The TOPPE driver/interpreter for GE scanners (`toppev2.e`)

This document applies to version 2.0 of the pulse sequence (`toppev2.e`).

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

`Version of this document: 2.0-2018/08/14`

https://toppemri.github.io/

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

Contents

Chapter 1

Source files

1.1 Introduction

This white paper provides a high-level description of the EPIC source code for the TOPPE driver of the TOPPE file format, suitable for distribution to the public (i.e., no EPIC-specific information is revealed here, as such information is reserved for research sites with access to the EPIC compiler). The main source files for the TOPPE driver (for GE scanners) are:

`jfn_rf_files2.c`, `jfn_rf_files2.h`	Defines a C structure ('rfstruct') containing all information pertaining to one module (i.e., one `.mod` file), including header info and the waveforms themselves. Also defines associated helper functions for reading a `.mod` file, managing waveform memory, etc.
`cores.c`, `cores.h`	Defines a C struct and associated helper functions for loading information from `modules.txt`. Also contains a function for reading `scanloop.txt`.
`toppev2.e`	Main EPIC file. Loads `scanloop.txt`, `modules.txt`, and the `.mod` files (listed in `modules.txt`), using the helper functions in the above .c/.h files. Executes sequence on the scanner, by stepping through each line in `scanloop.txt` in sequential order.

1.2 `jfn_rf_files2.c` and `jfn_rf_files2.h`

These files were originally used to load RF waveforms, hence the 'rf' in the file name. However, these files are now used to load `.mod` files regardless of whether they will be used for RF excitation, data readout, or only for playing out gradient waveforms (for, e.g., gradient spoiling or diffusion-encoding). This allows for a compact code base, and for flexibility in using a `.mod` file for several purposes; for example, the gradients associated with an RF excitation module can be used as readout waveforms during a gradient trajectory mapping experiment using the Duyn method.

```
/* jfn_rf_files2.h */
```

```c
typedef struct {
   char     fname[80];          /* rf file name */
   short    ncoils;             /* number of coils/channels */
   short    npre;               /* number of points before start of rf waveform */
   short    rfres;              /* number of points in rf waveform */
   short    res;                /* number of points in gradient waveform (>= rfres) */
   short    npulses;            /* number of different waveforms in .wav file */
   float    b1max;              /* Gauss */
   float    gmax;               /* Gauss/cm (4.0 on um3t) */
   short    dataoffset;         /* total header size (including ASCII and binary parts) */
   short    nparamsint16;       /* # of int16 parameters */
   short    nparamsfloat;       /* # of float parameters */
   short    paramsint16[32];    /* int16 parameters */
   float    paramsfloat[32];    /* float parameters */
   short*** rho;
   short*** theta;
   short**  gx;
   short**  gy;
   short**  gz;
} rfstruct;

int jfn_rf_getfilename(char *fname, int index, const char* directory);
int jfn_rf_readheader(char *fname, rfstruct *rfinfo);
int jfn_rf_allocatemem(rfstruct *rfinfo);
int jfn_rf_readwaveforms(rfstruct *rfinfo, int ishard);
int jfn_rf_freemem(rfstruct *rfinfo);
int readshort(short* i, int n, FILE* fid);
```

```c
/* jfn_rf_files2.c */

#include <string.h>
#include <math.h>

#include "jfn_globaldefs.h"
#include "jfn_rf_files2.h"


/* get name of rf file from the names listed in 'rffiles.txt' */
int jfn_rf_getfilename(char *fname, int index, const char* directory) {
    char tmpfname[80];
    int i;
    FILE *fid;

    sprintf(tmpfname, "%srffiles.txt", directory);
    if ((fid = fopen (tmpfname, "r")) == NULL)
        return(JFN_FAILURE);

    for (i = 0; i<index; i++)
        fscanf(fid, "%s\n", tmpfname);

    fscanf(fid, "%s\n", fname);

    fclose (fid);
    return JFN_SUCCESS;
}
```

```
short orderbyte(short in)
{
#ifdef LITTLE_ENDIAN

  char* sw;
  char tmp;
  sw = (char*)&in;
  tmp=sw[0];
  sw[0] = sw[1];
  sw[1] = tmp;

  /*
  in = htons(in);
  */
#endif
  return in;
}

unsigned short orderbyteu(unsigned short in)
{
#ifdef LITTLE_ENDIAN
  char* sw;
  char tmp;
  sw = (char*)&in;
  tmp=sw[0];
  sw[0] = sw[1];
  sw[1] = tmp;

  /*
  in = htons(in);
  */
#endif
  return in;
}


int jfn_rf_readheader(char *fname, rfstruct *rfinfo)
{
    FILE *fid;
    short nchar;
    int i;

    fprintf(stderr,"jfn_rf_readheader(): reading %s \n", fname);

    strcpy(rfinfo->fname, fname);

    if ((fid = fopen (fname, "r")) == NULL)
        return(JFN_FAILURE);

    /* go past ascii description */
    fread(&nchar, sizeof(short), (int) 1, fid);
    nchar = orderbyte(nchar);
    fseek(fid, nchar, SEEK_CUR);

    /* read rest of header */
    readshort(&(rfinfo->ncoils),  1, fid) ;
    readshort(&(rfinfo->res),     1, fid) ;
    readshort(&(rfinfo->npulses), 1, fid) ;
```

```
    fscanf(fid, "b1max:  %f\n", &(rfinfo->b1max));
    fscanf(fid, "gmax:   %f\n", &(rfinfo->gmax));

    fprintf(stderr, "\trfinfo.ncoils = %d \n", rfinfo->ncoils);
    fprintf(stderr, "\trfinfo.res    = %d \n", rfinfo->res);
    fprintf(stderr, "\trfinfo.npulses = %d \n", rfinfo->npulses);

    readshort(&(rfinfo->nparamsint16), 1, fid);
    readshort(rfinfo->paramsint16, rfinfo->nparamsint16, fid);

    rfinfo->npre  = rfinfo->paramsint16[0];
    rfinfo->rfres = rfinfo->paramsint16[1];

    readshort(&(rfinfo->nparamsfloat), 1, fid);
    for (i = 0; i < rfinfo->nparamsfloat; i++) {
        fscanf(fid, "%f\n", &(rfinfo->paramsfloat[i]));
    }

    rfinfo->dataoffset = ftell(fid);

    fclose (fid);
    return JFN_SUCCESS;
}


int readshort(short* i, int n, FILE *fid) {

    int j;

    fread(i, sizeof(short), n, fid);
    for (j=0; j<n; j++)
        i[j] = orderbyte(i[j]);

    return JFN_SUCCESS;
}


int readushort(unsigned short *i, int n, FILE *fid) {

    int j;

    fread(i, sizeof(unsigned short), n, fid);
    for (j=0; j<n; j++)
        i[j] = orderbyteu(i[j]);

    return JFN_SUCCESS;
}


int jfn_rf_readwaveforms(rfstruct *rfinfo, int ishard)
{
    FILE *fid;
    int p,c,i;   /* p = pulse number (0 or 1), c = coil #, i = waveform point index */
    int maxiamp = 32766;

    if ((fid = fopen (rfinfo->fname, "r")) == NULL)
        return(JFN_FAILURE);
```

```
    /* go past ascii and binary header */
    fseek(fid, rfinfo->dataoffset, SEEK_SET);

    /* fprintf(stderr,"jfn_rf_readwaveforms: ncoils, res, rfres, npre, dataoffset = %d, %d, %d, %d, 

    /* read waveforms. Note that we're only using rf channel 1 in the psd itself. */
    for (p = 0; p < rfinfo->npulses; p++)  {
        for (c = 0; c < rfinfo->ncoils; c++)
            readshort(rfinfo->rho[p][c],   (int) rfinfo->res, fid) ;
        for (c = 0; c < rfinfo->ncoils; c++)
            readshort(rfinfo->theta[p][c], (int) rfinfo->res, fid) ;
        readshort(rfinfo->gx[p], (int) rfinfo->res, fid) ;
        readshort(rfinfo->gy[p], (int) rfinfo->res, fid) ;
        readshort(rfinfo->gz[p], (int) rfinfo->res, fid) ;
    }

    /* set to hard pulse if selected */
    if (ishard) {
        for (p = 0; p < rfinfo->npulses; p++)  {
            for (c = 0; c < rfinfo->ncoils; c++) {
                for (i = 0; i < rfinfo->res; i++) {
                    rfinfo->rho[p][c][i]   = (short) maxiamp;
                    rfinfo->theta[p][c][i] = (short) 0;
                }
            }
        }
    }

    /* make sure EOS bit of last point is set */
    for (p = 0; p < rfinfo->npulses; p++)  {
        for (c = 0; c < rfinfo->ncoils; c++) {
            if (!(rfinfo->rho[p][c][rfinfo->res-1] % 2))
                rfinfo->rho[p][c][rfinfo->res-1]++;
            if (!(rfinfo->theta[p][c][rfinfo->res-1] % 2))
                rfinfo->theta[p][c][rfinfo->res-1]++;
        }
        if (!(rfinfo->gx[p][rfinfo->res-1] % 2))
            rfinfo->gx[p][rfinfo->res-1]++;
        if (!(rfinfo->gy[p][rfinfo->res-1] % 2))
            rfinfo->gy[p][rfinfo->res-1]++;
        if (!(rfinfo->gz[p][rfinfo->res-1] % 2))
            rfinfo->gz[p][rfinfo->res-1]++;
    }

    fclose (fid);
    return JFN_SUCCESS;
}


/* allocate memory for waveforms */
int jfn_rf_allocatemem(rfstruct *rfinfo) {

    <this section contain EPIC-related code and has been removed>

    return JFN_SUCCESS;
}
```

```
/* free memory */
int jfn_rf_freemem(rfstruct *rfinfo) {

    <this section contain EPIC-related code and has been removed>

    return JFN_SUCCESS;
}
```

1.2.1 *.mod files

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 `toppev2.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 `toppev2.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	0	1
2	0	0	32766	32766	−32638	1	0	1	1	0	0	0	0	0	1
3	0	0	0	0	32766	0	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	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/matlab

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 https://github.com/toppemri/matlab
```

1.3.2 Binary executable (driver/interpreter

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

1.3.3 https://github.com/toppeMRI/UserGuide

Latex source files for this user guide. To access, browse the website or copy the entire repository to local disk as follows:

```
git clone https://github.com/toppemri/UserGuide
```

1.3.4 https://toppemri.github.io/

The TOPPE website.