
The TOPPE pulse programming environment for GE MRI scanners

This document applies to version 1.0 of the pulse sequence (toppev1.e).
Tested on a GE Discovery MR750 scanner running software version DV25.1_R01.

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

Jon-Fredrik Nielsen, Ph.D.

jfnielse@umich.edu

This pdf is available from the TOPPE website: <https://toppemri.github.io/>.

This document is open-source. Latex code can be found in the following repository:
<https://github.com/toppeMRI/UserGuide>.

Alternatively, you can get a copy of the repository by typing the following in a Linux console:
`git clone https://github.com/toppemri/UserGuide.`

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
1.2.4 Legacy files	3
2 Using the toppe sequence	4
2.1 Creating .mod files	4
2.2 Creating modules.txt	4
2.3 Creating scanloop.txt	5
2.4 Pre-viewing your sequence with playseq.m	5
2.5 Compiling the toppe pulse sequence	5
2.6 Step-by-step scanner instructions	6
2.7 Checklist	8
2.8 Known bugs and limitations	8
3 Controlling sequence timing	10
4 Using toppe.e as an interpreter module for Pulseseq files	12
4.1 Pulseseq	12
4.2 Using toppev1.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 Pulseseq 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 `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-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 `toppev1.e` compatible with **Pulseseq**, 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 Pulseseq files.

This is the second ‘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 into toppev1.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 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 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 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, into toppev1.e. **We do not recommend using the slice=0 and view=0 indices (“slots”)**, hence the slice and view indices 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 `loadlab()` (i.e., which slice/e-cho/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.

1.2.4 Legacy files

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/matlab/>.

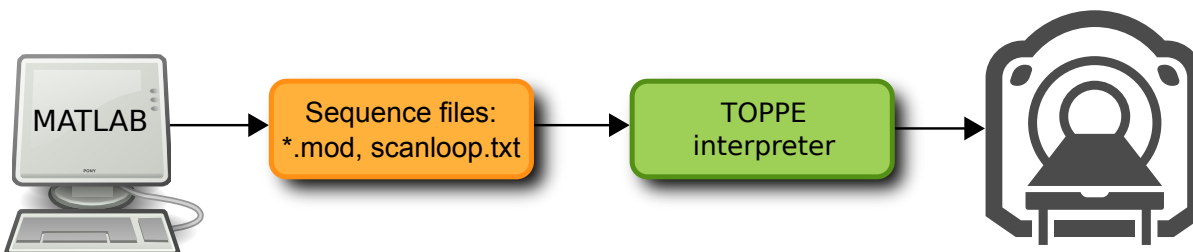


Figure 1.1: Overview of pulse sequence programming with `toppev1.e`. The TOPPE sequence files (orange) specify all details of the MR acquisition, such as RF phase cycling, gradient waveforms, and timing information. These files are created in MATLAB using the TOPPE MATLAB toolbox (<https://github.com/toppeMRI/toppe/>). The TOPPE sequence files are passed on to the TOPPE interpreter (green), a binary executable that executes the sequence on the scanner. Source code (EPIC) for the interpreter is available for download on the GE Collaboration Community site (<https://collaborate.mr.gehealthcare.com/>). The interpreter only needs to be compiled and installed once per scanner software upgrade. The interpreter, i.e., the `toppev1.e/toppev1.psd.o` executables, loads the `.mod` files listed in `modules.txt`, and plays them out according to the instructions in `scanloop.txt`. Each `.mod` file contains the RF and gradient waveform shapes for one module. A module 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 .mod files

The TOPPE MATLAB repository (<https://github.com/toppeMRI/matlab/>) includes the Matlab script **mat2mod.m** that writes rho, theta, gx, gy, and gz waveforms for a module to a .mod file. Important notes and caveats:

- All waveforms in a module must have the same length, i.e., they must be padded with zeroes as needed.
- Even if the module is not an RF excitation module, 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 module is a readout module, data will be acquired every 4 μ 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 module) 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 examples folder in the TOPPE MATLAB repository (<https://github.com/toppeMRI/matlab/>) contains several examples of how scanloop.txt can be created. Specifically, this is done in each example with the script **writeloop.m**.

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.

2.4 Pre-viewing your sequence with playseq.m

We recommend displaying your sequence in Matlab using playseq.m before attempting to play it on the scanner, to verify that the correct modules are played out in the intended order. playseq.m attempts to reproduce the exact module 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 topp pulse sequence

The current version of toppv1.e has been compiled for DV25, and has been tested on a GE Discovery MR750 3T scanner. The toppv1.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: toppv1 and toppv1.psd.o.

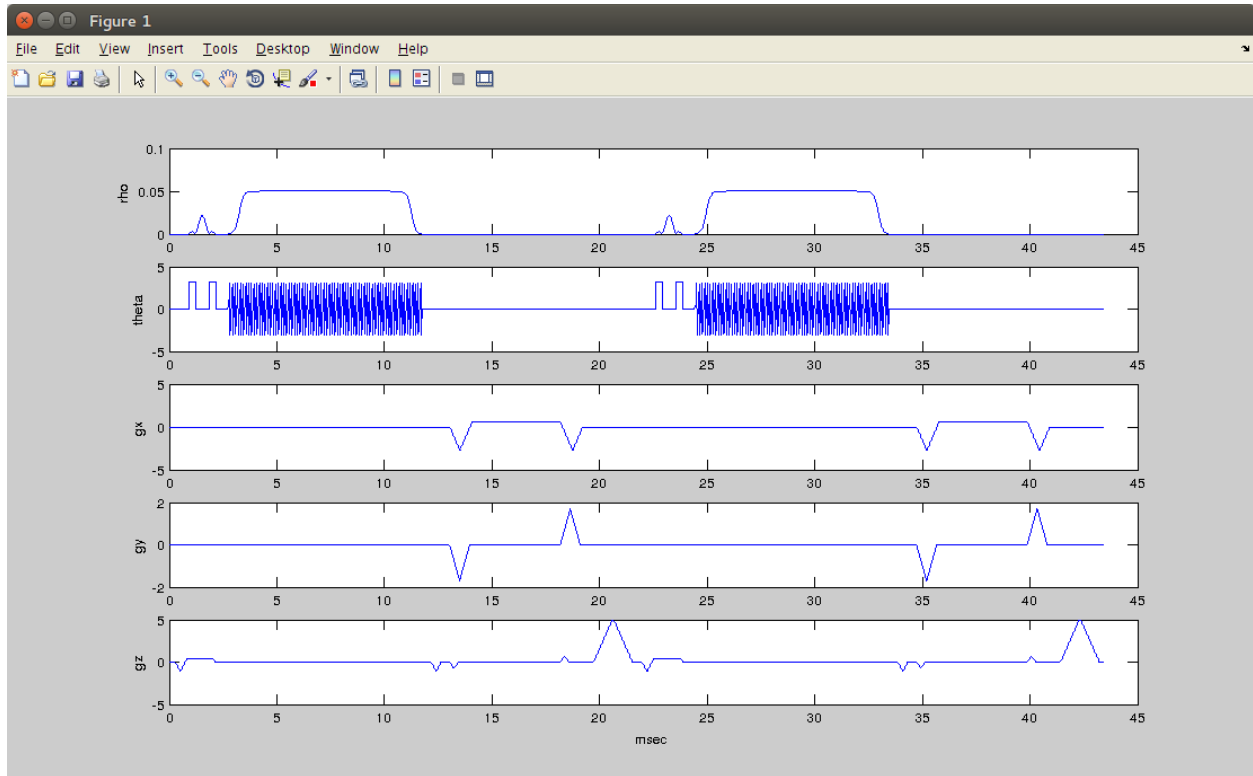


Figure 2.1: Example sequence display created with *scansim.m*. The sequence shown is a Bloch-Siebert B1 transmit mapping sequence with a 3D Cartesian readout. Like the *toppe* and *toppe.psd.o* executables on the scanner, *scansim.m* loads *modules.txt* and the *.mod* files listed therein, and *scanloop.txt*.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

2.6 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). This is only done once per scanner software upgrade.
2. Copy *modules.txt*, *scanloop.txt*, and all *.mod* files to */usr/g/bin/*.
3. Required files:
 - Make sure one of the readout (acquisition) *.mod* files is named *readout.mod*.
 - Make sure one of the RF excitation *.mod* files is named *tipdown.mod*.
4. Prescribe the *toppe* sequence:
 - Select Axial 2D pulse sequence; Family: 'Gradient Echo'; pulse: 'GRE'; PSD Name: 'toppev1'; click 'Accept'. (Fig. 2.2)
 - Set slice thickness to the design value.

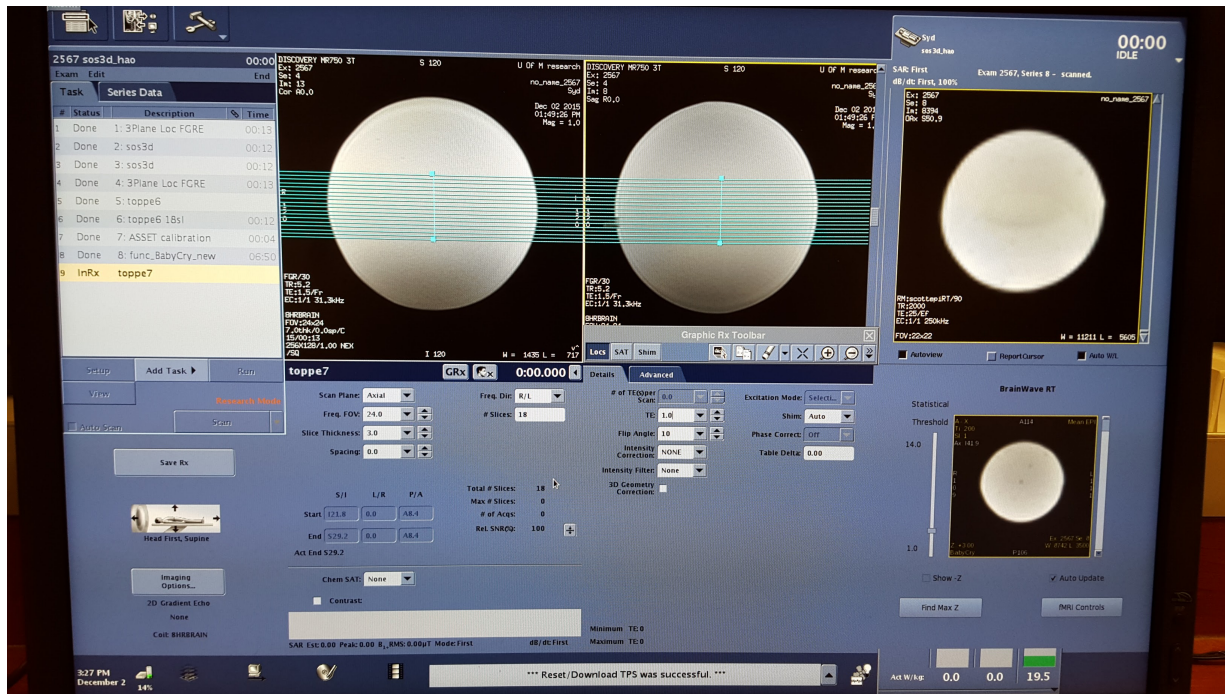


Figure 2.3: Scanner prescription, screenshot 2.

2.7 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 `loadtab()`), since data frames (“views”) for this slot are often flipped (reversed) with respect to the rest of the Pfile.

2.8 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.

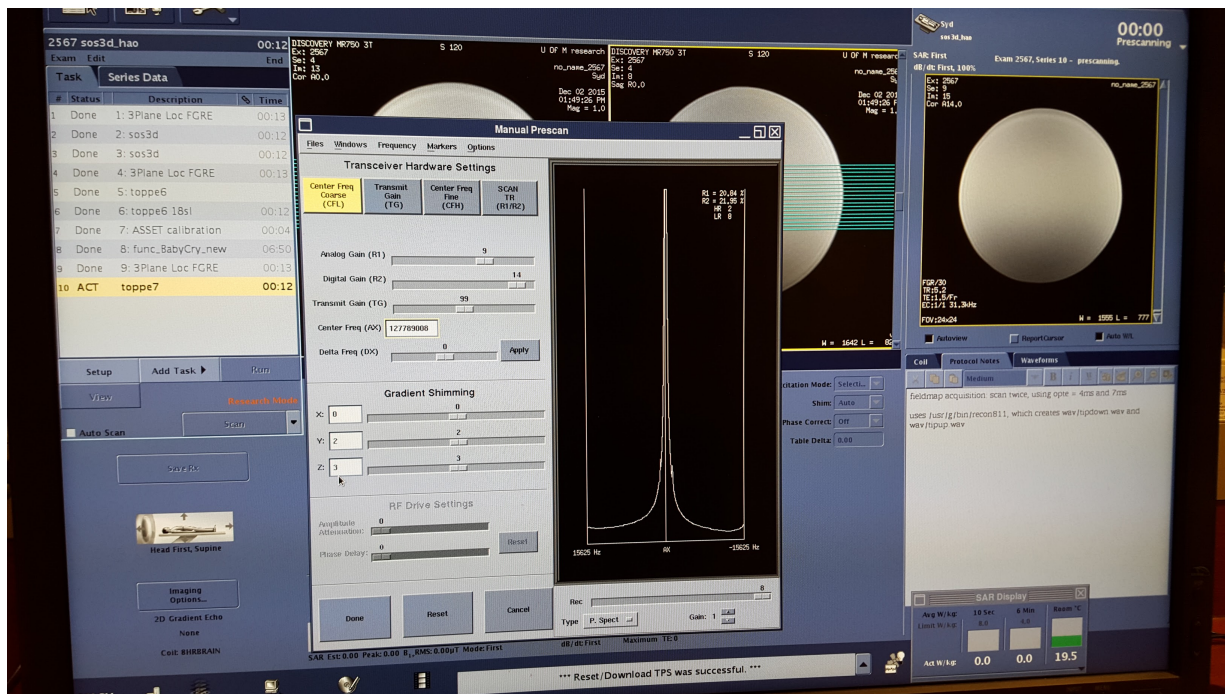


Figure 2.4: Scanner prescription, with manual prescan window.

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 '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 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.

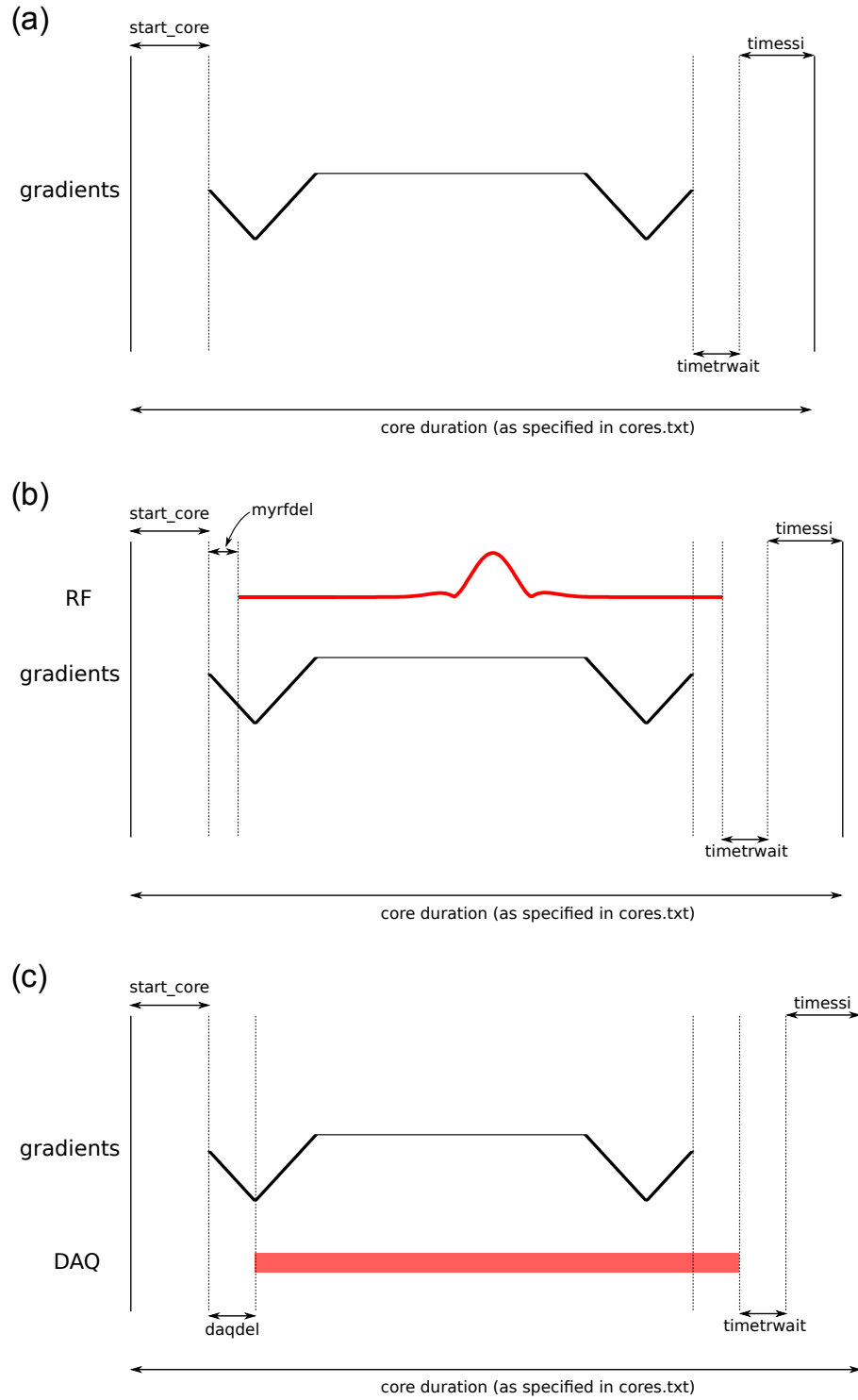


Figure 3.1: Detailed timing diagram for the three core types: (a) gradients-only, (b) RF excitation, and (c) data acquisition. The labels correspond to Control Variables (CVs) `intoppev1.e`. Note that `scansim.m` uses timing values in `timing.txt` (e.g., `example/timing.txt`) to reproduce the precise sequence timing one should expect to observe on the scanner.

Chapter 4

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

4.1 Pulseseq

An effort is currently underway to make `toppev1.e` compatible with Pulseseq, an open file format for compactly describing MR sequences. The Pulseseq file specification, along with supporting Matlab and C++ libraries, is available at

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

Pulseseq relies on vendor-dependent “interpreter modules” to load a Pulseseq (.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 Pulseseq platform and philosophy:

Layton K, Kroboth S, Jia F, Littin S, Yu H, Leupold J, Nielsen JF, Stöcker T, Zaitsev M. Pulseseq: 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 Pulseseq files, use the Matlab script **seq2ge.m** in the `pulseseq` 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 `pulseseq` 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/>