gr-MRI: A Software Package for Magnetic Resonance Imaging Using Software Defined Radios

By Christopher Jordan Hasselwander

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Biomedical Engineering

May, 2016

Nashville, Tennessee

Approved:

William A. Grissom, Ph.D.

Brett Byram, Ph.D.

ACKNOWLEDGMENTS

Firstly, I want to thank my advisor, Dr. Will Grissom for accepting me into his lab, when I reached out to him as an undergrad in his instrumentation class. His clear enthusiasm for his work, and teaching were what drew me to his lab, and I would likely not be here without him. He has provided a great environment in which I was given the opportunity to learn a great deal from him, but also given the flexibility to learn on my own.

I want to thank my lab mates and office mates for their friendship and for creating a unique and fun place to spend these last couple of years. I also want to Dr. Brett Byram and Dr. Mark Does for serving on my committee and helping me to mold this project.

Finally, and perhaps most importantly, I would like to thank my family for supporting me through this whole process, and for pretending to listen instead of calling me names when I carried on about MRI details for too long.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The research presented in this work focuses on the development of an open source software package, and specific hardware solution to implement a high fidelity magnetic resonance (MR) spectrometer using commercially-available software defined radios (SDR). Specifically we show that one can home-build a high performance MR spectrometer, with little hardware modification for roughly \$2,000. This chapter discusses the basic components of a magnetic resonance imaging (MRI) system, and MR spectrometer, and gives an overview of the use of field programmable gate array (FPGA) based devices for custom MR spectrometers. We will then discuss the specific gr-MRI software solution and system architecture.

1.1 MR System Overview

Modern commercial MRI and nuclear magnetic resonance (NMR) spectrometers are sophisticated devices with very high performance. Figure 1.1 shows a block diagram for a typical modern MR scanner. There are many basic components that comprise a modern RF spectrometer and MRI system, which are defined by the way that MR signals are produced, detected, and processed.

1.1.1 Magnet

The first necessary component of a MR, or NMR system is the magnet that creates the main magnetic field. Some major considerations regarding the magnet are its strength, stability, and field uniformity. The procession frequency at which protons spin in a magnetic field, or the Larmour Frequency, is directly dependent on the strength of the magnetic field. A larger main magnetic field will increase the Larmour Frequency, and thus increase the nec-

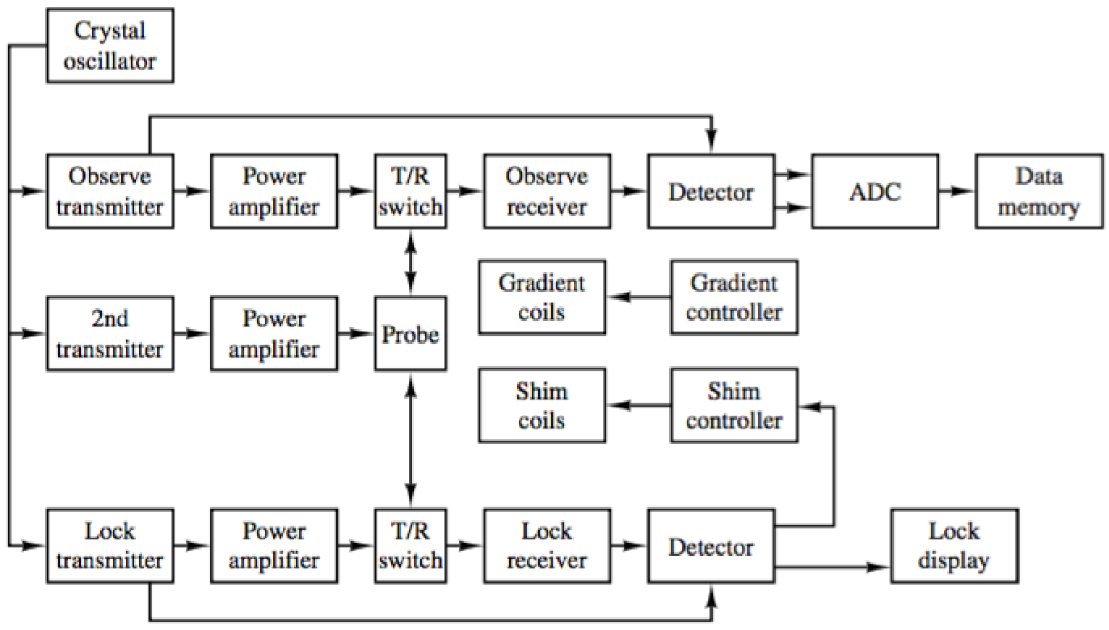


Figure 1.1: Block diagram of a typical 2 channel modern MR signal pathway [1].

essary center frequency of the spectrometer system. Additionally, the field must be uniform enough to observe spin features resulting from the physical structure that is being analyzed, which for imaging applications usually requires a field uniformity of roughly 0.1 ppm over the imaged volume [1]. The magnets used for this study were two aligned permanent magnets resulting in a 0.5 Tesla (T) main field, but most modern clinical scanners use super cooled electromagnets with field strengths ranging from 1.5T to upwards of 7T.

1.1.2 RF Probe

The most basic function of the RF probe is to couple nuclear spins to the RF transmitter or receiver. When transmitting, the RF probe generates a magnetic field perpendicular to the main magnetic field, affecting the magnetization vector in the sample. When receiving, the probe converts the magnetic field changes created by the relaxation of nuclear spins into an electrical current which is sent back to the MR spectrometer via an amplifier. The probe is electrically tuned to the Larmour frequency, and typically matched to 50 Ohms impedance.

There have been many sophisticated advances in the field of RF coils, but at its simplest, a probe can simply be a loop of wire.

1.1.3 Magnetic Field Gradients

Magnetic field gradients, referred to simply as gradients in the rest of the paper, are coils which play high voltage DC signals, which purposely create linear field inhomogeneities. MR imaging depends on the use of gradients to spatially map the received signals to specific frequencies generated by the gradients. Typically during an imaging experiment, gradients are pulsed to create transient field gradients with properties creating desired linear phase or frequency shifts across an object in three dimensions, resulting in the localization of signals. Additionally, gradients are often played during RF transmission to couple nuclear excitation to specific spatial locations.

1.1.4 MR Spectrometer

The MR spectrometer is responsible for the user controlled sequencing of RF and gradient pulses, as well as other gating pulses which control external hardware necessary for MRI. Any modern MR spectrometer consists of several basic components, which when combined, enable MR imaging.

1.1.4.1 Transmitter

The RF transmitter should be able to adapt to a wide range of potential frequencies and applications in MR. MR spectrometers usually use a highly stable 10MHz crystal oscillator, or something similar, which provides an RF source from which other frequencies can be derived [1]. The spectrometer usually includes a general purpose frequency synthesizer, so that it can produces RF pulses in a wide range of frequencies, as well as high speed, and high resolution digital to analog converters (DACs) to create the actual pulses. Modern spectrometers allow for user controlled RF transmission through time dependent amplitude modulation, which results in the ability to produce arbitrary pulse shapes. Additionally, spectrometers allow the user to modulate the phase of pulses, which is important for many imaging applications, such as moving the location of slice selective excitation or adiabatic swept pulse generation [2]. The signal produced by the spectrometer is generally too small to create the required magnetic field (normally around 0dBm), so the output from the transmitter is typically run through a power amplifier before being sent to the probe.

1.1.4.2 Receiver

The receive chain is responsible for converting the small signal detected by the probe into a digital representation of that signal, which is saved in memory. The first step in the receive chain is usually amplified 60-100dB by the preamplifier before the coupling with the spectrometer interface [1]. The spectrometer creates a complex signal by splitting the received signal into two streams and adding 90 degrees of phase to one before they are sent through independent high speed analog to digital converters (ADCs). Modern spectrometers typically used 12-16 bit ADCs which achieve high signal resolution, and typically the preamplifier is scaled so that the received signal uses the full dynamic range of the ADC. At minimum, the ADC must sample at least twice as fast as the highest frequency component in the signal to avoid aliasing, however it is typically advantageous to use as high a receive bandwidth as possible because noise voltage decreases with increased bandwidth. The high sampling rates create a high volume of data, which is typically down sampled before being sent to a computer, or other digital control system.

1.1.4.3 Digital Control

MR spectrometers also include some sort of high performance digital control such as a computer to coordinate the other spectrometer functions. The computer creates RF pulses, and sets timing for the transmission of those pulses, gradient pulses, and hardware control pulses in a pulse sequence, however most time critical pulses are loaded into the spectrometer and handled by hardware. The computer also specifies the timing and duration of data acquisition. Further, it defines the data sampling strategy, and keeps track of the data received, which is saved to memory for post processing.

1.2 Overview of Software Defined Radios in MR and NMR

While modern commercial spectrometers deliver high performance, 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 [3, 4], deliver point-of-care relaxometry measurements [5], hyperpolarize exogenous contrast agents [6], increase the number of receive channels in parallel imaging [7, 8, 9], implement parallel transmission [9, 10, 11], and acquire signals in NMR field monitoring probes concurrently with imaging [12, 13, 14]. In particular, many recent systems have been designed around field programmable gate arrays (FPGAs) which perform sequencing and signal processing functions [15, 3, 16, 17, 18, 5, 9]. 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 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 list ten compatible SDR vendors, many of which offer several SDR models [19]. SDR's 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 SDR's 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 flow-graph, the inputs and outputs of which are connected to the SDR via a driver interface.

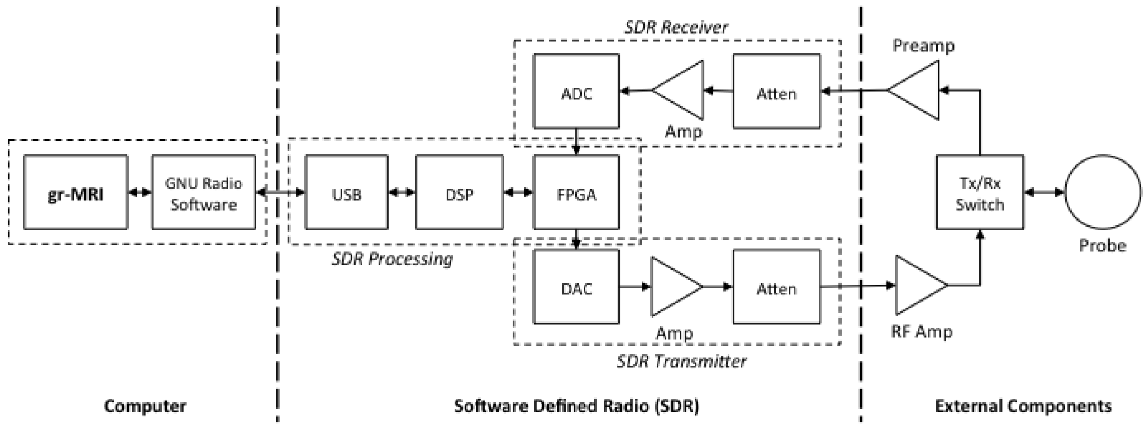


Figure 1.2: Block diagram showing the basic signal chain components for a single RF channel in SDR enabled MRI. The left side shows what the computer produces, the middle shows the signal processing that happens in the SDR, and the right shows external MRI specific hardware.

Figure 1.2 shows the basic signal chain for a single RF channel of SDR enabled MRI. The software exists in the computer, where it is executed and sends instructions to the SDR via a USB connection. The FPGA creates and sequences the final signals that are then converted to analog signals. The analog signal can be either amplified or attenuated before it is sent out of the radio to the RF amplifier, and to the Probe via the transmit / receive switch. The received signal is amplified by the preamp, and returned to the radio where it can be further amplified or attenuated. The ADC converts the signal to a digital signal, where the FPGA and digital signal processor (DSP) handle down converting and filtering the digital signal. The data stream is then returned to the computer and real time signal processing software. Similar signal chains exist for transmit only pulses such as gradient waveforms and hardware control pulses.

Chapter 2

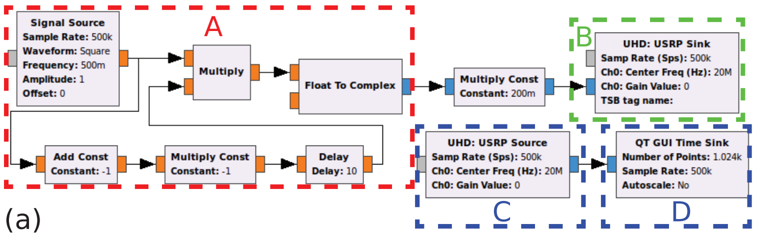
gr-MRI Software Package Overview and Architecture

2.1 Introduction

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 new 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. The full package and a detailed user guide can be downloaded at https://bitbucket.org/wgrissom/gnuradio-mri.

2.2 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, Figure 2.1a shows an implementation of the most basic NMR pulse sequence using GNU Radio software alone, 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; wires conduct signals in one direction. In this flowgraph, the blocks that produce a block excitation pulse are outlined in box A. The pulse is generated by duplicating a square wave, inverting and delaying its copy, and multiplying the copy by itself. Then the real-valued block pulse signal is converted to a complex signal type (with zero imaginary component) and passed into the USRP Sink block (box B), which interfaces via a driver to one or more transmit channels of a Universal Software Radio Peripheral SDR (USRP; Ettus Research, Santa Clara, CA, USA). The demodulated received signal comes back into the flowgraph via the USRP Source block (box C), which sends the amplitude of the signal to an oscilloscope block for continuous display (box D). The flowgraph is free-running, and does not record data. Figure 2.1b shows the oscilloscope window that appears when the flowgraph is executed, which displays the demodulated FID signal in real-time.



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

Multiply

Float To Complex

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

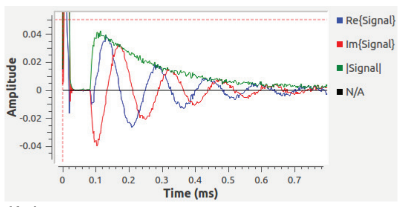


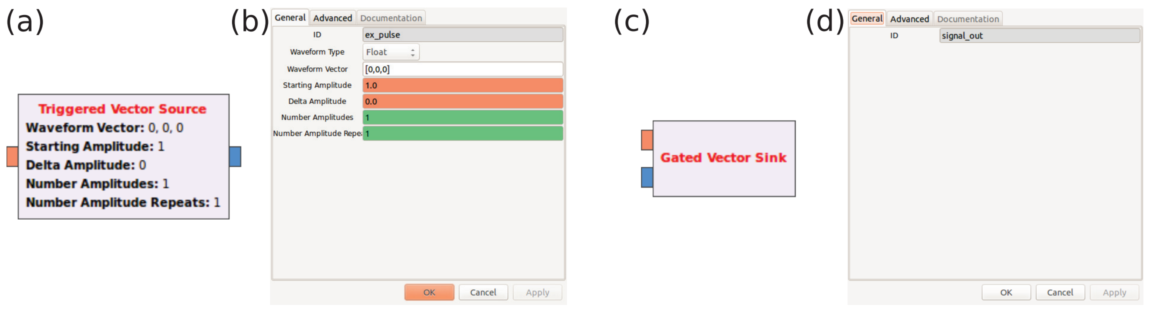
Figure 2.1: (a) Basic single-pulse GNU Radio flowgraph without gr-MRI elements. The outlined boxes contain: A) The excitation pulse generation blocks, B) the USRP Sink block which sends the excitation pulse to the radio, C) the USRP Source block which receives baseband RF signals from the radio, and D) the oscilloscope block to continuously display the received signal. (b) The oscilloscope window that appears when the flowgraph is executed, showing the FID signal in real-time.

This example illustrates that GNU Radio flowgraphs run continuously and are not inherently sequenced as is required for MR. Furthermore, the precise timing of transmitted signals is subject to delays on the PC, and the software lacks the ability to generate shaped waveforms such as sinc excitation pulses and gradient trapezoids, as well as the ability to change pulse amplitudes and phases between repetitions. It also lacks the ability to selectively record received signals over specific time intervals. To address these needs, gr-MRI pulse sequences use master clock signals that trigger provided RF and gradient pulse generation and signal recording blocks. gr-MRI further provides tools to calibrate gradient strength, center frequency, and RF power, and to synchronize sequence timing in order to compensate system delays between RF and gradient pulse transmission and signal reception. These tools and features are described in the next sections.

2.3 Sequenced pulse generation and signal recording in gr-MRI

gr-MRI Uses square wave signals with period equal to the sequence repetition time (TR) (or TR plus inversion time (TI) in an inversion recovery sequence, described further below) to trigger pulse sequence events. To generate shaped RF and gradient pulses, gr-MRI provides a C++-based *Triggered Vector Source* block, illustrated in Figure 2.2a. 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 (Figure 2.2b) 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, shown in Figure 2.2c, which writes a complex data stream from one input to an internal vector whenever its other input is high.

To initiate a gr-MRI scan sequence, the user invokes that sequence's Python script from the IPython shell [20], which creates the RF and gradient waveforms and loads them into Triggered Vector Sources in a new sequence 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 the 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 using provided functions described below.



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

General	Advanced	Documentation

ID	ex_pulse
Waveform Type	Float
Waveform Vector	[0,0,0]
Starting Amplitude	1.0
Delta Amplitude	0.0
Number Amplitudes	1
Number Amplitude Repe	1

Gated Vector Sink

General	Advanced	Documentation

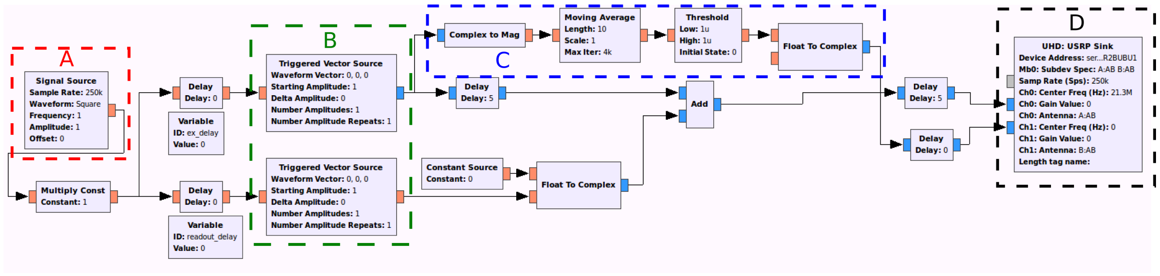
ID	signal_out

Figure 2.2: (a) The Triggered Vector Source GNU Radio block. The orange box on the left is the block's input socket and the blue box on the left is its output socket. (b) The Triggered Vector Source block's settings. (c) The Gated Vector Sink GNU Radio block. The blue box on the left represents the complex data input socket, and the orange box represents the gate socket which controls data recording. (c) The Gated Vector Sink block's settings.

2.4 gr-MRI Pulse sequences

2.4.1 Basic single-pulse: `FID.py`

Figures 2.3 and 2.4 show a gr-MRI-enabled flowgraph that implements the same single-pulse sequence as in Figure 2.1, but uses a Triggered Vector Source to generate arbitrary RF excitation pulses, and a Gated Vector Sink to record data. Figure 2.3 shows the flowgraph's transmit section. The outlined box A contains the TR clock signal generator that produces a square wave with period equal to the TR. The output signal triggers two Triggered Vector Sources in outlined box B, which are each loaded with a waveform and settings defined in the Python script. The top Triggered Vector Source produces a scalable block excitation pulse which is sent to the RF power amplifier, and the bottom one generates a longer block pulse that defines the signal recording interval. The latter signal is transmitted from the SDR and fed back into one of the SDR's receive channels to trigger the Gated Vector Sink in Figure 2.4 for signal recording. This loopback mechanism is necessary because variable USB latency during transmit makes it impossible to know exactly when the pulses were transmitted by the SDR with respect to the received signal's time basis. The signal is also used to compensate phase drift as described below. The blocks in box C 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 square pulse with an amplitude of 1 Volt 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 block 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 the SDR from two channels operating at the Larmor frequency. The settings of the USRP Sink block in box D define which signals are sent to which channels, and their center frequency.



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

Complex to Mag

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

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

Float To Complex

Delay
Delay: 5

Constant Source
Constant: 0

Float To Complex

Add

Delay
Delay: 5

Delay
Delay: 0

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

Figure 2.3: Sequence timing and RF excitation flowgraph of the single-pulse sequence `FID.py`. The outlined boxes contain: A) The TR clock square wave signal generator, which triggers all sequence events. B) Triggered Vector Sources that generate an RF excitation pulse (top) and a signal recording gate (bottom). C) A transmit-enable pulse is generated from the RF excitation pulse to unblank the RF amplifier. D) The USRP Sink block routes the baseband signals to three channels on the SDR, operating at RF (for the excitation pulse and the signal recording gate) or DC (for the transmit-enable pulse).

Figure 2.4 shows the receive section of the flowgraph for the single-pulse sequence. The far left block in outlined box A is the USRP Source which outputs baseband signals coming from the SDR's receive channels. The bottom data stream from this block is the complex RF data received from the scanner's preamplifier, and the top stream is the looped-back signal recording gate pulse, which serves two purposes in the flowgraph. First, it controls

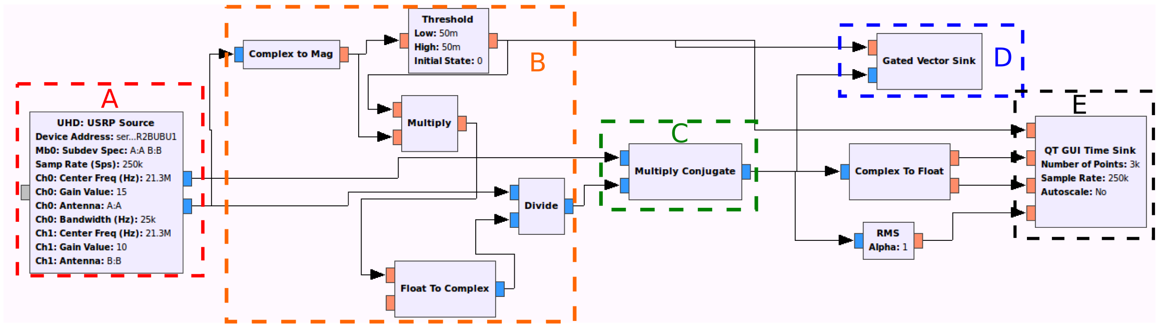


Figure 2.4: RF Receive flowgraph of the single-pulse sequence `FID.py`. The outlined boxes contain: A) the USRP Source, which converts received data into a complex data stream; B) a group of blocks used to normalize the signal recording gate pulse's amplitude while preserving its phase; C) a multiply conjugate block that removes the signal recording gate pulse's phase from the FID signal phase to correct phase drifts; D) the gated vector sink which stores the raw data to a vector; and E) a QT GUI sink used to display the received signal in real time.

the Gated Vector Sink in box D so that the FID signal is only recorded over the desired interval. Second, it is used to correct spectrometer phase drifts. This is done by the blocks in box B, which normalize the amplitude of signal recording gate pulse, but preserve its phase, and the received FID signal is multiplied with the normalized signal recording gate pulse's complex conjugate in box C. This works because the both the excitation and the signal recording gate pulses originated from the same daughterboard with the same phase. The far right block in box E is a *QT GUI Sink* that is used to display the received signal in real time. A representative QT GUI Sink window is shown in Figure 2.5 for the spin echo imaging sequence (described below).

The same strategies used to develop the FID 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 sequences 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 SDR's occurs via USRP Sink and Source blocks, which in spite of their vendor-specific name are compatible with all SDR's that are supported by GNURadio.

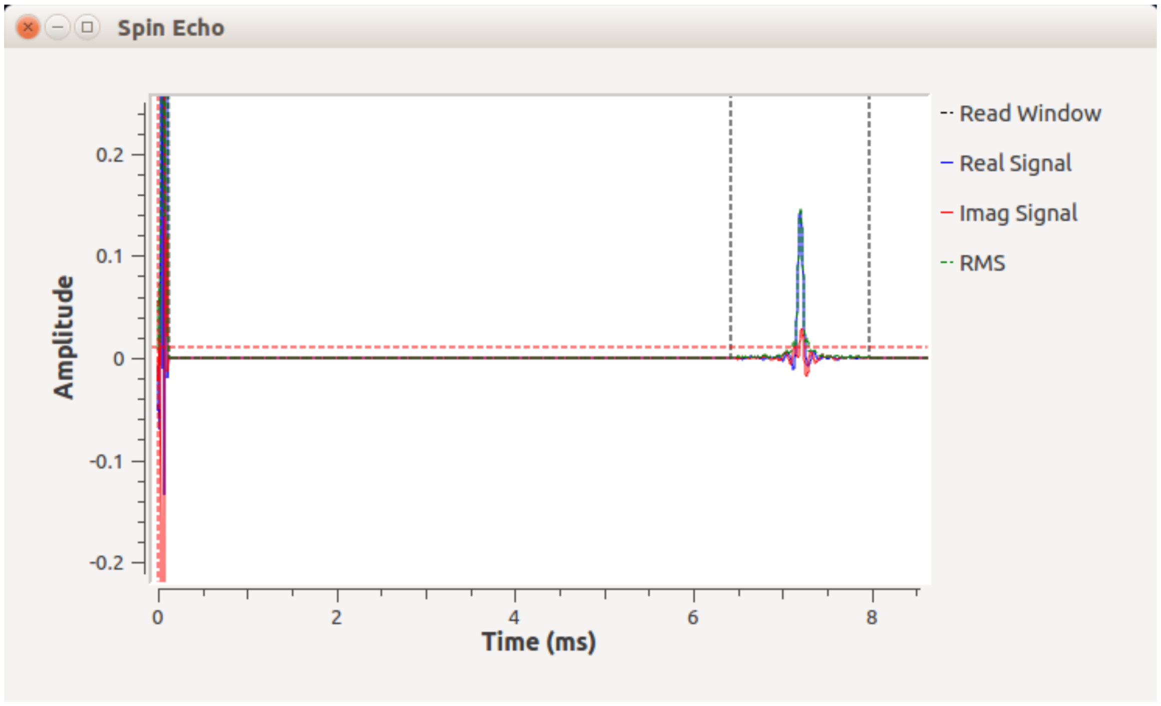


Figure 2.5: Screen shot of the receive data shown in the QT GUI window while running the spin echo sequence `spinecho.py`.

2.4.2 Gradient-recalled echo: `gradecho.py`

Figure 2.9a plots the gradient-recalled echo (GRE) pulse sequence generated by the script `gradecho.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, slice rewinding and readout prephasor trapezoids occur immediately after the pulse, followed by the readout gradient and acquisition window which are centered at the echo time (TE). Compared to the single-pulse sequence described above, the GRE sequence's flow graph includes three more Triggered Vector Sources to generate gradient pulse waveforms in three spatial dimensions.

Table 2.1 lists the sequence parameters that can be set in the configuration file `gre_config.txt` or during the prescan period before running the scan, and Table 2.2 shows the user-accessible functions defined by `gradecho.py` and the other imaging sequences described below. Many of the parameters in Table 2.1 also apply to the other sequences, as indicated. All scan parameters set in the config file can be changed dynamically while running the scan by calling `params.set_{param_name}({value})`, where `param_name` is the name of the parameter to be changed, and `value` is the new value to be assigned. Each time a parameter is changed, the parameter update functions also update all dependent pulse sequence parameters. Parameters can be saved to Python pickle (`.pkl`) files and can also be loaded from pickle files when operating in interactive prescan mode.

2.4.3 Spin echo: `spinecho.py`

Figure 2.10a plots the spin echo (SE) pulse sequence produced by `spinecho.py`. Compared to the GRE sequence, the SE sequence adds a block refocusing pulse midway between the excitation pulse and the readout window.

2.4.4 Inversion recovery: `invrecov.py`

Figure 2.11a plots the inversion recovery (IR) sequence produced by `invrecov.py`. Compared to the SE sequence, the IR sequence adds a block inversion pulse played TI (inversion time) seconds before each excitation and signal readout.

2.4.5 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 for how to edit the sequence. In the simplest case, the user need only define the RF pulse shape to generate a pulse sequence, however the user can define custom gradient waveforms, and new data acquisition windows if desired; multiple windows could also be

Param Name	Param Description	Units	Sequence
`TR`	Repetition time	s	GRE,SE,IR,FID
`CF`	Center frequency	Hz	GRE,SE,IR,FID
`offset`	Center frequency Offset	Hz	GRE,SE,IR,FID
`p90`	Excitation pulse length	s	GRE,SE,IR,FID
`p180`	Refocusing pulse length	s	SE,IR
`TE`	Echo time	s	GRE,SE,IR
`TI`	Inversion time	s	IR
`BW`	Readout bandwidth	Hz	GRE,SE,IR
`readout_length`	FID Readout length	s	FID
`NRO`	Matrix dimension in readout direction	integer	GRE,SE,IR
`NPE`	Number of phase encodes	integer	GRE,SE,IR
`FOVread`	Field of view in readout dimension	mm	GRE,SE,IR
`FOVphase`	Field of view in phase encode dimension	mm	GRE,SE,IR
`readdir`	Readout dimension (x=1,y=2,z=3)	integer	GRE,SE,IR
`phasedir`	Phase encode dimension	integer	GRE,SE,IR
`slicedir`	Slice dimension	integer	GRE,SE,IR
`pre_dur`	Readout prephasor duration	s	GRE,SE,IR
`phase_dur`	Phase encode gradient duration	s	GRE,SE,IR
`rewinder_length`	Slice gradient rewinder length	s	GRE,SE,IR
`slice_thick`	Slice thickness	mm	GRE,SE,IR
`nav`	Number of averages	integer	GRE,SE,IR,FID
`TBW`	Excitation pulse time-bandwidth product	unitless	GRE,SE,IR
`slice_shift`	Slice offset	Hz	GRE,SE,IR
`ex_type`	Excitation pulse shape (Square=0, Sinc=1)	integer	GRE,SE,IR
`ischopped`	RF Chopping (Off=0, On=1)	integer	GRE,SE,IR
`interactive_mode`	Pre-scan/Interactive mode (Off=0, On = 1)	integer	GRE,SE,IR
`param_file`	.pkl file name for automatically loaded scan parameters	string	GRE,SE,IR,FID
`leader_ID`	"Leader" radio serial ID (not interactive)	string	GRE,SE,IR,FID
`follower_ID`	"Follower" radio serial ID (not interactive)	string	GRE,SE,IR,FID

Table 2.1: Parameters that can be adjusted dynamically or in configuration files for each of the pulse sequences.

Function	**Description**	**Argument**
`params.set_{param}(value)`	Change value of parameter `{param}` and dependents	`value`: new value of parameter
`params.param_table()`	Display current sequence parameters	None
`sync()`	Synchronize two radios	None
`read_on()`	Enable readout gradient	None
`slice_on()`	Enable slice select gradient	None
`grads_off()`	Disable all gradients	None
`profile()`	Display 1D object profile with `params.nav` number of averages	None
`show_pulses()`	Plot pulse sequence	None
`params.save_params(file)`	Save scan parameters to pickle file	`file`: file name to save parameters to
`params.import_params(file)`	Import scan parameters from pickle file	`file`: file name to load parameters from
`calib_slice()`	Calibrate slice gradient rewinder	None
`calib_readout()`	Calibrate readout gradient prephasor to center signal	None
`run()`	Run imaging sequence and record data	None
`data.recon()`	Reconstruct image	None
`end()`	Stop sequence and program	None

Table 2.2: User interface functions defined by the three imaging pulse sequences.

defined for multi-echo sequences. The template includes function definitions that provide the same functionality as the full imaging pulse sequences, and the config file is preloaded with general parameter names that were used in the those pulse sequences.

2.5 System calibrations

2.5.1 Transmit/Receive Delay

Since a single SDR typically does not provide enough channels for an MRI scan, gr-MRI imaging sequences presume the use of two radios, one for generating RF waveforms and one for generating gradient waveforms. Due to sequence delays resulting from USB latency which can be on the order of 20 ms, radio synchronization is essential while running these two-radio pulse sequences. To address this, one radio is designated the ‘leader’ and the other the ‘follower’. It is assumed that the leader radio will produce the RF pulses and transmit enable pulse and the follower radio will generate the gradient waveforms. The gr-MRI sequences use a port on each radio to loop back a ten sample block synchronization pulse into dedicated receive channels on the follower SDR three times per TR. Figure 2.6 shows how these signals are processed in the sequence flowgraphs. The received synchronization pulses are each thresholded and combined as the real and imaginary parts of a single complex signal that is stored in a Vector Sink. While running a scan, a function in the imaging sequence script continuously checks the synchronization data from the Vector Sink three times per TR to compare the pulse timings. If a desynchronization is detected, the function pauses the main pulse sequence and sets global leader and follower delays to realign the transmitted pulses. When operating in interactive mode, the user can resynchronize the radios at any time by calling the `sync()` function. Desynchronization during data acquisition may corrupt the received data; the frequency with which desynchronization events occur with our setup was characterized as described below.

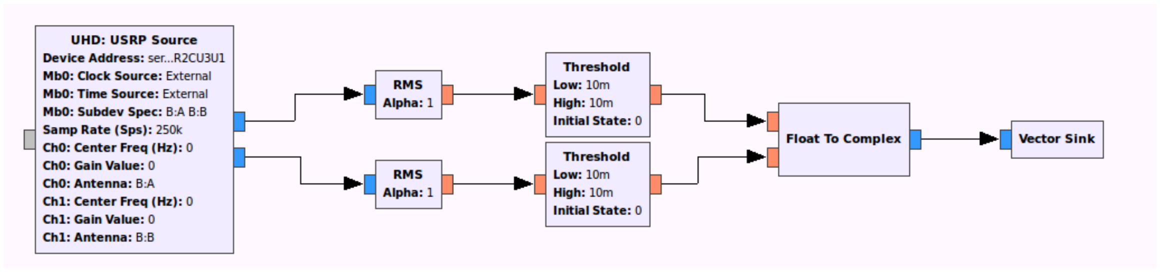


Figure 2.6: Synchronization section of the imaging sequence flowgraphs.

2.5.2 Center Frequency and Transmit Voltage

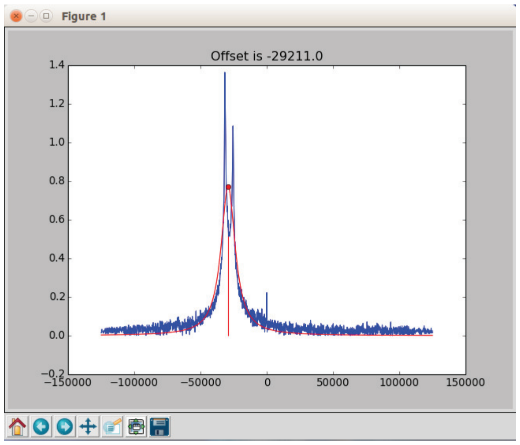
Automatic center frequency and power calibration functions are implemented as part of the `FID.py` pulse sequence, and are listed in Table 2.3. The center frequency offset calibration function is illustrated in Figure 2.7a. The user specifies the desired number of signal averages to use for FID measurement, and the system displays the resultant 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, illustrated in Figure 2.7b, finds the necessary RF pulse amplitude to achieve a 90 degree flip angle for a fixed-duration block 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.

2.5.3 Gradient strength

The gr-MRI package also provides a gradient strength calibration script called `grad_calibration.` which uses a spin echo sequence to measure one-dimensional profiles of an object of known

Function	Description	Argument
`calibrate_offset(nav)`	Find offset from true center frequency to default center frequency	`nav`: number of averages
`calibrate_power(nav)`	Find the optimal excitation pulse amplitude needed to excite a 90 degree flip angle	`nav`: number of averages
`save_calib()`	End calibration flow graph and saves calibration data to `cal.pkl`	None

Table 2.3: Center frequency and power calibration functions implemented in `FID.py`.

(a) 

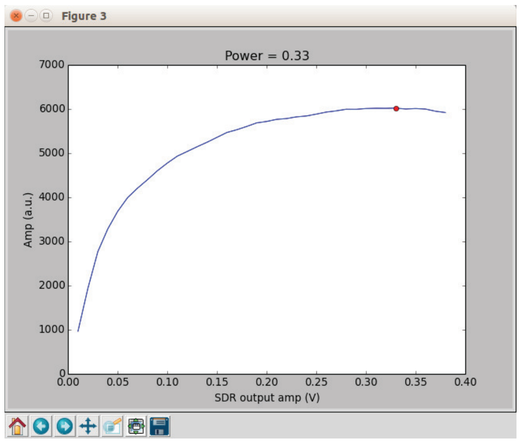
(b) 

Figure 2.7: User-visible plots that appear when running (a) the center frequency calibration function, and (b) the 90 degree pulse power calibration function. Both are defined in `FID.py`. The Lorentzian fit to the spectrum is plotted in (a) in red, and the vertical red line indicates the detected center frequency. I The detected 90-degree pulse power is indicated by the red dot in (b).

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 resultant frequency bandwidth of 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 gradient pulse amplitudes and step sizes. Table 2.4 shows all of the functions available to the user when running the `gradcalib.py` script. This calibration

Function	Description	Inputs
`sync()`	Synchronize the outputs of two radios.	None
`grad_calib(direction)`	Asks for phantom size in specified dimension and calibrates the gradient strength	`direction`: `x`, `y` or `z`
`save_calib()`	End the calibration flow graph and save calibration data to `gcal.pkl`	None

Table 2.4: Gradient calibration functions defined by the gradcalib.py script.

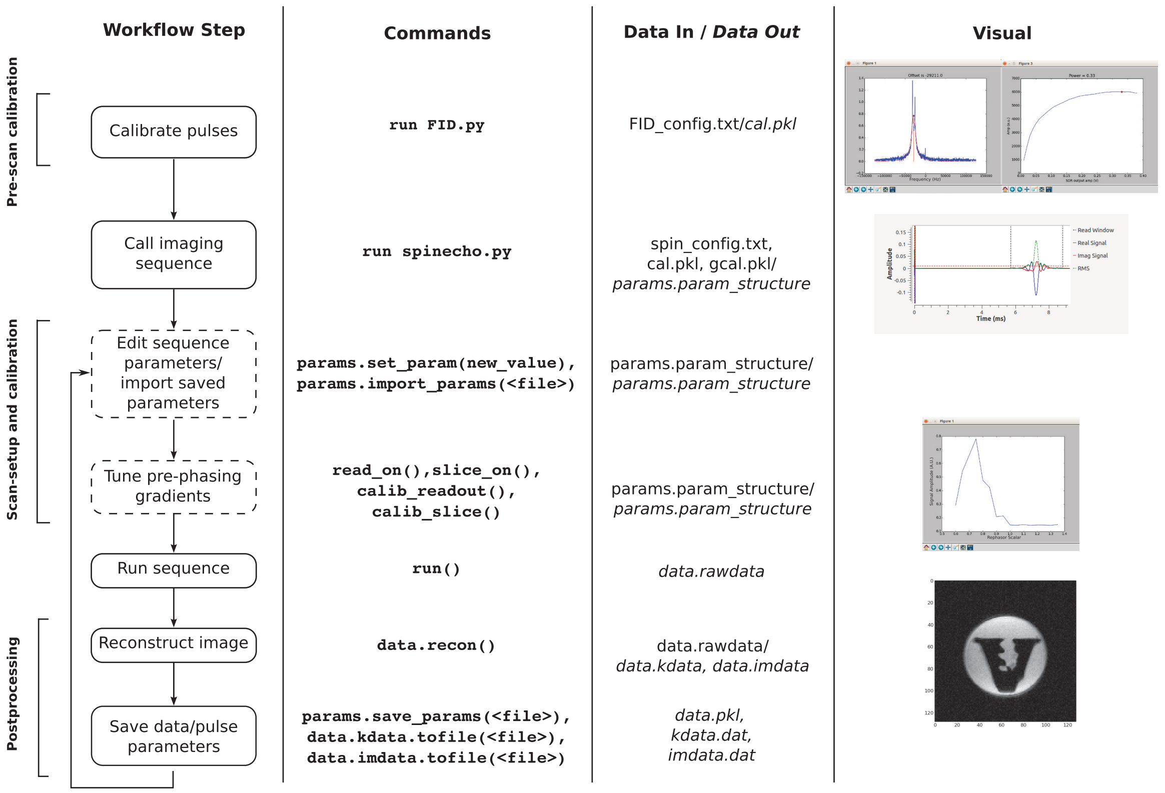
should only need to be done once for a given scanner.

2.6 Data processing and image reconstruction

All data from a sequence 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 an NPE×NRO 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 is present. The formatted k-space data matrix is returned in `data.kdata` and the image is returned in `data.imdata`.

2.7 Workflow summary

Figure 2.8 summarizes the workflow implemented by gr-MRI, for the `spinecho.py` imaging sequence. The diagram shows a workflow step, and the output that the step affects and (where applicable) the input the step receives. A gr-MRI imaging experiment comprises the following steps:



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

Figure 2.8: 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.

1. RF power and center frequency parameters are determined and saved using `FID.py`.

2. The user updates the imaging sequence's config file with the desired pulse sequence parameters for their scan (`spin_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 Figure 2.8 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 is optional but was required periodically on our scanner due to gradient amplifier nonlinearity, and 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.

2.8 Experiments

Experiments were performed on a 0.5 Tesla Oxford Maran tabletop scanner (Resonance Instruments, Witney, U.K.) to verify the imaging sequences and other functions of the gr-MRI software. 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 $1\times1\times1$ cm^3 cube phantom filled with $CuSO_4$-doped water. A solenoid RF coil was used for all scans.

Pulse	**Radio/Channel**	**Input/Output**	**Destination**
RF Excitation	Leader TX_A_A	Output	RF Amplifier
Signal recording	Leader TX_A_B	Output	Leader RX_B_B
Transmit-enable	Leader TX_B_A	Output	RF Amplifier
Leader sync	Leader TX_B_B	Output	Follower RX_B_B
RF Receive	Leader RX_A_A	Input	PC
Gx Gradient	Follower TX_B_B	Output	Gradient Amplifier
Gy Gradient	Follower TX_A_B	Output	Gradient Amplifier
Gz Gradient	Follower TX_A_A	Output	Gradient Amplifier
Follower Sync	Follower TX_B_A	Output	Follower RX_B_A

Table 2.5: Mapping of pulse waveforms to transmit and receive channels on the two SDRs used for the imaging experiments. The leader SDR generated and received all RF signals, and the follower SDR generated all gradient pulses.

2.8.1 SDR Hardware

All imaging scans used two Ettus Research USRP1 SDRs with connected clocks. Although the radios perform separate functions, there is a significant difference between the clock speed of individual radios. This results in a significant time drift between the two radios, which would be impractical to correct by the `sync()` function alone. We therefore connected the clocks following instructions provided on the GNU Radio website. The USRP1 comprises an Altera Cyclone FPGA, a 14 bit (-1 V to 1 V), 128 Ms/sec digital to analog converter, and a 12 bit (-1 V to 1 V), 64 Ms/sec 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 device connects to the PC by USB. In our experiments the leader radio 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 2.5 shows the mapping between imaging sequence pulses and radio channels. The naming convention for the channel used is *TX/RX Daughterboard_Slot A/B_Channel A/B*. 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.

2.8.2 Imaging Scans

Gradient-recalled echo, spin echo, and inversion recovery spin echo imaging experiments were run with the sequence parameters shown in Table 2.6. All scans used RF chopping to move DC artifacts to the edge of the FOV, and images were reconstructed using gr-MRI's 2D inverse FFT reconstruction. All sequences imaged an 11×8 mm^2 3D-printed Vanderbilt University logo-shaped phantom immersed in a 15 mm NMR tube filled with sunflower seed oil. The spin echo acquisitions were repeated 15 times with and without running interactive mode first, to record the number of leader-follower desynchronization events. The inversion recovery scan was repeated eight times with TI's of 10, 25, 50, 100, 150, 200, 300, and 400 ms.

Imaging Parameter	**Gradient Echo**	**Spin Echo**	**Inversion Recovery**
Python script	`gradecho.py`	`spinecho.py`	`invrecov.py`
TE (ms)	1.5	10	10
TR (ms)	1000	1000	750
Flip angle (degrees)	90	90	90
Matrix size	128×128	128×128	128×128
FOV (mm^2)	20×20	20×20	20×20
Slice Thickness (mm)	4	4	5
BW (kHz)	41.7	41.7	41.7
Averages	3	3	3

Table 2.6: Imaging scan parameters.

2.8.3 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

[21] (Figure 2.12, 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.

2.9 Results

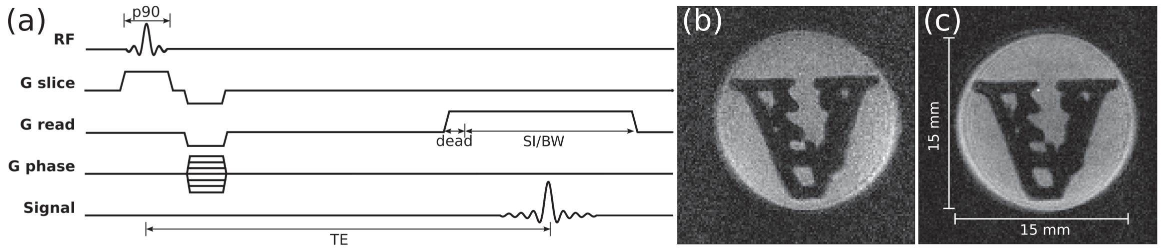


Figure 2.9: (a) Slice-selective gradient-recalled echo pulse sequence generated by `gradecho.py`. (b,c) Gradient-recalled echo images acquired using (b) the Maran spectrometer, and (c) gr-MRI.

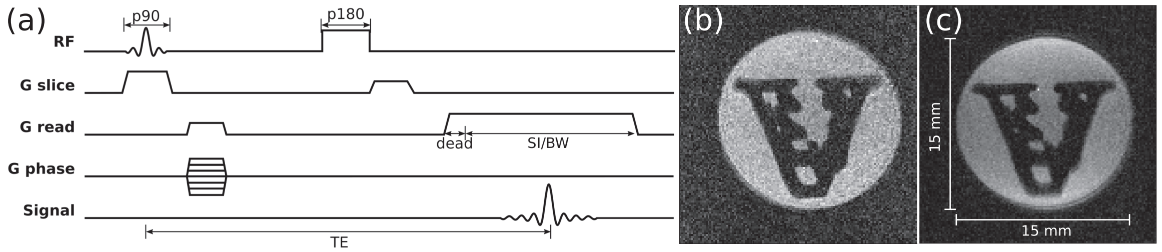


Figure 2.10: (a) Slice-selective spin echo pulse sequence generated by `slicespin.py`. (b,c) Spin echo images acquired using (b) the Maran spectrometer, and (c) gr-MRI.

2.9.1 Imaging

Figure 2.9c shows the image acquired with `gradecho.py`, and Figure 2.9b shows a gradient echo image acquired using the Maran spectrometer with closely-matched parameters. A visual comparison of the images confirms a lack of geometric distortions in the gr-MRI image, other than some blurring due to chemical shift in the frequency-encoded (horizontal) direction. The phantom dimensions measured from the gr-MRI image are 15.15

and 15.28 mm, which are close to the expected 15 mm dimensions. Figure 2.10c shows the image acquired with `spinecho.py`, and Figure 2.10b shows a spin echo image acquired using the Maran spectrometer. A visual comparison of the images again confirms a lack of geometric distortions in the gr-MRI image, other than some blurring due to chemical shift in the frequency-encoded (horizontal) direction. The phantom dimensions measured from the gr-MRI image are 15.25 and 14.85 mm, which again are close to the expected 15 mm dimensions. Figure 2.11b shows inversion recovery images acquired across a range of TIs. The signal is nulled at approximately 100 ms, which is consistent with the expected oil T_1 of approximately 150 ms. Some signal intensity variations are apparent in the short-TI images due to non-uniform inversion resulting from B_1^+ inhomogeneity.

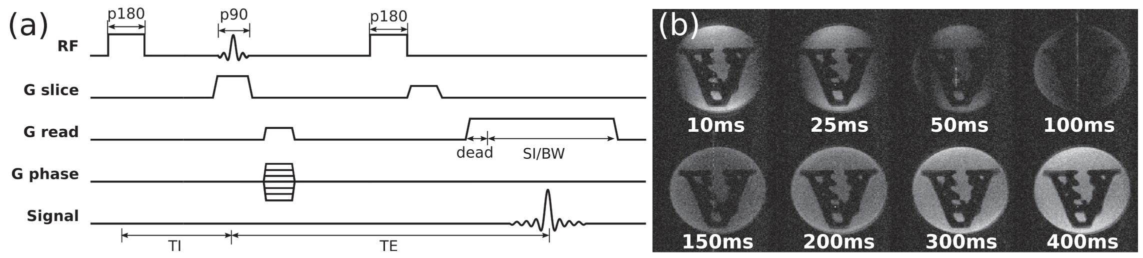


Figure 2.11: (a) Slice-selective inversion recovery spin echo pulse sequence generated by `invrecov.py`. (b) Images acquired using the sequence in (a) across a range of inversion times.

2.9.2 Synchronization Analysis

Table 2.7 shows how often desynchronization events occurred during the `spinecho.py` sequence, with and without running Interactive Mode prior to running the scan. The Table shows that the described 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 seconds, the buffer sizes were more likely to stabilize prior to running the full

	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%)

Table 2.7: Frequency of desynchronization events and recovery from them, with and without entering Interactive Mode before starting an imaging scan.

scan. In that case, only one desynchronization even 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.

2.9.3 Frequency-Swept Pulse Generation

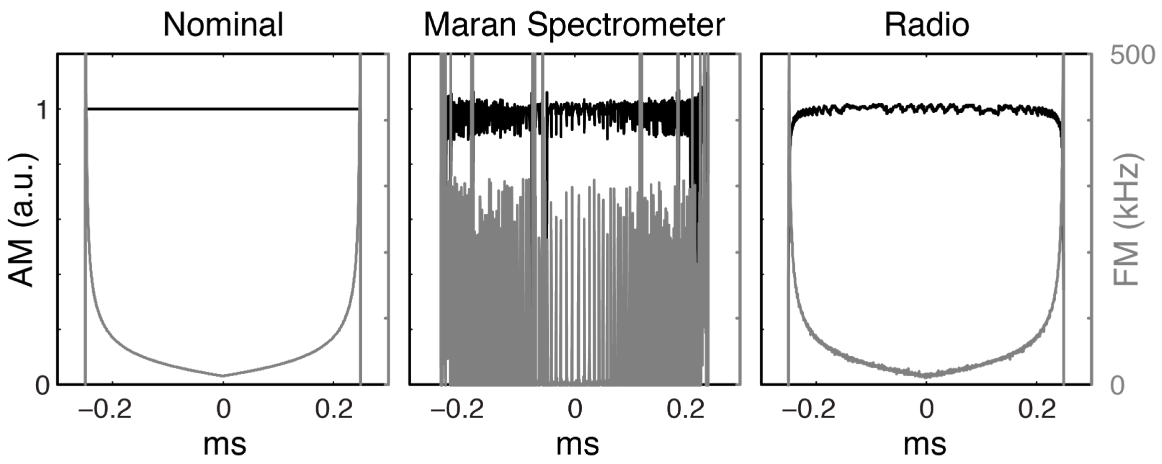


Figure 2.12: Comparison of nominal and measured amplitude modulation (AM) and frequency modulation (FM) waveforms of frequency-swept pulses generated by the Maran spectrometer and the SDR with the gr-MRI Triggered Vector Source block.

Figure 2.12 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 21 kHz. The gr-MRI errors were largest at the ends of the waveforms.

2.10 Discussion

2.10.1 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 a spin echo inversion recovery imaging sequence.

The imaging sequences were validated in 0.5 T phantom imaging experiments, and the gradient-recalled echo and spin echo images were 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 [22] and may be useful for RF encoding using the Bloch-Siegert shift [23, 24].

2.10.2 SDR Considerations

gr-MRI was validated in this work using Ettus Research USRP1's. The USRP1 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 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. For example, some newer SDRs have gigabit ethernet connections to the PC, which reduces latency by a few orders of magnitude 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. In addition, radios with more channels may be able to perform both RF and gradient functions.

Another consideration is Larmor frequency: the USRP1 can directly synthesize signals up to 64 MHz, so to operate at higher frequencies the user would need to use high-frequency transmit/receive daughterboards with onboard modulation/demodulation circuits, or implement mixing externally. Care would need to be taken to make sure that the reference oscillator signals for modulation and demodulation are phase-locked.

2.10.3 Custom Built Hardware

2.10.3.1 RF Coils

As stated in the Chapter 1, the RF probe is responsible for exciting magnetic spins in the sample, which then induce a current in the probe when the spins relax back to the main magnetic field. There are many sophisticated probes, however because our setup is small, we used a simple 10 turn solenoid coil. Solenoid coils are very sensitive, and create relatively homogeneous magnetic fields inside the coil. We wound a 10 turn coil tightly around the 15mm sample tube in order to place the conductor as close as possible to the sample, and secured it with insulating Kapton tape. We used 24 AWG insulated copper wire because it is easy to pack many close turns, and it is capable of carrying a 21MHz signal with low losses. When constructing a coil, one generally wants the smallest amount of real resistance possible. RF signals typically flow mostly on the surface of conductors, but depending on frequency and electrical properties of the conductor they penetrate a certain distance, which is referred to as the skin depth. If the wire gauge is too small, then the coil will add resistance, which reflects RF power and lowers the sensitivity of the coil. The skin depth for a 21MHz signal flowing on a copper wire is roughly 14 μm, which is far smaller than the 0.25 mm radius of 24 AWG wire.

2.10.3.2 Tuning and Matching Networks

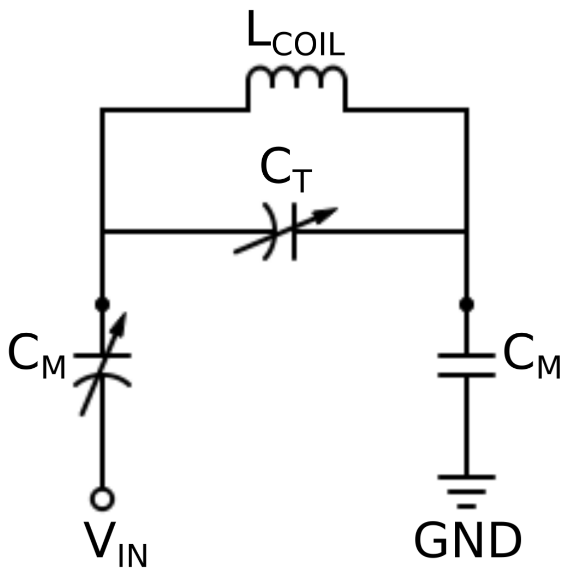


Figure 2.13: General ”pi” configuration for a tuning and matching network. C_M represents matching capacitors, and C
textsubscriptT represents the tuning capacitor.

In order to get the most sensitive coil, so that the transmit voltage remains low, and the greatest possible signal is detected, the coil must be coupled to the spins in the sample.

This means that the coil must be tuned to the Larmour Frequency, and impedance matched to 50 Ohms, which matches the impedance of the rest of the RF system. Fig 2.13 shows the general ”pi” configuration of a tuning and matching network, and is the configuration that was used for the presented experiments. The resonant frequency of an LC circuit can be calculated according to the following equation:

$$f = \frac{1}{(2\pi\sqrt{LC})}$$

In practice, when tuning and matching the coil, an inductance meter was used measure the inductance of the coil, and the necessary tuning capacitance was derived according the the previous equation, and assuming a resonant frequency of 21.3 MHz. The circuit was then connected to a network analyzer, and matching capacitors were chosen to minimize the reflected RF power at the center frequency. Variable capacitors were used in parallel with the tuning capacitor and one of the matching capacitors to allow optimization of the circuit when it was loaded with a sample, and placed into the magnet. The circuit used for the following experiments was tuned to 21.27 MHz and had a reflected power, or S11, of roughly -40dB at that frequency which is suitable for imaging.

2.10.3.3 Coil Housing

A custom coil housing was designed to accommodate these experiments as well as to enable future experiments. The housing, and experimental setup for the described imaging sessions is shown in Fig 2.14. The coil is shown on the left side of the box (a) and connects to the tuning and matching circuits (c) via a solderless junction (b). This housing allows the connection of two independent coils The two tuning and matching circuits shown in Fig 2.14c are constructed in the pi configuration described above, and contain variable capacitors that are accessed from outside of the housing. This allows us to tune and match the circuit dynamically while the coil is inside the magnet and while the circuit is connected to the receive chain. The tuning and matching circuits also contain active detuning circuits,

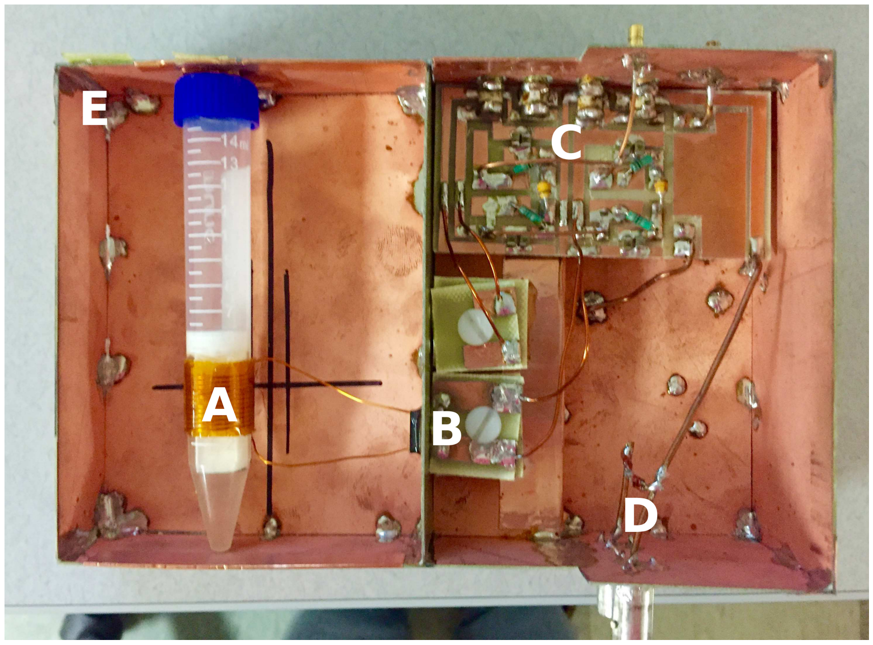


Figure 2.14: Custom built coil housing showing (a) the coil and sample, (b) the solderless junction that connects the coil to (c) the tuning and matching circuits. (d) A pair of crossed diodes connects the transmit port to the circuit during RF transmission, and isolates the transmit chain during receive. (e) The copper housing is grounded, which electrically shields the coil.

which will become important when using multiple closely tuned coils for transmit and receive. The coil shown here is used for transmit and receive, so the transmit and receive inputs and outputs are connected by a pair of crossed diodes, shown in Fig 2.14d. The crossed diodes allow the high power RF excitation pulses through during transmit, but isolate the transmit chain during receive. The entire housing box is made out of copper clad FR4 board. The copper housing serves as a common ground for the elements contained in the box, and electrically shields the coil from outside RF noise.

2.10.3.4 Unblanking Circuit

The dynamic range of the SDR output is -1V to 1V, and the required voltage to trip the TTL switch on the RF amplifier is 3.3V, so we could not directly use the output from the SDR to control the amplifier. To work around this issue, we built a small transistor switch

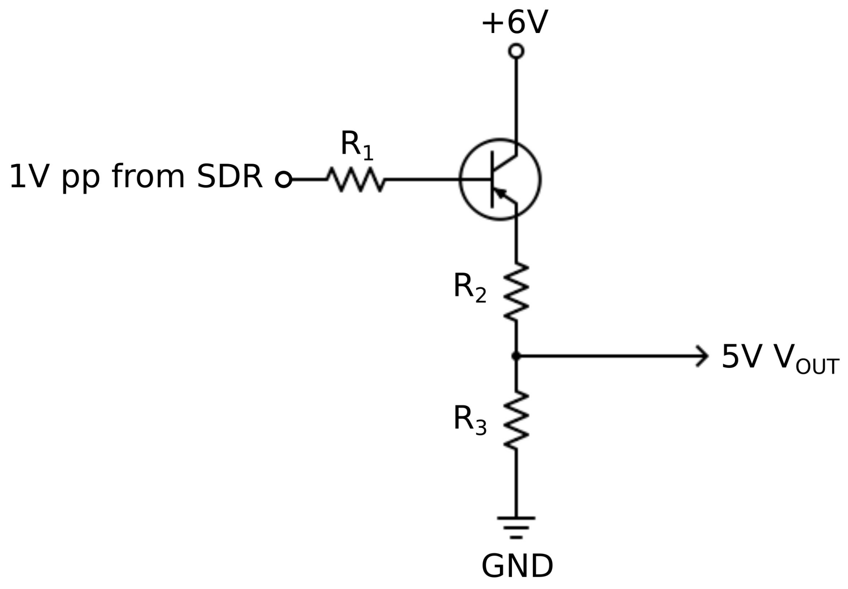


Figure 2.15: Transistor circuit used to convert the 1V square output from the SDR to a 5V TTL pulse used to enable the RF Amplifier during RF Transmission. Resistor values were chosen to limit the current drawn by the circuit, and to convert the 6V input from the SDR power jack to a 5V output.

that used the 1V pulse created by the SDR to drive a MOSFET, which allowed the 6V input to be connected to the output circuit. The circuit diagram is shown in Fig 2.15. The output circuit simply consisted of a voltage divider, shown as R2 and R3, which converts the input 6V signal to 5V and limits the current to roughly 10mA. The resistor labeled R1 was simply used to limit the current fed to the gating pin of the MOSFET.

Chapter 3

Conclusions and Future Work

gr-MRI Enables commercially-available software-defined radios to be used as low-cost custom MRI spectrometers, with fidelity that is comparable to or better than 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.

In the near future, we can use this SDR platform to implement frequency swept Bloch-Siegert pulses for RF gradient encoding, or to implement other advanced pulse sequences. This could include expanding the current library of pulse sequences to support a similar simple calibration of RF and gradient waveforms for non-cartesian trajectories such as radial or spiral. As it exists now, a user can create these pulses outside of the program, and load them before running a scan, but it would be nice to allow for an integrated system in which the pulses could be created when running the scan. Much in the way that GNU Radio is constructed, we would like to enable the integration of other user created functions in a sort of plug-in format, which could allow for a broadening of the system as a whole and enable novel pulse development and rapid testing of those pulses.

Future work would also include adapting this system to operate at higher fields, and to interface with clinical scanners. The USRP1 radio, or any other radio that runs the software is small, and would give researchers the ability to carry a custom spectrometer to any scanner on which they are working, so it would be beneficial for the system to operate at clinically relevant frequencies.

BIBLIOGRAPHY

[1] Howard D. W. Hill. *Spectrometers: A General Overview*, pages 4505–4518. Wiley, New York, 1996.

[2] Michael Garwood Alberto Tannus. Adiabatic pulses. *NMR in Biomedicine*, 10:423–434, 1997.

[3] S Jie, X Qin, L Ying, and L Gengying. Home-built magnetic resonance imaging system (0.3 T) with a complete digital spectrometer. *Rev Sci Instrum*, 76:105101, 2005.

[4] S M Wright, D G Brown, J R Porter, D C Spence, E Esparza, D C Cole, and F Russel Huson. A desktop magnetic resonance imaging system. *MAGMA*, 13:177–185, 2002.

[5] W K Peng, L Chen, and J Han. Development of miniaturized, portable magnetic resonance relaxometry system for point-of-care medical diagnosis. *Rev Sci Instrum*, 83:095115, 2012.

[6] S R Parnell, E B Woolley, S Boag, and C D Frost. Digital pulsed NMR spectrometer for nuclear spin-polarized ^{3}He and other hyperpolarized gases. *Meas Sci Technol*, 19:045601, 2008.

[7] J Bodurka, P J Ledden, P van Gelderen, R Chu, J A de Zwart, D Morris, and J H Duyn. Scalable multichannel MRI data acquisition system. *Magn Reson Med*, 51(1):165–171, 2004.

[8] W Tang, H Sun, and W Wang. A digital receiver module with direct data acquisition for magnetic resonance imaigng systems. *Rev Sci Instrum*, page 104701, 2012.

[9] P P Stang, S M Conolly, J M Santos, J M Pauly, and G C Scott. Medusa: A scalable MR console using USB. *IEEE Trans Med Imag*, 31(2):370–379, 2012.

[10] P Stang, A B Kerr, J M Pauly, and 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*, page 145, 2008.

[11] W A Grissom, D Xu, A B Kerr, J A Fessler, and D C Noll. Fast large-tip-angle multidimensional and parallel RF pulse design in MRI. *IEEE Trans Med Imaging*, 28(10):1548–1559, Oct. 2009.

[12] C Barmet, N De Zanche, B J Wilm, and K P Pruessmann. A transmit/receive system for magnetic field monitoring of in vivo MRI. *Magn Reson Med*, 62:269–276, 2009.

[13] P T Sipilä, S Greding, G Wachutka, and F Wiesinger. ^{2}H Transmit–receive NMR probes for magnetic field monitoring in MRI. *Magn Reson Med*, 65:1498–1506, 2011.

[14] P T Sipilä, R F Schulte, G Wachutka, and F Wiesinger. Digital multiband receiver for magnetic resonance. *Conc Magn Reson B: Magn Reson Engin*, 35B(4):210–220, 2009.

[15] L Gengying, J Yu, Y Xiaolong, and J Yun. Digital nuclear magnetic resonance spectrometer. *Rev Sci Instrum*, 72:4460, 2001.

[16] K Takeda. A highly integrated FPGA-based nuclear magnetic resonance spectrometer. *Rev Sci Instrum*, 78:033103, 2007.

[17] K Takeda. OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer. *J Magn Reson*, 192:218–229, 2008.

[18] W Tang and W Wang. A single-board NMR spectrometer based on a software defined radio architecture. *Meas Sci Technol*, 22:015902, 2011.

[19] A quick guide to hardware and GNU Radio.

[20] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[21] M Jankiewicz, J C Gore, and W A Grissom. Improved encoding pulses for Bloch-Siegert B_1^+ mapping. *J Magn Reson*, 226:79–87, 2013.

[22] M Garwood and L DelaBarre. The return of the frequency sweep: designing adiabatic pulses for contemporary NMR. *J Magn Reson*, 153(2):155–177, Dec 2001.

[23] R Kartäusch, T Driessle, T Kampf, T C Basse-Lüsebrink, U C Hoelscher, P M Jakob, F Fidler, and X Helluy. Spatial phase encoding exploting the Bloch-Siegert shift effect. *Magn Reson Mater Phy*, 2013.

[24] Z Cao, E Y Chekmenev, and W A Grissom. Frequency encoding by Bloch-Siegert shift. In *Proceedings 22nd Scientific Meeting, International Society for Magnetic Resonance in Medicine, Milan*, page 4220, 2014.