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.

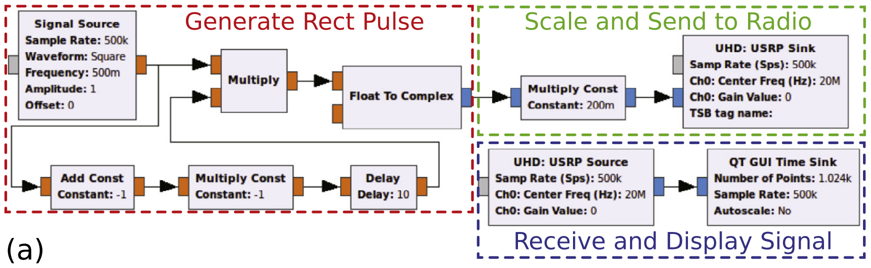
2. Software architecture and implementation

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

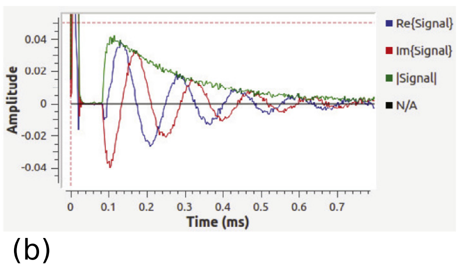


Fig. 1. (a) Basic single-pulse GNU Radio flowgraph without gr-MRI elements. The outlined boxes contain the rectangular excitation pulse generation blocks (red), blocks that scale the pulse to a desired amplitude and send it to the SDR (green), and the receive chain (blue), comprising a USRP Source block 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.)