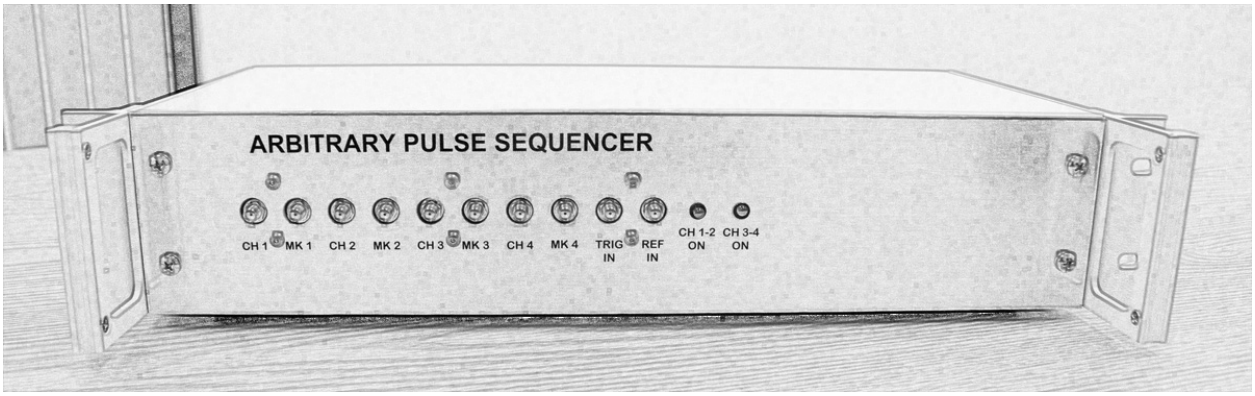
libaps Documentation

Release 1.3

BBN

Mar 03, 2017

Contents



The BBN Arbitrary Pulse Sequencer (APS) is an arbitrary waveform generator with an advanced sequencing ability. The sequencer allows for specification of individual operations (gates) to be defined as units in a waveform library, so that an algorithm/experiment can be defined by stringing together sequences of gates and delays. This results in a very compact description for efficient memory use.

Contents:

CHAPTER 1

Quick Start

TODO: Follow these steps to get up and running quickly...

CHAPTER 2

Installation Guide

Hardware

The BBN APS has separate enclosures for the power supply and analog front-end. A single power supply module has two 5.5V outputs and two 3.3V outputs, which is sufficient to power two analog modules. Using the supplied cables, connect a pair of power outputs with the same label (PS1 or PS2) to the power inputs on the rear of an analog module. It is important to maintain the labeled pairing when connecting to the analog modules in order to ensure proper power sequencing and optimal noise performance. These outputs are not hot-pluggable; one must ensure that the power switch on the front of the supply module is in the off position before connecting or disconnecting the power cables.

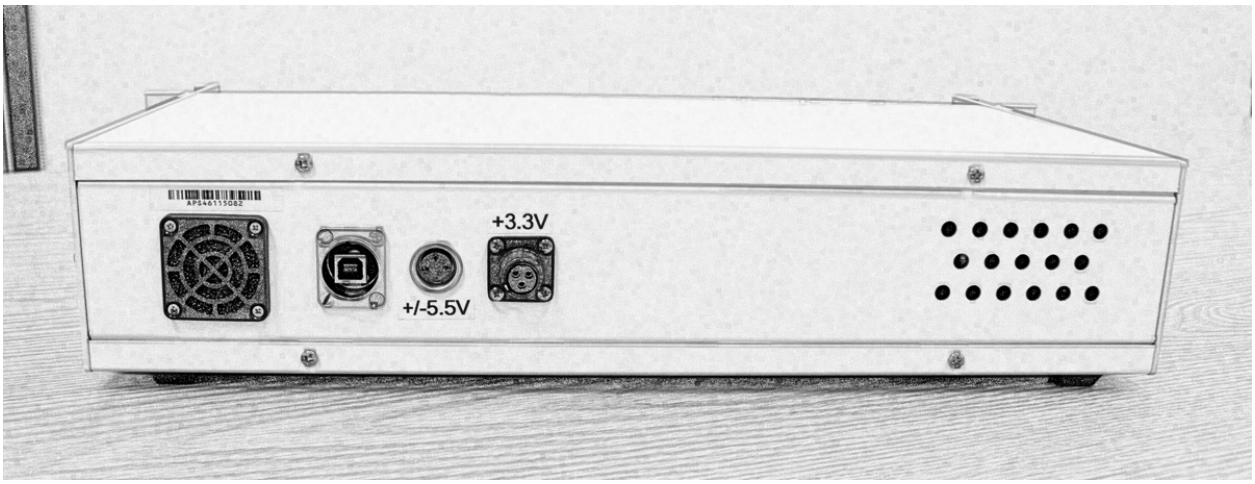


Fig. 2.1: **BBN APS rear panel.** The rear panel of the APS has a USB connect as well as 3.3V and 5V power supply inputs. Be careful to connect power outputs on the power supply with the same number in order to obtain the best noise performance from the device. Never unplug the cables from the power inputs while the power supply is on.

Once the power supply has been connected, turn the APS on with the power switch on the front of the power supply. At this point, the FPGAs in the analog module are in a blank state, awaiting upload of the pulse sequencer firmware

over the USB interface.

While the APS can run in a standalone configuration, we recommend running with a 10 MHz (+7 dBm) external reference. This reference must be supplied at the corresponding front panel input before powering on the device. Multiple devices can be syncronized by supplying an appropriate external trigger.

Software

USB Driver

The BBN APS requires a USB driver in order to communicate with the host PC. Prior to plugging the APS into the host computer, you should download and unzip the driver from the 'FTDI website <http://www.ftdichip.com/Drivers/D2XX.htm>'_. After connecting to a Windows XP/Vista/7 machine, the 'new hardware' wizard will open. Occasionally Windows will find an appropriate driver without further input, but more often you will need to supply the path to the FTDI driver folder.

On Linux, for normal user access to the device you will have to add a udev rule. Adding a file to /etc/rules.d such as `50-aps-usb-rules` with the line

```
# Make available to non-root users
SUBSYSTEM=="usb", ATTRS{idVendor}=="0403", ATTRS{idProduct}=="6001", GROUP="users",␣
↪MODE="0666"
```

should work. The kernel bundles a USB virtual com port driver that will take precedence over the FTDI driver libaps uses. See this FTDI guide for details but running the following commands after every plug-in or power cycle event will work.

```
sudo rmmod ftdi_sio
sudo rmmod usbserial
```

It should also be possible to automate unbinding the VCO driver using a more sophisticated udev rule.

libaps

The APS is driven by a C++ library with a C API. We have provided MATLAB, python, and LabVIEW bindings to this library such that use of the APS is as similar as possible in the various instrument control environments. The library is bundled into a release package that is available in release tab of the GitHub site. You simply need to add the relevant paths to your MATLAB or Python code. In particular for Python on Linux you will need to add the folder containing `libaps.so` to your `LD_LIBRARY_PATH`.

Python requirements

The BBN APS driver for Python requires Python 2.7 or later (but not Python 3+). You also need a working installation of NumPy and h5py. We recommend using the Anaconda python distribution.

MATLAB requirements

The BBN APS driver for MATLAB requires MATLAB 2010a or later. The driver does not depend on any toolkits, so a vanilla install is sufficient.

LabVIEW requirements

The BBN APS driver for LabVIEW requires only a relatively recent LabVIEW installation that supports object-oriented instrument classes (2008 or later).

Standalone GUI control program

A standalone Win32 application is available for controlling the BBN APS. This application is available on the downloads section of the BBN Qlab repository on Github.

CHAPTER 3

API

libaps exports a C API which can be called from any other language that has a foreign function interface that can interact with a C shared library.

Interface drivers for the BBN APS are available on several platforms including: MATLAB, Python, and LabVIEW. Effort has been made to use consistent naming across these interfaces, so that method signatures look similar in the various platforms. The following methods are available on all platforms, and are consider 'public' methods, in the sense that the API for these methods is expected to be consistent across major software versions. Use of methods outside of this list may result in broken code when receiving future software updates.

Methods

APS()

Inputs None

Outputs an APS object

Description This method instantiates an APS driver object. Creation of this object is the first step in all use cases of the driver.

connect(address)

Inputs integer or string *address* - device ID (integer) or serial number (string)

Outputs None

Description Opens the USB connection to the APS. May take as input a device ID or a device serial number. The device ID is determined by an alphanumeric sorting of the connected APS serial numbers. The first device in that sorted list has ID = 0. Consequently, if you only have one APS connected, you can assume that it is device 0.

disconnect()

Inputs None

Outputs None

Description Closes the USB connection to the APS. The APS driver allows only one open connection at a time, so it is important to include a call to `disconnect()` in your code.

init(force, bitfile)

Inputs integer *force* - (optional) 1 = force loading of FPGA firmware, 0 = do not force load (default); string *bitfile* - (optional) fullpath to a valid APS bitfile.

Outputs None

Description Performs all initialization tasks on the APS. This method should be called by all user code between `connect()` and all other commands. The driver attempts to detect whether initialization is necessary, and will skip most tasks if it detects that the APS is in a ready state. You can override these checks and force the driver to re-initialize the APS by calling this method with force = 1.

setAll(settings)

Inputs struct (Python dictionary) *settings* - complete APS settings structure

Outputs None

Description A single method for doing all setup tasks for the APS. The settings structure has the following elements:

- chan_n.enabled
- chan_n.amplitude
- chan_n.offset
- samplingRate
- triggerSource
- seqfile

where 'n' in the channel elements identifies the channel number (1-4). You can see an example usage of `setAll()` in the *Example*.

loadConfig(path)

Inputs string *path* - full path to an APS sequence configuration file

Outputs None

Description Loads a multi-channel sequence configuration files as described in *Sequence Mode and File Specification*. `loadConfig()` will enable any channel for which there is waveform/sequence data in the configuration file, and will set the run mode to RUN_SEQUENCE.

loadWaveform(channel, waveform[])

Inputs integer *channel* - target channel (1-4); integer/float array *waveform[]* - signed 14-bit waveform data in the range (-8191, 8192) or signed float data in the range (-1.0, 1.0)

Outputs None

Description Loads waveform data onto a channel of the APS. Also enables the channel. To load sequence data, see loadConfig() and/or setAll().

run()

Inputs None

Outputs None

Description Starts output on all enabled channels. See `setEnabled()` to see how to enable a channel.

stop()

Inputs None

Outputs None

Description Disables output on all enabled channels and resets the pulse sequencer back to the beginning of the sequence.

isRunning()

Inputs None

Outputs boolean

Description Returns *true* if any channel of the APS is currently running.

setRunMode(channel, mode)

Inputs integer *channel* - target channel (1-4); integer *mode* - RUN_WAVEFORM (0) or RUN_SEQUENCE (1)

Outputs None

Description Sets the run mode to either directly output the contents of waveform memory, or to function as a pulse sequencer, stepping through the loaded link list entries.

setOffset(channel, offset)

Inputs integer *channel* - target channel (1-4); float *offset* - normalized channel offset in range (-1.0, 1.0)

Outputs None

Description Sets the voltage offset of the specified channel. Note: the APS mimics a voltage offset by shifting the waveform data. Consequently, it is possible to introduce clipping of the waveform by using this method.

setAmplitude(channel, offset)

Inputs integer *channel* - target channel (1-4); float *offset* - channel amplitude/scale factor

Outputs None

Description Sets the channel scale factor. Note: the APS mimics channel amplitude by multiplying the waveform data by the channel scale factor. It is possible to introduce clipping of the waveform by using this method.

setEnabled(channel, enabled)

Inputs integer *channel* - target channel (1-4); bool *enabled* - enabled state of channel

Outputs None

Description Enables or disables the specified channel.

setTriggerDelay(channel, delay)

Deprecated - will not be supported in future releases

Inputs integer *channel* - target channel (1-4); integer *delay* - channel trigger/marker delay with respect to the analog output, specified in units of 4 sample increments (e.g. delay = 3 is a 12 sample delay)

Description Sets a fixed delay of the marker channel associated with a given analog output channel.

Properties

samplingRate

Description Set or get the sampling rate (in MS/s). Valid inputs are (1200, 600, 300, 100, or 40).

triggerSource

Description Set the trigger source. Valid inputs are 'internal' or 'external'.

Example

This example uses `setAll()` rather than calling individual methods.

```
% create settings structure
settings = struct();
settings.chan_1.enabled = true;
settings.chan_1.amplitude = 1.0;
settings.chan_1.offset = 0;
settings.chan_2.enabled = true;
settings.chan_2.amplitude = 1.0;
settings.chan_2.offset = 0;
settings.chan_3.enabled = true;
settings.chan_3.amplitude = 0.8;
settings.chan_3.offset = 0.1;
settings.chan_4.enabled = true;
```

```
settings.chan_4.amplitude = 1.2;
settings.chan_4.offset = -0.05;
settings.samplingRate = 1200;
settings.triggerSource = `external';
settings.seqfile = `Ramsey/Ramsey.h5';

aps = deviceDrivers.APS();
aps.connect(0);
aps.init();
aps.setAll(settings);
aps.run();

% acquire data...

aps.stop();
aps.disconnect();
```

The same thing could be accomplished with calls to individual methods:

```
aps = deviceDrivers.APS();
aps.connect(0);
aps.init();

% configure the APS
% set up channels
aps.setAmplitude(1, 1.0);
aps.setOffset(1, 0);
aps.setAmplitude(2, 1.0);
aps.setOffset(2, 0);
aps.setAmplitude(3, 0.8);
aps.setOffset(3, 0.1);
aps.setAmplitude(4, 1.2);
aps.setOffset(4, -0.05);

% load pulse sequence
aps.loadConfig(`Ramsey/Ramsey.h5');

% configure output rate and trigger source
aps.samplingRate = 1200;
aps.triggerSource = `external';

aps.run();

% acquire data...

aps.stop();
aps.disconnect();
```

CHAPTER 4

Sequence Mode and File Specification

In addition to outputting waveforms of up to 32,768 points, the APS supports a sequence mode that outputs a series of pulses from the waveform memory. This sequence definition table, known as a 'link list', can store up to 8,192 entries. The APS's dual-port memory architecture allows the driver software to stream data to the sequence memory while the device is running. If configured such that an individual sequence duration is longer than the transfer time ($\sim 10\,\mathrm{ms}$ per channel), then the APS can output a nearly continuous stream of pulses. Use of this streaming mode of operation is largely transparent to the user. One simply asks to load some sequence data to the APS, and if there are more entries than can fit onto the device memory, the driver will use streaming mode automatically.

The APS sequence mode requires the construction of a set of one or more link lists. The APS begins output of each link list upon receipt of a trigger, which can be supplied at the external trigger input of the analog module, or it can be generated internally. Each link list is composed of one or more entries, which can be any of the following types:

- **waveform** - an analog pulse defined by a section of waveform memory
- **time/amplitude pair** - a constant amplitude signal of a specified duration
- **delay** - a time/amplitude pair with amplitude zero

Each entry may also specify a transition in the output of the associated digital (marker) channel. In particular, one can specify the position of a rising edge, falling edge, or pulse at a delay from the beginning of the output of the entry.

The APS firmware allows for an additional division of each link list into sub-sequences known as 'mini link lists'. This allows the user to specify sequences with repeated sections, or sections which should wait for a trigger to output. One use of this feature is to specify sets of experiments which scan over some parameter such as a delay or a pulse height, and take the data in a 'round robin' mode, as supported by many digitizers.

Figure *Example pulse sequence: A simple pulse sequence that might occur in a Hahn echo experiment as implemented using a link list. Note that one can test very long delays because the data stored in waveform memory is independent of the experiment delays.* shows an example of a Hahn echo experiment ($\pi/2$-pulse, wait τ, π-pulse, wait τ, $\pi/2$-pulse) specified as a link list. The waveform memory (right panel) contains just two pulses, corresponding to the $\pi/2$ and π pulses. Then the link list (bottom) joins these pulses together with appropriate delays.

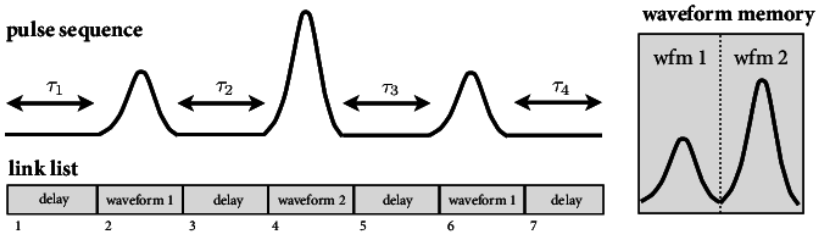


Fig. 4.1: **Example pulse sequence:** A simple pulse sequence that might occur in a Hahn echo experiment as implemented using a link list. Note that one can test very long delays because the data stored in waveform memory is independent of the experiment delays.

File Specification

Sequences are programmed using an HDF5 file with the following layout (n is an integer between 1-4):

/version - attribute indicating file version number
/channelDataFor - attribute containing array of integers specifying channels for which data is supplied in the file
/miniLLRepeat - attribute containing default number of times to repeat each mini link list (0 = play without repeats)
/chan_n/isListListData - integer attribute specifying whether link list data is supplied for this channel
/chan_n/isIQMode - integer attribute specifying whether this channel contains data for an I/Q pair (default = 1)
/chan_n/waveformLib - int16 vector of 14-bit waveform values, sign-extended to 16-bits
/chan_n/linkListData/length - integer attribute specifying number of link list entries
/chan_n/linkListData/addr - int16 vector of waveform addresses
/chan_n/linkListData/count - int16 vector of waveform lengths
/chan_n/linkListData/repeat - int16 vector of repeats
/chan_n/linkListData/trigger1 - int16 vector of offset counts for trigger1 pulses
/chan_n/linkListData/trigger2 - int16 vector of offset counts for trigger2 pulses

Link list field formats

An individual sequence entry consists of a value from each of the addr, count, repeat, trigger1, and trigger2 fields. Data is encoded in these fields in the following way:

Address: 16-bit offset in quad samples into waveform memory.

Count: length of waveform in quad samples minus one. For example, a waveform that is 16 samples long has count = 3. The minimum count is 2.

Repeat: 10-bit repeat count for the waveform. Bits 10-11 are reserved. Bit 15 is the START_MINILL flag. Bit 14 is the END_MINILL flag. Bit 13 is the WAIT_FOR_TRIG flag. Bit 12 is the TA_PAIR flag.

Trigger1/2: offset in quad samples to output a trigger pulse on the corresponding marker output channel. A value of zero means no pulse. Accordingly, it is not possible to have a pulse aligned with the first sample of a waveform.

CHAPTER 5

Specifications

Each unit has four 14-bit 1.2 GS/s analog output channels and four 300 MS/s digital marker channels. The clocks can be synchronized to an external 10 MHz reference. The device can be triggered internally or via an external trigger input. The device uses low-noise linear power supplies with an ultra-low-noise buffer amplifier on the DAC outputs to provide a system noise comparable to a 1 kOhm resistor.

The BBN Arbitrary Pulse Sequencer (APS) is USB slave module with four 1.2 GS/s 14-bit analog output channels and four 300 MS/s digital marker channels. A high-bandwidth amplifier buffers the DAC outputs to drive a 50 ohm load to $\pm$1V full-scale. High-speed LatticeSC3 FPGAs generate 600 MHz DDR sample streams for the DAC channels. Low-skew, low-jitter 300 MHz and 1.2 GHz clocks from a PLL clock generator drive the FPGAs and the DACs. The clocks can be phase locked to an external 10 MHz reference.

The voltage noise of the APS is limited by the noise performance of the last-stage output amplifier, an Analog Devices AD8099. The measured noise at the analog channel outputs is shown in Fig [fig:noise]. Careful engineering of the power supplies and analog/digital ground plane separation lead to system noise performance that is orders of magnitude better than some commonly used waveform generators.

Detailed Specifications

Parameter	Value
Analog channels	four 14-bit 1.2 GS/s outputs
Jitter	28 ps RMS (71 ps peak-to-peak)
Rise/fall time	2ns
Settling time	2 ns to 10%, 10 ns to 1%
Digital channels	four 300 MS/s capacitvely coupled outputs
Trigger input	1 V minimum into $50\,\Omega$, 5 V maximum; triggered on *rising* edge
Waveform memory	32,768 samples per channel
Sequence memory	8,192 entries per channel (unlimited with streaming)
Minimum sequence pulse length	12 samples (10 ns)
Maximum sequence pulse length	27 us waveform, or 8 s time-amplitude pair
Minimum sequence length	2 entries

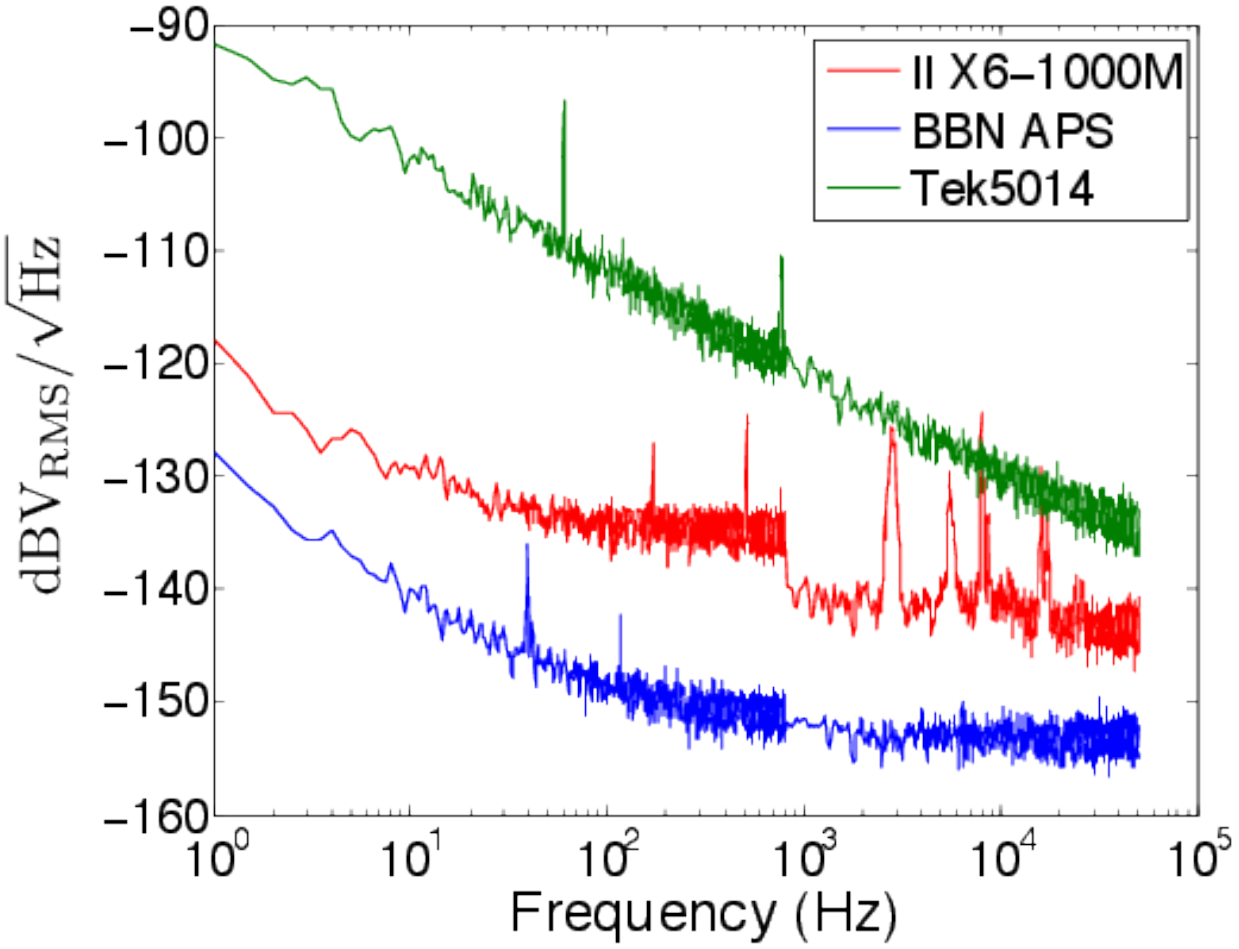


Fig. 5.1: **Comparison of AWG output noise.** Output noise power versus frequency for the Tektronix AWG5014, Innovative Integration X6-1000M, and BBN APS. The APS's linear power supplies and low-noise output amplifier lead to signficant improvements in the noise performance. The II X6 is significantly better than the Tek5014, but suffers from resonances in the noise spectrum because it is in a host PC environment.

CHAPTER 6

Indices and tables

- genindex
- search