Maximize a waveform generator’s memory

Joan Mercade - May 01, 2011

The architecture of an AWG (arbitrary waveform generator) looks like that of a DSO (digital storage oscilloscope) in reverse. Waveform memory plays a critical role in both instruments. DSOs capture waveforms and store them in memory, while AWGs create waveforms from data stored in memory.

AWGs must generate continuous signals for seconds, minutes, hours, or days to properly emulate actual systems, but they must do this within the limits of their memory. An AWG can have one of four architectures: the so-called “true arb,” DDS (direct-digital synthesis), interpolating, or interleaving, each of which interacts with the waveform memory in a different way (see a description of the four architectures). Fortunately, regardless of which type of AWG you have, you can program the instrument with loops and branches to reuse segments of memory and generate longer waveforms.

An AWG’s waveform $RL$ (record length) gives it the ability to generate nonrepeating signals. A waveform’s $TW$ (time window) is proportional to the $RL$ and is inversely proportional to the AWG’s $SR$ (sampling rate). For AWGs with a “true-arb” architecture, in which samples are converted by the DAC one after the other at the sampling speed, you can calculate the time window with this equation:

$$TW = RL/SR$$

As AWG sample rates grow, the amount of waveform memory needed must keep pace. Increasing the memory size at such high sampling rates raises two main issues. One is whether you can justify the cost of implementing the extra memory and the sophisticated memory-access architectures that feed the DAC at the required speeds. The other is whether you can afford to expend the effort and time required to calculate samples for complex signals and move them from mass storage devices (local or remote) to the waveform memory.

Memory access issues

AWGs combine ultra-high-speed RAM and massive demultiplexing of the memory access buses to
resolve these issues. **Figure 1** shows two AWG architectures: one with a mux and one without a mux. The mux lets the AWG read slower memory at a lower speed while keeping the overall data throughput constant.

**Figure 1.** Memory-access parallelization uses lower-cost memory at the expense of record-length granularity. Waveforms use \( n \) samples, where \( n \) is a multiple of the memory bus width. (\( N \) is the number of channels in the mux; \( SR \) is the sampling rate.)

There are always tradeoffs among design complexity, size, and cost. As a result, high-speed AWGs (those with sample rates greater than 2 GHz) typically limit the available waveform memory to some tens of megasamples. Even that much memory may limit time windows. For example, a 32-Msample, 12-Gsamples/s AWG can generate just over 2.5 ms of a nonrepeating waveform.

Just because an AWG has a large waveform memory doesn’t mean that you should use all of it. You should minimize waveform length to reduce calculation and transfer time. For short waveforms, you should consider increasing the amount of waveform memory beyond what you need for building the basic waveform. You’ll increase waveform quality, because you can increase the sampling rate (oversampling). A higher sampling rate will spread out quantization noise over frequency and improve signal-to-noise ratio. You can further improve signal quality by applying sigma-delta techniques. For periodic signals, simply store one cycle of the signal and repeat it.
Unfortunately, using short waveforms creates periodic quantization noise that will appear as high-amplitude spurious spectral lines in the output signal’s frequency spectrum. That noise will reduce the instrument’s SFDR (spurious-free dynamic range). Repeating the signal many times while ensuring that the signal does not repeat itself exactly in the waveform memory (such as by selecting a prime number for the record length or by selecting a number of cycles that is not a divisor of the selected record length) will result in a lower repetition rate for the quantization noise; likewise, adding some small amplitude dither will also result in a lower repetition rate. You’ll have about the same amount of noise power, but it will be spread over many more spectral lines. Thus, you’ll reduce the average power for each line and improve the SFDR.

Some AWGs have a specification for the minimum usable record length just as they do for the minimum sample rate. In addition to those limits, a parallel waveform-memory-access bus architecture forces the waveform length to be a multiple of a given integer to reflect the number of samples transferred simultaneously in each memory access cycle. This integer number (typically a power of 2) reflects the number of samples that the AWG transfers in parallel from the waveform memory to the DAC. You can call this characteristic the “record-length granularity.” AWGs with interleaving DACs double the record-length granularity because they use a two-DAC architecture, with each DAC connected to an independent memory bus.

You can update an AWG’s waveform memory from local, nonvolatile storage; from a remote computer; or from a DSO. But with any of these options, the update process may be unacceptably slow for applications such as automated test. Most high-performance AWGs allow for segmentation of their waveform memory, where each segment can hold a different number of samples, which maximizes flexibility. Switching from one segment to another is as easy as updating the sample pointer. Memory segmentation plays a fundamental role in sequencing and real-world interaction.

**Sequencing**

Many tests require long waveform memory records, and eventually, the necessary record length may exceed the available waveform memory. Sequencing is a popular way to increase a signal’s length. In sequencing, different waveform segments repeat (or are played back one after the other) through a user-defined sequence list.

For example, to produce a serial-data signal with one anomalous bit every second at 10 Gsamples/s, a typical AWG would need a 10-Gsample memory. A sequencing AWG could accomplish the same effect by repeating a 1-Msample record containing one bit 9999 times and then producing one anomalous bit (Figure 2). Doing this reduces waveform memory by 5000 times (2 Msamples versus 10 Gsamples), and it reduces the time needed to calculate, transfer, and load the necessary samples.

![Figure 2](image)

*Figure 2.* Waveform memory segmentation and sequencing lets you implement very long signals and generate signals that are responsive to real-world conditions. In this example, two different segments are combined through a sequencing list to produce a much longer signal.
Using signals with regularities may result in huge memory savings. Video-test signals are a good example. Each video line may contain similar sections, or synchronization signals: Lines with the same video information repeat many times in a single frame or field, and each frame or field requires similar synchronization signals at the beginning.

Straightforward sequencing—where each entry, or step, in the table (sequence list) specifies a waveform segment and the number of times it should be repeated before the AWG proceeds to the next step—may produce the necessary synchronization signals. Complex signals such as video-test signals, however, may require long sequences.

You can reduce memory usage by implementing nested sequences where you define “sequences of sequences” or even mix regular segments and other sequences. For a video-test signal, one low-level sequence could contain the segment with the line synchronization, and a second sequence could contain the luminance and chrominance information. Another sequence could contain the same segment with the line synchronization and the contents of the frame synchronization, and then a higher-level sequence could link all these sequences and include specific segments for each frame. With this structure, defining a new test pattern becomes as easy as changing the luminance segment entry in the sequence list.

Although memory segmentation and sequencing can improve the versatility of an AWG, they can introduce performance issues. First, there may be a limit to the number of segments that an AWG can support, regardless of its waveform-memory size. Memory-access considerations and the speed of the sequencer may also limit the minimum or the maximum segment size. The sequencer itself may also place a limit on the number of entries in the sequencing list and on the depth of nested sequences.

**Real-world interaction**

Engineers often use AWGs to generate predefined waveforms and sequences. Some applications, though, may require the AWG to interact with the DUT (device under test) at speeds beyond programmatic control. Many high-performance AWGs incorporate trigger-and-sequence or segment-control signals, providing real-time interaction with a DUT. Typical interactions include:

- **Basic sequence control.** You can instruct the sequencer to stop the output (typically by keeping the analog level established by the last sample) at the end of a given step until an external trigger signal activates. Then, the sequencer advances to the next step in the list.

- **Vectored segment control.** A series of input signals may change the segment that the AWG outputs. As an example, a commercially available high-performance AWG uses nine input lines (eight address lines plus a validation line) to select one of up to 256 waveform segments stored in memory. Although this solution provides extreme flexibility, its practical implementation in most test situations would force you to develop very complex hardware to map DUT behavior to AWG stimulus. Figure 3 shows the rear panel of a Tabor Instruments AWG. Each channel has a nine-pin connector for trigger signals.
• **Advanced sequence control.** Conditional branching (or jumps) may depend on the status of some external signals as defined for each entry in the sequence list. This capability provides the most flexibility because it allows similar capabilities to those allowed by vectored segment control, but without the need to develop complex external hardware.

You can trigger an action synchronously or asynchronously. In the first case, the AWG will wait for the end of the current step or segment before triggering. In the second case, the trigger takes place when the trigger conditions are satisfied. An AWG’s response to trigger and control inputs may be slower than expected, so you must know the minimum duration and maximum bandwidth that a trigger supports. Trigger delay or latency is another important factor as well.

Marker and trigger outputs in AWGs interact with the DUT or other test equipment through digital signaling. Marker outputs are typically fed from the waveform memory, providing timing and synchronization for other equipment. In some cases, you can associate markers with segments instead of individual samples. You may have some control on the marker output levels, easing the integration with the target circuits.

Some AWGs, instead of storing marker information, reserve some of the low bits otherwise connected to the DAC. Activating the markers may reduce the vertical resolution of the arbitrary waveform. Although markers share the same data path as the waveform data, the output circuitry is different. Thus, skew between the analog waveform output and the marker outputs is an issue, especially for AWGs with high sampling rates.

**Wrap-around issues**

Seamless, continuous waveform generation requires the looping or sequencing of waveforms or waveform segments. But many applications won’t tolerate waveforms that contain any discontinuities. Think of a serial data signal with a fraction of a bit at the end of the waveform. Connecting a looping signal like this to any clock-recovery system would end in complete havoc,
because any discontinuity or signal anomaly when looping or linking a signal results in a wrap-around artifact.

You can minimize or even cancel wrap-around artifacts by using some special techniques or by carefully designing the waveforms. For a single, looping segment, the number of symbols or cycles contained in a given segment must be an integer, and any convolution process (such as filtering) must keep consistency between the end and the beginning of the waveform. Even if you meet those conditions for every waveform or waveform segment, problems may arise when sequences of segments are involved.

IQ baseband generation of digitally modulated signals provides a good example. You can design any segment to be looped so it contains an integer number of symbols and then apply the required baseband filter through circular convolution (Figure 4). The result will be a signal without any artifact or discontinuity at any domain (time, spectral, or modulation).

![Baseband filter impulse response](image)

**Figure 4.** Wrap-around artifacts can introduce discontinuities into an AWG’s output. Adding “linking segments” provides perfect continuity.

If you must use two of those segments in sequence, the end of one segment and the beginning of the next one will be inconsistent. There are two solutions. In the first, you use the same symbols at the beginning of the two segments; the number of symbols depends on the accuracy required or the length of the convolution process. Of course, this method may be unacceptable if the application requires arbitrary symbols. In the second solution, you add a “linking segment” at the end of each segment that links with the symbols of the next segment. In this method, the first symbols of a segment are convolved with the last symbols of the previous segment. The result would be something like: A,…A, A, AB, B,…B, BC, C, C,…, C, CA..., where AB, BC, and CA are the linking
segments. The number of waveform segments will grow as well as the sequencing list, especially if you need multiple links, because any of those will require a specific linking segment. T&MW

Joan Mercadé is a telecom engineer with a degree from the Polytechnic University of Catalonia in Spain. He has worked in the test-and-measurement industry for over 20 years at companies including Philips and Tektronix. He now owns and runs Arbitrary Resources, an R&D and consulting company. joan.mercade@arbitrary.es.