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

1 Source files **1**

1.1 Introduction 1

1.2 `jfn_rf_files2.c` and `jfn_rf_files2.h` 1

1.3 `cores.c`, `cores.h` 6

1.4 `toppev2.e` 9

1.4.1 Loading modules.txt 9

1.4.2 Miscellaneous sequence information 10

1.4.3 Loading waveform shapes 10

1.4.4 Loading scanloop.txt 10

1.4.5 Sequence execution 11

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 */

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);
```

```
/* 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);

    /* 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) {

    int p,c;
    int npulses = rfinfo->npulses;

    rfinfo->rho   = (short***)AllocNode(sizeof(short**)*npulses);
    rfinfo->theta = (short***)AllocNode(sizeof(short**)*npulses);
```

```
    rfinfo->gx    = (short**)AllocNode(sizeof(short*)*npulses);
    rfinfo->gy    = (short**)AllocNode(sizeof(short*)*npulses);
    rfinfo->gz    = (short**)AllocNode(sizeof(short*)*npulses);

    for (p = 0; p < npulses; p++)  {
        rfinfo->rho[p]   = (short**)AllocNode(sizeof(short*)*rfinfo->ncoils);
        rfinfo->theta[p] = (short**)AllocNode(sizeof(short*)*rfinfo->ncoils);
        for (c=0; c < rfinfo->ncoils; c++) {
            rfinfo->rho[p][c]   = (short*)AllocNode(sizeof(short)*rfinfo->res);
            rfinfo->theta[p][c] = (short*)AllocNode(sizeof(short)*rfinfo->res);
        }
        rfinfo->gx[p] = (short*)AllocNode(sizeof(short)*rfinfo->res);
        rfinfo->gy[p] = (short*)AllocNode(sizeof(short)*rfinfo->res);
        rfinfo->gz[p] = (short*)AllocNode(sizeof(short)*rfinfo->res);
    }

    return JFN_SUCCESS;
}


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

    int p,c;
    int npulses = rfinfo->npulses;

    for (p = 0; p < npulses; p++)  {
        FreeNode(rfinfo->gz[p]);
        FreeNode(rfinfo->gy[p]);
        FreeNode(rfinfo->gx[p]);
        for (c=0; c < rfinfo->ncoils; c++) {
            FreeNode(rfinfo->rho[p][c]);
            FreeNode(rfinfo->theta[p][c]);
        }
        FreeNode(rfinfo->theta[p]);
        FreeNode(rfinfo->rho[p]);
    }

    FreeNode(rfinfo->gz);
    FreeNode(rfinfo->gy);
    FreeNode(rfinfo->gx);
    FreeNode(rfinfo->theta);
    FreeNode(rfinfo->rho);

    return JFN_SUCCESS;
}
```

1.3 cores.c, cores.h

```
/* cores.h */

typedef struct {
   int ncores;           /* number of modules (cores) */
   char** wavfiles;
```

```
   int* dur;     /* msec. 0 corresponds to minimum realizable duration. */
   int* hasRF;
   int* hasDAQ;
} corestruct;

int cores_getinfo(char *fname, corestruct* coredefinfo);
int cores_getloophdr(char *fname, int* loophdr);
int cores_readloop(char *fname, int* looparr);
```

```
/* cores.c */

#include "cores.h"
#include "jfn_rf_files2.h"   /* need readshort() etc */

int cores_getinfo(char *fname, corestruct* coredefinfo) {

    FILE *fid;
    short nchar,tmp;
    int i;
    char line[200];

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

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

    fgets(&line, 200, fid);   /* skip line */
    fscanf(fid, "%d\n", &(coredefinfo->ncores));
    fgets(&line, 200, fid);   /* skip line */

    coredefinfo->wavfiles = (char**)AllocNode(sizeof(char*)*(coredefinfo->ncores));
    coredefinfo->dur    = (int*)AllocNode(sizeof(int)*(coredefinfo->ncores));
    coredefinfo->hasRF  = (int*)AllocNode(sizeof(int)*(coredefinfo->ncores));
    coredefinfo->hasDAQ = (int*)AllocNode(sizeof(int)*(coredefinfo->ncores));

    for (i = 0; i < coredefinfo->ncores; i++)  {
        coredefinfo->wavfiles[i] = (char*)AllocNode(sizeof(char)*100);
        fscanf(fid, "%s\t%d\t%d\t%d\n",
            coredefinfo->wavfiles[i],
            &(coredefinfo->dur[i]),
            &(coredefinfo->hasRF[i]),
            &(coredefinfo->hasDAQ[i]));
        fprintf(stderr, "cores_getinfo(): %s\t%d\t%d\t%d\n",
            coredefinfo->wavfiles[i],
            coredefinfo->dur[i],
            coredefinfo->hasRF[i],
            coredefinfo->hasDAQ[i]);
    }

    fclose (fid);
    return JFN_SUCCESS;
}


/* free memory */
int cores_freemem(corestruct *coredefinfo) {
```

```
    int i,c;
    int ncores = coredefinfo->ncores;

    FreeNode(coredefinfo->dur);
    FreeNode(coredefinfo->hasRF);
    FreeNode(coredefinfo->hasDAQ);

    for (i = 0; i < ncores; i++)  {
        FreeNode(coredefinfo->wavfiles[i]);
    }

    FreeNode(coredefinfo->wavfiles);

    return JFN_SUCCESS;
}

int cores_getloophdr(char *fname, int* loophdr) {

    FILE *fid;
    int nt, i;
    char line[200];

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

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

    fgets(&line, 200, fid);   /* go past first line */
    fscanf(fid, "%d\t%d\t%d\t%d\n", &(loophdr[0]), &(loophdr[1]), &(loophdr[2]), &(loophdr[3]));
    fprintf(stderr, "\tcores_getloophdr: Found %d startseq() calls in loop file\n", loophdr[0]);

    fclose (fid);

    return JFN_SUCCESS;
}


/* read .loop file */
int cores_readloop(char *fname, int* looparr) {

    FILE *fid;
    int nt, i, j;
    char line[200];
    int loophdr[4];

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

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

    fgets(&line, 200, fid);   /* skip line */
    fscanf(fid, "%d\t%d\t%d\t%d\n", &(loophdr[0]), &(loophdr[1]), &(loophdr[2]), &(loophdr[3]));
    nt = loophdr[0];

    fgets(&line, 200, fid);   /* skip line */

    /* load loop array */
    for (i = 0; i < nt; i++)  {
```

```
        for (j = 0; j < NL-1; j++) {
            fscanf(fid, "%d\t", &(looparr[i*NL+j]));
        }
        fscanf(fid, "%d\n", &(looparr[i*NL+NL-1]));
    }

    for (i = nt-20; i < nt; i++)  {
        for (j = 0; j < NL-1; j++) {
            fprintf(stderr, "%d\t", looparr[i*NL+j]);
        }
        fprintf(stderr, "%d\n", looparr[i*NL+NL-1]);

    }

    fclose (fid);

    return JFN_SUCCESS;
}
```

1.4 toppev2.e

toppev2.e is written in EPIC, a proprietary C-based programming language developed by GE, and can therefore not be listed in full here [1]. Only code excerpts are presented here.

1.4.1 Loading modules.txt

Prior to scanning, toppev2.e first reads modules.txt:

```
    #define MAXCORES 20 /* determined empirically */

    corestruct coredefinfo;  /* contains list of .mod files and related info */

    int hasRF[MAXCORES];
    int hasDAQ[MAXCORES];
    int coredur[MAXCORES];

    /* get total number of unique modules/cores, and fill hasDAQ and hasRF arrays */
    cores_getinfo("modules.txt", &coredefinfo);
    ncores = coredefinfo.ncores;
    for (j=0; j<ncores; j++) {
        hasRF[j] = coredefinfo.hasRF[j];
        hasDAQ[j] = coredefinfo.hasDAQ[j];
    }
```

At this point, coredefinfo also contains a list of .mod file names.

[1] For access to toppev2.e, see https://toppemri.github.io/

1.4.2 Miscellaneous sequence information

Miscellaneous other information is also loaded early on during sequence prescription. For example, the number of samples to acquire (per readout) is loaded from `readout.mod`:

```
    rfstruct roinfo;          /* see jfn_rf_files2.c/h */
    jfn_rf_readheader("readout.mod", &roinfo);
    ndaq = roinfo.rfres;
```

1.4.3 Loading waveform shapes

`toppev2.e` then loads all waveforms for the `.mod` files contained in the `coredefinfo` struct:

```
    rfstruct* coresinfo;  /* array of rfstruct (.mod file info) for array of cores */

    coresinfo = (rfstruct *) AllocNode(ncores*sizeof(rfstruct));

    for ( j=0; j<ncores; j++ ) {

        /* read and load .mod file for core j */
        jfn_rf_readheader(coredefinfo.wavfiles[j], &coresinfo[j]);
        jfn_rf_allocatemem(&coresinfo[j]);
        jfn_rf_readwaveforms(&coresinfo[j], ishard);
```

The struct array `coresinfo` now contains all sequence waveforms, for all modules.

The waveforms are then moved to sequence hardware, using EPIC functions (not shown).

1.4.4 Loading scanloop.txt

The final task before executing the sequence is to load the scan loop description:

```
    int looparr[9000000];    /* scan loop definition, loaded from scanloop.txt */

    /* get scan loop header info */
    cores_getloophdr("scanloop.txt", loophdr);
    nstartseq = loophdr[0];      /* number of rows */
    maxslice  = loophdr[1];
    maxecho   = loophdr[2];
    maxview   = loophdr[3];

    /* fill looparr (needed in scan()) */
    cores_readloop("scanloop.txt", looparr);
```

The scan loop information is now stored in the 2D array `looparr`.

See the TOPPE user guide for a detailed description of `scanloop.txt`.

1.4.5 Sequence execution

`toppev2.e` "simply" steps through each row in `looparr`, and executes the waveforms (RF and gradient) associated with the module indexed by the first column (modules are numbered in the order of appearance in `modules.txt`). Each line corresponds to the execution of one `.mod file`.