Pulseq on GE

Jon-Fredrik Nielsen

November 16, 2023

Outline

Pulseq on GE manual

Pulseq on GE v1.0 (DRAFT)

Instructions for running Pulseq sequences on GE using the TOPPE[1] universal pulse sequence interpreter

This document applies to version 6 of the interpreter (`tv6.e`).

This version of the manual contains GE specific information and can be distributed to sites with a GE research scanner.

This document was last modified on Sun Nov 5 16:32:56 2023

LaTeX source for this manual can be found at:
https://github.com/jfnielsen/TOPPEpsdSourceCode/tree/UserGuide/v6/

An occasionally updated PDF version of this manual can be found at:
https://drive.google.com/file/d/1eToTYUtFipf6UaAohOfuroNdFi4h8cOs/view?usp=sharing

- Redacted version for the public: `https://toppemri.github.io/`
- Full version in the MR Software Sharing forum on `gecares.com`

History

Original 'TOPPE' interpreter [1]:

- Developed independently of Pulseq; tailored to GE
- Proved that cross-vendor sequence programming is possible
- Shortcomings:
  - gap between blocks ($\simeq$200-400μs)
  - Gradients had to start and end at 0 within each block
  - Different version of interpreter source code for each scanner software version
- Workarounds possible

[1] Nielsen JF, Noll DC. TOPPE: A framework for rapid prototyping of MR pulse sequences. Magnetic resonance in medicine. 2018 Jun;79(6):3128-34.

Pulseq philosophy: assemble blocks on the fly

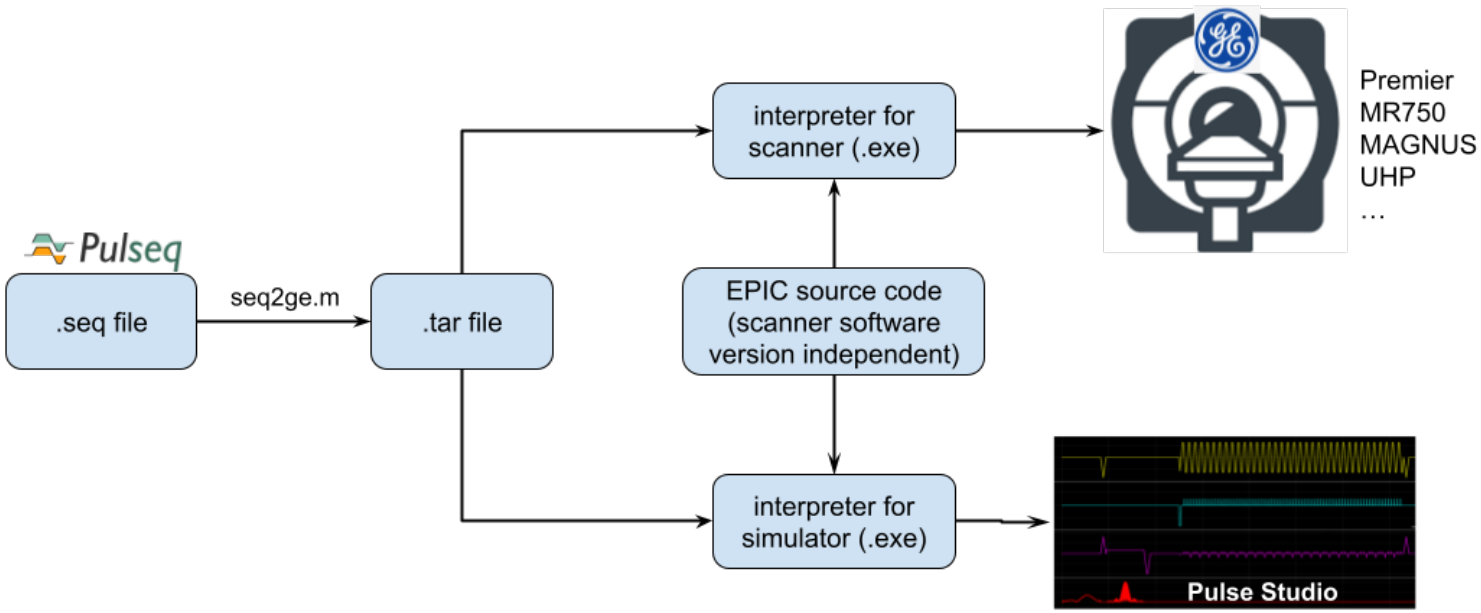
- Define blocks on the fly during sequence execution (streaming)

```
# Format of blocks:
# NUM DUR RF  GX  GY  GZ  ADC  EXT
[BLOCKS]
  1 316   1   0   0   1   0   1
  2 100   0   2   3   4   0   0
  3 223   0   0   0   0   0   0
  4  46   0   5   0   0   1   0
  5 100   0   6   7   8   0   0
  6 256   0   0   0   0   0   0
```

- Not directly compatible with EPIC

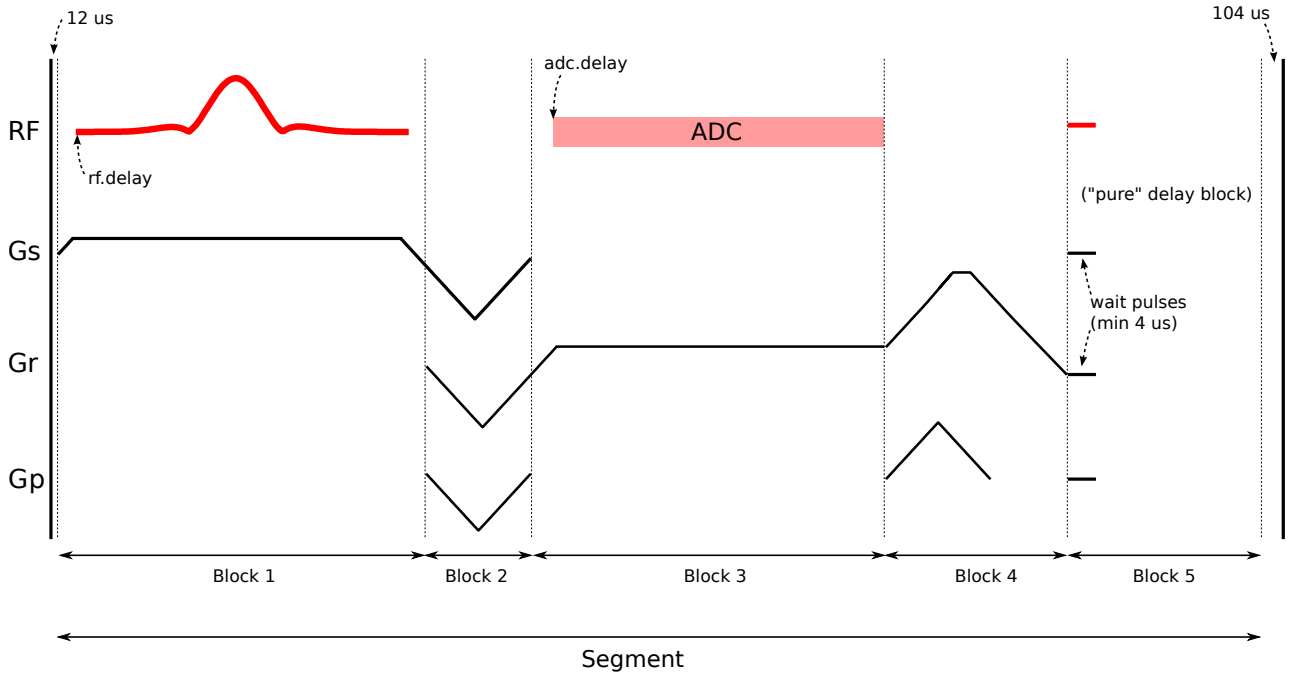
New Pulseq interpreter (2023)

- Time gaps eliminated (mostly)
- Gradient waveforms can cross block boundaries
- A single set of source files for multiple software versions
- Reuses some basic TOPPE functionality (e.g., file formats), but mostly rewritten from scratch
- Sequence represented as a collection of scaled 'segments'



Segment concept

- Segment = a fixed sub-sequence of Pulseq blocks that are always executed together
- Can consist of any number of blocks (even 100s)



Convert a .seq file to segment representation

- Step 1: Label the start of each segment[2]

```
segmentID = 1;   % any non-negative integer
seq.addBlock(rf, gz, mr.makeLabel('SET', 'LIN', segmentID));
```

- Step 2: Convert

```
>> ceq = seq2ceq('gre2d.seq');
   ceq =
                nMax: 192
       nParentBlocks: 4
        parentBlocks: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
            segments: [1x1 struct]
          nSegments: 1
                loop: [192x10 double]
```

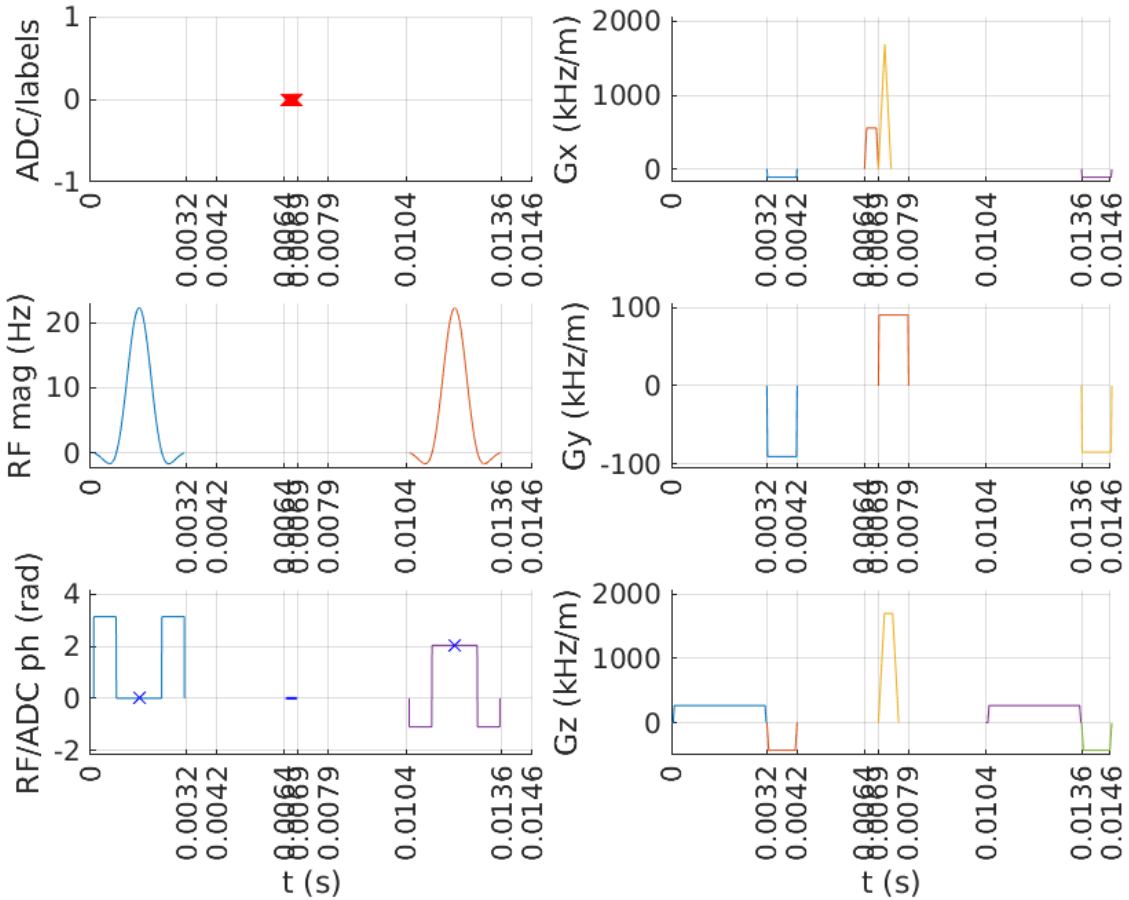
'ceq' object:

- A vendor-independent sequence description
- May find use in other contexts (e.g., other vendor platforms, simulation tools, etc)

[2] Need to do manually for now

gre2d.seq

```
>> seq.plot('blockRange', [1 8], 'showBlocks', true);
```



'Parent' blocks

Unique Pulseq blocks (up to a scaling/phase factor)

```
>> ceq.parentBlocks{1}
ans =

  struct with fields:

    blockDuration: 0.0032
               rf: [1x1 struct]
               gx: []
               gy: []
               gz: [1x1 struct]
              adc: []
            label: [1x1 struct]
              amp: [1x1 struct]
```

Segment definitions

In our example, there's only one segment:

```
>> ceq.segments

ans =

  struct with fields:

           groupID: 1
    nBlocksInGroup: 6
          blockIDs: [1 2 0 3 4 0]      % parent block IDs (0 = wait period)
```

Scan loop definition

Dynamic scan information such as waveform amplitude scaling, RF/ADC phase, etc

```
>> ceq.loop(1:8,:)

ans =

1.0e+06 *

0.0000    0.0000    0.0000         0         0         0         0    0.2667         0
0.0000
0.0000    0.0000         0         0         0   -0.1105   -0.0907   -0.4312         0
0.0000
0.0000         0         0         0         0         0         0         0         0
0.0000
0.0000    0.0000         0         0         0    0.5556         0         0         0
0.0000
0.0000    0.0000         0         0         0    1.6931    0.0907    1.7021         0
0.0000
0.0000         0         0         0         0         0         0         0         0
0.0000
0.0000    0.0000    0.0000    0.0000         0         0         0    0.2667         0
0.0000
0.0000    0.0000         0         0         0   -0.1105   -0.0850   -0.4312         0
0.0000
```

Write segment representation to file

- Convert to GE hardware units and write to file:

```
>> ceq2ge(ceq, sysGE, 'gre2d.tar');

    ceq2ge: Writing block 192/192
    preflightcheck: Checking max 10s SAR, time interval 0—10s (scan duration: 0s)
        Predicted peak 10s SAR in 150 lbs subject: 0.0 W/kg
    Sequence file gre2d.tar ready for execution on GE scanners
```

- Both steps can be combined into one:

```
>> seq2ge('gre2d.seq', sysGE, 'gre2d.tar');
```

- The .tar file can now be simulated and executed on GE scanners

Summary of file creation steps

1. Define the start of each segment in the .seq file

```
segmentID = 1;   % any non-negative integer
seq.addBlock(rf, gz, mr.makeLabel('SET', 'LIN', segmentID));
```

2. Convert to .tar file

```
>> seq2ge('grd2d.seq', sysGE, 'gre2d.tar');
```

Simulate sequence in Pulse Studio

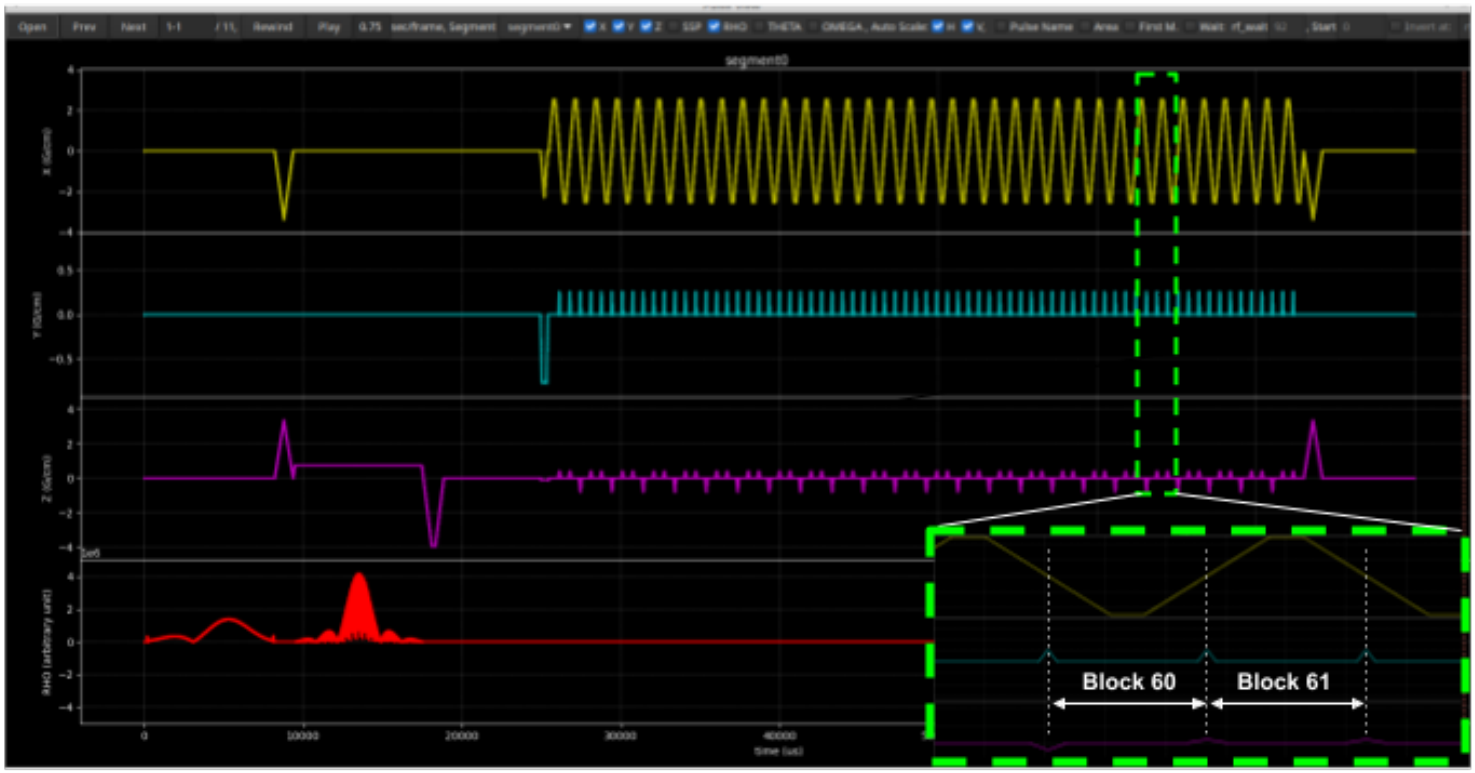
Pulse Studio

- GE's new sequence simulator, available for software version MR30.2
- Use WTools to create waveform files, then view with Pulse View
- Display whole (or parts of) sequence: waveforms, timing, SSP pulses
- GE distributes the Python source for Pulse View
  - Wants to collaborate with the community to improve Pulse View

Simulation steps

1. Compile interpreter for simulation (MR30.2)
2. Simulate scan loop in WTools (instructions in Pulseq on GE manual)
3. Load waveform files in Pulse View

Simulate sequence in Pulse Studio



Execute on scanner

1. Copy .tar file to folder or your choice (/any/path/)
2. Place toppe<ID>.entry file in /usr/g/research/pulseq/v6/ (<ID> = integer of your choice)
3. Replace the 1st line in toppe<ID>.entry with /any/path/
4. Set sequence control variables (CVs): TODO
5. Scan

Steps 1-4 are the same as for the Pulse Studio simulator

Additional instructions in Pulseq on GE manual

Pulseq on GE workflow summary

1. Define the start of each segment in the .seq file

```
segmentID = 1;   % any non-negative integer
seq.addBlock(rf, gz, mr.makeLabel('SET', 'LIN', segmentID));
```

2. Convert to .tar file

```
>> seq2ge('grd2d.seq', sysGE, 'gre2d.tar');
```

3. Simulate in Pulse Studio
   1. Compile interpreter for simulation (MR30.2)
   2. Simulate scan loop in WTools
   3. Load waveform files in Pulse View

4. Run sequence on scanner
   1. Compile interpreter for your scanner software version
   2. Prescribe and run

Dynamic imaging using 'runs' feature

- Dynamic sequences (e.g., fMRI) can be very long $\rightarrow$ large .seq files
- Redundancy: typically a short sub-sequence is simply repeated
  - But pay attention to RF phase cycling (e.g., RF spoiling)
- Pulseq exploits this on the interpreter level
  - The .seq file only needs to define the sub-sequence
  - Set number of repetitions ('runs') on UI
  - Shorter scan file $\rightarrow$ loads faster on scanner
  - New feature currently in 'develop' branch

TTL trigger output pulses

- TTL output pulses are specified in the Pulseq file as follows:

```
trig_out = mr.makeDigitalOutputPulse('ext1', 'delay', 100e-6);
seq.addBlock(gx, trig_out);   % block containing a gradient and a TTL trigger
```

- seq2ge.m captures TTL triggers

- New GE interpreter:
  - Trigger[3] starts at the requested time
  - Trigger duration defaults to 500us, use 'trigdur' control variable (CV) to change
  - Trigger channel (DABOUT) is set on UI
  - Trigger cannot be the only event inside a block (may change in future)
  - Built-in support for Skope calibration scan[4]

[3] Implemented by Dinank Gupta

[4] Thanks to Andy Derbyshire for this feature

Acknowledgments

Interpreter code and testing:

- Rolf Schulte (GE) provided C code for PNS calculation, advised on implementation of safety checks, and provided example code for dynamic waveform loading (via the fidall psd which he is the author of).
- Kang Wang (GE) consulted on various implementation and usability aspects, e.g., FOV shifting and C++ code integration.
- Ante Zhu and Afis Ajala (GE) helped with testing and debugging on the MAGNUS head gradient system.
- Sherry Huang (GE) has been central to connecting the development team with core GE resources.
- Dinank Gupta added TTL trigger out support.
- John Derbyshire contributed code for adding the Skope field camera calibration scan in MPS2.
- Matteo Cencini added son-of-recon CV and example scripts.
- Dirk Poot added error handling code to an early version (v2).
- Alexei Ouriadov added broadband support in an early version (v2)

MATLAB toolboxes:

- Amos Cao wrote the convenient toppe.write2loop() MATLAB function, and made many other contributions.
- Melissa Haskell made a number of contributions, including model-based reconstruction support for spiral imaging.
- Rolf Schulte (GE) provided the toppe.pns() MATLAB script for PNS calculation.
- Github user moedn proposed use of the MIT license

If you feel someone should be added to this list, please let jfnielse@umich.edu know – any omissions are entirely unintentional!