
The TOPPE pulse programming environment for GE MRI scanners

This document applies to toppe.e version 0.9-7d.

Tested on a GE Discovery MR750 scanner running software version DV25.

Version of this document: 0.9-7d-2016/03/30

Jon-Fredrik Nielsen, Ph.D.

`jfnielse@umich.edu`

Contents

1 Overview	1
1.1 Introduction	1
1.2 Required external files	1
1.2.1 *.wav files	2
1.2.2 cores.txt	2
1.2.3 scanloop.txt	2
2 Using the toppe sequence	4
2.1 Creating .wav files	4
2.2 Creating cores.txt	4
2.3 Creating scanloop.txt	4
2.4 Testing your files with scansim.m	5
2.5 Compiling the toppe pulse sequence	6
2.6 Legacy files that must exist in /usr/g/bin/ on scanner	6
2.7 Step-by-step scanner instructions	6
2.8 Known bugs and limitations	8
3 Controlling sequence timing	10
4 Using toppe.e as an interpreter module for Pulseq files	12
4.1 Pulseq	12
4.2 Using toppe.e to play .seq files	12
Appendices	13
A Tools for RF and gradient waveform design	14
A.1 Matlab scripts included in this distribution	14
A.2 John Pauly's RF pulse design code (Matlab)	14
A.3 Brian Hargreaves' spiral gradient design code (Matlab)	14
A.4 Generating Pulseq files	14

Chapter 1

Overview

1.1 Introduction

Implementing research pulse sequences on GE MR scanners requires EPIC programming, a time-consuming and error-prone task with a steep learning curve. Moreover, pulse sequences need to be recompiled after each scanner software upgrade, which is sometimes problematic.

This user guide describes the “`toppe.e`”¹ pulse sequence for GE scanners, which allows the entire sequence to be specified with a set of external files created with a high-level software tool such as Matlab. This makes it possible to play arbitrary sequences of RF pulses and gradient waveforms, which enables **rapid prototyping of sequences without the need for low-level EPIC programming**. With `toppe.e`, the task of pulse programming a GE scanner becomes one of creating the various external files that define the sequence.

`toppe.e` was developed as a research tool at the fMRI laboratory at University of Michigan, and has to date been used in several projects including stack-of-spirals imaging, Bloch-Siebert B1+ mapping, echo-shifted RF-spoiled imaging (PRESTO), steady-state imaging with 3D tailored RF excitation, and dual-echo steady-state (DESS) imaging.

We are currently making `toppe.e` compatible with **Pulseseq**, an open file format for compactly describing MR sequences. See Chapter 4 for more information about using `toppe.e` as a GE “interpreter module” for Pulseseq files.

This is the first ‘beta’ release of `toppe.e`, and **your feedback** is most welcome.

1.2 Required external files

In addition to the `toppe` and `toppe.psd.o` executables, which only need to be compiled and installed once for each scanner software upgrade, the following files are needed to run a particular scan:

¹“The End Of Pulse Programming”, rearranged; pronounced “top dot e”

1.2.1 *.wav files

toppe.e creates several unique “cores” (or modules), with each core/module associated with one .wav file (Fig. 1.1). For example, an RF excitation module may be defined by a file called tipdown.wav that specifies the RF amplitude and phase waveforms (rho and theta) and all three gradients. Similarly, a Cartesian (spin-warp) gradient-echo readout may be defined in a file readout.wav that contains readout and phase-encode gradient waveforms. Finally, a spoiler gradient can be defined in a file spoiler.wav. Each .wav file is unique up to a scaling factor, and typically only a few .wav files are needed. Note that each .wav file gives rise to a separate createseq() call in toppe.e.

1.2.2 cores.txt

The various *.wav files needed to define a scan are listed in a small text file named cores.txt, which simply contains a line for each .wav file specifying the file name, core duration, and whether it is an RF excitation module, readout module, or gradients-only module. Values are tab-separated. A cores.txt file for our simple spin-warp imaging example may look like this:

```
Total number of unique cores
3
wavfile_name      duration(us)      hasRF?  hasDAQ?
tipdown.wav 0      1      0
readout.wav 0      0      1
spoiler.wav 0      0      0
```

A duration of 0 means that the minimum core duration for that .wav file will be used.

1.2.3 scanloop.txt

Finally, the complete MR scan loop is specified in scanloop.txt, in which each line corresponds to a separate startseq() call in toppe.e. A scanloop.txt file for single-slice, RF-spoiled spin-warp imaging with 256 phase-encodes might begin like this:

```
nt maxslice      maxecho maxview
768 0      0      768
Core iarf iath iagx iagy iagz slice echo view dabon rot rfph recph textra
1 32766 32766 0 0 32766 0 0 0 0 0 0 0 0
2 0 0 32766 32766 -32638 0 0 1 1 0 0 0 0
3 0 0 0 0 32766 0 0 0 0 0 0 0 0
1 32766 32766 0 0 32766 0 0 0 0 0 21298 0 0
2 0 0 32766 32766 -32382 0 0 2 1 0 0 21298 0
3 0 0 0 0 32766 0 0 0 0 0 0 0 0
...
```

where nt is the total number of startseq() calls (256 phase-encodes \times 3 cores per TR), and maxslice, maxecho, and maxview correspond to rhnslices-1, rhnecho-1, and rhnframes, respectively, in toppe.e. Each row in scanloop.txt specifies which core to play and with what RF and gradient amplitudes; where to store the data using loadtab() (i.e., which slice/echo/view); whether the core is used to acquire data; in-plane (kx,ky) rotation angle; the RF and acquisition phase; and a value 'textra' (in μ s) by which the core

duration is extended (see Ch. 3). Values are tab-separated. For long scans, `scanloop.txt` can contain many tens of thousands of lines.

cores.txt:

wavfile_name	duration(us)	hasRF?	hasDAQ?
Total number of unique cores			
3			
tipdown.wav	0	1	0
readout.wav	0	0	1
tipup.wav	2000	1	0

scanloop.txt:

core	ia_rf	ia_th	ia_gx	ia_gy	ia_gz	slice	echo	view	dabon	rot	rfphs	recphs	textra
1	1820	32768	32768	32768	32768	0	0	0	0	0	0	0	0
2	1820	32768	32768	-32222	-29492	0	0	1	1	0	0	0	0
3	1820	32768	32768	32222	29492	0	1	1	0	0	0	0	0
1	1820	32768	32768	32768	32768	0	0	0	0	0	0	0	0
2	1820	32768	32768	-31130	-29492	0	0	2	1	0	0	0	0
3	1820	32768	32768	31130	29492	0	1	2	0	0	0	0	0
1	1820	32768	32768	32768	32768	0	0	0	0	0	0	0	0
2	1820	32768	32768	-30038	-29492	0	0	3	1	0	0	0	0
3	1820	32768	32768	30038	29492	0	1	3	0	0	0	0	0
:													

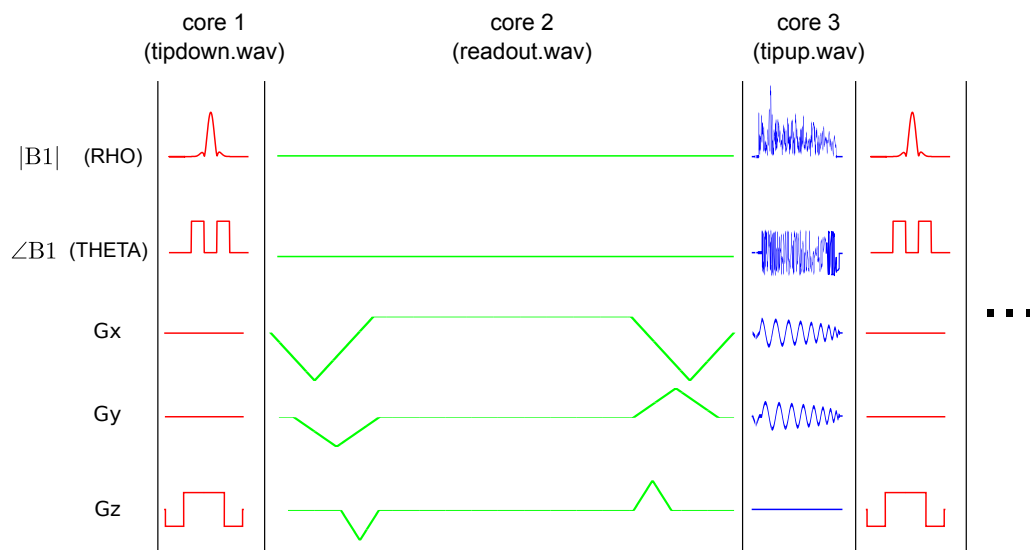


Figure 1.1: Overview of pulse sequence programming with `toppe`. The `toppe/toppe.psd.o` executables load the .wav files listed in `cores.txt`, and play them out according to the instructions in `scanloop.txt`. Each .wav file contains the RF and gradient waveform shapes for one core. A core is dedicated to either RF excitation, data acquisition, or gradients-only. With this setup, arbitrary sequences of RF and gradient pulses can be played out, which enables rapid pulse sequence prototyping without the need for EPIC programming. For detailing sequence timing information, see Ch. 3.

Chapter 2

Using the toppe sequence

2.1 Creating .wav files

The `matlab` folder in the toppe distribution (tar file) includes the Matlab script **mat2wav.m** that writes rho, theta, gx, gy, and gz waveforms for a core/module to a .wav file. Important notes and caveats:

- All waveforms in a core must have the same length, i.e., they must be padded with zeroes as needed.
- Even if the core is not an RF excitation core, you must create a non-zero 'dummy' RF pulse to ensure that the .wav file will be loaded correctly on the scanner (hopefully this bug will be fixed in future releases). A simple hard RF pulse of low amplitude (e.g., 0.01 Gauss) seems to work well in most cases.
- If the core is a readout core, data will be acquired every 4us for the entire duration of the waveforms in the .wav file. Depending on your readout trajectory, you may therefore need to discard some of the data (near the beginning and/or end of the core) before reconstructing.
- If more than one readout .wav file is used, they must all be the same length (readout windows of different widths are not permitted).
- For backward compatibility, the following must be done (this may change in future releases):
 - One of the readout .wav files must be named **readout.wav**
 - One of the RF excitation .wav files must be listed in a file **rfiles.txt** in `/usr/g/bin/`

2.2 Creating cores.txt

`cores.txt` can simply be created by hand, as specified above.

2.3 Creating scanloop.txt

The toppe distribution contains an example Matlab file in the `example` directory, **writeloop.m**, that creates `scanloop.txt` for a scan that combines Bloch-Siegert mapping, and SPGR acquisitions with different flip angle and TR, into one long scan.

2.4 Testing your files with scansim.m

We recommend displaying your sequence in Matlab using `scansim.m` before attempting to play it on the scanner, to verify that the correct cores are played out in the intended order. `scansim.m` attempts to reproduce the exact core timing seen on the scanner, using CV values in the file `timing.txt`. To use `scansim.m`, first use `readloop.m` to load a `scanloop.txt` file. `scansim.m` and `readloop.m` can be found in the `matlab` directory. As an example, do the following in the example directory:

```
>> addpath('../matlab/');
>> d = readloop('scanloop.txt');
>> startseqStart = 10000;
>> startseqEnd = startseqStart+19; % display 20 consecutive startseq() calls
% Load the .wav files listed in cores.txt and display part of sequence.
% Exact timing information is loaded from 'timing.txt':
>> scansim(startseqStart,startseqEnd,d);
```

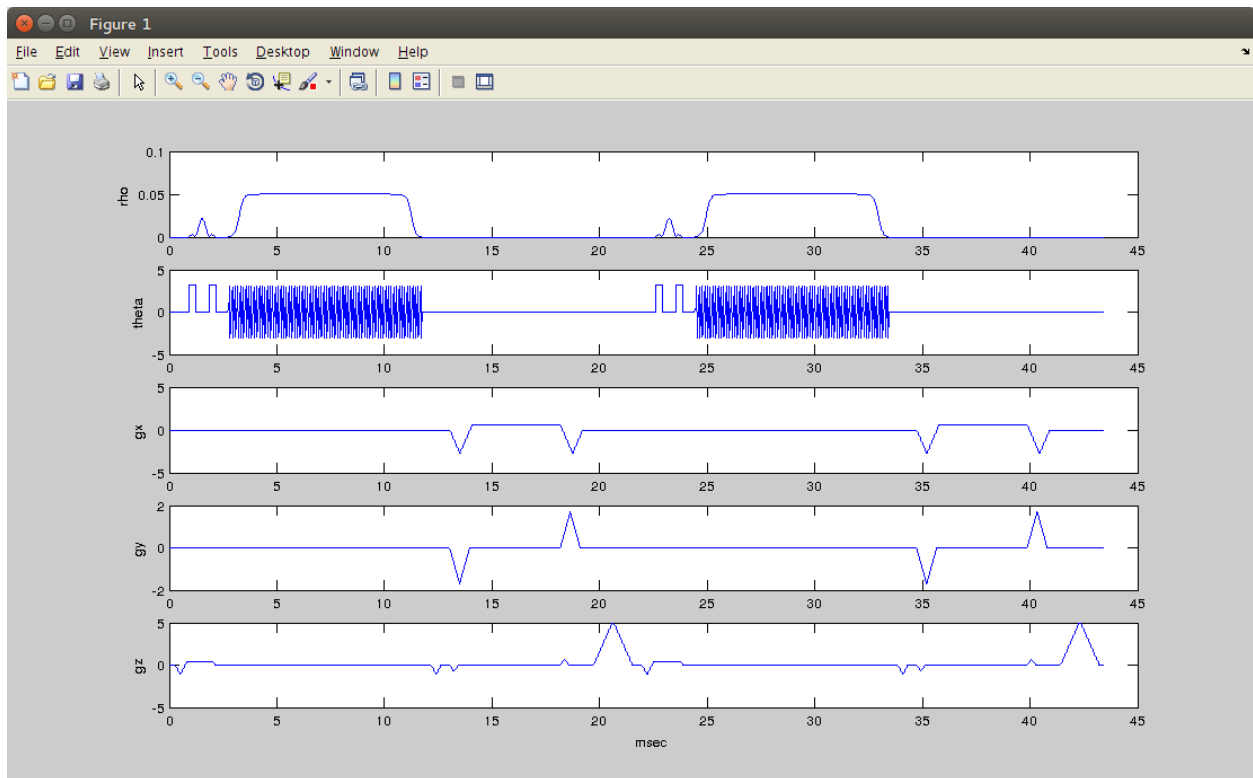


Figure 2.1: Example sequence display created with `scansim.m`. The sequence shown is a Bloch-Siebert B1 transmit mapping sequence with a 3D Cartesian readout. Like the `toppe` and `toppe.psd.o` executables on the scanner, `scansim.m` loads `cores.txt` and the `.wav` files listed therein, and `scanloop.txt`. In addition, `scansim.m` obtains exact sequence timing information from the file `timing.txt`.

2.5 Compiling the toppe pulse sequence

The current version of `toppe.e` has been compiled for DV25, and has been tested on a GE Discovery MR750 3T scanner. The `toppe.e` source code is in the `psd` directory.

To compile, follow the usual EPIC compilation steps. First, check compiler version:

```
which psdmake
```

With the compiler at University of Michigan, the output is

```
/ESE_DV25.0_R01/psd/bin/psdmake
```

Prepare directory for compilation and compile:

```
prep_psd_dir  
psdmake hw
```

This will create two executables: `toppe` and `toppe.psd.o`.

2.6 Legacy files that must exist in `/usr/g/bin/` on scanner

The README file in the `legacy` directory lists the files that must exist in `/usr/g/bin/` on the scanner for the `toppe` executables to run.

2.7 Step-by-step scanner instructions

Follow these steps to prescribe and run the `toppe` sequence:

1. Copy `toppe` and `toppe.psd.o` to `/usr/g/bin/` on the scanner host computer (console).
2. Copy `cores.txt`, `scanloop.txt`, and all `.wav` files to `/usr/g/bin/`.
3. Make sure the required **legacy files** exist:
 - Make sure one of the readout (acquisition) `.wav` files is named `readout.wav`.
 - Create a file `rffiles.txt` in `/usr/g/bin/`, containing the name of one of the RF `.wav` files.
 - Copy `sech_7360.rho`, `sech_7360.theta`, and `myhanning.rho` from the `legacy` directory in this distribution to `/usr/g/bin/` on the scanner.
4. Run a localizer sequence, including auto-prescan.
5. Prescribe the `toppe` sequence:
 - Select Axial 2D pulse sequence; Family: 'Gradient Echo'; pulse: 'GRE'; PSD Name: 'toppe'; click 'Accept'. (Fig. 2.2)
 - Freq. FOV: 24 (Fig. 2.3)
 - Set slice thickness to the design value.

- Set slice spacing to 0.
 - Freq. Dir: R/L
 - Set number of slices to rhnslices (equal to maxslice+1, see cores.txt).
 - TE: 1.0 (doesn't matter)
 - Flip angle: 10 (doesn't matter)
6. Save and download the sequence. Do NOT click the scan button yet.
 7. Do a 'dummy' manual prescan, by entering into manual prescan (Fig. 2.4), adjusting receive gains if needed, and exiting (this will avoid auto-prescan, which doesn't work at the time of writing).
 8. Click scan button. This will create a Pfile in /usr/g/mrraw/.
 9. If you see evidence of overranging in the reconstructed image you will need to reduce the receive gains (in manual prescan) and scan again.
 10. To run a different scan with the same number of slices, simply overwrite cores.txt and scanloop.txt and make sure the .wav files for the next sequence exist in /usr/g/bin/. Then download the sequence (right-click) and hit Scan button. You do not need to prescribe a new sequence every time you load a new set of external files.

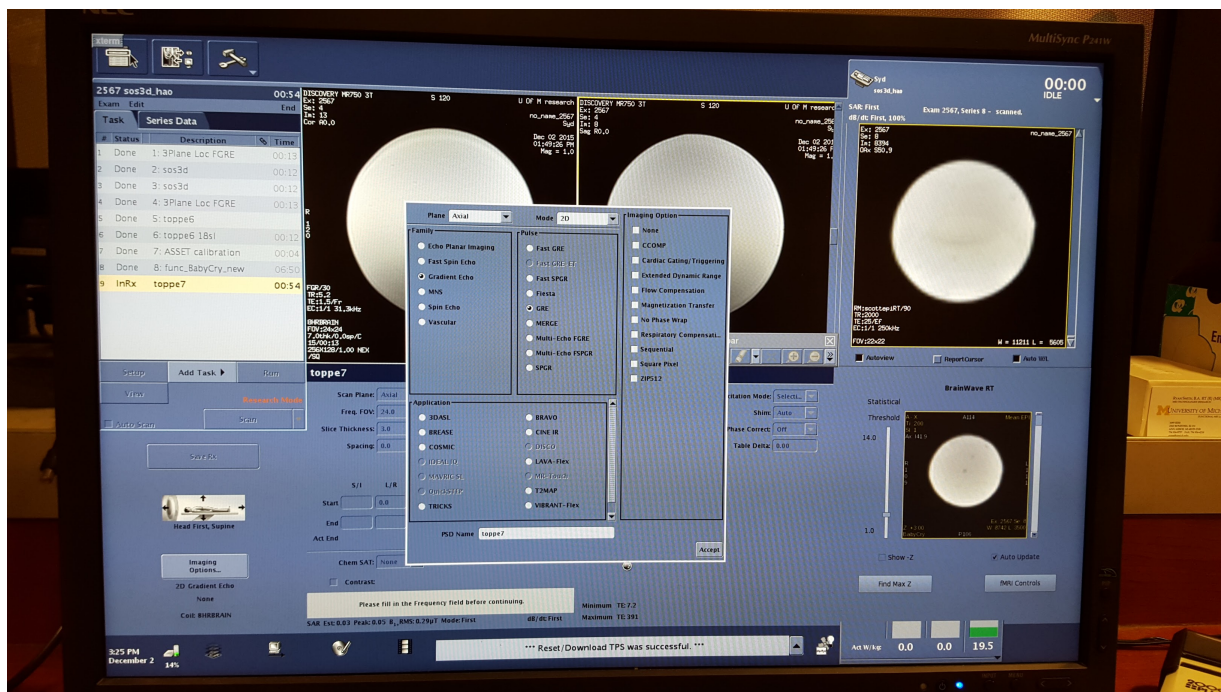


Figure 2.2: Scanner prescription, screenshot 1.

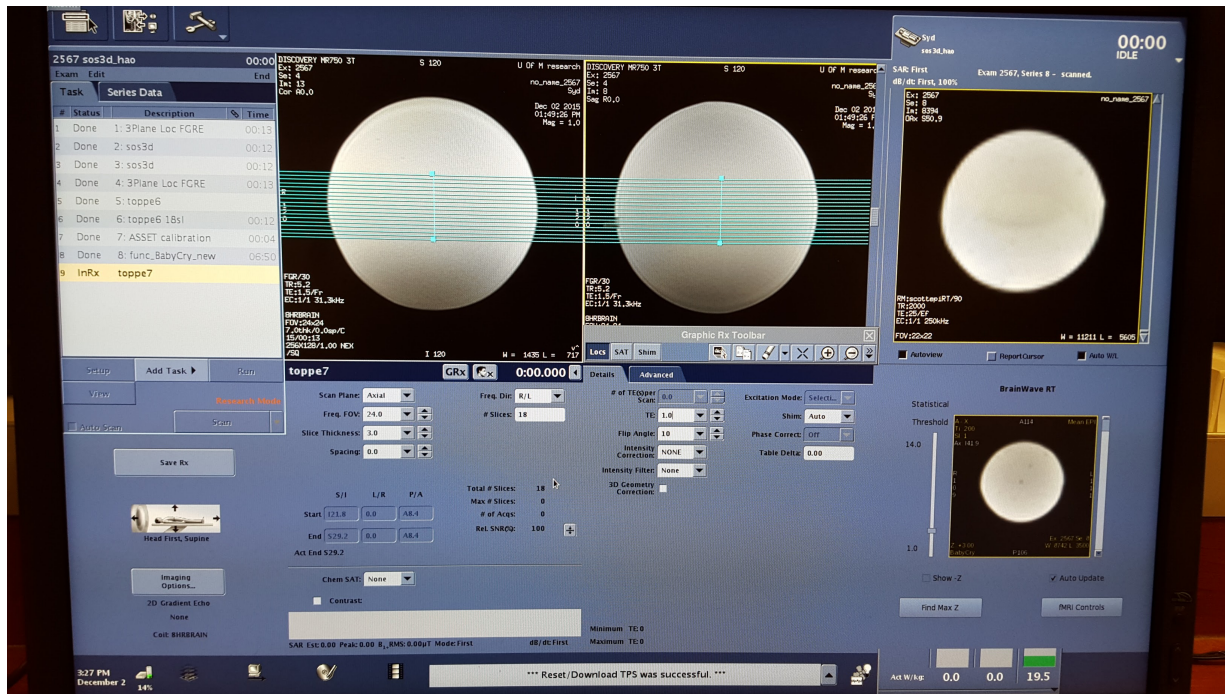


Figure 2.3: Scanner prescription, screenshot 2.

2.8 Known bugs and limitations

- Data is saved in **reversed order**, so be sure to account for this in your reconstruction.
- Auto prescan doesn't work, hence the use of manual prescan above.
- B1 scaling across multiple RF pulses has not been verified. May need to expand `rfpulse` struct.
- `toppe.e` does not support cardiac/respiratory gating at the moment. If other groups have a need for this we believe gating can be easily added.
- `toppe.e` does not currently do any SAR, PNS, or gradient heating checks.

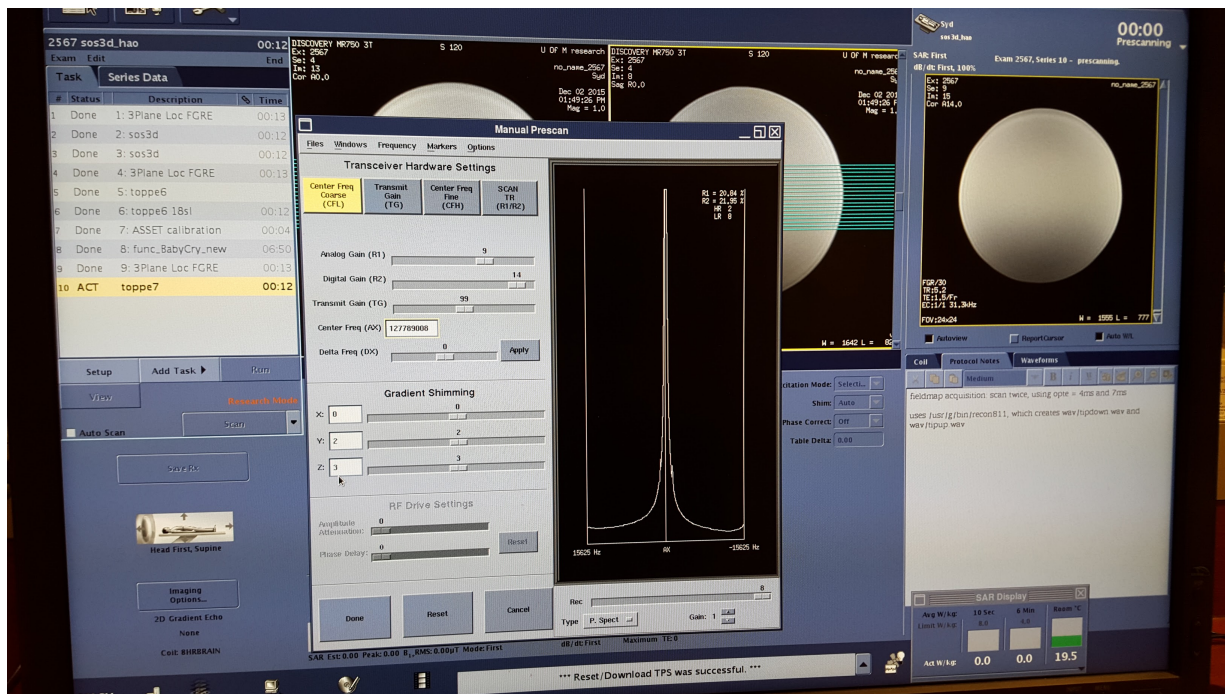


Figure 2.4: Scanner prescription, with manual prescan window.

Chapter 3

Controlling sequence timing

The default core duration is set to the value specified in `cores.txt`, however the duration can be extended in real-time by setting the value of the 'extra' column in `scanloop.txt` to a non-zero value. This allows the sequence timing to be controlled dynamically, e.g., for the purpose of varying TE or TR.

If the minimum core duration exceeds the prescribed duration in `cores.txt`, the minimum core duration is used (without warning). It is therefore perfectly fine to set the core duration in `cores.txt` to '0', since this guarantees that the minimum duration will be used which is often the desired behavior.

Figure 3.1 shows detailed timing information for the three core types: gradients-only, RF excitation, and data acquisition. For gradients-only cores, the minimum core duration is equal to the waveform duration plus the controls variables (CVs) 'start_core', 'timetrwait', and 'timessi'. These have been determined empirically, and are currently set to 224us, 64us, and 100us, respectively. For RF and acquisition cores, the core duration must be extended by 'myrfdel' and 'daqdel', respectively, to account for gradient delays with respect to RF transmission and data acquisition, respectively.

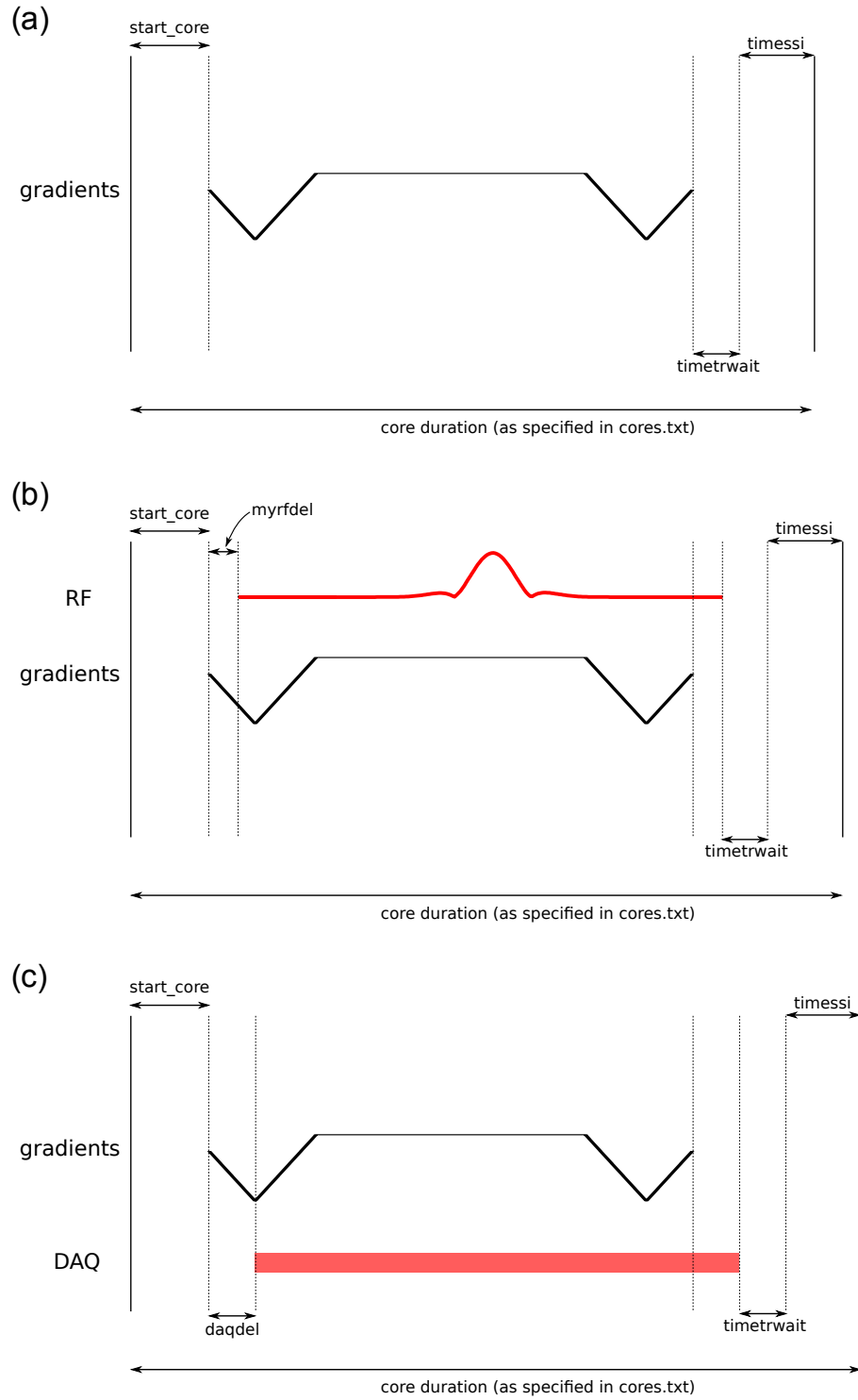


Figure 3.1: Detailed timing diagram for the three core types: (a) gradients-only, (b) RF excitation, and (c) data acquisition. The labels correspond to Control Variables (CVs) in `toppe.e`. Note that `scansim.m` uses timing values in `timing.txt` (e.g., `example/timing.txt`) to reproduce the precise sequence timing one should expect to observe on the scanner.

Chapter 4

Using `toppe.e` as an interpreter module for Pulseq files

4.1 Pulseq

An effort is currently underway to make `toppe.e` compatible with Pulseq, an open file format for compactly describing MR sequences. The Pulseq file specification, along with supporting Matlab and C++ libraries, is available at

<http://pulseq.github.io/>

Pulseq relies on vendor-dependent “interpreter modules” to load a Pulseq (.seq) file onto a particular scanner platform. `toppe.e` can serve as the interpreter module for GE scanners. Interpreters currently also exist for Siemens and Bruker scanners, enabling truly platform-independent MR pulse programming. The following publication has more information about the Pulseq platform and philosophy:

Layton K, Kroboth S, Jia F, Littin S, Yu H, Leupold J, Nielsen JF, Stöcker T, Zaitsev M. Pulseq: A rapid and hardware-independent pulse sequence prototyping framework. Magn Reson Med 2016 (to appear).

4.2 Using `toppe.e` to play .seq files

To use `toppe.e` as a GE interpreter module for Pulseq files, use the Matlab script **seq2ge.m** in the `pulseq` directory in this distribution. `seq2ge.m` takes as input a .seq file and outputs the various files needed by `toppe.e` (`cores.txt`, `scanloop.txt`, and `.wav` files). For an example, see `main.m` in the `pulseq` directory.

Appendices

Appendix A

Tools for RF and gradient waveform design

A.1 Matlab scripts included in this distribution

My own Matlab scripts for generating slice-selective RF pulses, balanced cartesian readouts, spoiler gradients, etc, are included in the `wavgen` directory in this distribution. The code is provided as-is, and is undocumented at the moment.

A.2 John Pauly's RF pulse design code (Matlab)

John Pauly has made his Shinnar-Le Roux code available for download at

<http://rsl.stanford.edu/research/software.html>

The code included in the `wavgen/tipdown` directory in this distribution uses Pauly's code to generate SLR slice-select pulses.

A.3 Brian Hargreaves' spiral gradient design code (Matlab)

Brian Hargreaves has made his spiral readout gradient design code available for download at

<http://mrsrl.stanford.edu/~brian/vdspiral/>

A.4 Generating Pulseseq files

Pulseseq provides tools for waveform and sequence creation, available on the Pulseseq web page. Alternatively, sequences can be designed, simulated, and exported in Pulseseq (.seq) format using JEMRIS, available at

<http://www.jemris.org/>