MaRCoS, an open-source electronic control system for low-field MRI

Negnevitsky, V.; Vives-Gilabert, Y.; Algarín, J.M.; Craven-Brightman, L.; Pellicer-Guridi, R.; O'Reilly, T.; ... ; Menküc, B.

Journal of Magnetic Resonance 350 (2023) 107424

Contents lists available at ScienceDirect

Journal of Magnetic Resonance

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

MaRCoS, an open-source electronic control system for low-field MRI

Vlad Negnevitsky [a], Yolanda Vives-Gilabert [b], José M. Algarín [c], Lincoln Craven-Brightman [d], Rubén Pellicer-Guridi [e], Thomas O'Reilly [f], Jason P. Stockmann [d], Andrew Webb [f], Joseba Alonso [c], Benjamin Menküc [g,*]

[a] Oxford Ionics Ltd, Oxford OX5 1PF, UK
[b] Intelligent Data Analysis Laboratory, Department of Electronic Engineering, Universitat de València, Valencia, Spain
[c] MRILab, Institute for Molecular Imaging and Instrumentation (i3M), Spanish National Research Council (CSIC) and Universitat Politècnica de València (UPV), Valencia, Spain
[d] Massachusetts General Hospital, A. A. Martinos Center for Biomedical Imaging, Charlestown, MA, USA
[e] Asociación de Investigación MPC, Manuel de Lardizábal 5, Donostia-San Sebastián 20018, Spain
[f] Department of Radiology, Leiden University Medical Center, Albinusdreef 2, Leiden 2333, Netherlands
[g] University of Applied Sciences and Arts Dortmund, Sonnenstr. 96, Dortmund 44139, Germany

ARTICLE INFO

Article history:
Received 20 November 2022
Revised 20 February 2023
Accepted 13 March 2023
Available online 21 March 2023

Keywords:
MRI
Low-field MRI
MRI control system
MRI sequence programming

ABSTRACT

Every magnetic resonance imaging (MRI) device requires an electronic control system that handles pulse sequences and signal detection and processing. Here we provide details on the architecture and performance of MaRCoS, a MAgnetic Resonance COntrol System developed by an open international community of low-field MRI researchers. MaRCoS is inexpensive and can handle cycle-accurate sequences without hard length limitations, rapid bursts of events, and arbitrary waveforms. It has also been readily adapted to meet the requirements of the various academic and private institutions participating in its development. We describe the MaRCoS hardware, firmware and software that enable all of the above, including a Python-based graphical user interface for pulse sequence implementation, data processing and image reconstruction.

1. Introduction

The electronic control system or "console" is an indispensable component of any MRI scanner. It handles the sequences of electromagnetic pulses (both radiofrequency and gradient) in a time-synchronous fashion, as well as the acquisition and digitization of MR signals, so they can be processed for image reconstruction. At a higher level, the control system also serves as an interface between the user and the scanner itself.

MRI makes use of pulses in different regimes of the electromagnetic spectrum, radiofrequency in the 1–100 MHz range, and gradient in the low kHz range, which are combined to fulfill the requirements for imaging: the creation of phase coherent precessing magnetization using a resonant transmit coil, and spatial encoding, with the encoded signals being detected via Faraday induction by a resonant receiver coil [1]. These tasks necessitate a high degree of phase coherence between the various pulses, placing strong constraints on the time control. Modern field-programmable gate-array (FPGA) modules are perfectly suited for fast, time-synchronous tasks and are thus often at the core of MRI consoles.

The consoles provided by the major scanner manufacturers tend to be tailored to their specific setups [2,3], but there exist also more generic concepts that are somewhat hardware-independent and could be used in a wide range of scanners. Although a number of relatively low cost consoles have been developed for MRI, e.g. Pure Devices GmbH (Rimpar, Germany), Magritek Ltd (Wellington, New Zealand), RS2D (Mundolsheim, France), MR Solutions (Guildford, UK) or Niumag Corporation (Suzhou, China), these inevitably comprise much proprietary hardware and software, and so do not lend themselves to open source development. In developing an open solution, it is also important to note that many low-field systems actually require quite sophisticated operation to overcome the specific challenges of low-field MRI [4–17].

Among the projects opened to the community [2,18–26], the Open-source Console for Real-time Acquistion (OCRA [21]) is notable for its flexibility, inexpensive off-the-shelf components, community focus, and real-time capabilities. The MAgnetic Resonance COntrol System (MaRCoS [3]) uses the same versatile hardware, however its software, firmware and FPGA firmware have been replaced to go beyond the limitations of OCRA.

* Corresponding author.
E-mail address: benjamin.menkuec@fh-dortmund.de (Benjamin Menküc).

https://doi.org/10.1016/j.jmr.2023.107424
1090-7807/

In this article we discuss the MaRCoS rationale and its main performance advances, then present the MaRCoS hardware, firmware and desktop software. We then discuss the directions in which MaRCoS development is proceeding, and provide some information for readers interested in trying MaRCoS out.

2. System goals and overview

MaRCoS was started by a partnership of low-field MRI academic groups [3] aiming to replace a range of proprietary consoles with a unified, inexpensive yet high-performance platform suitable for some unconventional experiments.

The main project goals were to use open-source or off-the-shelf hardware, be easily programmable, have no hard limitations on the sequence length or complexity, and to be scalable to multiple channels in the future. Additional goals were to allow the transmit and receive modulation frequencies to be independently varied, and to support sampling rates up to several megahertz for the purpose of investigating highly oversampled data acquisition [27]. Initially the existing OCRA platform was used and extended[1], however its limitations on sequence length, complexity and timing precision, as well as the low-level 'assembly'-style sequence programming, led to a complete rewrite of the server, FPGA firmware, and client software to create the MaRCoS system.

The core of MaRCoS is the Red Pitaya SDRLab 122–16 [28], a commercial board with two analog inputs for digitising received data and two analog outputs for generating the RF transmit waveforms, with a bandwidth of around 50 MHz making it suitable for proton MRI at field strengths of up to 1.17 Tesla. Two receive/transmit channels are run in parallel, with frequency up/down-conversion and the bulk of the filtering handled digitally. Three digital outputs can be used for controlling RF switches or other peripherals, and one input is used for triggering acquisitions using external devices[2]. The SDRLab also controls either a GPA-FHDO [29] or an OCRA1 [30] four-channel gradient board. The MaRCoS hardware is controlled from a PC running either Linux or Windows (using WSL) over gigabit Ethernet. Fig. 1 shows the overall system architecture.

On the SDRLab, sequence and acquisition data are streamed to and from an FPGA, allowing for much longer sequences than designs which first load a sequence into FPGA memory and then execute it; this is often a limiting factor when an FPGA having only several megabytes of RAM is used[3]. MaRCoS does not use raster clocks[4] or impose any timing constraints on events beyond that of the hardware clock; the architecture permits any change in an internal or external parameter to occur at any time with cycle-accuracy. This includes changing the RF modulator/demodulator frequencies, RF envelope amplitudes and phases, gradient channel voltages, receiver sampling rates and acquisition timing, and digital outputs. Sequences are programmed at a low level using simple arrays of times and values for each parameter, which are converted to hardware instructions by the marcos_client Python library. There are several intermediate text-based interfaces to suit different styles of programming, as well as a GUI for running a range of calibrations and predefined acquisition routines.

To our knowledge MaRCoS is the first inexpensive system that is capable of handling unlimited, cycle-accurate sequences, handling rapid bursts of events, creating arbitrary waveforms, treating RF and gradients in a uniform way, and independently controlling the frequency, phase and amplitude of the different TX and RX channels in each TR. It also has several less common features, needed for specialised experiments that were in planning when MaRCoS was begun: using multiple acquisition dwell times within a single sequence, handling dwell times as short as 50 ns to allow extreme oversampling of the receiver signal, and allowing the user to freely program digital inputs/outputs in the sequence. The hardware, firmware and software are described in more detail below.

3. MaRCoS hardware

3.1. SDRLab-based console

The core of the SDRLab is a Xilinx Zynq-7020 system-on-chip (SoC), integrating two embedded ARM processors and a field-programmable gate array (FPGA). MaRCoS runs Linux[5] and a custom C++ server on the processors, which controls the sequencing and digital signal processing (DSP) on the FPGA as well as digital and analog I/O. The Linux OS handles networking and provides various standard services such as SSH.

The FPGA part of the Zynq communicates with a dual-channel 16-bit analog–digital converter (ADC) and a dual-channel 14-bit digital-analog converter (DAC), which together provide two independent channels for MRI[6]. The system is clocked from a 122.88 MHz crystal oscillator[7], providing an analog baseband bandwidth of $\sim$ 50 MHz. The Zynq also controls multiple digital outputs and inputs, which communicate with the gradient board and external devices such as TR switches.

3.2. GPA-FHDO and OCRA1 gradient boards

MaRCoS currently supports two gradient DAC boards, the GPA-FHDO and the OCRA1, shown in Fig. 2. The GPA-FHDO is a gradient DAC that also includes a linear power stage that can deliver $\pm$10 A on four channels. The 4-channel 16-bit DAC[8] supports serial-peripheral interface (SPI) clocks up to 50 MHz. Due to limitations of the SPI isolator[9] on the GPA-FHDO, the maximum SPI clock is around 40 MHz, which results in an effective sample rate of around 100 kSPS if the four channels are updated in parallel[10]. The GPA-FHDO also monitors the current of each channel, which can be used to automatically calibrate non-linearities and offsets in the analog circuitry. The ADC[11] has a maximum sample rate of 500 kSPS for each channel. This is limited by the SPI isolator in the same way as the DAC sample rate, however since the ADC is mainly used for calibrating the system during the prescan, the sample rate of the ADC is not a critical factor. An adapter PCB for the SDRLab is available to easily connect to the GPA-FHDO. It includes a buffer for the TX-gate signal and a connector for an optional fan. A plugin module is also available for the GPA-FHDO to generate a $\pm$12 V bipolar output, so that external analog gradient amplifiers can be used. The GPA-FHDO and adapter PCB designs are fully open-source [29], and a set of PCBs can be manufactured and assembled for below $400.

[1] The authors rewrote the OCRA server and client to use a `msgpack`-based protocol, support the open-source PulSeq hardware-independent sequencing language, and support the newly-developed GPA-FHDO gradient board.

[2] It will also be used for synchronizing multiple devices during multi-channel operation, which is currently under development.

[3] Currently the sequences are stored in $\sim$ 500 MB DDR memory, however an update will allow sequences of arbitrary length to be streamed from a PC; Section 4.1 contains more detail.

[4] A raster clock system constrains output changes to only occur on the edges of a slow 'raster' clock, limiting the flexibility of event timing.

[5] It works with either the Debian-based image provided with the SDRLab, or a minimal image based on Yocto Linux v2.7.

[6] ADC: Analog Devices LTC2185. DAC: Analog Devices AD9767.

[7] Abracon ABLNO-122.880 MHz.

[8] Texas Instruments DAC80504.

[9] Analog Devices ADUM4150.

[10] The SPI bandwidth can be used to update the channels unevenly, e.g. one channel at 400 kSPS while the others are idle.

[11] Texas Instruments ADS8684.

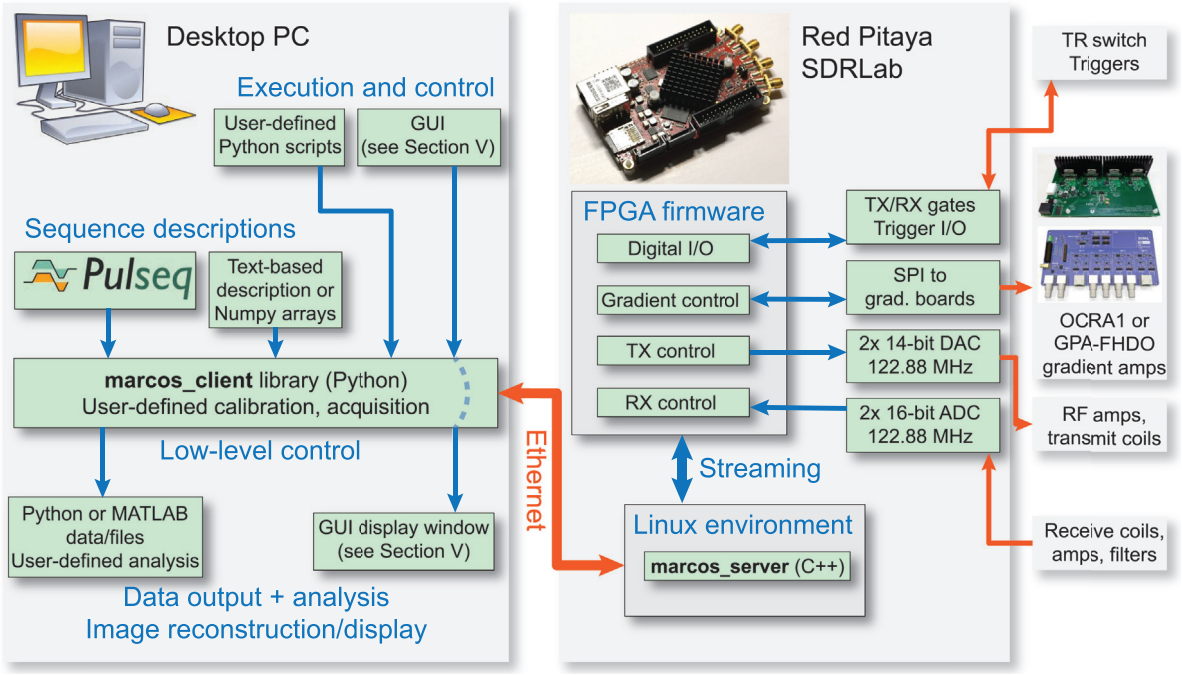

Fig. 1. MaRCoS system architecture showing the main components in the desktop software stack and the embedded hardware and software on the SDRLab. There are various ways to program a sequence, including a graphical user interface (GUI), direct Numpy arrays and PulSeq, which all control the marcos_client library. This communicates with the SDRLab, executing the sequence and returning the acquired data.

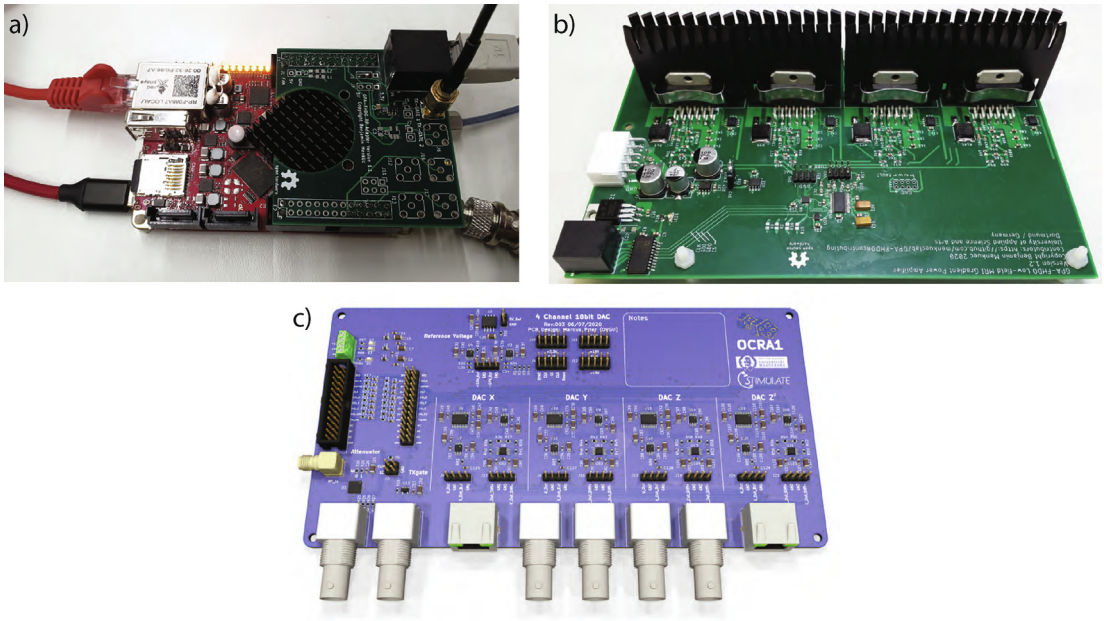

Fig. 2. MaRCoS hardware components. **a)** Red Pitaya SDRLab with a GPA-FHDO adapter PCB, **b)** GPA-FHDO board, and **c)** rendering of the OCRA1 board.

The OCRA1 is a gradient board with ±10 V single-ended and ±20 V differential voltage outputs on four channels. The manufacturing files and schematics are available at [30]. It uses four 18-bit DACs[12]. The OCRA1 DACs can be run at their maximum SPI clock frequency of 35 MHz, which results in an effective sample rate of ∼ 400 kSPS. The effective sample rate for the OCRA1 is higher than the GPA-FHDO, mainly because the OCRA1 uses four separate SPI interfaces for its DACs rather than a single four-channel DAC. Unlike the GPA-FHDO, the OCRA1 does not have a power stage. Hence, it requires external analog gradient power amplifiers, which users can choose according to their needs. Another difference between the two boards is that the OCRA1 has an on-board RF attenuator that could be used for flip-angle calibration instead of varying the RF amplitude directly from the SDRLab (MaRCoS does not currently support controlling the attenuator). The OCRA1 also has an onboard TX gate buffer similar to the SDRLab adapter PCB for the GPA-FHDO.

[12] Analog Devices AD5781B.

4. MaRCoS server and FPGA firmware

The MaRCoS architecture was designed from the ground up to achieve the goals in Section 2. The FPGA firmware receives instructions from the MaRCoS server and translates them into hardware commands, as well as down-converting, filtering and storing the receive data. The server in turn receives instruction binaries and control commands from the client PC, and sends back acquired data. Fig. 3 shows the tightly coupled server-firmware architecture, which relies on the server streaming data to and from the firmware in real time, and the firmware converting between the asynchronous data streams and synchronous (precisely timed) input and output data using several layers of buffering.

4.1. MaRCoS embedded server

The MaRCoS embedded server[13] is tightly coupled to the FPGA firmware and has five major roles: receiving sequences from the client PC, controlling the system at a high level (e.g. running or stopping sequences and checking the firmware status), streaming a steady supply of instructions to the FPGA, regularly reading the FPGA receive buffers and streaming the raw input data into large arrays in the system memory, and sending these arrays back to the client PC once the sequence is finished. When a sequence is executing, the server tries to balance keeping the FPGA instruction buffer near-full against keeping the receive buffers near-empty, since failing to supply enough instructions would result in an unpredictable timing delay, and failing to read the received data would result in a buffer overflow and data loss. The server continuously tracks the free space in the various buffers and allocates its time based on what is most urgent, and the firmware records 'buffer-full' and 'buffer-empty' events so that the user is always alerted if (and at what point) their sequence overloaded the server. The server can sustain a steady rate of ~ 1.5 million combined reads plus writes per second, which is enough for arbitrary-waveform RF envelopes, gradient outputs and acquisition rates with bandwidths of several hundred kHz. Bursts of $\sim 20\,000$ output instructions or acquisition samples are possible with several-MHz bandwidths owing to the large buffers[14]

The server currently receives a sequence from the client PC, executes it, and returns the data in three separate stages. Sequences can be hundreds of megabytes in size[15], however large amounts of sequence and acquisition data can take tens of seconds to transfer to and from the SDRLab. This could be improved by transferring and executing the sequences in stages, or streaming them from the PC in a similar way to the streaming between the server and FPGA firmware. The Python-based client software is currently the performance bottleneck, however, therefore optimizing the data transfer is not yet critical. The client is discussed in Section 5.1.

4.2. FPGA firmware structure

The FPGA firmware executes the transmit, gradient, digital I/O and control instructions sent from the embedded server while simultaneously acquiring ADC data, filtering it and storing it in the receive buffers. The central firmware module is an instruction decoder and sequencer, whose role is to interpret instructions from the circular instruction buffer which is kept filled by the server. The decoder controls 24 output/timing first-in first-out (FIFO) buffers, which can store 8 words each. Most instructions tell a particular FIFO to output a data word at a precise time, and these time-synchronous data words control the synchronous firmware modules as shown in Fig. 3. These include the various transmit/receive cores, the gradient DAC controllers and digital I/O, as well as cores whose properties are typically controlled asynchronously in other MRI consoles, such as the numerically-controlled oscillator frequency and phase and the cascaded integrator-comb (CIC) filter decimation[16]. The uniformity of the FIFOs allows each instruction simply to specify a FIFO index, time delay and output word; the complexity of choosing the delays and values is handled entirely in the client software on the PC. This neatly decouples complex sequence timing and design from low-level hardware or firmware details, since virtually any pattern of FIFO outputs could be executed with no low-level changes.

4.3. Receive and transmit RF signal processing

The FPGA firmware performs both the down-conversion and low-pass filtering of the received signal, and the up-conversion of the transmitted signal envelope. These are traditionally done with external analog components, however can be performed digitally in MaRCoS because the Larmor frequency of LF-MRI systems falls below the 61 MHz Nyquist frequency of the SDRLab. On the receive path, the analog signal is digitized by a 16-bit ADC at 122.88 MHz. This results in a data rate of 2 Gbps per channel, which is challenging to transfer from the FPGA to the client PC using inexpensive hardware.

The data rate can be greatly reduced by digitally down-converting, lowpass-filtering and downsampling. The down-conversion is carried out by multiplying the signal from the ADC by the I and Q outputs of a quadrature oscillator operating at the Larmor frequency. The numerically-controlled oscillator (NCO) uses a direct digital synthesizer (DDS), whose frequency and phase can be altered during runtime through the output FIFOs. The firmware has three independent NCOs, and the NCO used by each of the four TX and RX complex multipliers can be independently selected and switched mid-sequence.

In order to avoid higher-frequency noise aliasing into the passband spectral region, low-pass filtering is needed before downsampling. An efficient way to combine decimation and filtering is to use a CIC filter, which consists of a decimator and a moving average low-pass filter. The decimation rate of the CIC is configured using output FIFOs, in a similar way to the NCO properties.

MaRCoS can use either proprietary (Xilinx) or open-source cores for the complex multiplier, NCO and CIC. The three latter cores were written for MaRCoS, and to the authors' knowledge are among the most feature-complete open-source cores that implement this functionality. They are available at [31–33].

To improve the aliasing performance and stopband suppression, multiple CIC filter stages can be cascaded. The MaRCoS firmware uses four six-stage CIC filters in total, one each for the i and q outputs of each receive channel's complex multiplier. Their outputs are rounded to 32 bits before entering the receiver buffers, and the decimation factor is adjustable from 4 to 4095. As the decimation factor of a CIC is altered, the output signal is scaled, and this effect must be compensated to avoid losing dynamic range. The proprietary CIC cores compensate the scaling internally and keep it in a range between 0.5 and 1. The scaling of the open-source cores is configurable; the compiler precomputes values that keep

[13] The server is a Linux user-space executable written in C++ running on the SDRLab, and is started/stopped, recompiled, etc. via SSH.

[14] Once *sustained* rates in the MHz range are required, however, MaRCoS must be modified to support direct-memory access (DMA) in the firmware and server and/or extended with firmware-level data compression. This is well beyond current experimental demands.

[15] The total size of the MR sequence plus acquired data is limited by the memory that the server can reserve within Linux, which is around 500 MB.

[16] Controlling these with the same timing precision as the RF envelopes and gradients allows in-sequence alterations to the oscillator frequency, acquistion rate, etc. and guarantees that the behaviour of a sequence is reproducible down to the clock cycle.

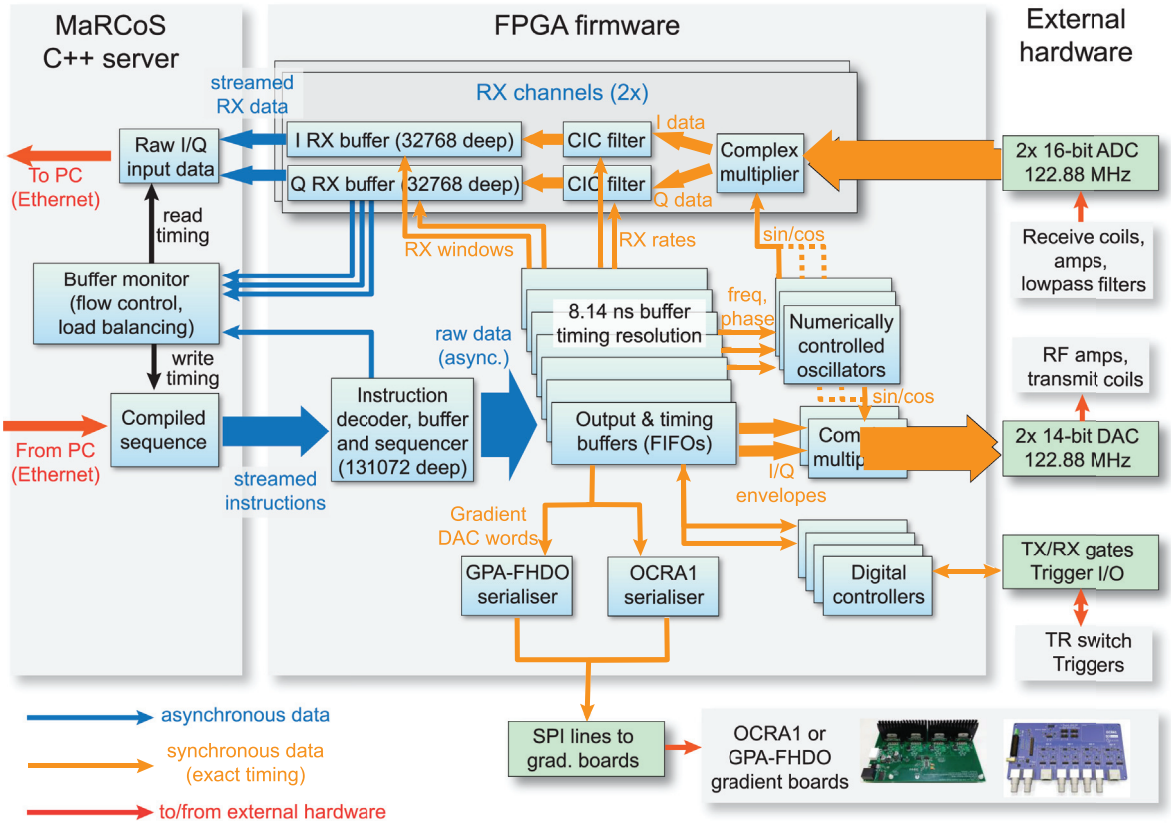

Fig. 3. MaRCoS server and FPGA firmware architecture on the SDRLab. The server receives a sequence from the client PC via Ethernet and streams it to the FPGA firmware, where it is translated into time-synchronous hardware operations including RF and gradient outputs. The firmware receives data from the ADCs, demodulates and filters it, and saves it into RX buffers, from which it is read by the server and sent to the PC.

the scaling in the 0.5 – 1 range, which are written in during the pulse sequence before each change in the decimation factor.

The aliased noise of a CIC increases towards the edge of the passband. The frequency response is also not perfectly flat across the passband, thus it is not ideal to use the entire bandwidth of the CIC filter. Instead, usually a second decimation and filtering stage follows the CIC with a small decimation factor, usually 2–4. In the case of MaRCoS, the data rate is already greatly reduced after the CIC, and the second decimation stage is not implemented inside the FPGA; users are free to implement their own in software. It is good practice to use a finite-impulse-response (FIR) filter as the second decimation stage, which is also tuned to compensate the falling frequency response of the CIC passband. These filters are called CIC compensation filters.

The transmit chain is simpler than the receive chain, because no filtering is needed. The complex baseband envelope data, supplied by several output FIFOs, is up-converted by multiplying the signal by the I output of a quadrature oscillator, in a reverse process to the down-conversion. The real part of the up-converted signal is then sent to the DAC. A low-pass filter is usually applied in the analog RF chain after the RF power amplifier, in order to remove higher harmonics.

4.4. Gradient control and digital I/O

To control the gradient boards, data from the FIFOs is serialized by dedicated modules for the GPA-FHDO and the OCRA1. The OCRA1 module sends data over four parallel SPI links sharing a clock and enable line, to control the four DACs on the OCRA1 board. The GPA-FHDO module has a single, bidirectional SPI link, which either controls the four-channel DAC or reads the four-channel current-sensing ADC. Both modules have adjustable SPI clock frequencies, to optimize the gradient update rate based on the experimental setup and cabling. They also report errors to the sequencer if they cannot serialize the data they are being provided in time.

The general digital outputs are handled by a single FIFO, and mapped directly to output pins on the SDRLab. Eight LEDs are controlled in the same way, for debugging and monitoring purposes. Digital inputs are read at precise times by a special 'read' instruction, and a 'trigger' instruction can pause the sequencer until a particular digital input changes its state[17].

4.5. Electronic performance and stability

The electronic noise levels, amplitude and voltage fluctuations, and timing jitter of MaRCoS should be low enough that they never limit the quality of reconstructed images or cause artefacts/ghosting.

The FPGA firmware is clock cycle-accurate, and residual jitter in the ADCs or DACs comes from the phase noise of the crystal oscillator as well as voltage and temperature fluctuations. Based on its datasheet [34] the oscillator has a phase noise of −162 dBc/Hz at a 10 kHz offset, and a maximum RMS jitter of 125 fs over a 12 kHz bandwidth; this level of jitter is not easily measurable with common laboratory equipment. A pair of RF pulses 100 ms apart was sampled using a 1 GSPS oscilloscope, and the timing jitter between them was below the measurement resolution of 1 ns. This mea-

[17] This will be used for multi-device synchronization in the future.

surement is likely too coarse by orders of magnitude to capture the jitter, however it sets an upper bound.

We have also run a phase stability experiment using a CPMG pulse sequence [35,36], with 100 repetitions of an echo train with 50 echoes, 10 ms echo spacing and 1 s repetition time. The phase of the acquired echoes was stable down to 180 mrad (standard deviation) for all echoes in the train, and down to $\sim$ 60 mrad for 20% of them.

The SDRLab ADC and DAC can be characterized by their spurious-free dynamic range (SFDR), which are 90 dBc and 75 dBc from their datasheets. We have not independently measured this, although informal online discussions by SDRLab users indicate the ADC performance is better than the DAC, due to the DAC clock coming from the FPGA whereas the ADC is clocked directly from the crystal oscillator [37].

Additionally we have measured the GPA-FHDO voltage drift over several hours to be below 10 μV, which is well below what would affect typical MRI experiments.

The phase and magnitude of control pulses, as well as their timing, is sufficiently stable for demanding MRI applications [3]. Although more thorough characterization is required, the MaRCoS hardware is sufficiently stable for imaging with a range of sequences including gradient echo, turbo spin echo and inversion recovery, using different sampling trajectories including Cartesian and radial [3].

5. Desktop software platform

5.1. MaRCoS client library

As discussed in Section 4.1, the SDRLab executes sequences sent from the client PC, which runs the marcos_client Python library. This provides a class for creating and sending sequences from the PC, and various helper routines for controlling the SDRLab hardware, calibrating the GPA-FHDO and testing various aspects of the MaRCoS system. At the marcos_client level, a sequence is specified by supplying pairs of (time, value) arrays for each hardware output and control, such as the TX envelope, gradient waveforms, etc.[18]. The library scales and rounds the arrays to machine units, applies time offsets to cancel out latencies in the gradient serializers and other hardware, and compiles these values and times into binary instructions. These are encoded via the `msgpack` protocol and sent to the MaRCoS server, which executes the sequence and returns the acquired data. The acquired data is then scaled and passed to the user code, where it can be decimated, filtered, then analysed or reconstructed into an image.

The marcos_client library is the foundation for the higher-level interfaces of MaRCoS, including text-based interfaces such as PulSeq and a graphical interface. Internally marcos_client is not yet optimized, and takes tens of seconds to compile when sequences consisting of hundreds of complex TRs with arbitrary waveforms are run. As this becomes a bottleneck, the code can be sped up in a range of ways, or ported to a language such as C++ or Rust for further performance gains – these will become increasingly important as MaRCoS is scaled to handle more than two TX and RX channels. As mentioned previously the client–server architecture can also be modified to support streaming sequences, which will further improve performance from the user's perspective. For sequences with simple RF and gradient waveforms, the performance overhead from marcos_client and the client–server communications is still minimal. Some typical examples are listed in Table 1. More complex trajectories such as spirals will cause longer proportional delays due to compilation and sequence transfer, however these are yet to be fully benchmarked.

The MaRCoS server-FPGA firmware stack can be emulated on a PC by using the Verilator tool [38] to create a clock cycle-accurate C++ model of the FPGA firmware, and compiling the MaRCoS server on desktop Linux so that it directly interacts with the model. The emulated stack behaves like a virtual SDRLab on the network. It accepts MaRCoS binary sequences and can produce plots of the emulated hardware outputs, as well as complete debugging information from the FPGA firmware, and is useful for new users to learn the MaRCoS software without requiring any hardware. The marcos_client library also has a unit test suite that checks the behaviour of the emulated stack against pre-defined reference outputs, which greatly simplifies development and debugging of the MaRCoS server and FPGA firmware.

So far the emulated stack has been sufficient to debug and optimise the FPGA firmware, however as MaRCoS is used for increasingly complex sequences and protocols, memory and bus contention will increase and the interaction between the embedded server and FPGA firmware will become increasingly non-deterministic. To resolve more complex bugs between the server and firmware, we plan to carry out in situ debugging using Xilinx ChipScope or ultimately custom cores to monitor the real-time dataflow around the design.

5.2. Text-based interfaces

In addition to the direct Python interface of marcos_client, we have created a wrapper that enables MaRCoS users to program pulse sequences in the open-source PulSeq language using either Matlab [39] or Python [40]. The wrapper converts pulse sequences from the PulSeq text file format (which describes RF pulses, gradient pulses, and other sequence events) into Numpy arrays that marcos_client takes as inputs, and is available at [41]. Advantages of PulSeq are its comparative simplicity and its ability to compile the same script to generate sequences for multiple platforms (including Siemens, GE, and Bruker), helping enable reproducible research across sites. The MaRCoS console is now part of this PulSeq pulse sequence programming ecosystem. PulSeq also provides an easy-to-use tool for students in university courses. Some tools built on top of this include basic scripts for finding the center frequency, calibrating the RF transmit power, calibrating the gradient amplitudes, and basic B0 shimming [42].

Another MaRCoS wrapper, included with the marcos_client library, provides a domain-specific language that is similar to Magritek consoles, offering another style of programming. Due to the flexibility of the array-style format accepted by marcos_client, new text-based wrappers are easy to write.

5.3. Graphical User Interface (GUI)

A GUI was designed and developed to facilitate the interaction between the users and MaRCoS. It is available to download at the PhysioMRI GitHub repository [43]. We note that the GUI is intended to be used in laboratory environments and not yet for clinical purposes due to lack of required certification. It offers highly customizable sequences through numerous input parameters that can be modified by the user.

The GUI is written in Python, which allows for fast application development and execution, and relies on the PyQt5 library. PyQt5 is a Python wrapper of Qt, which is a widely used GUI toolkit written in C++. The current version of the GUI has been tested on Ubuntu 20.04.4 LTS and Windows 10, and it is expected to work under macOS systems.

[18] For example, to specify two pulses of 70% full-scale amplitude starting at 20 μs and 100 μs from the beginning of the sequence, each with a duration of 30 μs, one would supply the array pair ([20], [0.7, 0, 0.7, 0]).

Table 1
Typical real-world scan times of various sequences, showing the total overhead due to sequence compilation, transfer and execution, and acquired data transfer. TR times are always 200 ms. Acquired data processing times on the PC (filtering, transforms etc.) are not included. The nominal scan times are how long the sequences would take to run with zero overhead.

Sequence	Matrix size	Echo train length	Nominal scan time (s)	Total overhead (s)
Spin echo	60×60	–	12	0.09
Gradient echo	60×60	–	12	0.06
Turbo spin echo	60×60	5	2.4	0.07
Gradient echo	120×120×10	-	240	1.44
Turbo spin echo	60×60×10	5	24	0.54
Turbo spin echo	120×120×10	5	48	1.83

When the GUI starts, the latest FPGA firmware and MaRCoS server are loaded onto the SDRLab. The MaRCoS server is immediately run afterwards and the first window of the GUI appears, called the *Session Window*. The session window allows the user to select the kind of element that is going to be scanned, which can be a human subject or phantom, and to introduce an ID related to it. Other information can also be introduced for subjects and phantoms, i.e. demographic data such as sex, height and weight in the case of subjects. The acquisitions made in the subsequent windows will be related to this session ID: however, the user can change it at any time by clicking the corresponding icon to come back to the session window. After that, the GUI main window is launched (Fig. 4).

The GUI main window was developed combining two methodologies: 1) graphically by manually placing the different layouts in the window by means of Qt Designer [44], which generates a ui file; and 2) hand-coding the dynamic addition of elements and widgets specific to the selected sequence into the previously defined layouts.

The communication between the GUI and the marcos_client library is largely performed through the sequence classes, also written in Python, which act as an intermediate layer as shown in Fig. 4. This intermediate layer provides some advantages: 1) the GUI can be modified or even replaced while retaining the sequence definitions and their parameterization; and 2) users can run sequences even without executing the GUI, which is useful when debugging new sequence classes or running automated scripts. Experiments are run with a predefined 6-fold oversampling factor applied to the acquisition bandwidth required by the sequence. Once the acquired data is received, the decimation FIR filter implemented in Scipy is applied to extract the signal with the originally required bandwidth. This oversampling factor compensates the non-flat frequency response of the CIC filter applied in the FPGA, as explained above in Section 4.3.

A number of sequences have already been developed and introduced into the GUI repository, which are [45]: 3D turbo spin echo, 3D gradient echo, and 2D HASTE, while new sequences can be integrated very easily. In order to facilitate this, a Python class for each new sequence has to be created. These sequence classes inherit from a parent class (`mriBlankSequence`) that contains many methods common to all sequences, e.g. RF pulses, gradient pulses, readout or save data.

It is essential that new sequence classes include three basic elements: the input parameters, their corresponding default values and the associated labels that will be displayed in the GUI main window. Moreover, they have to include four methods. The first, `sequenceInfo`, displays information related to the author of the sequence class as well as any useful information related to the sequence itself. The `sequenceTime` method estimates the duration of the acquisition every time that a GUI input parameter is changed, and prints the result to the console. The `sequenceRun`

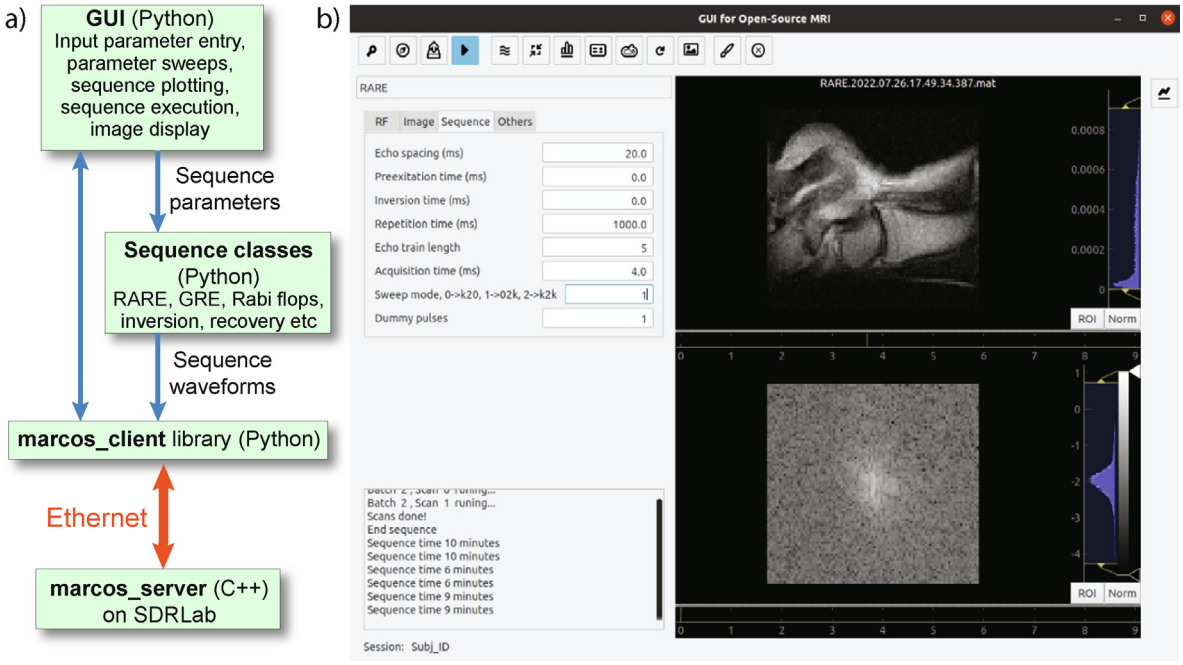

Fig. 4. MaRCoS GUI overview. **a)** Communication and software hierarchy between the GUI and the MaRCoS server on the SDRLab. **b)** Main window of the GUI.

method is where the sequence structure and parameterization is defined. It takes the parameters supplied by the user (i.e. pulse times, frequencies, gradient voltages, repetitions etc.) and converts them into Numpy arrays, which are then passed to the marcos_client library as described in Section 5.2. The marcos_client library compiles the sequence data into a batch of instructions, consisting of hundreds of TRs with precise timing and phase coherence maintained between them; it then executes the sequence on the SDRLab and returns the acquired data to the GUI. Finally, `sequenceAnalysis` processes the acquired data and defines the plots that the GUI displays.

The GUI also integrates a calibration window with useful functions that helps to calibrate the scanner. At the moment, the calibration functions already developed are: Rabi flops for flip angle calibration, inversion recovery sequence to measure T_1, resonant frequency to compensate for any field drift, noise measurement to characterize the system in new environments, a CPMG pulse sequence to measure T_2, shimming and generic slice selection. The calibration window was designed following the same procedure as the main window and its appearance is very similar.

Other functionalities implemented in the GUI main window cover the export of the selected sequence and loading and saving input parameters from and to files. The export is also useful for the batch acquisition functionality, which performs multiple measurements sequentially without human intervention. In this case, the user exports the parameters of the different experiments to be performed and uploads them in the desired order to the batch window. The GUI also includes the possibility to linearly sweep two parameters for any image or calibration sequence.

Finally, the GUI is linked to an XNAT server [46]. XNAT is an open source imaging informatics platform that facilitates common management, productivity, and quality assurance tasks for imaging and associated data. By enabling the corresponding icon in the GUI main window, it is possible to automatically upload the recently acquired MRIs to the XNAT if it is running on the PC. Data is saved in Matlab files (.mat) and NIfTI format.

The GUI is routinely used at i3M. However, there are a number of developments which we wish to add, such as an automatic method that checks if the sequence parameters are within hardware limits and avoids executing sequences that are out of bounds, the use of sequences that offer the possibility to manage multiple TX or RX channels, and the use of different frequencies for TX and RX.

Additional sequences such as balanced steady state free precession, echo planar imaging or zero echo time, as well as new sampling trajectories beyond Cartesian trajectories, will enrich the possibilities offered by the GUI. Other useful utilities, which are often featured in commercial imaging systems, include the possibility to set the field of view directly from a scout image, or be able to select a region of interest and show the mean value and standard deviation to get a quick estimation of the signal-to-noise ratio. Another longer-term aim is to make the GUI accessible to non-expert users, i.e. to develop protocols to generate a standardized set of images from different sequences.

6. Conclusion

This paper has described the design and initial performance of MaRCoS, an open-source MRI electronic control system based on a Red Pitaya SDRLab platform and custom gradient boards. The system has two transmit channels, two receive channels, and can drive three gradient DACs with another channel also available. The control and testing can be performed in readily available open-source software. Critical measures of phase stability, timing accuracy, and data transfer rates have been evaluated. The receiver channel can directly digitize the MR signal, with significant oversampling enabling the minimization of quantization noise. CIC filters are used in the final stage of signal processing. A graphical user interface has been designed in Python, which provides a critical interface in being able to write new imaging sequences and to assess image performance. The development of the MaRCoS system is just beginning, and a number of future advances and desirable extensions have also been listed. The low cost means that interested readers can be up-and-running for around $1000 (for the SDRLab and gradient board). There are a number of constantly upgraded and interactive resources to get started, including a wiki page [47] and Slack channel [48].

Author contributions

J. Alonso, A. Webb and V. Negnevitsky conceived and planned the MaRCoS project, with major technical contributions and advice from B. Menküc and J. P. Stockmann. B. Menküc wrote the GPA-FHDO serializer, CIC filter, complex multiplier and NCO modules of the MaRCoS firmware, and V. Negnevitsky wrote the rest of the firmware. L. Craven-Brightman wrote the MaRCoS-PulSeq library. Y. Vives-Gilabert and J. M. Algarín wrote the MaRCoS GUI. V. Negnevitsky wrote the MaRCoS server, client and auxiliary tools. B. Menküc, J. M. Algarín, R. Pellicer-Guridi, T. O'Reilly, Y. Vives-Gilabert and L. Craven-Brightman tested and characterized the system, in the research laboratories of J. Alonso, B. Menküc, A. Webb and J. P. Stockmann. V. Negnevitsky, B. Menküc, J. Alonso, Y. Vives-Gilabert, J. M. Algarín, J. P. Stockmann and A. Webb prepared the manuscript, with input from all authors.

Data availability

Data will be made available on request.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Enormous credit goes to Thomas Witzel, Marcus Prier, Suma Anand and David Schote for their work on the OCRA system as well as many fruitful discussions. Thanks to Danny Park for preparing the original SDRLab Linux image used in both OCRA and MaRCoS, and Maxim Zaitsev for fruitful discussions on rounding and timing in PulSeq. This work was supported by the Ministerio de Ciencia e Innovación of Spain through research grant PID2019-111436RB-C21. The work was co-financed by the European Union through the Programa Operativo del Fondo Europeo de Desarrollo Regional (FEDER) of the Comunitat Valenciana (IDIFEDER/2018/022 and IDIFEDER/2021/004), Future and Emerging Technologies (FET) grant 101034644, and ERC Advanced Grant 101021218 PASMAR; the National Institutes of Health NIBIB under Grant No. U24-EB028984; and the HIFF program of the University of Applied Sciences and Arts Dortmund.

References

[1] E.M. Haacke, R.W. Brown, M.R. Thompson, R. Venkatesan, et al., Magnetic Resonance Imaging: Physical Principles and Sequence Design, vol. 82, Wiley-liss New York, 1999.

[2] 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. Imaging 31 (2) (2012) 370–379, https://doi.org/10.1109/TMI.2011.2169681.

[3] T. Guallart-Naval, T. O'reilly, J.M. Algarín, R. Pellicer-Guridi, Y. Vives-Gilabert, Lincoln Craven-Brightman, V. Negnevitsky, B. Menküc, Fernando Galve, J.P. Stockmann, A. Webb, J. Alonso, Benchmarking the performance of a low-cost magnetic resonance control system at multiple sites in the open MaRCoS community, NMR Biomed. (2022) e4825arXiv:2203.11314, https://doi.org/10.1002/NBM.4825. https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/10.1002/nbm.4825.

[4] P.C. McDaniel, C.Z. Cooley, J.P. Stockmann, L.L. Wald, The MR Cap: A single-sided MRI system designed for potential point-of-care limited field-of-view brain imaging, Magn. Reson. Med. 82 (5) (2019) 1946–1960, https://doi.org/10.1002/MRM.27861, URL https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.27861.

[5] M. Nakagomi, M. Kajiwara, J. Matsuzaki, K. Tanabe, S. Hoshiai, Y. Okamoto, Y. Terada, Development of a small car-mounted magnetic resonance imaging system for human elbows using a 0.2 T permanent magnet, J. Magn. Reson. 304 (2019) 1–6, https://doi.org/10.1016/j.jmr.2019.04.017.

[6] C.Z. Cooley, P.C. McDaniel, J.P. Stockmann, S.A. Srinivas, S.F. Cauley, M. Śliwiak, C.R. Sappo, C.F. Vaughn, B. Guerin, M.S. Rosen, M.H. Lev, L.L. Wald, A portable scanner for magnetic resonance imaging of the brain, Nature Biomed. Eng. 2020 5:3 5 (3) (2020) 229–239. https://doi.org/10.1038/s41551-020-00641-5. https://www.nature.com/articles/s41551-020-00641-5.

[7] T. O'Reilly, W.M. Teeuwisse, D. Gans, K. Koolstra, A.G. Webb, In vivo 3D brain and extremity MRI at 50 mT using a permanent magnet Halbach array, Magn. Reson. Med. (2020), https://doi.org/10.1002/mrm.28396, mrm.28396. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/mrm.28396.

[8] K.N. Sheth, M.H. Mazurek, M.M. Yuen, B.A. Cahn, J.T. Shah, A. Ward, J.A. Kim, E.J. Gilmore, G.J. Falcone, N. Petersen, K.T. Gobeske, F. Kaddouh, D.Y. Hwang, J. Schindler, L. Sansing, C. Matouk, J. Rothberg, G. Sze, J. Siner, M.S. Rosen, S. Spudich, W.T. Kimberly, Assessment of brain injury using portable, low-field magnetic resonance imaging at the bedside of critically Ill Patients, JAMA, Neurology 78 (1) (2021) 41–47, https://doi.org/10.1001/JAMANEUROL.2020.3263, URL https://jamanetwork.com/journals/jamaneurology/fullarticle/2769858.

[9] M.H. Mazurek, M.M. Yuen, B.A. Cahn, M.S. Rosen, K.T. Gobeske, E.J. Gilmore, D. Hwang, F. Kaddouh, J.A. Kim, G. Falcone, N. Petersen, J. Siner, S. Spudich, G. Sze, W.T. Kimberly, K.N. Sheth, Low-Field, Portable Magnetic Resonance Imaging at the Bedside to Assess Brain Injury in Patients with Severe COVID-19 (1349), Neurology 96 (15 Supplement).

[10] Y. Liu, A.T.L. Leong, Y. Zhao, L. Xiao, H.K.F. Mak, A. Chun, O. Tsang, G.K.K. Lau, G. K.K. Leung, E.X. Wu, X. Linfang, A low-cost and shielding-free ultra-low-field brain MRI scanner, Nat. Commun. 12 (1) (2021) 1–14, https://doi.org/10.1038/s41467-021-27317-1, URL https://www.nature.com/articles/s41467-021-27317-1.

[11] T. Guallart-Naval, J. Algarín, R. Pellicer-Guridi, F. Galve, Y. Vives-Gilabert, R. Bosch, E. Pallás, J. González, J. Rigla, P. Martínez, F. Lloris, J. Borreguero, Á. Marcos-Perucho, V. Negnevitsky, L. Martí-Bonmatí, A. Ríos, J.M. Benlloch, J. Alonso, Portable magnetic resonance imaging of patients indoors, outdoors and at home, Scientific Reports 12 (1) (2022) 1–11, https://doi.org/10.1038/s41598-022-17472-w, URL https://www.nature.com/articles/s41598-022-17472-w.

[12] T. O'Reilly, A.G. Webb, In vivo T1 and T2 relaxation time maps of brain tissue, skeletal muscle, and lipid measured in healthy volunteers at 50 mT, Magn. Reson. Med. 87 (2) (2021) 884–895, https://doi.org/10.1002/MRM.29009, URL https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.29009.

[13] M. Sarracanie, Fast Quantitative Low-Field Magnetic Resonance Imaging With OPTIMUM - Optimized Magnetic Resonance Fingerprinting Using a Stationary Steady-State Cartesian Approach and Accelerated Acquisition Schedules, Investigative Radiology. URL https://journals.lww.com/investigativeradiology/Fulltext/9000/Fast_Quantitative_Low_Field_Magnetic_Resonance.98659.aspx.

[14] J.M. Algarín, E. Díaz-Caballero, J. Borreguero, F. Galve, D. Grau-Ruiz, J.P. Rigla, R. Bosch, J.M. González, E. Pallás, M. Corberán, C. Gramage, S. Aja-Fernández, A. Ríos, J.M. Benlloch, J. Alonso, Simultaneous imaging of hard and soft biological tissues in a low-field dental MRI scanner, Scientific Reports 10 (1) (2020) 21470, https://doi.org/10.1038/s41598-020-78456-2, arXiv:/arxiv.org/abs/2005.01462.

[15] J. Borreguero, J.M. Gonzalez, E. Pallas, J.P. Rigla, J.M. Algarin, R. Bosch, F. Galve, D. Grau-Ruiz, R. Pellicer, A. Rios, J.M. Benlloch, J. Alonso, Prepolarized MRI of Hard Tissues and Solid-State Matter, NMR Biomed. (2022), https://doi.org/10.1002/NBM.4737, URL https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/10.1002/nbm.4737.

[16] J. Borreguero, F. Galve, J.M. Algarín, J.M. Benlloch, J. Alonso, Slice-selective Zero Echo Time imaging of ultra-short T2 tissues based on spin-locking, arXiv preprint arXiv:2201.06305arXiv:2201.06305v1.

[17] C.Z. Cooley, J.P. Stockmann, T. Witzel, C. LaPierre, A. Mareyam, F. Jia, M. Zaitsev, Y. Wenhui, W. Zheng, P. Stang, G. Scott, E. Adalsteinsson, J.K. White, L.L. Wald, Design and implementation of a low-cost, tabletop MRI scanner for education and research prototyping, J. Magn. Reson. 310 (2020) 106625, https://doi.org/10.1016/j.jmr.2019.106625.

[18] W. Tang, W. Wang, W. Liu, Y. Ma, X. Tang, L. Xiao, J.H. Gao, A home-built digital optical MRI console using high-speed serial links, Magn. Reson. Med. 74 (2) (2015) 578–588, https://doi.org/10.1002/MRM.25403, URL https://onlinelibrary.wiley.com/doi/10.1002/mrm.25403.

[19] C.J. Hasselwander, Z. Cao, W.A. Grissom, gr-MRI: A software package for magnetic resonance imaging using software defined radios, J. Magn. Reson. 270 (2016) 47–55, https://doi.org/10.1016/j.jmr.2016.06.023.

[20] S. Anand, J.P. Stockmann, L.L. Wald, T. Witzel, A low-cost (<$500 USD) FPGA-based console capable of real-time control, in: Proceedings of the 2018 ISMRM & SMRT annual meeting and exhibition in Paris, ISMRM, 2018, p. 948.

[21] OCRA MRI. https://openmri.github.io/ocra/.

[22] K. Takeda, Chapter 7 - Highly Customized NMR Systems Using an Open-Resource, Home-Built Spectrometer, Vol. 74 of Annual Reports on NMR Spectroscopy, Academic Press, 2011, pp. 355–393. https://doi.org/10.1016/B978-0-08-097072-1.00007-8. URL https://www.sciencedirect.com/science/article/pii/B9780080970721000078.

[23] C.A. Michal, A low-cost multi-channel software-defined radio-based NMR spectrometer and ultra-affordable digital pulse programmer, Concepts Magnetic Reson. Part B: Magnetic Reson. Eng. 48B (3) (2018) e21401, arXiv: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cmr.b.21401, doi: 10.1002/cmr.b.21401. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cmr.b.21401.

[24] A. Ang, M. Bourne, S. Obruchkov, R. Dykstra, Construction of a pxie platform for instrumentation development, in: 2017 Eleventh International Conference on Sensing Technology (ICST), 2017, pp. 1–4, https://doi.org/10.1109/ICSensT.2017.8304504.

[25] D. Ariando, C. Chen, M. Greer, S. Mandal, An autonomous, highly portable NMR spectrometer based on a low-cost System-on-Chip (SoC), J. Magn. Reson. 299 (2019) 74–92, https://doi.org/10.1016/J.JMR.2018.12.007.

[26] J. Zhen, C. Dykstra, G. Gouws, S. Obruchkov, R. Dykstra, Mobile low field magnetic resonance hardware development, J. Magn. Reson. 322 (2021) 106852, https://doi.org/10.1016/J.JMR.2020.106852.

[27] F. Galve, J. Alonso, J.M. Algarín, J.M. Benlloch, Magnetic Resonance Imaging method with zero echo time and slice selection.

[28] Red Pitaya SDRLab, https://redpitaya.com/sdrlab-122-16/.

[29] B. Menküc, GPA-FHDO GitHub project website, https://github.com/menkueclab/GPA-FHDO.

[30] OCRA1 - SPI controlled 4 channel 18 bit DAC and RF attenuator. https://zeugmatographix.org/ocra/2020/11/27/ocra1-spi-controlled-4-channel-18bit-dac-and-rf-attenutator/.

[31] Complex multiplier core Git repository. https://github.com/catkira/complex_multiplier.

[32] CIC core Git repository. https://github.com/catkira/CIC.

[33] DDS core Git repository. https://github.com/catkira/DDS.

[34] Abracon ABLNO-122.880MHz datasheet. https://abracon.com/Precisiontiming/ABLNO.pdf.

[35] H.Y. Carr, Steady-state free precession in nuclear magnetic resonance, Phys. Rev. 112 (5) (1958) 1693–1701, https://doi.org/10.1103/PhysRev.112.1693.

[36] S. Meiboom, D. Gill, Modified spin-echo method for measuring nuclear relaxation times, Rev. Sci. Instrum. 29 (8) (1958) 688–691, https://doi.org/10.1063/1.1716296.

[37] STEMlab 16–122.88 - TX composite noise measurements [German]. https://saure.org/phpBB_04/viewtopic.php?f=35&t=519.

[38] Verilator, a fast open-source Verilog/SystemVerilog simulator. https://www.veripool.org/verilator/.

[39] K.J. Layton, S. Kroboth, F. Jia, S. Littin, H. Yu, J. Leupold, J.F. Nielsen, T. Stöcker, M. Zaitsev, Pulseq: A rapid and hardware-independent pulse sequence prototyping framework, Magn. Reson. Med. 77 (4) (2017) 1544–1552, https://doi.org/10.1002/MRM.26235/ASSET/SUPINFO/MRM26235-SUP-0001-SUPPINFO.PDF, URL https://onlinelibrary.wiley.com/doi/10.1002/mrm.26235.

[40] K.S. Ravi, S. Geethanath, J.T. Vaughan, PyPulseq: A Python package for MRI pulse sequence design, J. Open Source Softw. 4 (42) (2019) 1725, https://doi.org/10.21105/joss.01725.

[41] FLOCRA-PulSeq Git repository. https://github.com/stockmann-lab/flocra-pulseq.

[42] MGH_MARCOS Project. https://github.com/stockmann-lab/mgh_marcos.

[43] PhysioMRI Git repository. https://github.com/yvives/PhysioMRI_GUI.

[44] Qt Designer. https://doc.qt.io/qt-6/qtdesigner-manual.html.

[45] M. Bernstein, K. King, X. Zhou, Handbook of MRI Pulse Sequences, Elsevier Academic Press, 2004.

[46] XNAT, open source imaging informatics. https://www.xnat.org/.

[47] MaRCoS Git repository and wiki. https://github.com/vnegnev/marcos_extras/wiki.

[48] MaRCoS Slack group. https://marcos-system.slack.com.