Pattern Generator Design Doc

Blake Johnson

11/22/2010

The purpose of the pulse generation module is to simplify the specification of common time-domain qubit experiments. The idea is that many time-domain experiments fall into one of two categories:

- Concatenated primitives, such as: $\{+X_{\pi/2}, +Y_{\pi}, -X_{\pi/2}, ...\}$
- Sequences of experiments with a changing parameter, such as a pulse amplitude, a pulse duration, or a delay time.

Often, experiments contain some combination of these two. For instance, a Hahn spin echo experiment might be succinctly defined by the sequence:

$$\{X_{\pi/2}, \tau/2, Y_{\pi}, \tau/2, X_{\pi/2}\}$$

where τ is a variable time delay, and I use the shorthand notation of specifying rotations as $X_{\theta} = R_X(\theta) = e^{-i\theta\sigma_x/2}$. A spin echo experiment consists of a set of such pulse sequences for a given list of delays, τ.

The module is designed such that experiments can be specified by lists of pulse symbols, where the parameters that construct a particular pulse can be static or they can change in a systematic way. The constructors for these pulses are such that one only has to specify parameters if they differ from the default values.

To use the module, one specifies an instance of the PatternGen class, i.e.

```
pg = PatternGen();
```

One then constructs an experiment by specifying a list of `pulse()` functions with their associated parameters. The syntax for the `pulse` function is as follows:

```
pg.pulse( sym, 'paramname', paramvalue )
```

where *sym* is a pulse symbol that can be one of the following:

- Qubit pulse symbols: Xp, Xm, X90p, X90m, X45p, X45m, Xtheta, Yp, Ym, Y90p, Y90m, Y45p, Y45m, Ytheta, Up, Um, U90p, U90m, Utheta, QId
- Flux pulse symbols: Zf, ZId
- Measurement pulse symbols: M, MId

where Xp = $+X_{\pi}$, Xm = $-X_{\pi}$, X90p = $+X_{\pi/2}$, etc. A 'U' pulse is a rotation about an arbitrary axis in the X-Y plane. 'QId', 'ZId', and 'MId' pulses indicate the identity operation (do nothing), and are frequently used as placeholders for a delay in an experiment. The pulse function accepts many optional parameters. All parameters may be specified as a single value or as a list (vector) of values. The parameters are:

- *amp* - pulse amplitude

- *width* - total pulse width, in time steps
- *duration* - pulse duration (must be greater than or equal to the pulse width), to allow centering a pulse in a larger time bin
- *sigma* - for specifying pulse shapes such as Gaussians or hyberbolic tangents
- *angle* - rotation angle in the X-Y plane, only allowed for 'U' pulses
- *pType* - specifies the function used to generate the pulse. Allowed values are 'square', 'gaussian', 'gaussOn', 'gaussOff', and 'tanh'.

Each pulse symbol determines the value of the *amp* and *angle* parameters. If you want to specify the pulse amplitude or angle, you need to use *Xtheta* or *Ytheta* (amplitude), or *Up*, *U90m*, etc (angle), or *Utheta* (both amplitude and angle). The sigma and pType parameters can always be overriden by user input, though the default behavior is that qubit pulse symbols use the 'gauss' *pType*, while flux and measurement pulses use the 'square' *pType*.

Example (Ramsey experiment):

```
delays = 0:50:1000;
RamseyExpt = {pg.pulse('X90p'), pg.pulse('QId', 'width', delays)
pg.pulse('X90p')};
```

Example (amplitude Rabi):

```
RabiAmps = 0:80:8000;
RabiExpt = {pg.pulse('Xtheta', 'amp', RabiAmps)};
```

To perform the experiment, this pulse description must be converted into a sequence of pulse patterns that can be output by an AWG. The function that does this conversion is called `GetPatternSeq`. It does this by aligning the right edge of the pulse train (the last time point) to a fixed time within a larger total pattern length. It has the following prototype:

```
GetPatternSeq(pulse_list, pattern_number, delay, fixed_point,
cycle_length)
```

where *pulse_list* is a list of `pulse()`'s, *pattern_number* is an index into any parameters that were specified as lists (vectors), *delay* allows to account for channel-to-channel delays caused by varying cable lengths or switching times in electronics, and *fixed_point* specifies where the right edge of the sequence should lie within the total pattern of length *cycle_length*. *GetPatternSeq* returns two vectors corresponding to the I and Q output to be sent to the modulator. Note that this is the case even for flux and measurement pulses which are always single channel.

The *PatternGen* module also defines a utility function called `bufferPulse()` that allows for automated generation of digital pulses that 'gate' the microwave sources to provide a better on-off ratio during identity operations and reset periods between experiments. `bufferPulse`

outputs 1 whenever there is a non-zero output in the base pattern and 0 otherwise. The method takes several inputs to account for the particular limitations of the pulse modulation modes on the Agilent E8257 and N8153A sources. The function prototype is:

```
bufferPulse(pattern, zero_level, padding, reset_time, delay)
```

where the parameters are:

- *pattern* - the pulse pattern to buffer
- *zero_level* - the DAC value which gives 0V output. By convention, this is usually 8192 on a Tek5014 with the offset set to 0.
- *padding* - increase the width of the buffer pulse by this number of points in *each direction*. Accommodates finite rise time of the pulse.
- *reset_time* - minimum time between pulses. If the gap between pulses is shorter than this time, keep the output high (on)
- *delay* - shifts the buffer earlier in time by this number of points to account for delay between the buffer signal and the output of the pulse by the uW source.