The TOPPE[1] pulse programming environment

for GE MRI scanners

This document applies to version 2b of the pulse sequence (`toppev2b.e`).

Tested on a GE Discovery MR750 scanner running software version DV26.

`Version of this document: 2b-2019/01/14`

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

This document is open-source. Latex code can be found in the TOPPE repository:
https://github.com/toppeMRI/toppe.
Alternatively, you can get a copy of the repository by typing the following in a Linux console:
`git@github.com:toppeMRI/toppe.git`

Other resources:
Wiki: https://github.com/toppeMRI/toppemri.github.io/wiki
Discussion forum: https://github.com/orgs/toppeMRI/teams/discussion-forum

Jon-Fredrik Nielsen, Ph.D.
jfnielse@umich.edu

[1] "**T**he **E**nd **O**f **P**ulse **P**rogramming", rearranged; pronounced "toppee"

Contents

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" pulse sequence for GE scanners, which allows the entire sequence to be specified with a set of 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 `toppev2b.e`, the task of pulse programming a GE scanner becomes one of creating the various files that define the sequence.

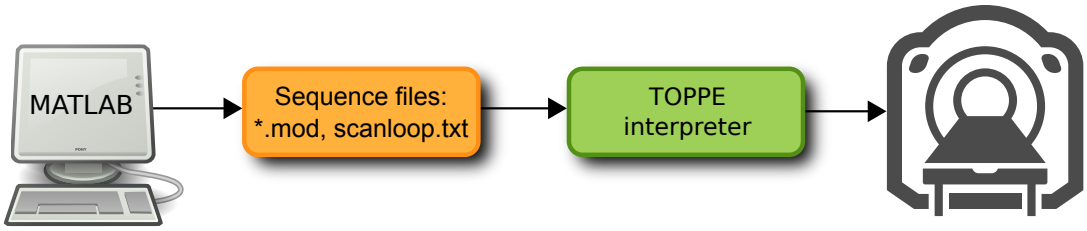

Figure 1.1: Overview of pulse sequence programming with `toppev2b.e`. *The TOPPE sequence files (orange) specify all details of the MR acquisition, such as RF phase cycling, gradient waveforms, and timing information. These files are created in MATLAB using the TOPPE MATLAB toolbox (https://github.com/toppeMRI/matlab/). The TOPPE sequence files are passed on to the TOPPE interpreter (green), a binary executable that executes the sequence on the scanner. The interpreter only needs to be compiled and installed once per scanner software upgrade. 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.*

`toppev2b.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 `toppev2b.e` compatible with **Pulseq**, an open file format for compactly describing MR sequences. See Chapter 4 for more information about using `toppev2b.e` as a GE "interpreter module"

for Pulseq files.

Your feedback is most welcome.

1.2 Required files

In addition to the `toppe` and `toppe.psd.o` executables, the following files are needed to run a particular scan (see Fig. 1.2):

1.2.1 ⋆.mod files

`toppev2b.e` creates several unique "cores" (or modules), with each core/module associated with one .mod file (Fig. 1.2). 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 `toppev2b.e`. Also, each .mod file can contain multiple waveform shapes, that can be selected dynamically (column 16 in `scanloop.txt`).

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 in `toppev2b.e`. Each line in scanloop.txt must contain the following tab-separated values:

column #	entry	units/type
1	module number	positive integer, starting at 1
2	rf waveform (rho) amplitude	signed even short int16 (-32766 to +32766)
3	phase waveform (theta) amplitude	signed even short int16 (-32766 to +32766)
4	Gx waveform amplitude	signed even short int16 (-32766 to +32766)
5	Gy waveform amplitude	signed even short int16 (-32766 to +32766)
6	Gz waveform amplitude	signed even short int16 (-32766 to +32766)
7	data storage 'slice' index	positive integer, starting at 1
8	data storage 'echo' index	positive integer, starting at 0
9	data storage 'view' index	positive integer, starting at 1
10	turn on/off data acquisition	one of two integers: 0 (off) or 1 (on)
11	in-plane (x-y) rotation angle	signed even short int16: -32766 (-pi rad) to +32766 (+pi rad)
12	RF transmit phase	signed even short int16: -32766 (-pi rad) to +32766 (+pi rad)
13	receive phase	signed even short int16: -32766 (-pi rad) to +32766 (+pi rad)
14	time added to end of module	positive integer, in microseconds
15	RF transmit frequency offset	integer, in Hz
16	waveform number	positive integer, starting at 1

Example: 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   1
2   0   0   32766   32766   −32638  1   0   1   1   0   0   0   0   1
3   0   0   0   0   32766   0   0   0   0   0   0   0   0   1
1   32766   32766   0   0   32766   0   0   0   0   0   21298   0   0   0   1
2   0   0   32766   32766   −32382  1   0   2   1   0   0   21298   0   0   1
3   0   0   0   0   32766   0   0   0   0   0   0   0   0   1
...
```

where nt is the total number of `startseq()` calls (256 phase-encodes $\times$ 3 cores per TR), and `maxslice`, `maxecho`, and `maxview` correspond to the maximum values of slice, echo, and view, respectively. Values are tab-separated. For long scans, `scanloop.txt` can contain many tens of thousands of lines.

1.3 Source code and other resources

TOPPE is open-source and is available at the following sites:

1.3.1 https://github.com/toppeMRI/toppe

Matlab scripts for creating and viewing TOPPE files. Also contains complete sequence examples. To access the code, you can either browse the website, or copy the entire repository to local disk as follows:

```
git clone git@github.com:toppeMRI/toppe.git
```

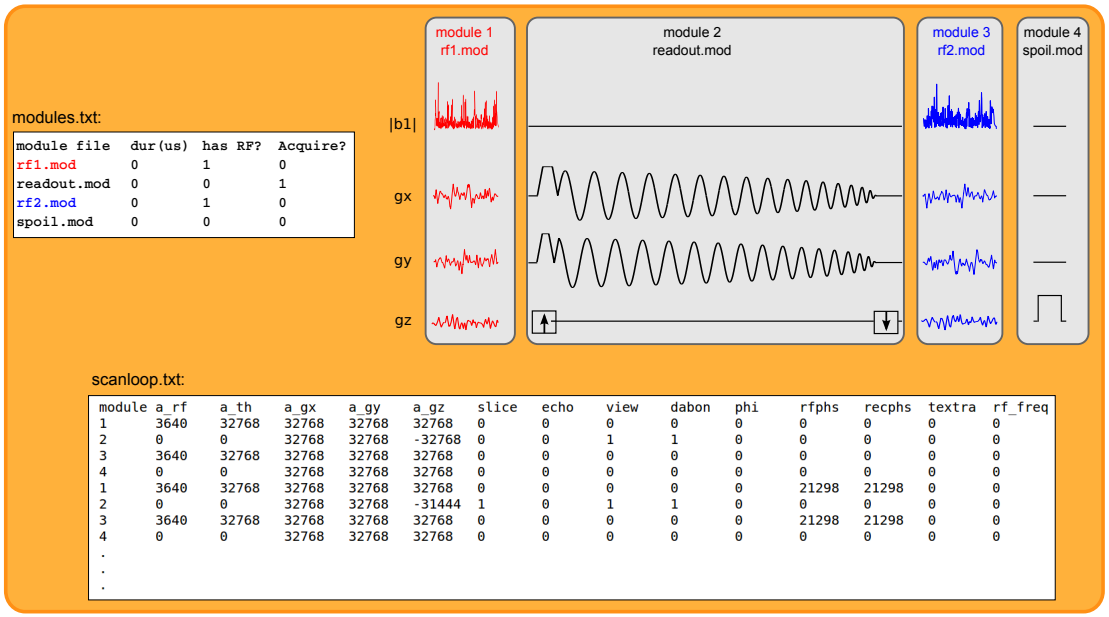

Figure 1.2: Overview of the TOPPE file structure. A TOPPE MR sequence is comprised of a (usually) small number of unique modules that are listed in the file `'modules.txt'` (a). Each module contains a set of arbitrary gradient waveforms and a complex RF waveform (if applicable), and is contained within a file with extension `'.mod'`. These modules are unique up to waveform scale factors, and to a rotation in the logical transverse (k_{xy}) plane. (b) Example of four different modules within an MR sequence: two 3D RF excitations, one spiral-in readout, and one gradient spoiler. (c) The file `'scanloop.txt'` lists the order in which to execute the modules, and specifies all other dynamic sequence information. Each row specifies the module number as listed in `modules.txt`; RF and gradient waveform amplitudes; where to store the acquired data in memory ('slice/echo/view' indices). in-plane rotation angle ('phi'); transmit and receive phase; a parameter 'textra' by which to extend module duration (for dynamic TR changes); and RF transmit frequency (for slice offsets). For detailing sequence timing information, see Ch. 3.

1.3.2 Binary executable (driver/interpreter

See https://toppemri.github.io/ for more info.

1.3.3 https://toppemri.github.io/

The TOPPE website.

Chapter 2

Using the toppe sequence

2.1 Getting started: running an example sequence

The MATLAB code repository contains several complete pulse sequence examples, such as 3D spoiled gradient-echo (SPGR) and stack-of-spirals echo-shifted dynamic imaging (PRESTO fMRI). We recommend starting by running of these sequences. See the TOPPE website (https://toppemri.github.io/) for details.

2.2 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.
- In each module (.mod file), all gradient waveforms must begin and end at 0.
- 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.
- If the module is a readout module, data will be acquired every 4 μs 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**

For some tips on waveform design, see Appendix A.

2.3 Creating modules.txt

`modules.txt` can simply be created by hand, as specified above. Columns are tab-separated.

2.4 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, the number of slices should be even. In addition, avoid loading the `slice`$= 0$ slot with data.

2.5 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`. For examples of how to use `playseq.m`, see the readme file in the `examples` folder in the Matlab repository (https://github.com/toppeMRI/toppe).

2.6 Compiling the toppe pulse sequence

The current version of `toppev2b.e` has been compiled for DV26, and has been tested on a GE Discovery MR750 3T scanner.

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

```
which psdqmake
```

Then prepare directory for compilation and compile:

```
prep_psd_dir
psdqmake hw
```

This will create two executables: `toppev2b` and `toppev2b.psd.o`.

2.7 Step-by-step scanner instructions

Follow these steps to prescribe and run the toppe sequence:

1. Copy `toppev2b` and `toppev2b.psd.o` to /usr/g/bin/ on the scanner host computer (console). This only needs to be done once per scanner software upgrade.
2. Copy `modules.txt`, `scanloop.txt`, and all .mod files to /usr/g/bin/.

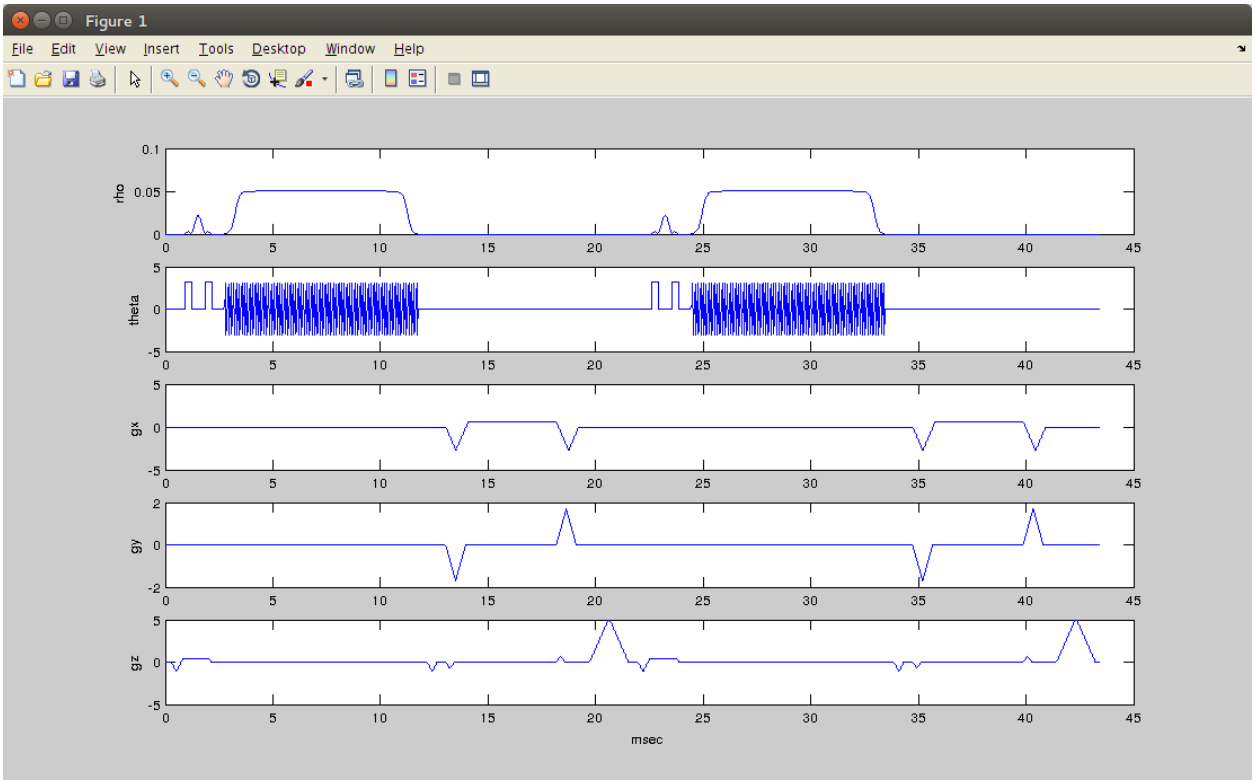

Figure 2.1: Example sequence display created with `playseq.m`. The sequence shown is a Bloch-Siegert B1 transmit mapping sequence with a 3D Cartesian readout. Like the `toppe` and `toppe.psd.o` executables on the scanner, `playseq.m` loads `modules.txt` and the .mod files listed therein, and `scanloop.txt`.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

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: 'toppev2b'; click 'Accept'. (Fig. 2.2)
    - Prescribe a number of slices that is larger than the maximum 'slice' value in scanloop.txt.
    - Other settings do not matter but must be specified. Suggested values are: Slice thickness 3, slice spacing 0.
5. Save and download the sequence, and run autoprescan.
6. Click scan button. This will create a Pfile in /usr/g/mrraw/.
7. To run a different TOPPE scan, simply overwrite modules.txt and scanloop.txt and make sure the new .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.

2.8 Checklist

- One of the RF files must be named `tipdown.mod`
- One of the readout files must be named `readout.mod`
- Make sure your .mod files comply with the requirements listed in Section 2.2.

In addition, remember the following recommendations, which have been determined empirically:

- It seems that the number of 'slices' should be even to avoid corrupt data.
- It seems safest to avoid storing data in the `slice`=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.

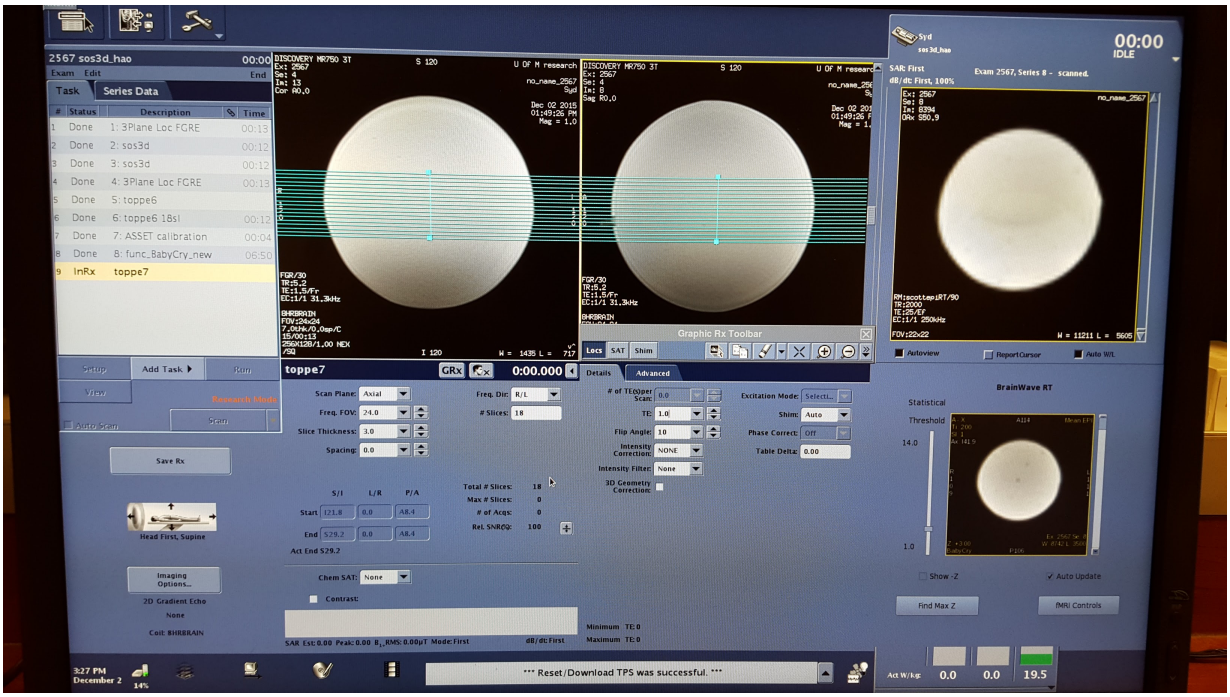

Figure 2.3: Scanner prescription, screenshot 2.

- We have observed empirically that data for the last slice is sometimes flipped.
- B1 scaling across multiple RF pulses has not been verified. May need to expand `rfpulse` struct.
- `toppev2b.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.
- `toppev2b.e` does not currently do any checks for SAR, PNS, or gradient heating.

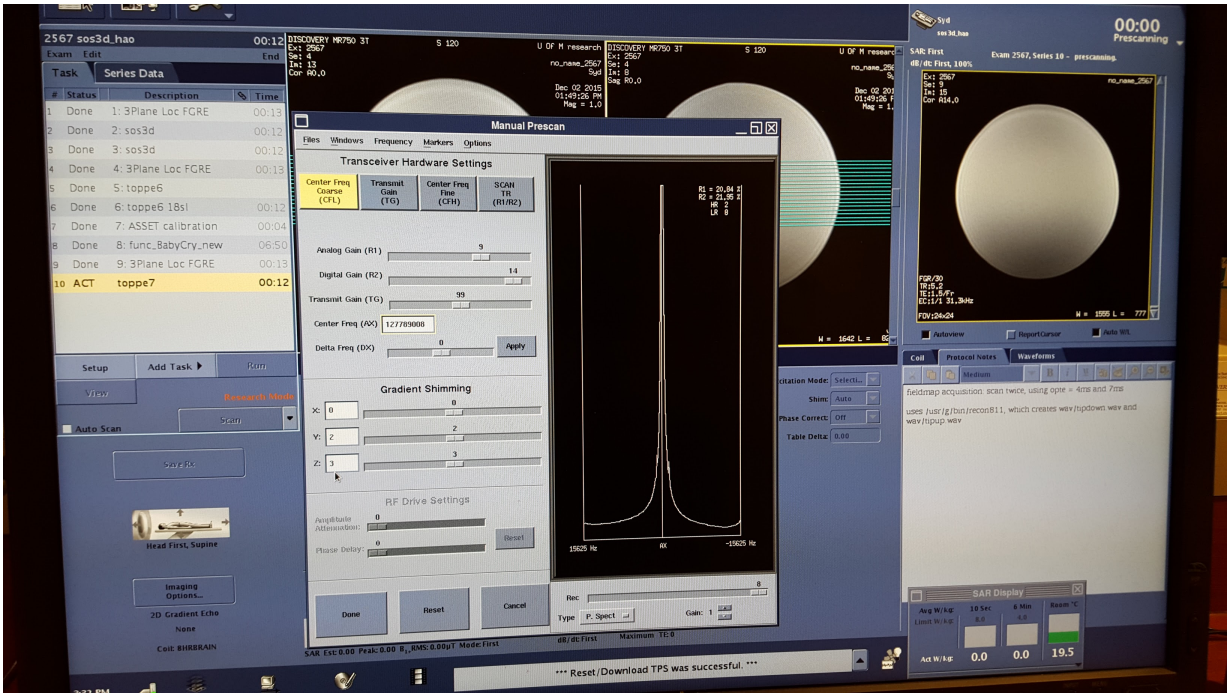

Figure 2.4: Scanner prescription, with manual prescan window.

Chapter 3

Controlling sequence timing

The default module duration is set to the value specified in `modules.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 module duration exceeds the prescribed duration in `modules.txt`, the minimum module duration is used (without warning). It is therefore perfectly fine to set the module duration in `modules.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 module types: gradients-only, RF excitation, and data acquisition. For gradients-only modules, the minimum module 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 modules, the module duration must be extended by 'myrfdel' and 'daqdel', respectively, to account for gradient delays with respect to RF transmission and data acquisition, respectively.

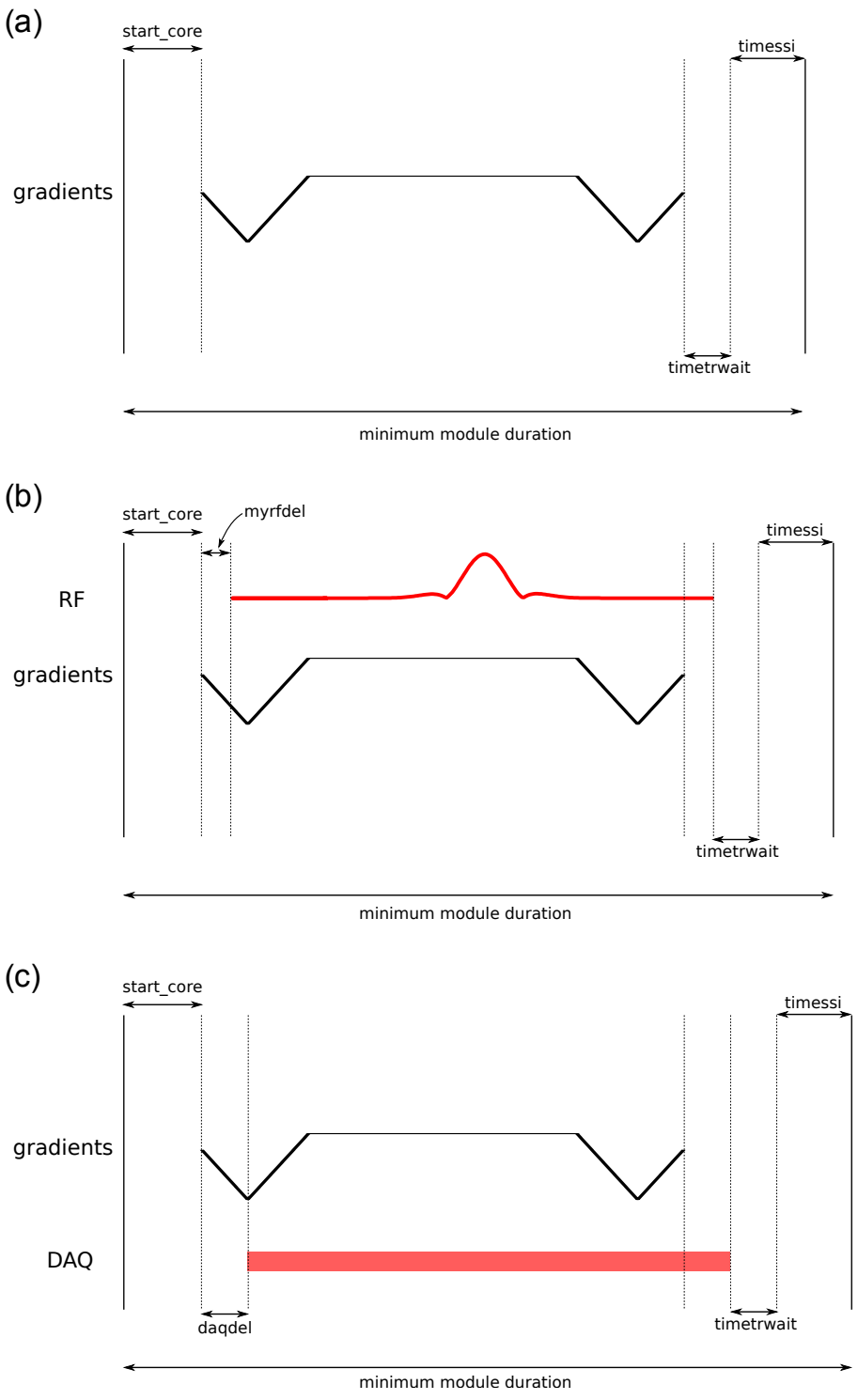

Figure 3.1: Detailed timing diagram for the three module types: (a) gradients-only, (b) RF excitation, and (c) data acquisition. The labels correspond to Control Variables (CVs) in `toppev2b.e`. *Note that* `playseq.m` *uses timing values in* `timing.txt` *(see sequence examples in the Matlab repository) 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 `toppev2b.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. `toppev2b.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: http://onlinelibrary.wiley.com/doi/10.1002/mrm.26235/abstract.

4.2 Using `toppev2b.e` to play .seq files

To use`toppev2b.e` as a GE interpreter module for Pulseq files, use the Matlab script **seq2ge.m** in the `pulseq` directory in the beta distribution (v0.9). `seq2ge.m` takes as input a .seq file and outputs the various files needed by`toppev2b.e` (`modules.txt`, `scanloop.txt`, and .mod 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/