Journal of Magnetic Resonance 270 (2016) 47–55

Contents lists available at ScienceDirect

Journal of Magnetic Resonance

journal homepage: www.elsevier.com/locate/jmr

gr-MRI: A software package for magnetic resonance imaging using software defined radios

Christopher J. Hasselwander [a,c], Zhipeng Cao [a,c], William A. Grissom [a,b,c,*]

[a] Department of Biomedical Engineering, Vanderbilt University, Nashville, TN, USA
[b] Department of Radiology and Radiological Sciences, Vanderbilt University, Nashville, TN, USA
[c] Vanderbilt University Institute of Imaging Science, Nashville, TN, USA

A R T I C L E I N F O

Article history:
Received 18 March 2016
Revised 3 June 2016
Accepted 30 June 2016
Available online 1 July 2016

Keywords:
Software-defined radio
Spectrometers
Open-source software
MRI
Frequency-swept RF pulses

A B S T R A C T

The goal of this work is to develop software that enables the rapid implementation of custom MRI spectrometers using commercially-available software defined radios (SDRs). The developed gr-MRI software package comprises a set of Python scripts, flowgraphs, and signal generation and recording blocks for GNU Radio, an open-source SDR software package that is widely used in communications research. gr-MRI implements basic event sequencing functionality, and tools for system calibrations, multi-radio synchronization, and MR signal processing and image reconstruction. It includes four pulse sequences: a single-pulse sequence to record free induction signals, a gradient-recalled echo imaging sequence, a spin echo imaging sequence, and an inversion recovery spin echo imaging sequence. The sequences were used to perform phantom imaging scans with a 0.5 Tesla tabletop MRI scanner and two commercially-available SDRs. One SDR was used for RF excitation and reception, and the other for gradient pulse generation. The total SDR hardware cost was approximately $2000. The frequency of radio desynchronization events and the frequency with which the software recovered from those events was also measured, and the SDR's ability to generate frequency-swept RF waveforms was validated and compared to the scanner's commercial spectrometer. The spin echo images geometrically matched those acquired using the commercial spectrometer, with no unexpected distortions. Desynchronization events were more likely to occur at the very beginning of an imaging scan, but were nearly eliminated if the user invoked the sequence for a short period before beginning data recording. The SDR produced a 500 kHz bandwidth frequency-swept pulse with high fidelity, while the commercial spectrometer produced a waveform with large frequency spike errors. In conclusion, the developed gr-MRI software can be used to develop high-fidelity, low-cost custom MRI spectrometers using commercially-available SDRs.

1. Introduction

Modern commercial magnetic resonance imaging (MRI) and nuclear magnetic resonance (NMR) spectrometers are sophisticated devices with very high performance. However, many research and development applications in magnetic resonance require more configurable, portable, or scalable spectrometers at a low cost. For example, spectrometers have been developed in-house to meet the unique needs of low-field MRI scanners [1,2], deliver point-of-care relaxometry measurements [3], hyperpolarize exogenous contrast agents [4], increase the number of receive channels in parallel imaging [5–7], implement parallel transmission [7–9], and acquire signals in NMR field monitoring probes concurrently with imaging [10–12]. In particular, many recent systems have been designed around field programmable gate arrays (FPGAs) which perform sequencing and signal processing functions [1,3,7,13–16]. FPGAs are particularly well-suited for MR at Larmor frequencies of tens to hundreds of megahertz since they can process multiple streams of transmitted and received data in parallel at high speeds.

While FPGAs are well-suited to application in high-frequency MR spectrometers, replicating current FPGA-based spectrometers is challenging for non-electronics experts due to the steep learning curve involved in FPGA programming, and since most are based on custom circuitboard designs that would be difficult for non-experts to recreate. At the same time, communications research has benefited in recent years from the development of the open-source GNU

* Corresponding author at: Institute of Imaging Science, Vanderbilt University, 1161 21st Ave. South, MCN AA-3114, Nashville, TN 37232-2310, USA.

E-mail addresses: christopher.j.hasselwander@vanderbilt.edu (C.J. Hasselwander), zhipeng.cao@vanderbilt.edu (Z. Cao), will.grissom@vanderbilt.edu (W.A. Grissom).

http://dx.doi.org/10.1016/j.jmr.2016.06.023
1090-7807/© 2016 The Authors. Published by Elsevier Inc.

Radio software (gnuradio.org), which enables non-hardware experts to build custom software radios that can be used with a wide range of low-cost software-defined radio (SDR) devices; at the time of writing, the GNU Radio website listed ten compatible SDR vendors, many of which offer several SDR models [17]. SDRs typically comprise analog-to-digital and digital-to-analog converters, an FPGA for basic filtering and signal down- and up-conversion, and a USB interface. They can be thought of as PC sound cards that operate at RF frequencies, in that they act as an interface between the digital computer and the analog world, while the PC handles most of the real-time digital signal manipulations. Depending on their feature set, commercial GNU Radio-compatible SDRs currently cost between a few hundred and a few thousand dollars and ship with FPGA software images, so the user can focus on implementing the functionality of their radios on the PC side. Software radios are built in the Python programming language (python.org) in GNU Radio, by connecting modular signal processing components together into a flowgraph, the inputs and outputs of which are connected to the SDR via a driver interface.

We describe an open-source software package that extends the functionality of GNU Radio to perform MRI experiments. The package comprises a set of Python scripts and two C++-based GNU Radio flowgraph elements. It implements system timing calibrations, center frequency and transmit power optimization, shaped RF and gradient pulses, image reconstruction, and three representative MR imaging sequences: gradient echo, spin echo, and inversion recovery. It was used to operate a commercial 0.5 Tesla tabletop MRI scanner with a pair of commercially-available SDRs that generated all RF and gradient pulses and sampled received signals. Overall, the software will enable users to rapidly implement custom MRI spectrometers, without recreating or developing hardware. Since it is built on top of the active GNU Radio project, the software will be compatible with a wide range of current and future SDR devices. By convention, extensions to GNU Radio are prefixed with 'gr-', so the software is called gr-MRI.

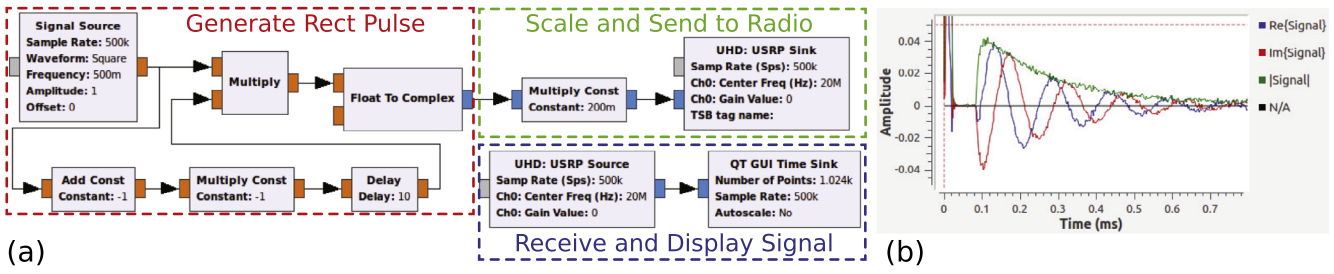
2. Software architecture and implementation

2.1. A basic single-pulse sequence in GNU Radio

To illustrate how GNU Radio works and to motivate the architecture and features of the gr-MRI package, Fig. 1a shows an implementation of the most basic NMR pulse sequence using standard GNU Radio, without gr-MRI. The sequence comprises a single-pulse excitation with simultaneous reception of the free induction decay (FID) signal. Specifically, the figure shows a graphical representation of this sequence's flowgraph in GNU Radio Companion, a GUI-based flowgraph editor packaged with GNU Radio. A GNU Radio flowgraph is made up of signal generation, signal processing, and input and output blocks, which are connected by virtual wires that transmit baseband signals between them. The virtual wires connect to the blocks at orange ports if they are real-valued floating point signals, and at blue ports if they are complex-valued floating point signals. Wires conduct signals in one direction, indicated by the arrows. A signal can be connected to as many inputs as desired, but each input can accept only one signal. All the signal processing implemented in a flowgraph happens in real-time on the PC, with inputs and outputs that are connected via a USB or other interface to stream data continuously between the PC and the SDR. All signals in the flowgraph are at baseband; modulation to and from the RF Larmor frequency is performed digitally by the FPGA chip in the SDR.

In the flowgraph of Fig. 1a, the blocks that produce a baseband rectangular excitation pulse are outlined in red. The pulse is made by generating a square wave signal with period equal to the sequence repetition time (TR) and range zero to one, duplicating it and subtracting one from its copy to shift it to a range of −1 to zero so that the copy is half a period out of phase and negated compared to the original, then negating the copy and delaying it by ten samples, and multiplying it with the original signal to obtain a ten-sample rectangular pulse. The pulse repeats once per TR and its duration in seconds is determined by the sample rate of the flowgraph; in Fig. 1a the 500 kHz sample rate of the flowgraph dictates that the ten sample pulse is twenty microseconds long. Then the real-valued rectangular pulse signal is converted to a complex signal type (with zero imaginary component) and passed into the USRP Sink block (green box), which interfaces to the transmit channels of a Universal Software Radio Peripheral SDR (USRP; Ettus Research, Santa Clara, CA, USA). In this case, only one transmit channel is used. The demodulated received signal comes back into the flowgraph via the USRP Source block (blue box) which interfaces to the receive channels of the SDR; in this case, only one receive channel is used. The received signal is then sent to an oscilloscope block for continuous display. The Larmor frequency is specified as an argument to the USRP Sink and Source so that the SDR's FPGA digitally modulates the excitation pulse from baseband to the Larmor frequency, and demodulates received signals to baseband from the Larmor frequency. The flowgraph is free-running, and does not record data. Fig. 1b shows the oscilloscope window that appears when the flowgraph is executed, which displays the demodulated FID signal in real-time.

This example illustrates that GNU Radio flowgraphs run continuously and are not inherently sequenced as is required for MR experiments. Furthermore, the software lacks the ability to generate shaped waveforms such as sinc excitation pulses and gradient trapezoids with specific timing, as well as the ability to change pulse amplitudes and phases between repetitions. It also lacks



Signal Source
Sample Rate: 500k
Waveform: Square
Frequency: 500m
Amplitude: 1
Offset: 0

Add Const
Constant: -1

Multiply Const
Constant: -1

Delay
Delay: 10

Multiply Const
Constant: 200m

UHD: USRP Sink
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 20M
Ch0: Gain Value: 0
TSB tag name:

UHD: USRP Source
Samp Rate (Sps): 500k
Ch0: Center Freq (Hz): 20M
Ch0: Gain Value: 0

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 500k
Autoscale: No

Fig. 1. (a) Basic single-pulse GNU Radio flowgraph without gr-MRI elements. The outlined boxes contain the rectangular excitation pulse generation blocks (red), blocks that scale the pulse to a desired amplitude and send it to the SDR (green), and the receive chain (blue), comprising a USRP Source block that brings received baseband RF signals back from the radio into the flowgraph, and an oscilloscope block to continuously display the received signal. Orange ports on the blocks denote real-valued floating point inputs and outputs, and blue ports denote complex-valued floating point inputs and outputs. (b) The oscilloscope window that appears when the flowgraph is executed, showing the baseband FID signal in real-time. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

the ability to selectively record received signals over specific time intervals, since the relative timing of transmitted and received signals within the flowgraph is subject to unknown PC-SDR communication delays. The ability to average repeated signals further requires that spectrometer phase drifts are removed from received signals. To address these needs, gr-MRI pulse sequences use master clock signals that trigger all sequence events, and sequences can be dynamically started, stopped, and restarted. gr-MRI provides shaped RF and gradient pulse generation blocks and signal recording blocks, and tools to calibrate gradient strength, center frequency, and RF power. It can synchronize sequence timing between multiple radios, compensate spectrometer phase drifts, and record received signals over desired time intervals. These tools and features are described in the next sections.

2.2. Sequenced pulse generation and signal recording in gr-MRI

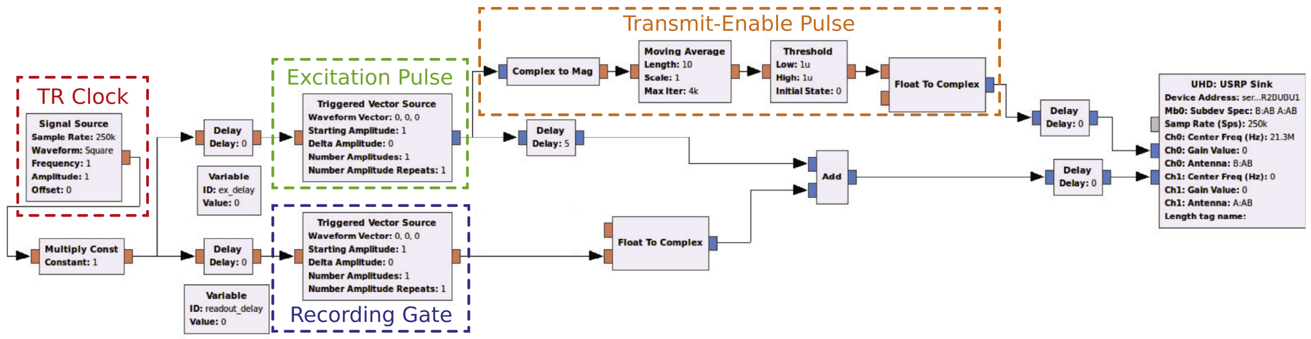
gr-MRI uses square wave signals with period equal to the sequence TR (or TR plus inversion time (TI) in the inversion recovery sequence described below) as master clock signals to trigger pulse sequence events. To generate shaped RF and gradient pulses, gr-MRI provides a C++-based *Triggered Vector Source* block. The block plays real- or complex-valued samples from a user-provided vector that is typically loaded when it is constructed but can also be changed during sequence execution. After its pulse has been played completely, a Triggered Vector Source plays zeros until it receives another trigger. The settings allow the user to specify amplitude stepping for phase encode gradients and RF chopping, and how many times to repeat each step to accommodate averaging. Received signals are recorded using the *Gated Vector Sink* block, which writes a complex data stream from one input to an internal vector whenever its other input is high. The data in that vector is accessible to the user during and after a scan.

2.3. gr-MRI Pulse sequences

Sequence execution. To initiate a gr-MRI scan sequence, the user invokes that sequence's Python script from the IPython shell [18], which loads the sequence and flowgraph parameters from a configuration file, and creates the RF and gradient waveforms and loads them into Triggered Vector Sources in the sequence's flowgraph. The script then optionally launches the flowgraph into an interactive prescan period, during which the user can dynamically adjust sequence parameters such as timing and pulse amplitude settings from the command line, and observe the effect on the received signal which is displayed in real time. Finally the user invokes the full scan, and the entire sequence is run while the script saves raw data into a Gated Vector Sink. After the scan, the user can extract the data from the Gated Vector Sink into a k-space matrix, and reconstruct an image.

Basic single-pulse: `FID.py`. Figs. 2 and 3 show the two parts of a gr-MRI-enabled flowgraph that implements the same single-pulse sequence as in Fig. 1. Fig. 2 shows the flowgraph's transmit section. The red outlined box contains the TR clock signal generator that produces a square wave with period equal to the TR. The clock square wave signal passes through a Multiply Const block which is used to switch the sequence on and off dynamically without starting and stopping the whole flowgraph, by switching the block's `Constant` variable to one or zero. The clock signal triggers two Triggered Vector Sources, which are each loaded with a waveform and settings defined in the Python script. The top Triggered Vector Source produces a scalable rectangular excitation pulse which is sent to the RF power amplifier, and the bottom one generates a longer rectangular pulse that defines the signal recording gate. The latter signal is transmitted from the SDR and directly fed back into one of its receive channels to trigger the Gated Vector Sink in Fig. 3 for signal recording. This loopback mechanism is necessary to account for the aforementioned PC-SDR communication delays which would otherwise prevent accurate timing of signal recording. The looped-back signal is also used to compensate spectrometer phase drift as described below. The blocks outlined in orange generate a transmit-enable pulse that unblanks the RF power amplifier during the excitation pulse, by routing the absolute value of the RF pulse through a 10-sample moving average filter and a thresholding operation, resulting in a rectangular pulse with an amplitude of 1 V that is 10 samples longer than the RF pulse. All other pulses are delayed by 5 samples to center the RF pulses in the transmit-enable window. The transmit-enable pulse is output from the SDR from a channel operating at DC. The rectangular excitation pulse and the signal recording gate pulse are combined as the real and imaginary parts of a single complex-valued signal, and are output from one SDR transmit channel operating at the Larmor frequency. The settings of the USRP Sink block at the end of the flowgraph define which signals are sent to which SDR transmit channels, and their center frequency. The entire flowgraph runs at a baseband sampling rate of 250 kHz, corresponding to a dwell time of 4 µs. The sampling rate can be changed by the user as desired and as allowed by the hardware.

Fig. 3 shows the receive section of the flowgraph for the single-pulse sequence. The far left block is the USRP Source which outputs baseband signals coming from the SDR's receive channels. The top data stream from this block is the complex RF data received from the scanner's preamplifier, and the bottom stream is the looped-back signal recording gate pulse, which serves two purposes in



Signal Source
Sample Rate: 250k
Waveform: Square
Frequency: 1
Amplitude: 1
Offset: 0

Multiply Const
Constant: 1

Delay
Delay: 0

Variable
ID: ex_delay
Value: 0

Delay
Delay: 0

Variable
ID: readout_delay
Value: 0

Triggered Vector Source
Waveform Vector: 0, 0, 0
Starting Amplitude: 1
Delta Amplitude: 0
Number Amplitudes: 1
Number Amplitude Repeats: 1

Triggered Vector Source
Waveform Vector: 0, 0, 0
Starting Amplitude: 1
Delta Amplitude: 0
Number Amplitudes: 1
Number Amplitude Repeats: 1

Moving Average
Length: 10
Scale: 1
Max Iter: 4k

Threshold
Low: 1u
High: 1u
Initial State: 0

Delay
Delay: 5

Delay
Delay: 0

Delay
Delay: 0

UHD: USRP Sink
Device Address: ser...R2BUBU1
Mb0: Subdev Spec: B:AB A:AB
Samp Rate (Sps): 250k
Ch0: Center Freq (Hz): 21.3M
Ch0: Gain Value: 0
Ch0: Antenna: B:AB
Ch1: Center Freq (Hz): 0
Ch1: Gain Value: 0
Ch1: Antenna: A:AB
Length tag name:

Fig. 2. Sequence timing and RF excitation portion of the single-pulse sequence flowgraph generated by `FID.py`. The red outlined box contains the TR clock square wave signal generator, which triggers a pair of Triggered Vector Sources to generate an RF excitation pulse (green box) and a signal recording gate (blue box). A transmit-enable pulse is generated from the excitation pulse's envelope to unblank the RF amplifier (orange). The USRP Sink block on the right routes the baseband signals to two channels on the SDR, operating at RF (for the excitation pulse and the signal recording gate signal) or DC (for the transmit-enable pulse). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

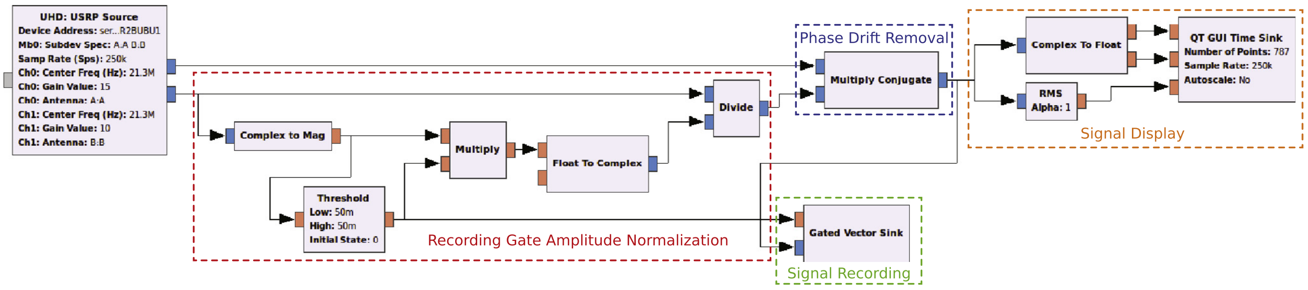


Fig. 3. RF receive portion of the single-pulse sequence flowgraph generated by `FID.py`. The red outlined box contains a chain of blocks that normalize the signal recording gate pulse's amplitude while preserving its phase. That magnitude-normalized signal is then used to remove spectrometer phase drifts from the FID signal phase using the Multiply Conjugate block in the blue box. The phase-corrected signal is recorded by the Gated Vector Sink in the green box, and is displayed in real time by the QT GUI Time Sink block in the orange box. The Gated Vector Sink is gated by the signal recording gate pulse. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

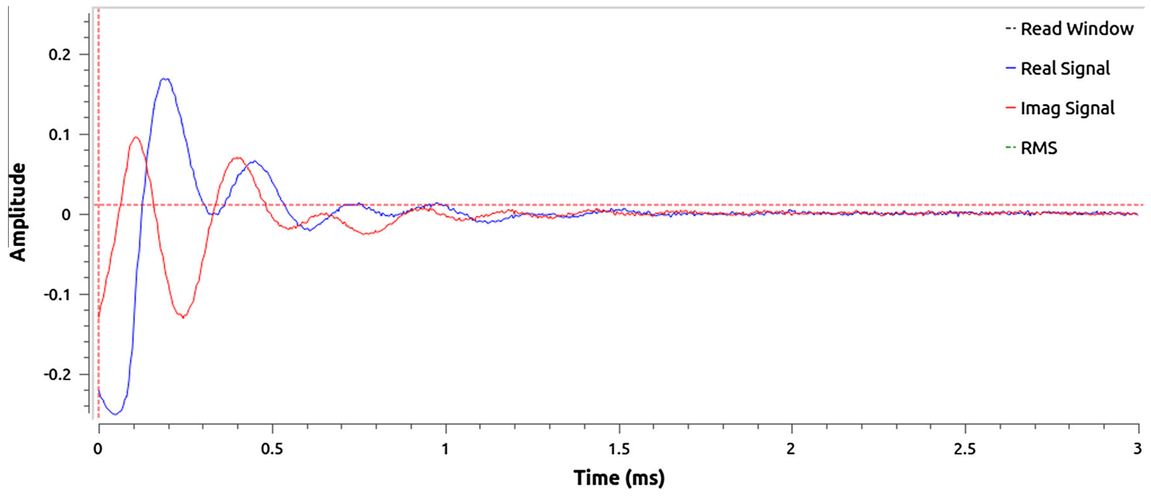


Fig. 4. Screen shot of a received baseband FID signal shown in the QT GUI window while running the single-pulse sequence `FID.py`.

the flowgraph. First, it controls the Gated Vector Sink in the green box so that the FID signal is only recorded over the desired interval. Second, it is used to correct spectrometer phase drifts. To do that, the blocks in the red box first normalize the magnitude of the signal recording gate pulse but preserve its phase by dividing the pulse by its magnitude, and then the received FID signal is multiplied with the normalized signal recording gate pulse's complex conjugate using the *Multiply Conjugate* block in the blue box. This phase removal procedure works because the excitation and signal recording gate pulses originate from the same transmit channel so they experience the same phase drift. The orange box on the far right contains a *QT GUI Time Sink* block that displays the magnitude and real and imaginary parts of the signal in real time. A representative QT GUI Time Sink window is shown in Fig. 4.

The gr-MRI single-pulse sequence implemented in `FID.py` addresses the shortcomings of the non-gr-MRI sequence. Specifically, PC-SDR communication delays and spectrometer phase drifts are compensated using the looped-back signal recording gate pulse, enabling timed signal recording and averaging, and the sequence can generate shaped RF pulse waveforms and change pulse amplitudes and phases between TRs. The same strategies used to develop the single-pulse sequence were applied to enable gradient waveform generation for 2D and 3D imaging. The gr-MRI package includes three common imaging sequences, and a template sequence to enable development of new sequences. These are described next. The software was built using GNU Radio version 3.7.5.1, and uses the scientific Python libraries NumPy (http://www.numpy.org) and Matplotlib (http://matplotlib.org). All communication with SDRs occurs via USRP Sink and Source blocks, which in spite of their vendor-specific name are compatible with all SDRs that are supported by GNU Radio. The full package and a detailed user guide can be downloaded at https://bitbucket.org/wgrissom/gnuradio-mri.

Spin echo: *`spinecho.py`*. Fig. 5 plots the spin echo (SE) pulse sequence produced by `spinecho.py`, and defines some relevant scan parameters. By default the sequence plays a Hamming-windowed sinc excitation pulse concurrently with a slice-select gradient trapezoid in the z direction to localize the excitation to a slice in the sample. Phase encoding and readout prephasor trapezoids occur immediately after the pulse. A rectangular refocusing pulse is played midway between the excitation pulse and the readout window. This is followed by the slice rewinder and the readout gradient and acquisition window which are centered at the echo time (TE). Compared to the single-pulse sequence described above, the SE sequence's flow graph includes three more Triggered Vector Sources to generate gradient pulse waveforms in three spatial dimensions. A video demonstrating the SE sequence can be viewed at https://youtu.be/diGM2nWundI.

Table S1 lists the sequence parameters that can be set in the SE sequence's configuration file `spinecho_config.txt` or dynamically during the prescan period before running the scan. Each time a parameter is changed, all dependent parameters are updated. Parameters can be saved to or loaded from Python pickle (`.pkl`) files. Table S2 lists the user-callable functions defined by `spinecho.py`. This includes functions that set, store and load parameters, and various scan control functions. There is a function to plot the

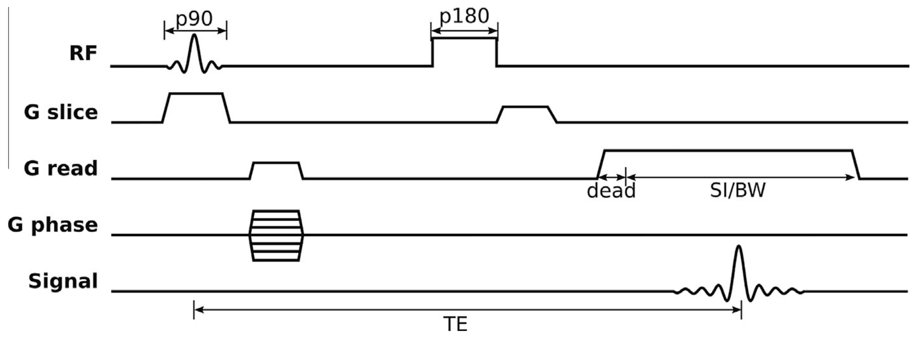


Fig. 5. Slice-selective spin echo pulse sequence generated by `spinecho.py`.

current pulse sequence. Most of the sequence parameters and all the user-callable functions also apply to the other sequences.

Gradient-recalled echo: *`gradecho.py`*. Fig. S1a plots the gradient-recalled echo (GRE) pulse sequence generated by the script `gradecho.py`. Compared to the SE sequence, the GRE sequence lacks a refocusing pulse, and the slice rewinder gradient immediately follows the excitation pulse.

Inversion recovery: *`invrecov.py`*. Fig. S2a plots the inversion recovery (IR) sequence produced by `invrecov.py`. Compared to the SE sequence, the IR sequence adds a rectangular inversion pulse played TI (inversion time) seconds before each excitation and signal readout.

Template sequence: *`template.py`*. gr-MRI also provides a template sequence to enable rapid development of custom pulse sequences. The template is based on `gradecho.py`, and is heavily commented to give instructions on how to edit the sequence. In the simplest case, the user only needs to define the RF pulse shape to generate a valid pulse sequence, however the user can define custom gradient waveforms, and new data acquisition windows if desired; multiple windows could also be defined for multi-echo sequences. The template includes function definitions that provide the same functionality as the full imaging pulse sequences, and its configuration file is preloaded with general parameter names that were used in those pulse sequences.

2.4. System calibrations

Synchronizing Two SDRs. Since one SDR typically does not provide enough channels for an MRI scan, gr-MRI imaging sequences assume the use of two SDRs, one for generating RF waveforms and one for generating gradient waveforms. However, each SDR can experience a different PC-SDR communication delay (the same delay that necessitates the looped-back recording gate signal described previously), resulting in a relative delay between the two. We have empirically found that this relative delay can be as long as 20 ms, and that it can vary somewhat during scanning. Thus, it is essential to continuously synchronize the two SDRs to ensure that gradient waveforms are played at the correct times with respect to the RF pulses and signal acquisition. To achieve this, one SDR is designated the ‘leader’ and the other the ‘follower’. It is assumed that the leader will produce the RF pulses and the transmit enable pulse and the follower will generate the gradient waveforms. Each SDR generates a ten sample rectangular synchronization pulse from a dedicated transmit channel three times per TR, and both synchronization pulses are sent to dedicated receive channels on the follower SDR. The two pulse signals are continuously recorded, and a monitoring function in the gr-MRI imaging sequence script periodically checks the data to compare pulse timings. A desynchronization event is detected if the pulses do not overlap in time. When that happens, the pulse sequence is paused and the function adds a delay to the transmitted signals of the SDR whose pulse appears earlier, to resynchronize. The delays are implemented using `Delay` blocks placed directly in front of each SDR’s USRP Sink. Desynchronization during data acquisition may corrupt the received data, so the frequency with which desynchronization events occurred with our setup was characterized as described below.

Center frequency and transmit voltage. Automatic center frequency and power calibration functions are implemented as part of the `FID.py` pulse sequence, and are listed in Table S3. For center frequency offset calibration, the user specifies the desired number of signal averages to use for FID measurement, and the system displays the FFT of the averaged signal, along with a fit Lorentzian curve. The new center frequency offset is defined to be the frequency corresponding to the peak of the Lorentzian curve, and is set automatically when the script has completed. This calibration can be run multiple times for best accuracy, since SNR increases as the radio’s frequency approaches the Larmor frequency. The power calibration finds the necessary RF pulse amplitude to achieve a 90° flip angle for a fixed-duration rectangular pulse. The user specifies the desired number of signal averages, and the script acquires signals across a range of pulse amplitudes until a maximum received signal amplitude has been reached. Both functions save the optimized parameters to a Python pickle file named `cal.pkl` which is loaded and used by the imaging pulse sequences.

Gradient strength. The gr-MRI package also provides a gradient strength calibration script called `grad_calibration.py`, which uses a spin echo sequence to measure one-dimensional profiles of an object of known size. The user defines the gradient dimension to calibrate, and the object size in that dimension. The sequence then calibrates the gradient strength as a function of radio output voltage based on pre-loaded gradient amplitudes and the frequency bandwidth of the object’s profile. Gradient strengths are saved in units of Gauss/mm/Volt to the file `gcal.pkl` and are used by the imaging sequences to convert desired image FOV and matrix size parameters to gradient pulse amplitudes and step sizes. Table S4 lists the functions available to the user when running the `grad_calibration.py` script. This calibration should only need to be done once for a given scanner.

2.5. Data processing and image reconstruction

All received data is automatically transferred to an object of class `data` after a sequence is run. The object contains the raw data from the sequence’s Gated Vector Sink, and a time stamp indicating when the sequence was initiated. When the `data.recon()` function is called, the data is reformatted (and averaged, if averaging was performed) to create a 2D matrix. If the readout dimension was oversampled with respect to the desired readout bandwidth, the matrix is decimated to the specified matrix size using a 60-tap anti-aliasing FIR filter. Finally, a 2D inverse FFT reconstruction is performed, which corrects for RF chopping if it was used. The formatted k-space data matrix is returned in `data.kdata` and the image is returned in `data.imdata`.

2.6. Workflow summary

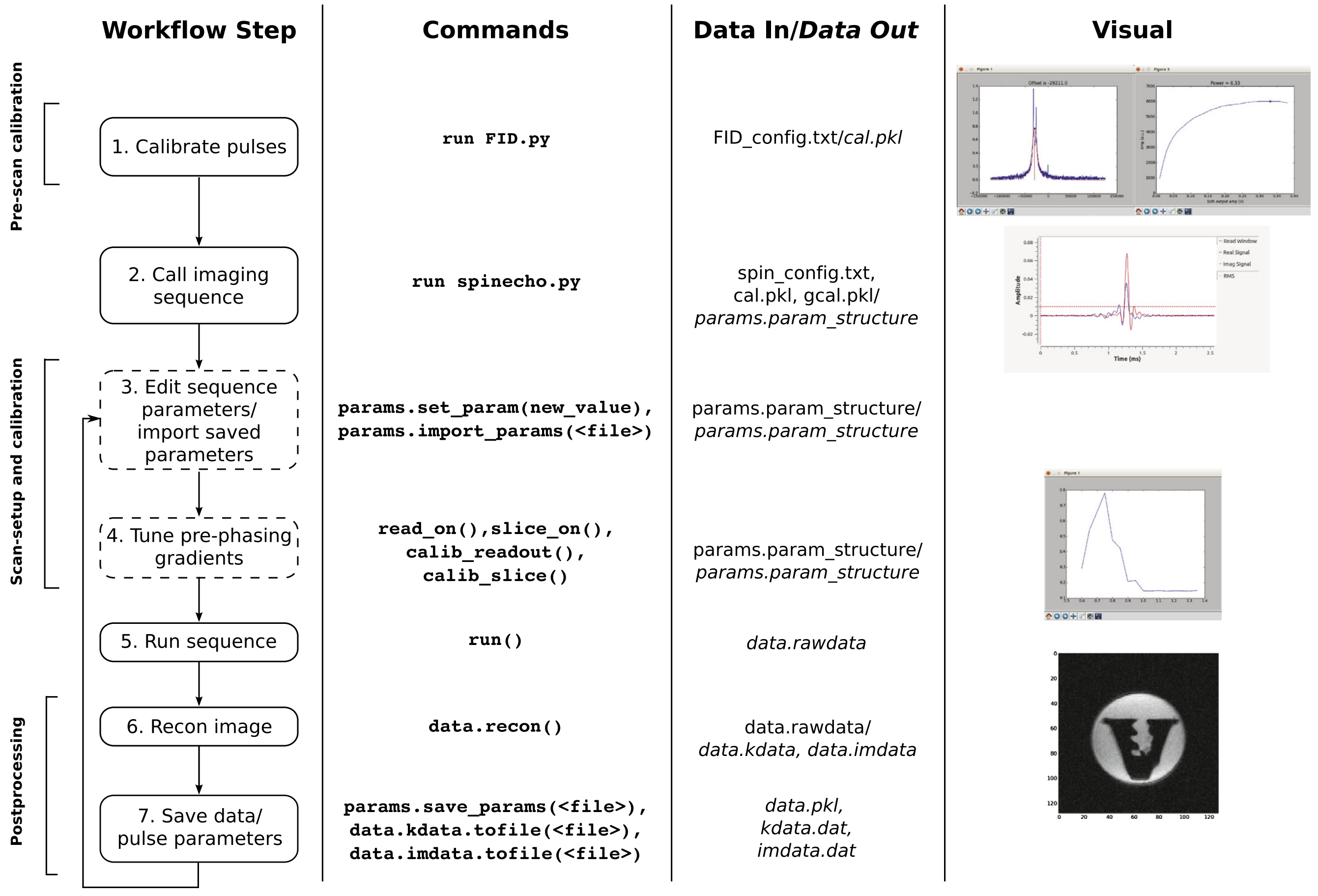
Fig. 6 summarizes the workflow implemented by gr-MRI, for the `spinecho.py` imaging sequence. The diagram shows a workflow step, the output that the step creates and (where applicable) the input the step receives. A gr-MRI imaging experiment comprises the following steps:

1. RF power and center frequency parameters are calibrated and saved using `FID.py`.
2. The user updates the imaging sequence's config file with the desired pulse sequence parameters for their scan (`spinecho_config.txt`). Then they invoke the imaging sequence script (`spinecho.py`). The calibrated parameters from Step 1 are automatically loaded, as well as the parameters from the config file.
3. Interactive Mode starts and the raw signal is displayed in real-time, while the user dynamically adjusts sequence parameters or loads saved parameters. This step is optional.
4. The user optionally fine-tunes gradient pulse moments to center the signal in the acquisition window. The figure for the "Tune pre-phasing gradients" step of Fig. 6 shows an example of a plot that is shown when tuning the slice gradient. The plot is integrated signal amplitude versus a parameter `rephase_fudge`, which is calibrated by the script and will be used to scale the slice rephasing gradient during imaging. The maximum point on this plot corresponds to the scaling of the rephasing gradient amplitude needed to fully cancel the through-slice phase. This step was required periodically on our scanner due to gradient amplifier nonlinearity, but it is optional and it may not be necessary on other scanners.
5. The user runs the sequence. The raw signal from each TR is displayed in a GUI window, and the current phase encode line index is reported, so the user can monitor the scan.
6. After the scan has completed, the user can save the raw k-space data for external reconstruction, or call `data.recon()` to reconstruct an image.
7. The user can save data or parameters, change parameters, or begin a new scan.

3. Validation experiments

3.1. Experimental setup

Experiments were performed on a 0.5 Tesla Oxford Maran tabletop scanner (Resonance Instruments, Witney, U.K.) to validate the imaging sequences and other functions of the gr-MRI software. The scanner has a Maran DRX 2 spectrometer, a Tomco BT00500-AlphaS 500 W RF amplifier (Tomco Technologies, Stepney, Australia), and Analogue Crowne Micro-Tech 600 gradient amplifiers (Crown Audio, Elkhart, Indiana, U.S.A.). The software ran on a PC running Ubuntu 14.04 (Canonical, London, U.K.), with 16 GB RAM and a 4 GHz Intel Core i7-4790 CPU (Intel Corp, Santa Clara, CA, U.S.A.). Prior to imaging scans, gradient calibration was performed using a $1 \times 1 \times 1\ \mathrm{cm}^3$ cube phantom filled with $CuSO_4$-doped



Workflow Step	Commands	Data In/*Data Out*
1. Calibrate pulses	`run FID.py`	FID_config.txt/*cal.pkl*
2. Call imaging sequence	`run spinecho.py`	spin_config.txt, cal.pkl, gcal.pkl/ *params.param_structure*
3. Edit sequence parameters/ import saved parameters	`params.set_param(new_value), params.import_params(<file>)`	params.param_structure/ *params.param_structure*
4. Tune pre-phasing gradients	`read_on(),slice_on(), calib_readout(), calib_slice()`	params.param_structure/ *params.param_structure*
5. Run sequence	`run()`	*data.rawdata*
6. Recon image	`data.recon()`	data.rawdata/ *data.kdata, data.imdata*
7. Save data/ pulse parameters	`params.save_params(<file>), data.kdata.tofile(<file>), data.imdata.tofile(<file>)`	*data.pkl, kdata.dat, imdata.dat*

Fig. 6. Workflow illustration for the `spinecho.py` imaging sequence. Dotted lines in the workflow steps indicate optional steps that can be turned on or off using the `interactive_mode` parameter in the configuration file. The second column shows commands as the user would enter them into the IPython shell. The third column describes the data that is used or produced by each command, and the fourth column shows any images or plots that are created.

water. A solenoid RF coil was used for all scans, which was made with 8 turns of 22 gauge wire, with 16 mm diameter and 10 mm length. All scans used the scanner's preamplifier (45 dB gain) and transmit/receive switch in passive mode. A wiring diagram for the experiments is shown in Fig. 7.

SDR hardware. SDR imaging scans used two Ettus Research USRP1 SDRs, one for RF transmit and receive, and the other for gradient waveform generation. To minimize timing drifts between the two radios, their clocks were connected using instructions provided on the GNU Radio website. The USRP1 comprises an Altera Cyclone FPGA, a 14 bit (−1 V to +1 V), 128 Ms/s digital to analog converter, and a 12 bit (−1 V to +1 V), 64 Ms/s analog to digital converter. The USRP1 can accommodate up to two transmit daughterboards and two receive daughterboards, each of which provides two channels. Each daughterboard can be driven at a unique frequency. The USRP1 connects to the PC by USB. The leader radio in our setup produced the RF excitation pulse and the signal recording gate pulse on one transmit daughterboard, and DC transmit-enable and synchronizing pulses on a second transmit daughterboard. The follower radio produced the gradient pulses and another synchronizing pulse. Table S5 lists the mapping between imaging sequence signals and radio channels. Because the USRP1 produces a maximum pulse amplitude of 1 V but our RF amplifier required at least 3.3 V to unblank, the transmit-enable pulse was used to drive a transistor switch that connected the SDR's 6 V power supply to the RF amplifier's unblanking input.

Imaging scans. An SE imaging scan was run with parameters: TE/TR = 10/1000 ms, 90° flip angle, 128×128 image matrix, 20×20 mm^2 field-of-view, 4 mm slice thickness, 41.7 kHz readout bandwidth, 3 averages. An SE scan was also acquired using the Maran DRX 2 spectrometer with the same parameters. The scans used RF chopping to move DC artifacts to the edge of the FOV. The sequences imaged an 11×8 mm^2 3D-printed Vanderbilt University logo-shaped phantom immersed in a 15 mm-diameter NMR tube filled with sunflower seed oil. The gr-MRI scan was repeated 15 times with and without running interactive mode first, to record the number of leader-follower desynchronization events. Gradient-recalled echo and inversion recovery spin echo images of the phantom were also acquired. The parameters for those scans are listed in Table S6 and the results are shown in Figs. S1 and S2.

Frequency-swept pulse generation. To validate the software and the SDR's ability to generate frequency-swept waveforms, an experiment was performed in which a Triggered Vector Source was used to produce a 500 µs frequency-swept waveform originally designed for Bloch-Siegert B_1^+ mapping [19] (Fig. 9, left). The small signal RF pulse was looped back into a USRP1 receive channel and recorded. For comparison, the same waveform was generated using the Maran scanner's spectrometer, and recorded the same way.

3.2. Results

Imaging scans. Fig. 8 shows images acquired with gr-MRI's `spinecho.py` sequence and the Maran spectrometer with closely-matched parameters. A visual comparison of the images confirms a lack of geometric distortions in the gr-MRI image. Even though the images were acquired using the same signal chain up to the spectrometer, the gr-MRI image has higher SNR than the Maran image. This may be due to a difference in the noise figure of the Maran's receiver, and we do not expect that the USRP1 with gr-MRI would broadly have better noise performance than other spectrometers.

Synchronization analysis. Table 1 shows how often desynchronization events occurred during the `spinecho.py` sequence, with and without running Interactive Mode prior to running the scan. The gr-MRI monitoring function corrected the desynchronization event each time by adjusting the leader and follower delays to compensate. Desynchronization events occurred more frequently immediately after starting a flowgraph because the USB interface dynamically optimizes the data buffer sizes. By running interactive mode first for approximately 30 s, buffering was more likely to stabilize prior to running the full scan. In that case, only one desynchronization event occurred, and there were no data corruptions. When Interactive Mode was not run, one of the ten desynchronization events led to data corruption. Data corruption is the result of a desynchronization event that occurs between the most recent check for synchronization and the time at which the pulses are transmitted.

Frequency-swept pulse generation. Fig. 9 compares small-signal frequency-swept pulses generated by the Maran and the USRP1 with gr-MRI. The USRP1 played the waveform with high fidelity, while the Maran's waveform contained spurious dropouts and undesired phase plateaus due to limited temporal and phase resolution, resulting in large spikes in the transmitted FM waveform. The RMS error of the Maran's amplitude waveform was 24.8%, while that of the radio's was 4.0%. The RMS error of the Maran's frequency waveform was 247 kHz, while that of the radio's was

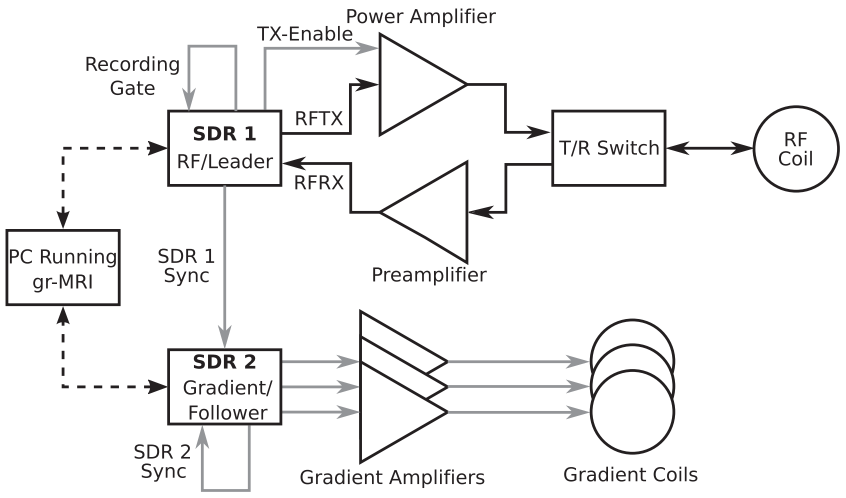


Fig. 7. Hardware wiring diagram for the imaging experiments. The dashed black wires represent USB digital baseband signals sent between the computer running gr-MRI and the SDRs. The gray wires represent DC gradient and control signals, and the solid black wires represent RF signals.

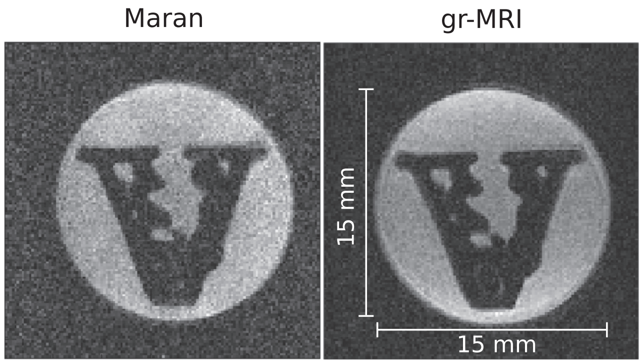


Fig. 8. Spin echo images acquired using (left) the Maran spectrometer, and (right) gr-MRI. The horizontal direction is frequency-encoded.

Table 1
Frequency of desynchronization events and recovery from them, with and without entering Interactive Mode before starting a spin echo imaging scan.

	Interactive mode off	Interactive mode on
TR (s)	1	1
Scan time (s)	128	128
Scans run	15	15
Desync detected	10 (66%)	1 (6.6%)
Desync corrected	10 (100%)	1 (100%)
Data corrupted	1 (10%)	0 (0%)

21 kHz. The gr-MRI errors were largest at the ends of the waveforms.

4. Discussion

4.1. Summary of results

We presented the gr-MRI software package, which comprises a set of Python scripts, flowgraphs, and signal generation and recording blocks for GNU Radio, an open-source SDR software package that is widely used in communications research. gr-MRI implements basic sequencing functionality, and tools for system calibrations, multi-radio synchronization, and MR signal processing and image reconstruction. It includes four pulse sequences: a single-pulse sequence to record free induction signals, a gradient-recalled echo imaging sequence, a spin echo imaging sequence, and an inversion recovery spin echo imaging sequence.

The imaging sequences were validated in 0.5 T phantom imaging experiments and compared to images acquired using a commercial MRI spectrometer. The gr-MRI images were free of distortions, and had similar overall geometry and quality to the commercial spectrometer's images. The ability to generate frequency-swept pulses using gr-MRI was also validated, which is needed for adiabatic excitations [20] and may be useful for RF encoding using the Bloch-Siegert shift [21,22].

4.2. Comparison with other research spectrometers

The primary advantage of gr-MRI is that it enables off-the-shelf SDRs to be used as MRI spectrometers, and requires minimal digital and RF electronics expertise. The primary tradeoff is that the hardware is not optimized for MRI, which necessitated the development of the synchronization methods described here. However, most previously-described MRI spectrometers have been based on new circuit board designs [13,2,1,14,15,6,7,6] that a user would have to replicate. Furthermore, the only spectrometer with open-sourced hardware schematics is the OPENCORE NMR platform [14,15], and it does not provide gradient control channels. Several spectrometers have been based wholly or in part on National Instruments hardware that is straightforward to use but considerably more expensive than SDRs [2,12]. A further advantage of gr-MRI (which it inherits from GNU Radio) is that it is not tied to a single device, so it will be compatible with many existing and future SDRs.

In gr-MRI, imaging scans are prescribed and controlled via an iPython command-line interface. Some previous spectrometers have been based on National Instruments hardware and have GUIs implemented in National Instruments' LabView [2] software. The Medusa console [7] provides a MATLAB-based user interface. LabView and MATLAB interfaces would likely be easier for novices to use than the gr-MRI command-line interface, though it would be straightforward to build a GUI for gr-MRI in the future using Python GUI elements. Some previous spectrometers have also provided more advanced pulse programming capabilities, enabling users to define pulse sequences in text files or even graphical editors [1,2]. However, to our knowledge that software is not publicly available.

4.3. SDR hardware requirements & considerations

gr-MRI was validated in this work using Ettus Research USRP1s. The USRP1 has 12 bit ADCs and 14 bit DACs. This is within the range of commercial spectrometers which typically have between 12 and 16 bit ADCs and DACs. The USRP1's maximum Larmor frequency is 32 MHz, which is dictated by its 64 MS/s receiver sampling rate. Without additional hardware, this corresponds to an upper field strength limit of 0.75 Tesla for proton imaging, though other nuclei with lower gyromagnetic ratios could be imaged at higher field strengths. Operation at higher frequencies would require high-frequency transmit/receive daughterboards with onboard modulation/demodulation circuits, or external mixing. Care would need to be taken to make sure that the reference oscillator signals for modulation and demodulation are phase-locked; the loopback phase corrections implemented in gr-MRI may also

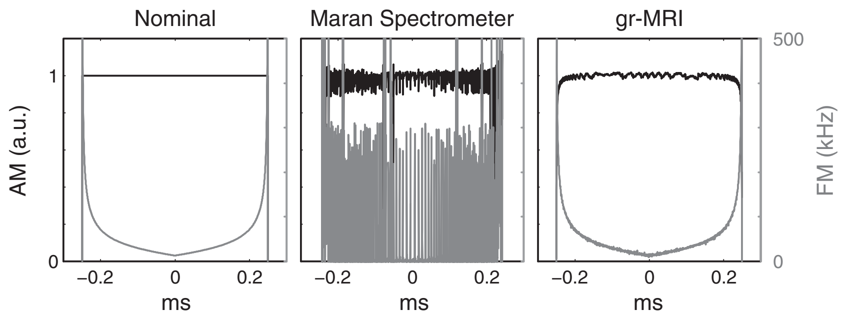


Fig. 9. Comparison of nominal and measured amplitude modulation (AM) and frequency modulation (FM) waveforms of frequency-swept pulses generated by the Maran spectrometer and the USRP1 SDR with the gr-MRI Triggered Vector Source block.

be helpful in that scenario. Another consideration is baseband RF and gradient sampling rate and readout bandwidth. On an SDR the limits for these parameters are dictated by the baseband sampling rate. Today's clinical MRI scanners use baseband RF and gradient waveform resolutions between 4 and 10 μs. The imaging results presented in our validations used the coarsest resolution that can be selected on the USRP1, 4 μs. However, the USRP1 can be used with baseband sampling rates as high as 8 MS/s. Thus we expect that the baseband resolution of modern SDRs is within the range of what is required for MRI, even at high readout bandwidths. Finally, we also note that since baseband transmitted and received data are continuously streamed between the PC and SDR, PC-SDR data transfer rates and buffering are independent of receiver duty cycle in a pulse sequence, so long readouts and multi-echo scans could be immediately implemented with gr-MRI.

While the USRP1 is well-suited to MRI at low field strengths, it is the first Ettus SDR and has a more basic feature set than more recent models (though it was available for purchase from Ettus at the time of writing). The gr-MRI software is compatible with any GNU Radio-compatible SDR, though some of the features we developed based on the USRP1's capabilities may not be required for newer SDRs and could be removed as desired by users. For example, some newer SDRs have gigabit ethernet connections to the PC, which reduces latency considerably compared to USB. We used looped-back signals as a general solution to latency and between-radio delays; with lower and more stable latency, pulse sequences may not require this. Additionally, some modern SDRs allow for time synchronization via time-stamped transmissions or with a MIMO cable. These simplifications would free up valuable I/O channels. Commercial SDRs with onboard ARM-based Linux PCs that can run GNU Radio are also becoming available for embedded applications (such as the Ettus E300 series), which in the future may enable sequences to be run directly from the SDR and received data to be processed and stored on it until the user requests it. SDRs with more channels than the USRP1 may be able to perform both RF and gradient functions.

To obtain a 6 V RF amplifier unblanking pulse, we used the SDR's 1 V transmit enable pulse to turn on a transistor switch that connected a 6 V DC power source to the amplifier whenever the pulse was high. It may be possible to use SDR I/O pins to produce these and other control signals (such as signals needed for coil detuning circuits); such pins exist on the USRP1 motherboard but were not supported by GNU Radio at the time of writing.

5. Conclusion

gr-MRI enables commercially-available software-defined radios to be used as low-cost custom MRI spectrometers, with fidelity that is comparable to commercial spectrometers. It was designed to be highly customizable and reconfigurable. This will make it easier for researchers and engineers to develop custom spectrometers, without requiring significant spectrometer hardware development or FPGA programming.

Acknowledgments

This work was supported by NIH grant R21 EB 018521. The authors would like to thank Pooja Gaur for help in editing the manuscript.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.jmr.2016.06.023.

References

[1] S. Jie, X. Qin, L. Ying, L. Gengying, Home-built magnetic resonance imaging system (0.3 T) with a complete digital spectrometer, Rev. Sci. Instrum. 76 (2005) 105101.
[2] S.M. Wright, D.G. Brown, J.R. Porter, D.C. Spence, E. Esparza, D.C. Cole, F. Russel Huson, A desktop magnetic resonance imaging system, MAGMA 13 (2002) 177–185.
[3] W.K. Peng, L. Chen, J. Han, Development of miniaturized, portable magnetic resonance relaxometry system for point-of-care medical diagnosis, Rev. Sci. Instrum. 83 (2012) 095115.
[4] S.R. Parnell, E.B. Woolley, S. Boag, C.D. Frost, Digital pulsed NMR spectrometer for nuclear spin-polarized ^{3}He and other hyperpolarized gases, Meas. Sci. Technol. 19 (2008) 045601.
[5] J. Bodurka, P.J. Ledden, P. van Gelderen, R. Chu, J.A. de Zwart, D. Morris, J.H. Duyn, Scalable multichannel MRI data acquisition system, Magn. Reson. Med. 51 (1) (2004) 165–171.
[6] W. Tang, H. Sun, W. Wang, A digital receiver module with direct data acquisition for magnetic resonance imaging systems, Rev. Sci. Instrum. (2012) 104701.
[7] P.P. Stang, S.M. Conolly, J.M. Santos, J.M. Pauly, G.C. Scott, Medusa: a scalable MR console using USB, IEEE Trans. Med. Imag. 31 (2) (2012) 370–379.
[8] P. Stang, A.B. Kerr, J.M. Pauly, G. Scott, An extensible transmit array system using vector modulation and measurement, in: Proceedings 16th Scientific Meeting, International Society for Magnetic Resonance in Medicine, Toronto, 2008, p. 145.
[9] W.A. Grissom, D. Xu, A.B. Kerr, J.A. Fessler, D.C. Noll, Fast large-tip-angle multidimensional and parallel RF pulse design in MRI, IEEE Trans. Med. Imag. 28 (10) (2009) 1548–1559.
[10] C. Barmet, N. De Zanche, B.J. Wilm, K.P. Pruessmann, A transmit/receive system for magnetic field monitoring of in vivo MRI, Magn. Reson. Med. 62 (2009) 269–276.
[11] P.T. Sipilä, S. Greding, G. Wachutka, F. Wiesinger, ^{2}H transmit–receive NMR probes for magnetic field monitoring in MRI, Magn. Reson. Med. 65 (2011) 1498–1506.
[12] P.T. Sipilä, R.F. Schulte, G. Wachutka, F. Wiesinger, Digital multiband receiver for magnetic resonance, Conc. Magn. Reson. Part B: Magn. Reson. Eng. 35B (4) (2009) 210–220.
[13] L. Gengying, J. Yu, Y. Xiaolong, J. Yun, Digital nuclear magnetic resonance spectrometer, Rev. Sci. Instrum. 72 (2001) 4460.
[14] K. Takeda, A highly integrated FPGA-based nuclear magnetic resonance spectrometer, Rev. Sci. Instrum. 78 (2007) 033103.
[15] K. Takeda, OPENCORE NMR: open-source core modules for implementing an integrated FPGA-based NMR spectrometer, J. Magn. Reson. 192 (2008) 218–229.
[16] W. Tang, W. Wang, A single-board NMR spectrometer based on a software defined radio architecture, Meas. Sci. Technol. 22 (2011) 015902.
[17] A Quick Guide to Hardware and GNU Radio. <https://gnuradio.org/redmine/projects/gnuradio/wiki/Hardware> (Accessed: December 2015).
[18] F. Pérez, B.E. Granger, IPython: a system for interactive scientific computing, Comput. Sci. Eng. 9 (3) (2007) 21–29.
[19] M. Jankiewicz, J.C. Gore, W.A. Grissom, Improved encoding pulses for Bloch-Siegert B_1^+ mapping, J. Magn. Reson. 226 (2013) 79–87.
[20] M. Garwood, L. DelaBarre, The return of the frequency sweep: designing adiabatic pulses for contemporary NMR, J. Magn. Reson. 153 (2) (2001) 155–177.
[21] R. Kartäusch, T. Driessle, T. Kampf, T.C. Basse-Lüsebrink, U.C. Hoelscher, P.M. Jakob, F. Fidler, X. Helluy, Spatial phase encoding exploiting the Bloch-Siegert shift effect, Magn. Reson. Mater. Phys. 27 (2014) 363–371.
[22] Z. Cao, E.Y. Chekmenev, W.A. Grissom, Frequency encoding by Bloch-Siegert shift, in: Proceedings 22nd Scientific Meeting, International Society for Magnetic Resonance in Medicine, Milan, 2014, p. 4220.