An Extensible, Graphical Environment for Pulse Sequence Design and Simulation

W. R. Overall[1], and J. M. Pauly[1]

[1]Electrical Engineering, Stanford University, Stanford, CA, United States

Introduction: Pulse-sequence conceptualization, simulation, and hardware programming are usually distinct development stages, with similar code rewritten at each step. Additionally, each hardware vendor maintains a unique, proprietary programming environment. As a result, no freely available code library exists for standard pulse sequences, and sequences are not portable across vendors. We have developed a simple, extendable, freely available environment for rapid pulse sequence programming and simulation. Our software, named 'SpinBench' [1], combines computation speed with easy expandability by isolating the optimized simulator code from the plugin modules that provide additional functionality. Using this software, many pulse sequences can be designed and simulated using the software's built-in graphical tools. Plugin APIs are provided to allow specialized pulse types or simulations of unusual physical phenomena.

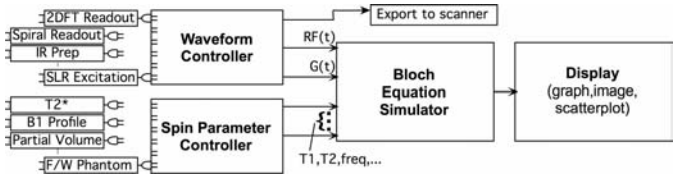

Figure 1: *Block diagram of our software architecture. Plugins are activated as desired to produce arbitrary waveforms or spin conditions; software controllers translate plugin state for input to a highly parallelized Bloch simulator, which computes values for subsequent*

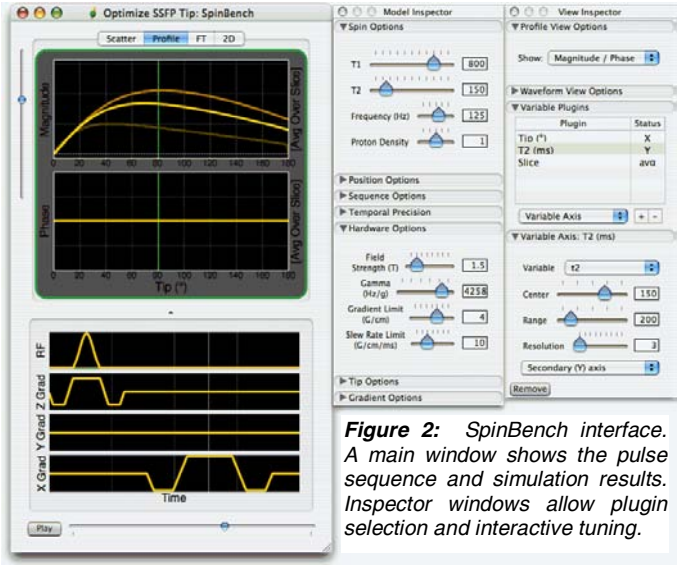

Figure 2: *SpinBench interface. A main window shows the pulse sequence and simulation results. Inspector windows allow plugin selection and interactive tuning.*

Methods: Our software leverages the XCode software development environment available within Mac OS X, which provides high-level document operations including Load, Save, Cut, Paste, and Undo without explicit programming effort. The software structure (Fig. 1) includes a multithreaded Bloch-equation simulator that uses all available processors for fast simulation. To optimize simulation speed, this simulator block accepts only a minimum set of inputs (T1, T2, frequency, etc.). A plugin architecture is used to specify higher-level structures including waveform types (trapezoid, spiral, slice-selection, etc.) or spin environments (motion, T2*, receiver phase, phantom characteristics, etc.). Software controllers organize the active plugins and translate their state into inputs recognized by the Bloch simulator. After simulation, outputs can be displayed as a graph, image, Fourier transform, or scatterplot. Any floating-point Bloch input or plugin parameter can be specified as the independent axis or axes.

Significant time savings result from distilling parameters to the minimum set prior to the time-consuming Bloch simulation. The default simulation algorithm computes signal independently for each value over the desired range [2], with culling operators used to skip unnecessary iterations. While other simulation techniques (including [3,4]) may be faster in certain cases, this approach places no assumptions on the experiment, and its independent measurements are extremely well suited to parallelization.

An external plugin-writing environment is provided for extensibility, consisting of a set of documented Objective-C APIs and a code library of existing plugins for use as a starting point. Plugins typically require a minimum of coding, and may or may not have an associated inspector pane for graphical selection of parameter values. A plugin may specify new RF pulses, gradient waveforms, unique physical conditions, or simulate a specific phantom. If SpinBench does not have the appropriate plugins when a sequence file is loaded, a window notifies the user and directs them to the plugin's author, if known. APIs also exist for converting sequences and data to and from other file types; this capability may be used to directly produce code for use on major vendors' hardware. In this way, SpinBench can be a platform-independent sequence-development tool.

Results: The program interface (Fig. 2) consists of a main window showing the current pulse sequence and simulator results, and inspector windows that allow parameter and plugin manipulation. Fig. 2 plots a simulation of slice-selective SSFP signal for 3 different T2 values as a function of tip. This simulation required 4 sec. on a 2.5-GHz Dual-PowerPC G5 system. For comparison, the same simulation required more than 2 min. of computation time when implemented in Matlab. Computation times generally range from instantaneous to minutes depending upon simulation complexity. Several plugin modules have been created and used in our research, including the magnetic dipole signal arising from an iron particle in a rephased SSFP experiment as shown in Fig. 3. This simulation was completed in 3 min., and the plugin consists of less than 60 lines of code, most of which is housekeeping code copied from an existing plugin.

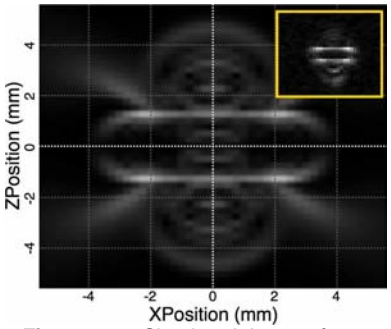

Figure 3: *Simulated image from a paramagnetic particle using a positive-contrast SSFP pulse sequence. Inset: experimental data confirms simulation.*

Discussion: We have developed and demonstrated a flexible, vendor-independent MR signal simulator and pulse-programming environment. Parallel code execution allows fast simulation without sacrificing accuracy, and plugin APIs allow arbitrary functionality. The software, plugin APIs, and a library of sample plugins are available for download over the web [1].

Further code optimization is possible to fully exploit parallelism and to avoid unnecessary computations. We will also explore the sequence optimizations made possible by integrated spin simulation, including automated contrast optimization between selected tissues and automated k-space trajectory determination.

References:

[1]. http://www-mrsrl.stanford.edu/SpinBench/

[2]. Summers, RM, *et al*. MRM 3(3): 363-376, 1986.

[3]. Magland, J, *et al*. Proc. 14th ISMRM: 2365, 2006.

[4]. Kwan, RKS, *et al.* IEEE Trans Med Imag 18(11): 1085-1097, 1999.