, SpinCore PulseBlaster ESR pulse generator (PCI) and Acqiris DP235 digitizer (PCI), Agilent EGS4420 RF generator (GPIB)	22.8%	238%
State-of-the-art commercial spectrometers	~18%	~200%

GPIB = general purpose interface bus; PCI = peripheral component interface.

* The overhead for dummy devices is calculated according to the formula OH = (real experiment length/calculated experiment length for repetition time 1 ms) $\times$ 100%.

The type of the device interface bus is specified in brackets. Microsoft Windows 2000 Pro (TM) SP3 is used in all SpecMan configurations. The calculated experiment lengths are 655 s for HYSCORE and 5 s for Davies ENDOR.

The HYSCORE experiment stresses the pulse programmer. A significant part of the pulse sequence is changing from one experimental point to another. Contrary to that in the ENDOR experiment, the pulse sequence is the same for every point and only the RF is changed. Thus the key factor is the performance of the RF generator.

The SpecMan program has a special dummy driver, which can emulate the behavior of inputs and outputs of any "real" device, accept commands and generate required data. The computational overhead estimated using a dummy device configuration is presented in the first row of Table 1. Because it is complicated to precisely emulate a 1 ms delay in the software, the repetition time was set to 0 ms. As expected, the overhead values are similar for the HYSCORE and ENDOR experiments and decrease with increasing of processor speed. The second row of the table contains data for devices employing a GPIB (*10*) for communication. The transfer rate of commands over GPIB (about 1 Mb/s) limits the performance of the system. The performance of the HYSCORE experiment, which requires significant reprogramming, suffers especially. Modern high-speed pulse programmers and acquisition systems are built on different versions of the PCI bus (*11*). The PCI devices provide significantly shorter reprogramming times. For example, the spectrometer based on the PulseBlaster ESR pulse programmer and the Acqiris DP-series digitizer shows an excellent performance in HYSCORE experiments (see Table 1, row 3). The programming delays in this configuration are comparable with the computational ones. Using the special mode of the SpecMan program, which fit the device programming into the repetition time of the last shot, programming delays can be decreased even further.

CONCLUSION

We have demonstrated that the SpecMan program is well suited for modern pulse EPR experiments. It delivers a low-cost, easy-to-maintain solution for developing new experiments and for conducting routine applications. Simple interfacing and a front end free of technical details decreases the learning time needed to operate the spectrometer. The flexible kernel of the program guarantees fast adaptation of the program to all experimental needs. A number of special features, such as random acquisition, real-time base line correction, elimination of erroneous accumulated traces, and an auto-save option, give experimentalists unique possibilities that were not available until now. Many laboratories benefit already from the program. SpecMan is under continuous development. Planned improvements include multiple acquisitions during the pulse sequence, programming of a whole experiment at once for "intelligent" devices, and additional modules for real-time processing. Detailed help and a fully functional demonstration version are available from the official Web sites of the program (*12*).

ACKNOWLEDGMENTS

We gratefully acknowledge those individuals who on different stages of the project participated in programming and discussions: Rene Tschagellar, Jurg Keller,

and Tatjana Rubinov. The ideas implemented in the spectrometer control program of Jaap Shane [see ref. (*4*)] had a significant influence on the SpecMan design.

REFERENCES

1. Debbins J, Gould K, Halleppanavar V, Polzin J, Radick M, Sat G, Thomas D, Trevino S, Haworth R. 2002. Novel software architecture for rapid development of magnetic resonance applications. Concepts Magn Reson (Magn Reson Engineering) 15(3):216–237.
2. Borland Software Corporation. 2002. C++ builder. Available at: http://www.borland.com/cbuilder/.
3. Donnelly C, Stallman R. Bison. 1995. The YACC-compatible parser generator. Boston: Free Software Foundation.
4. Shane JJ, Gromov I, Vega S, Goldfarb D. 1998. A versatile pulsed X-band ENDOR spectrometer. Rev Sci Instrum 69(9):3357–3364.
5. Gromov I, Krymov V, Manikandan P, Arieli D, Goldfarb D. 1999. A W-band pulsed ENDOR spectrometer: setup and application to transition metal centers. J Magn Reson 139:8–17.
6. Schosseler P, Wacker TH, Schweiger A. 1994. Pulsed ELDOR detected NMR. Chem Phys Lett 224:319–324.
7. Epel B, Arieli D, Baute D, Goldfarb D. 2003. Improving W-band pulsed ENDOR sensitivity-random acquisition and pulsed special TRIPLE. J Magn Res 164(1): 78–83.
8. Hoefer P, Grupp A, Nebenfuehr H, Mehring M. 1986. Hyperfine sublevel correlation (HYSCORE) spectroscopy: a 2D ESR investigation of the squaric acid radical. Chem Phys Lett 132:279–282.
9. Davies ER. 1974. New pulse ENDOR technique. Phys Lett A 47:1–2.
10. ANSI/IEEE Standard 488.1-1987, ANSI/IEEE Standard 488.2-1992
11. IEEE Standard P1386.1
12. SpecMan: the shell for pulse EPR experiments. 2004. Available at: http://www.esr.ethz.ch/specman/recent/sm_main.html, http://www.geocities.com/boep777/sm_main.html.