The TOPPE pulse programming environment

for GE MRI scanners

This document applies to toppe.e version 1.0.

Tested on a GE Discovery MR750 scanner running software version DV25.1_R01.

Version of this document: 1.0-2017/03/02

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 `*.mod` files . . . 2

1.2.2 `modules.txt` . . . 2

1.2.3 `scanloop.txt` . . . 2

2 Using the toppe sequence **5**

2.1 Creating .mod files . . . 5

2.2 Creating modules.txt . . . 5

2.3 Creating scanloop.txt . . . 5

2.4 Testing your files with `scansim.m` . . . 6

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 . . . 7

2.8 Checklist . . . 9

2.9 Known bugs and limitations . . . 9

3 Controlling sequence timing **11**

4 Using toppe.e as an interpreter module for Pulseq files **13**

4.1 Pulseq . . . 13

4.2 Using `toppev1.e` to play .seq files . . . 13

Appendices **14**

A Tools for RF and gradient waveform design **15**

A.1 Matlab scripts included in this distribution . . . 15

A.2 John Pauly's RF pulse design code (Matlab) . . . 15

A.3 Brian Hargreaves' spiral gradient design code (Matlab) . . . 15

A.4 Generating Pulseq files 15

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`toppev1.e`, the task of pulse programming a GE scanner becomes one of creating the various external files that define the sequence.

`toppev1.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-Siegert 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 `toppev1.e` compatible with **Pulseq**, an open file format for compactly describing MR sequences. See Chapter 4 for more information about using`toppev1.e` as a GE "interpreter module" for Pulseq files.

This is the first 'beta' release of`toppev1.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 ⋆.mod files

toppev1.e creates several unique "cores" (or modules), with each core/module associated with one .mod-file (Fig. 1.1). For example, an RF excitation module may be defined by a file called tipdown.mod 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.mod that contains readout and phase-encode gradient waveforms. Finally, a spoiler gradient can be defined in a file spoiler.mod. Each .mod file is unique up to waveform scale factors and to a rotation in the logical xy-plane, and typically only a few .mod files are needed. Note that each .mod file gives rise to a separate createseq() call in toppev1.e.

1.2.2 modules.txt

The various ⋆.mod files needed to define a scan are listed in a small text file named modules.txt, which simply contains a line for each .mod 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 modules.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.mod 0   1   0
readout.mod 0   0   1
spoiler.mod 0   0   0
```

A duration of 0 means that the minimum core duration for that .mod 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 intoppev1.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 1   0   768
Core iarf iath iagx iagy iagz slice echo view dabon rot rfph recph textra freq
1   32766   32766   0   0   32766   0   0   0   0   0   0   0   0
2   0   0   32766   32766   -32638  1   0   1   1   0   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   0
2   0   0   32766   32766   -32382  1   0   2   1   0   0   21298   0   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 × 3 cores per TR), and maxslice, maxecho, and maxview correspond to rhnslices−1, rhnecho−1, and rhnframes, respecively, intoppev1.e. **We do not recommend using the** slice=0 **and** view=0 **indeces ("slots")**, hence the slice and view indeces both start at 1 in the above example. Each row in scanloop.txt specifies which core to play

and with what RF and gradient amplitudes; where to store the data using `loaddab()` (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 μs) 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.

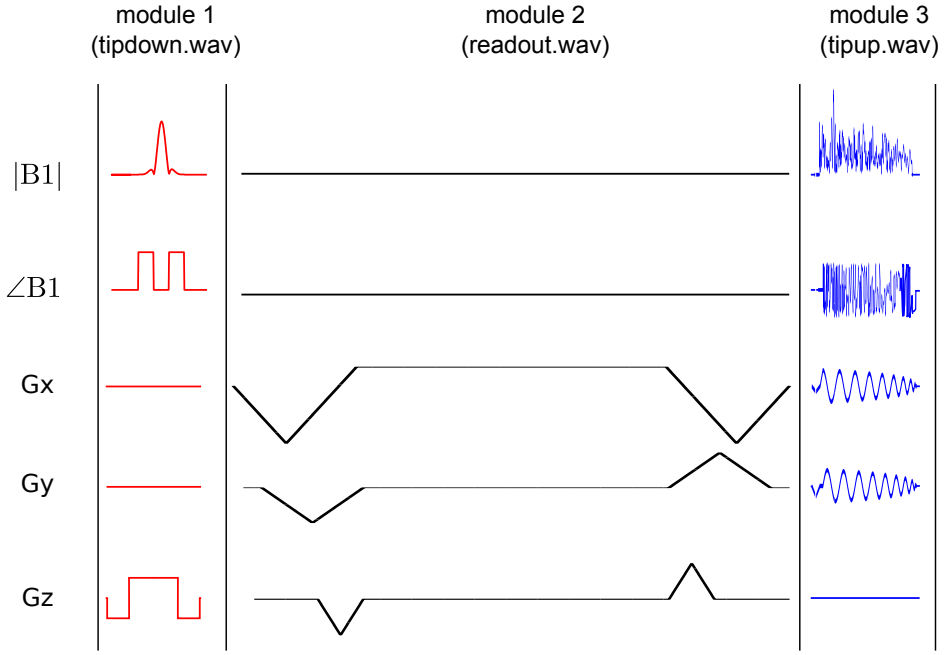
cores.txt:

Total number of unique cores
3

wavfile_name	duration(us)	hasRF?	hasDAQ?
tipdown.wav	2000	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
1	1820	32768	32768	32768	32768	0	0	0	0
2	1820	32768	32768	-32222	-29492	0	0	1	1
3	1820	32768	32768	32222	29492	0	1	1	0
1	1820	32768	32768	32768	32768	0	0	0	0
2	1820	32768	32768	-31130	-29492	0	0	2	1
3	1820	32768	32768	31130	29492	0	1	2	0
1	1820	32768	32768	32768	32768	0	0	0	0
2	1820	32768	32768	-30038	-29492	0	0	3	1
3	1820	32768	32768	30038	29492	0	1	3	0
...									



Detailed core timing:

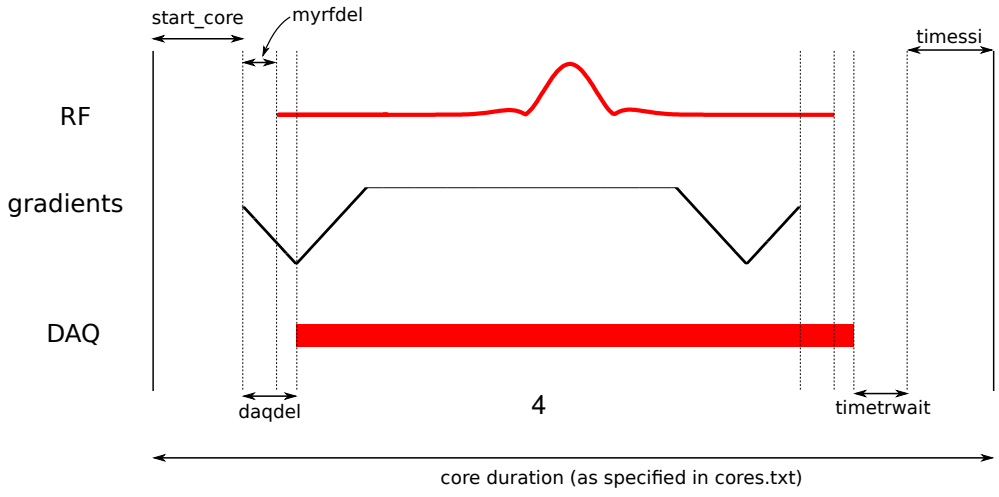


Figure 1.1: Overview of pulse sequence programming with `toppev1.e`. *The toppe/toppe.psd.o executables load the .mod files listed in* `modules.txt` *and play them out according to the instructions in* `scanloop.txt`. *Each .mod*

Chapter 2

Using the toppe sequence

2.1 Creating .mod 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 .mod 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 .mod 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 .mod 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 .mod 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 .mod files must be named **readout.mod**
  - One of the RF excitation .mod files must be named **tipdown.mod**

2.2 Creating modules.txt

`modules.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.

We have determined empirically that to avoid data corruption, `rhnslices` **should be even**. In addition, for now, avoid loading the `dabslice`$= 0$ slot with data. This means that there will be an odd number of slice slots available for useful data. On the console, prescribe `rhnslices`-1 slices, i.e., the number of loaddab slice slots filled with the desired data.

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 .mod files listed in modules.txt and display part of sequence.
% Exact timing information is loaded from 'timing.txt':
>> scansim(startseqStart,startseqEnd,d);
```

2.5 Compiling the toppe pulse sequence

The current version of `toppev1.e` has been compiled for DV25, and has been tested on a GE Discovery MR750 3T scanner. The`toppev1.e` source code is in the `psd` directory.

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

```
which psdqmake
```

With the compiler at University of Michigan, the output is

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

Prepare directory for compilation and compile:

```
prep_psd_dir
psdqmake hw
```

This will create two executables: `toppev1` and `toppev1.psd.o`.

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

The files `legacy1.rho`, `legacy1.theta`, and `legacy2.rho` must be copied to /usr/g/bin/ on the scanner. These files are included in the subfolders of `matlab/seqlib/` in the TOPPE MATLAB repository, available at https://github.com/toppeMRI/toppe/.

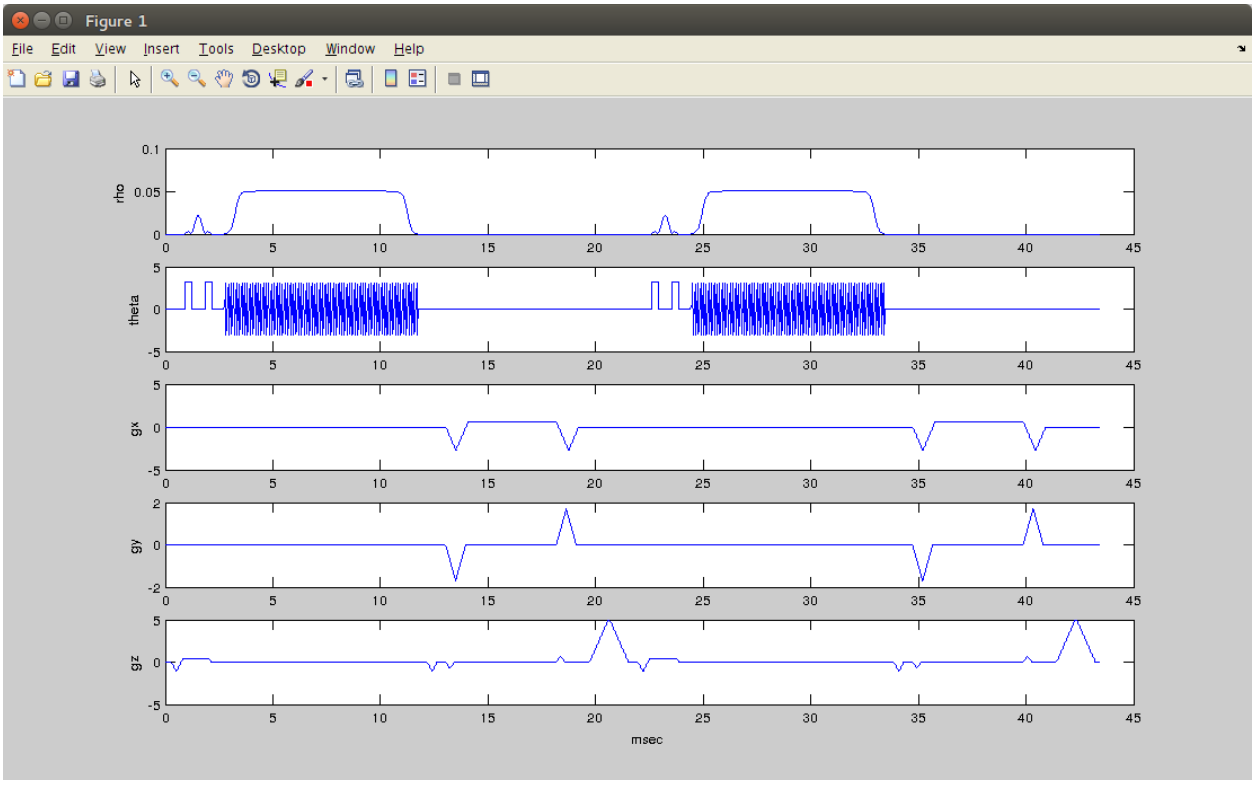


Figure 2.1: Example sequence display created with `scansim.m`. The sequence shown is a Bloch-Siegert B1 transmit mapping sequence with a 3D Cartesian readout. Like the `toppe` and `toppe.psd.o` executables on the scanner, `scansim.m` loads `modules.txt` and the .mod files listed therein, and `scanloop.txt`.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

2.7 Step-by-step scanner instructions

Follow these steps to prescribe and run the toppe sequence:

1. Copy `toppev1` and `toppev1.psd.o` to /usr/g/bin/ on the scanner host computer (console).
2. Copy `modules.txt`, `scanloop.txt`, and all .mod files to /usr/g/bin/.
3. Make sure the following required files exist:
   - Make sure one of the readout (acquisition) .mod files is named `readout.mod`.
   - Create a file `rffiles.txt` in /usr/g/bin/, containing the name of one of the RF .mod 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: 'toppev1'; click 'Accept'. (Fig. 2.2)
- Freq. FOV: 24 (Fig. 2.3)
- Set slice thickness to the design value.
- Set slice spacing to 0.
- Set number of slices to `rhnslices`-1 (see discussion of scanloop.txt above).
- TE: 1.0 (doesn't matter)
- Flip angle: 10 (doesn't matter)

6. Save and download the sequence, and run autoprescan.
7. If autoprescan fails, do a 'dummy' manual prescan: Enter into manual prescan (Fig. 2.4), adjust receive gains if needed, and exit (this will avoid auto-prescan when clicking the 'scan' button).
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 modules.txt and scanloop.txt and make sure the .mod 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.

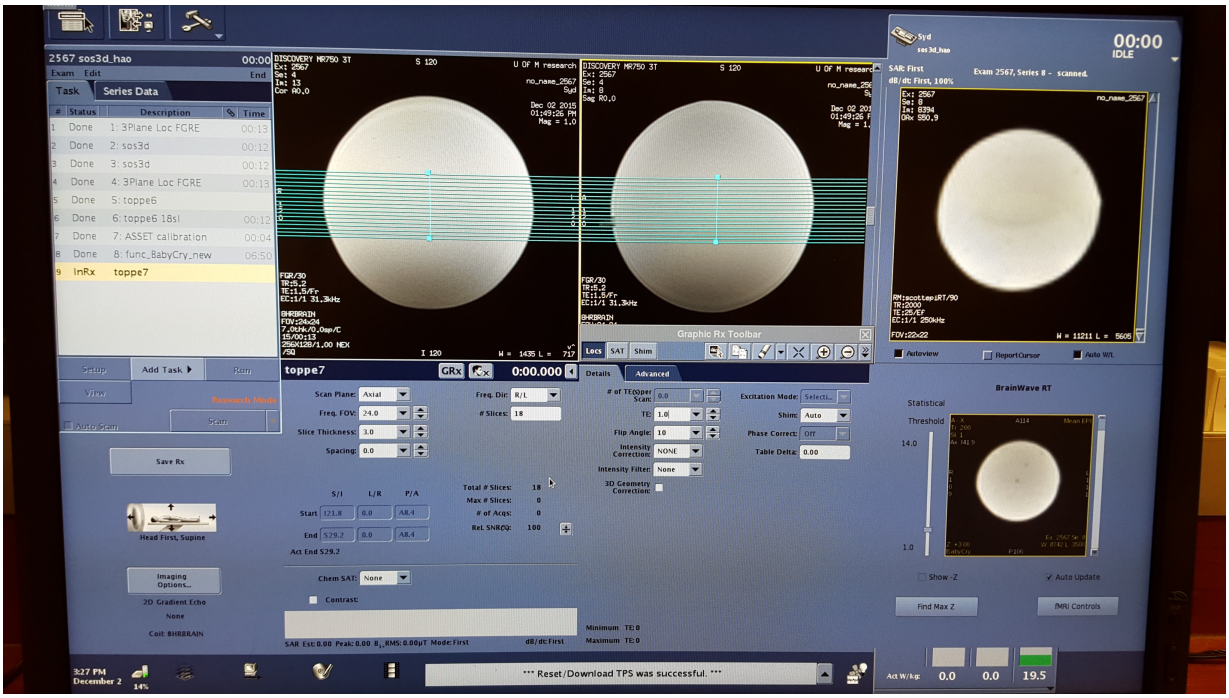


Figure 2.3: Scanner prescription, screenshot 2.

2.8 Checklist

Remember the following recommendations, which have been determined empirically:

- It seems that `rhnslices` should be even to avoid corrupt data.
- It seems safest to avoid storing data in the `dabslice=0` slot (in `loaddab()`), since data frames ("views") for this slot are often flipped (reversed) with respect to the rest of the Pfile.

2.9 Known bugs and limitations

- Data may be saved in **reverse order** (due to oeff and eeff flags), so keep an eye out for this. Inspect your raw data.
- Auto prescan may not work, hence the use of manual prescan above.
- B1 scaling across multiple RF pulses has not been verified. May need to expand `rfpulse` struct.
- `toppev1.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.
- `toppev1.e` does not currently do any checks for SAR, PNS, or gradient heating.

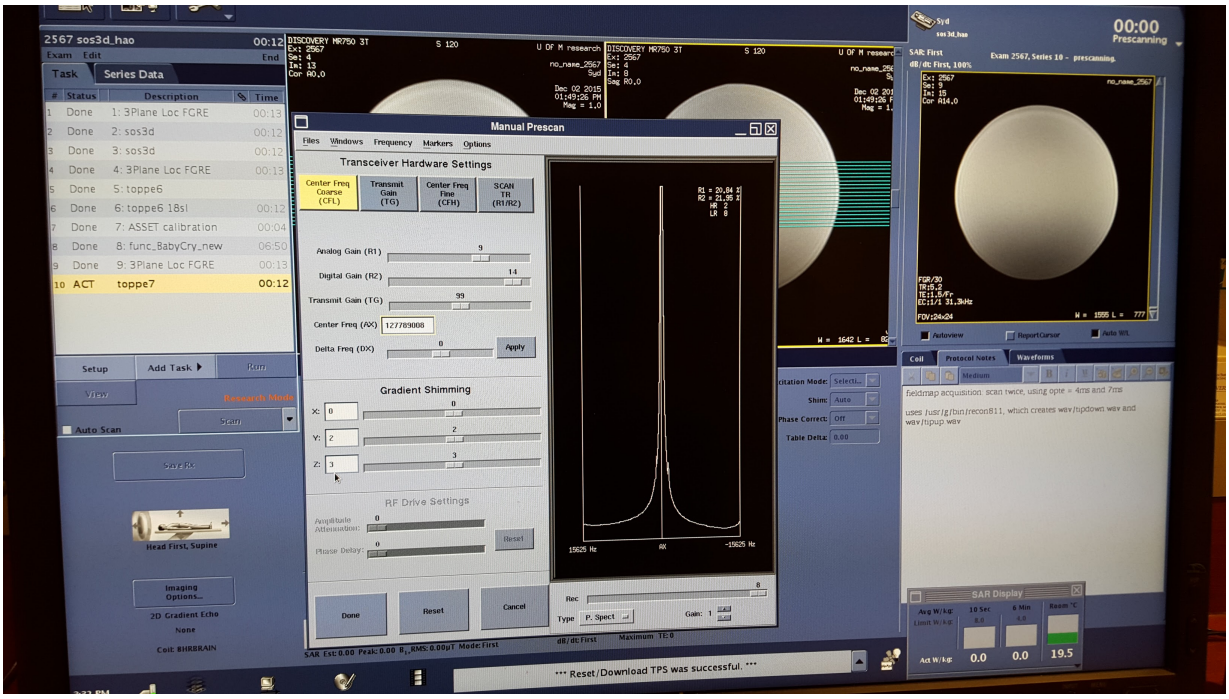


Figure 2.4: Scanner prescription, with manual prescan window.

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 'textra' 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 during a scan.

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.

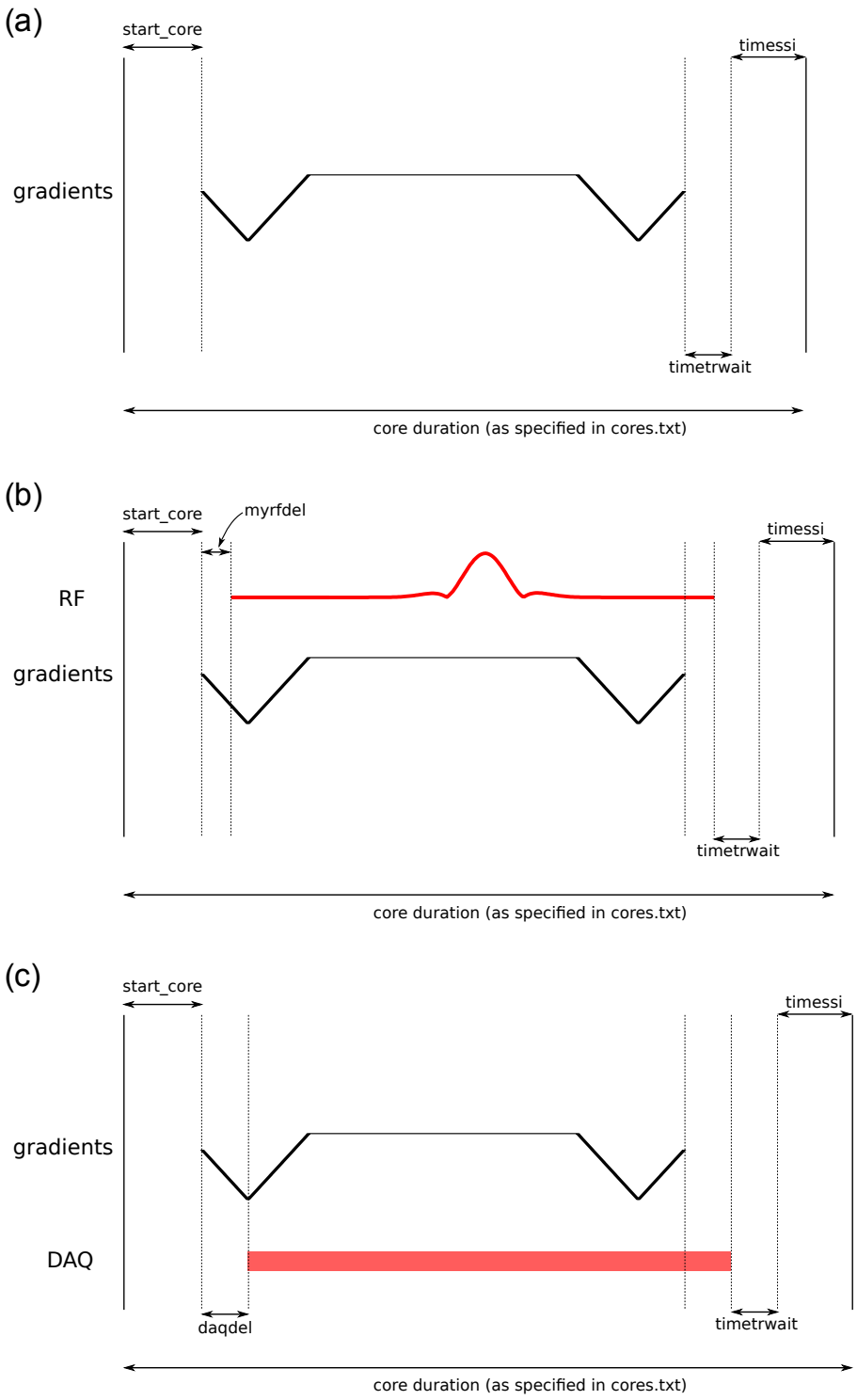


Figure 3.1: Detailed timing diagram for the three core types: (a) gradients-only, (b) RF excitation, and (c) data acquisition. The labels correspond to Control Variables (CVs) in `toppev1.e`*. Note that* `scansim.m` *uses timing values in* `timing.txt` *(e.g.,* `example/timing.txt`*) to reproduce the 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`toppev1.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. `toppev1.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 `toppev1.e` to play .seq files

To use`toppev1.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`toppev1.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 Pulseq files

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

http://www.jemris.org/