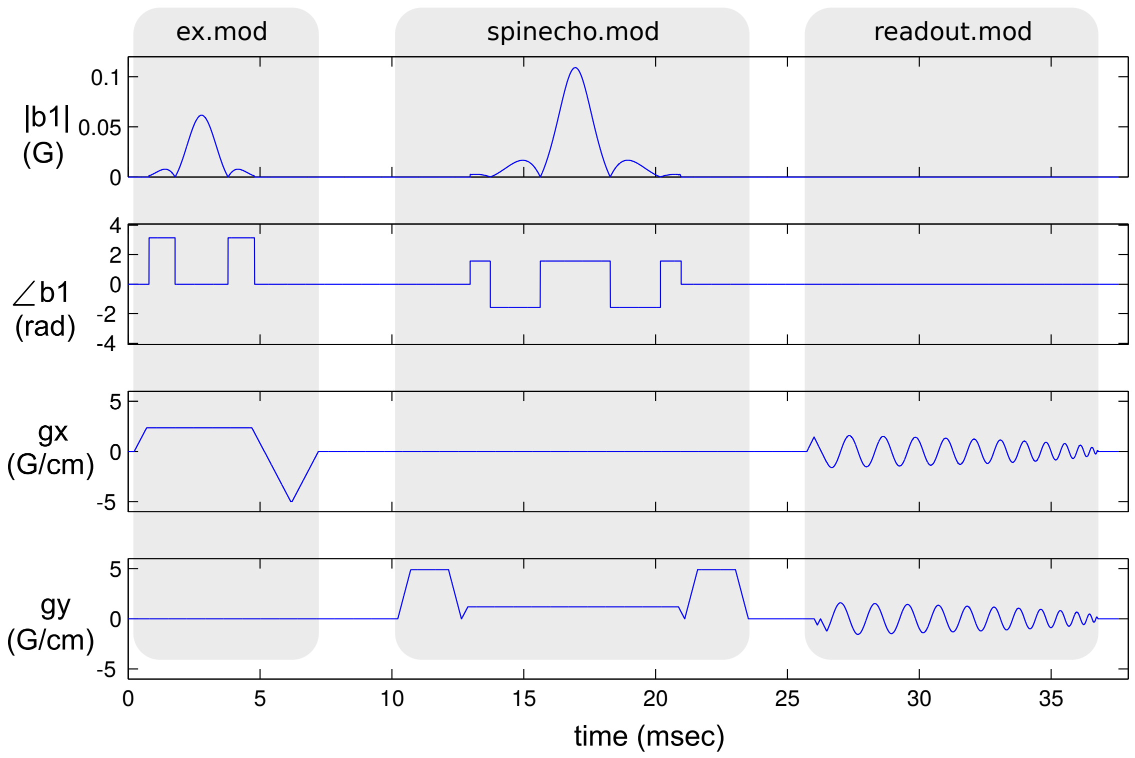
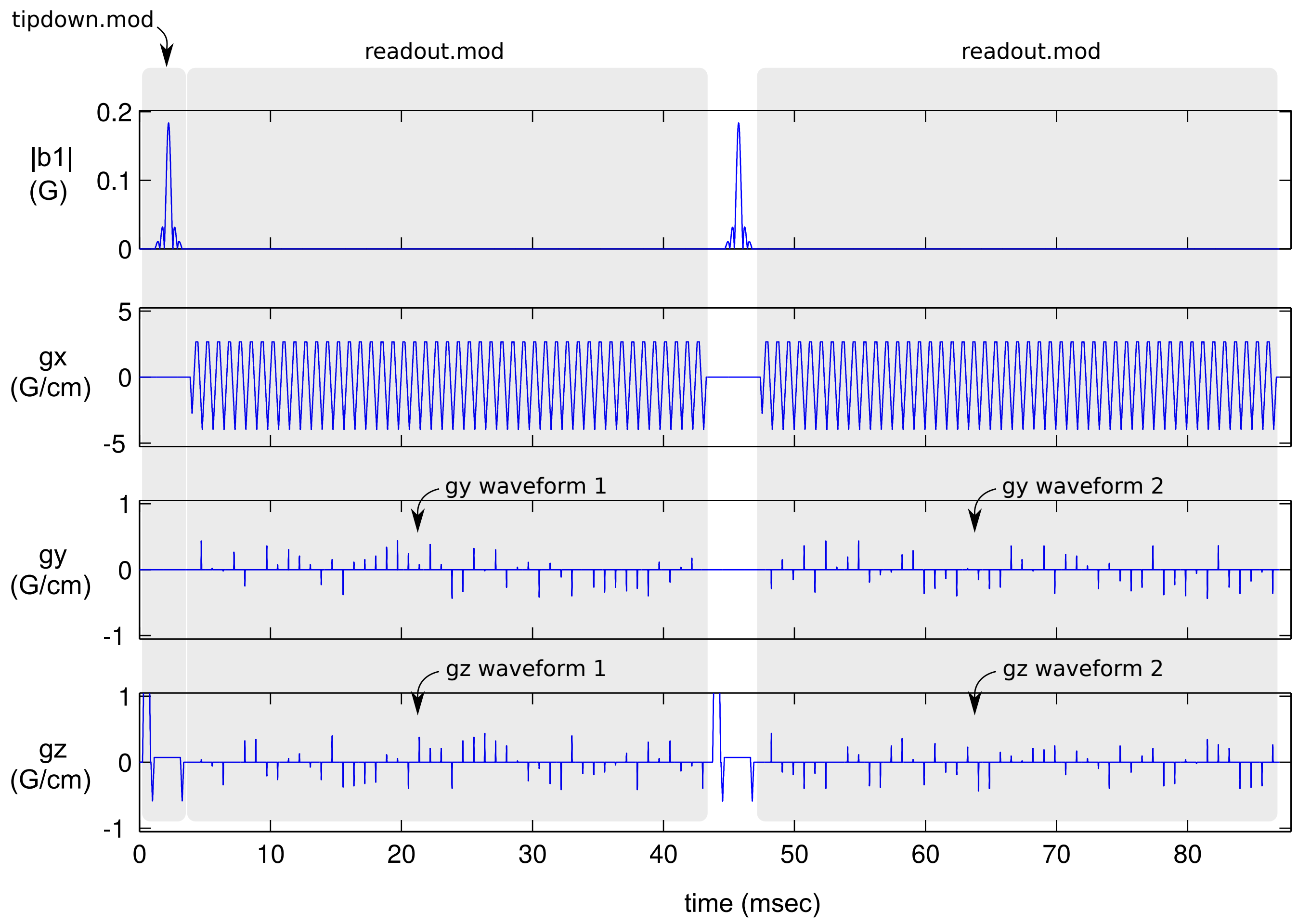
SUPPORTING TEXT S1

On the relationship between TOPPE and Pulseq

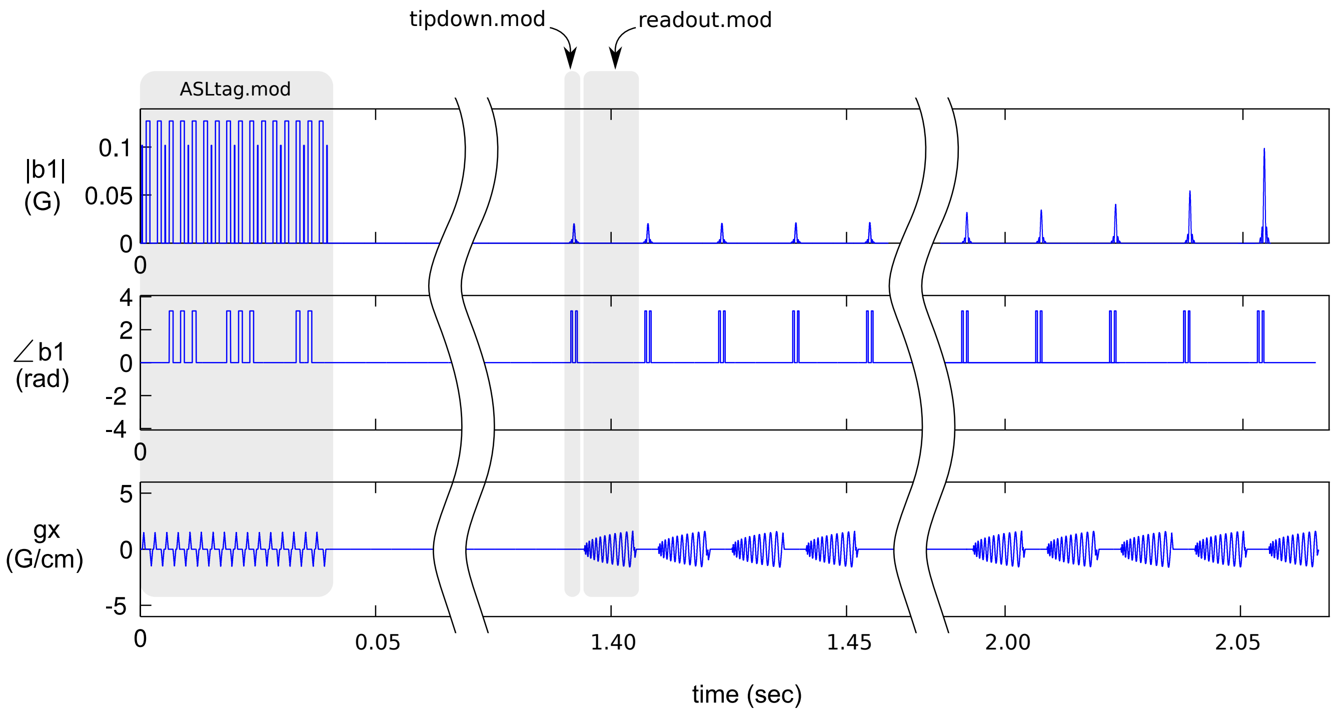
On a conceptual level, TOPPE is similar to Pulseq in that in both approaches, sequence specification is separated from hardware execution; our article thus presents TOPPE and Pulseq as parallel works. However, as we see it, on a practical level the role of TOPPE is not to "compete" with Pulseq, but to complement it by allowing execution of a platform-independent sequence specification on GE scanners. Based on our understanding of Pulseq and EPIC, we believe it would be rather awkward to attempt to write a GE driver/interpreter that interprets Pulseq files directly, without the intermediate Pulseq to TOPPE file conversion step proposed in our article. In particular, the hierarchical structure of Pulseq does not lend itself to easy parsing and implementation in EPIC. Specifically, EPIC requires that a relatively small number of sequence modules be defined prior to execution, each specifying waveforms on all three gradient axes as well as a (complex) RF waveform and/or data acquisition window. During scanning, amplitude waveforms and RF/acquisition frequency and phase can be assigned dynamically. The TOPPE file format is a simple reflection of the EPIC code structure, which greatly simplifies the task of writing a GE hardware driver. Pulseq, on the other hand, specifies sequential execution of sequence "blocks", in which the composition of each block is allocated dynamically, i.e., the presence of individual gradient or RF "events" in a block is specified during execution. To our knowledge, EPIC does not contain a programming mechanism that mirrors this way of representing a sequence. Thus, we see the TOPPE file format – and not just the TOPPE GE driver – as an essential piece of the proposed platform.



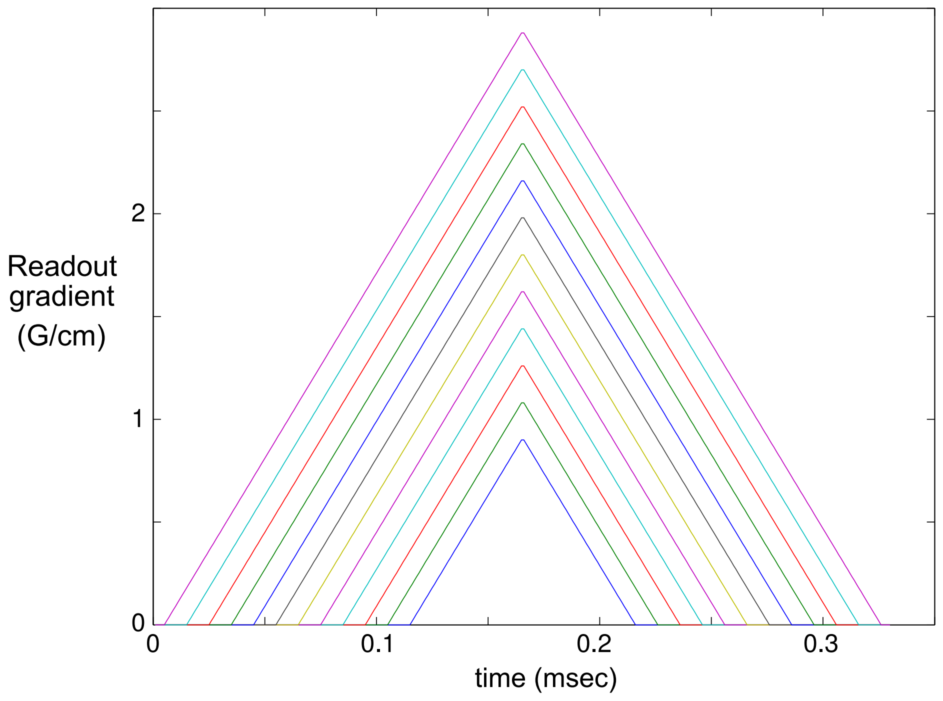
Supporting Figure S1: Pulse sequence diagram for the k-space trajectory measurements shown in Fig. 3.



Supporting Figure S2: Example of a TOPPE sequence with multiple (64) different gy and gz waveforms associated with one `.mod` file. Only two of the waveforms are shown. All 64 waveforms are loaded into scanner memory during sequence prescription. During sequence execution, the particular waveform to be played out is selected on-the-fly according to the corresponding entry in `scanloop.txt`. In this example, each waveform is of duration 40 ms. We successfully ran this sequence and switched the waveforms during sequence execution (not shown), according to the instructions in `scanloop.txt`. With the possibility of up to 20 different `.mod` files, each containing multiple waveforms, we believe TOPPE will be sufficiently flexible for most applications. However, we have observed that there appears to be a limit on total waveform memory allowed by our GE scanner: for example, the multi-waveform `.mod` file shown here appears to nearly reach the total memory limit.



Supporting Figure S3: Example of a relatively complex pulse sequence that is not easily implemented using traditional programming techniques: Arterial Spin Labeling with stack-of-spirals readout. Following a (velocity-selective) spin tagging pulse and a transit delay of ∼1.3 sec, a train of alternating RF excitation and data readout modules are played out (total duration ∼0.7 sec). kz (partition) encode ordering is center-out (not shown). The flip angle is increased to maintain constant signal following each excitation. For a given readout repetition interval and assuming tissue T1 of gray matter at 3T, we obtained the optimal flip angle schedule using the Matlab function `fmincon`. Implementing such a sequence using traditional programming techniques would be relatively laborious, and would in any case likely require the ASL tagging pulse, flip angle schedule, and perhaps the spiral readouts to be loaded from external files.



Supporting Figure S4: Another example of a relatively complex pulse sequence that is not easily implemented using traditional programming techniques: A set of 12 triangular readout gradients used to measure gradient impulse response function, as implemented in (15). Although the waveforms differ only by their time to peak (ranging from 50 to 160 μs in 10 μs increments), there is no direct way to encode these waveforms programmatically during run-time. In TOPPE, one can either associate each waveform with its own `.mod` file, or load all 12 waveforms into a single module (after zero-padding to ensure equal waveform duration).