arbWaveCom

Generated by Doxygen 1.8.9.1

Fri Jul 17 2015 14:24:17

Contents

Chapter 1

AWG Data/Communications format

How data is communicated to and from the AWG

1.1 Device Documentation

Device documentation is hosted at exodus.poly.edu/~kurt/:

- User Manual (pdf)
- Programmer's Manual (pdf)

1.2 Arbitrary precision numbers as ASCII

To send numbers of arbitrary lengths of decimal digits, the AWG uses the format:

#<Number of decimal digits in the number><The number itself>

All digits are sent as ASCII, not binary. Numbers longer than 9 digits are not representable, nor should they be needed.

1.2.1 Examples

Number	AWG Representation
1	#11
31	#231
193	#3193
14253697	#814253697

1.3 Marker Data Format

See the Programmer's Manual, page 2-112

The command invoked to send marker data to the AWG is MARKER:DATA

It is formatted as

```
MARKER:DATA <Number of points in ASCII Number Format><That many bytes of marker data, formatted as below>
```

1.3.1 Individual Marker Point Format

Points are represented as binary bytes, with the LSB corresponding to Marker 2 and the next lowest bit Marker 1. For clarity:

Marker 1	Marker 2	Marker Value
L	L	0 (00)
L	H	1 (01)
H	L	2 (10)
H	H	3 (11)

1.3.2 Examples

Marker Pattern	Command ([] means send binary representation)
6 points, Marker 1 toggling every point, Marker 2 every other point	MARKER:DATA #16[0][2][1][3][0][2]
10 points, Marker 1 high on only first point only, Marker 2 only on the sixth	MARKER:DATA #210[2][0][0][0][0][1][0][0][0][0]

1.4 Curve (Waveform) Data Format

See the manuals:

- Equipment Manual, page 4A-15 (PDF page 149)
- Programmer's Manual, page 2-59 (PDF page 85) and following

The command used to transfer data to the AWG is `CURVe` Here we assume the data width has already been set to 1 (byte). In that case, the command format as

```
CURVe <Number of points in ASCII Number Format><That many bytes output data, formatted as below>
```

1.4.1 Output Point Format

The generator has 8-bits of output resolution, and to have the ability to output 0 it has slightly asymmetric output ranges. They also chose a data format that has output values linearly increasing with binary values. As a result, you get:

- [0, 254] maps onto [-1, 1]
- 127 onto 0
- 255 is 1.0079

1.4.2 Examples

Curve Pattern	Command ([] means send binary representation)
6 points, 50% duty cycle, 2 point period, ranging from 0 to 1	CURVe #16[127][255][127][255][127][255]

12 points, Triangle wave from [-1,1]	CURVe #212[127][169][213][255][213][169][127][85][42][0][42][85]

Note

Actual waveforms are restricted to have lengths that are multiples of 32, but for simplicity these are shorter.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 Option subsystem return codes

Return codes used internally to indicate how to respond to parsing options.

Macros

- #define OPT_RET_OK 0

  Indicates successful option parsing. Continue program operation.
- #define OPT_RET_ERR -1

  Indicates an error occurred while parseing options. Exit with non-zero code.
- #define OPT_RET_EXIT -2

  Indicates program should exit normally without further activity.

5.1.1 Detailed Description

Return codes used internally to indicate how to respond to parsing options.

5.2 Option encoding bitmasks

Bit masks for encoding the flags for the various options that can be set on the command line.

Macros

- #define OPT_TEMPLATE_MASK (1u << 0)

  Flag for requested template file output. 0 is unset, 1 is set.
- #define OPT_RANDAMP_MASK (1u << 1)

  Flag for using random amplitudes for output. 0 is unset, 1 is set.
- #define OPT_HELPREQ_MASK (1u << 2)

  Flag for user-requested help. 0 is unset, 1 is set.
- #define OPT_FROMCMD_MASK (1u << 15)

  Flag indicating user input frequency specification via command-line options. 0 is unset, 1 is set.
- #define OPT_STARTSET_MASK (1u << 8)

  Flag indicating user specified a start frequency, found in progOptions::start_f. 0 is unset, 1 is set.
- #define OPT_STOPSET_MASK (1u << 9)

  Flag indicating user specified a stop frequency, found in progOptions::stop_f. 0 is unset, 1 is set.
- #define OPT_AMPSET_MASK (1u << 10)

  Flag indicating user specified an amplitude. If OPT_RANDAMP_MASK is not also set, found in progOptions::amplitude. 0 is unset, 1 is set.
- #define OPT_PERIODSET_MASK (1u << 11)

  Flag indicating pulse duration is set, found in progOptions::tooth_period. 0 is unset, 1 is set.
- #define OPT_NUMSET_MASK (1u << 12)

  Flag indicating the number of teeth is set, found in progOptions::num_f. 0 is unset, 1 is set.
- #define OPT_ALLSET_MASK (OPT_STARTSET_MASK | OPT_STOPSET_MASK | OPT_AMPSET_MASK | OPT_PERIODSET_MASK | OPT_NUMSET_MASK)

  Pre-combined set of flags for checking if all needed command line options are set.

5.2.1 Detailed Description

Bit masks for encoding the flags for the various options that can be set on the command line.

Options these flags refer to are stored in progOptions::flags

5.3 User-requested help message

Sets up the text displayed when invoked with -h/–help.

Macros

- #define SUFFIX_EXE ".exe"

  Pick the correct suffix for referring to the execuatable in the help text.
- #define ANY_ALL_TEXT "If any of -s, -e, -n, and -p are supplied, they all must be supplied.\n"

  Text for warning when not supplying all of the required options.

Variables

- const char helpText [ ]

  The text displayed when invoked with -h/–help.

5.3.1 Detailed Description

Sets up the text displayed when invoked with -h/–help.

5.3.2 Macro Definition Documentation

5.3.2.1 #define SUFFIX_EXE ".exe"

Pick the correct suffix for referring to the execuatable in the help text.

On Windows builds, this will evaluate to '.exe' on Linux/Unix it will evaluate to the empty string. For documentation purposes the _WIN32 version will be shown.

5.3.3 Variable Documentation

5.3.3.1 const char helpText[ ]

Initial value:

```
= "Usage:  genAWGpattern" SUFFIX_EXE " [options]\n\
\n\
       genAWGpattern" SUFFIX_EXE " [options] -s <freq> -e <freq> -n <count>\n\
       -p <period> [-r|-a <amplitude>]\n\
\n\
Options:\n\
  -h | --help           Displays this information\n\
  -t | --template       Output blank template to " TEMPLATE_FILENAME "\n\
\n\
  -d | --debug          Output debug information.\n\
  -q | --quiet          Suppress normal output.  Does not suppress debug output\n\
\n\
  -i | --input-file     Path to an input file\n\
  -f | --clock-freq     MHz. Sets the target sample clock on the AWG\n\
\n\
Command Line Pulse Specification:\n\
  WARNING: " ANY_ALL_TEXT "\
  -s | --start-freq     MHz. Lowest frequency in pulse\n\
  -e | --end-freq       MHz. High frequency in pulse\n\
  -n | --number-freq    Total number of pulse teeth\n\
  -p | --tooth-period   ns. Time per tooth\n\
```

```
  -r | --random-amp      Randomly select amplitude for each tooth from [0.1, 1.0]\n\
  -a | --fixed-amp       Same amplitude for every tooth (range 0 to 1)\n\
\n\
\n\
Generates a file whose content is suitable for streaming directly over a serial\n\
connection to an AWG2040 to program it with a specified frequency pulse-train.\n\
Pulse parameters are read in from '" INPUT_FILENAME "' (Unless overridden by the '-i'\n\
option), OR are specified using the 'Command Line Pulse' set of options.\n\
\n\
If both '-i' and any of the 'Command Line Pulse' set are supplied, the last one\n\
will be honored.  Likewise the last of any repeated options are honored.\n"
```

The text displayed when invoked with -h/–help.

The help text to display for user-requested help.

The full text of the help message is printed here, including whatever platform-dependent changes need to be made. The text also depends on SUFFIX_EXE, TEMPLATE_FILENAME, ANY_ALL_TEXT, and INPUT_FILENAME.

5.4 genBinary subsystem return codes

Return codes used internally to indicate kinds of failures.

Macros

- #define GEN_BINARY_EPARSE -1

  An error occurred while parsing a value.
- #define GEN_BINARY_ERESIZE -2

  An error occurred while resizing an array.

5.4.1 Detailed Description

Return codes used internally to indicate kinds of failures.

Chapter 6

Data Structure Documentation

6.1 freqList Struct Reference

Holds all the information needed to describe a train of frequency pulses.

`#include <genBinary.h>`

Data Fields

- unsigned int freqCount

  The number of frequency pulses actually used in freqList, ampList, and durList.
- unsigned int actualSize

  The total number of spaces for values in the arrays freqList, ampList, and durList.
- double * freqList

  Array of frequency values, in MHz.
- double * ampList

  Array of relative amplitude values, on interval [0,1].
- double * durList

  Array of pulse durations, in ns.

6.1.1 Detailed Description

Holds all the information needed to describe a train of frequency pulses.

Stores pointers to arrays containing the frequencies, amplitudes, and durations of each pulse. The utilized and actual sizes of the arrays are actually stored. The arrays are in matched order, i.e. the first pulse represented by the 0th element of each array.

The documentation for this struct was generated from the following file:

- genBinary/genBinary.h

6.2 progOptions Struct Reference

Structure to hold values for command-line options.

```
#include <defOptions.h>
```

Data Fields

- uint32_t flags

  Bit flags used to indicate on-off states for various options. See Option encoding bitmasks.
- double amplitude

  Amplitude chosen for combs with constant output-amplitude. In range [0, 1] (fraction of max).
- double start_f

  Start frequency, sets the lowest frequency to be output in the series of pulses. In MHz.
- double stop_f

  Stop frequency, sets the highest frequency to be output in the series of pulses. In MHz.
- unsigned int num_f

  The number of individual pulse to generate.
- double clock_freq

  The sample output frequency. In MHz.
- double tooth_period

  The length of each pulse. In ns.
- char * inputPath

  C-string for a command-line specified frequency specification file path.

6.2.1 Detailed Description

Structure to hold values for command-line options.

Expected initialization found in OPT_INIT_VAL

See also

helpText
progOptions

The documentation for this struct was generated from the following file:

- defOptions/defOptions.h

Chapter 7

File Documentation

7.1 defOptions/defOptions.h File Reference

Functions and parameters for dealing with command-line options.

```
#include <inttypes.h>
#include "filenames.h"
```

Data Structures

- struct progOptions

  Structure to hold values for command-line options.

Macros

- #define OPT_RET_OK 0

  Indicates successful option parsing. Continue program operation.
- #define OPT_RET_ERR -1

  Indicates an error occurred while parseing options. Exit with non-zero code.
- #define OPT_RET_EXIT -2

  Indicates program should exit normally without further activity.
- #define OPT_TEMPLATE_MASK (1u << 0)

  Flag for requested template file output. 0 is unset, 1 is set.
- #define OPT_RANDAMP_MASK (1u << 1)

  Flag for using random amplitudes for output. 0 is unset, 1 is set.
- #define OPT_HELPREQ_MASK (1u << 2)

  Flag for user-requested help. 0 is unset, 1 is set.
- #define OPT_FROMCMD_MASK (1u << 15)

  Flag indicating user input frequency specification via command-line options. 0 is unset, 1 is set.
- #define OPT_STARTSET_MASK (1u << 8)

  Flag indicating user specified a start frequency, found in progOptions::start_f. 0 is unset, 1 is set.
- #define OPT_STOPSET_MASK (1u << 9)

  Flag indicating user specified a stop frequency, found in progOptions::stop_f. 0 is unset, 1 is set.

- #define OPT_AMPSET_MASK (1u << 10)

  Flag indicating user specified an amplitude. If OPT_RANDAMP_MASK is not also set, found in progOptions::amplitude. 0 is unset, 1 is set.

- #define OPT_PERIODSET_MASK (1u << 11)

  Flag indicating pulse duration is set, found in progOptions::tooth_period. 0 is unset, 1 is set.

- #define OPT_NUMSET_MASK (1u << 12)

  Flag indicating the number of teeth is set, found in progOptions::num_f. 0 is unset, 1 is set.

- #define OPT_ALLSET_MASK (OPT_STARTSET_MASK | OPT_STOPSET_MASK | OPT_AMPSET_MASK | OPT_PERIODSET_MASK | OPT_NUMSET_MASK)

  Pre-combined set of flags for checking if all needed command line options are set.

- #define OPT_INIT_VAL {0, 0.0, 0.0, 0.0, 0, 1024.0, 0.0, NULL}

  Initialization data for a progOptions instantiation.

Typedefs

- typedef struct progOptions progOptions_type

  Structure to hold values for command-line options.

Functions

- int parseOptions (int argc, char *argv[ ], progOptions_type *options)

  Takes command-line arguments and parses them.

- void printOptions (const progOptions_type *toPrint, const char *idStr)

  Takes a progOptions structure and prints a formatted version to stdout.

- void printBitSetting (uint32_t flags, unsigned int mask, const char *title)

  Takes a bitfield and mask and prints the selected bit's state to stdout.

Variables

- int g_opt_debug

  Whether -d/–debug has been set. 0 is unset, 1 is set.

- int g_opt_quiet

  Whether -q/–quiet has been set. 0 is unset, 1 is set.

- const char helpText [ ]

  The help text to display for user-requested help.

7.1.1 Detailed Description

Functions and parameters for dealing with command-line options.

Contains the functions used to handle parsing of input flags, printing out debug information about set options, and masks used to decode flags.

7.1.2 Typedef Documentation

7.1.2.1 typedef struct progOptions progOptions_type

Structure to hold values for command-line options.

Expected initialization found in OPT_INIT_VAL

See also

helpText
progOptions

7.1.3 Function Documentation

7.1.3.1 int parseOptions (int *argc*, char * *argv[]*, progOptions_type * *options*)

Takes command-line arguments and parses them.

Uses getopt_long() to process long & short flags passed in to the command line into appropriate settings. The flags and their values are described in helpText

If –debug is set, it will begin printing out options as they are set, and will print out the full options array via printOptions() when processing is complete.

Parameters

in	*argc*	argc as passed in to main()
in	*argv*	argv as passed in to main()
out	*options*	Pointer to the structure containing the set options from the command line input. Must be appropriately allocated before calling this function.

Returns

One of the values defined in Option subsystem return codes.

7.1.3.2 void printBitSetting (uint32_t *flags*, unsigned int *mask*, const char * *title*)

Takes a bitfield and mask and prints the selected bit's state to stdout.

Prints out the state as 'set' or 'unset', along with an optional title explaining what has been decoded.

Warning

Will happily accept a mask with multiple bits set, but will report only the state of the highest set bit.

Parameters

in	*flags*	A bitfield with an encoded value
in	*mask*	The bitmask for the value to decode

in	*title*	The name to print as a label for the decoded value. If a Null pointer is passed it will list only the bit number.

7.1.3.3 void printOptions (const progOptions_type * *toPrint*, const char * *idStr*)

Takes a progOptions structure and prints a formatted version to stdout.

The formatted output includes the values of any numeric option, as well as the decoded contents of the progOptions::flags field via calls to printBitSetting().

Parameters

in	*toPrint*	The options whose setting will be printed
in	*idStr*	A pointer to a string containing the name toPrint will be referred to on screen. If a Null pointer is passed, the display will default to "options"

7.2 defOptions/defOptions_int.h File Reference

Sets up global flags for debug and quiet options.

```
#include "filenames.h"
```

Macros

- #define SUFFIX_EXE ".exe"

  Pick the correct suffix for referring to the execuatable in the help text.
- #define ANY_ALL_TEXT "If any of -s, -e, -n, and -p are supplied, they all must be supplied.\n"

  Text for warning when not supplying all of the required options.

Variables

- int g_opt_debug = 0

  Whether -d/–debug has been set. 0 is unset, 1 is set.
- int g_opt_quiet = 0

  Whether -q/–quiet has been set. 0 is unset, 1 is set.
- const char helpText [ ]

  The text displayed when invoked with -h/–help.

7.2.1 Detailed Description

Sets up global flags for debug and quiet options.

7.3 defOptions/filenames.h File Reference

File names are hard-coded here for use throughout the program.

Macros

- #define TEMPLATE_FILENAME "template.txt"

  File to output the frequency specification template to.
- #define INPUT_FILENAME "freqSpec.txt"

  File to read for frequency specification input.
- #define OUTPUT_ROOT "awgOutput"

  File name stem used.

7.3.1 Detailed Description

File names are hard-coded here for use throughout the program.

7.4 defOptions/templateContents.h File Reference

Content of the template file printed with the -t or –template flags.

Variables

- const char templateStr [ ]

  The full template file contents.

7.4.1 Detailed Description

Content of the template file printed with the -t or –template flags.

Contained here in its entirety to allow easy editing in the future.

7.4.2 Variable Documentation

7.4.2.1 const char templateStr[ ]

Initial value:

```
="# Lines starting with '#' are comments\n\
# All other lines should be in the following format\n\
# freq [MHz], duration [ns], amplitude [relative, [0,1] ]\n\
# durations are a goal, not a guarantee, will be rounded to nearest 1/2 cycle of freq (including 0!)\n\
# amplitudes relative scales, where 1 is full-scale.\n\
# Output is only 8-bit, so effective amplitude resolution is 1/127 ~ 0.008\n\
#\n\
# Example line of 111 MHz for 30ns, with 3/4 full scale amplitude\n\
# 100, 30, 0.75\n\
"
```

The full template file contents.

The template file will be written to disk when the full program is called with the -t or –template flags. This gives an example of how to specify the output pulses, as well as a description of how it will actually work internally.

7.5 genBinary/genBinary.h File Reference

Handles generating AWG command string from user-supplied options.

Data Structures

- struct freqList

  Holds all the information needed to describe a train of frequency pulses.

Macros

- #define AWG_ZERO_VAL 127

  The integer value that corresponds to a zero-volt output on the AWG.
- #define PI 3.141592653589793

  Pi to double precision.
- #define TWO_PI 6.283185307179586

  Twice pi to double precision.
- #define DEFAULT_FREQ_LIST_SIZE 8

  Default frequency list size, tradeoff between minimum memory footprint and overhead for expansion if too small.
- #define GEN_BINARY_EPARSE -1

  An error occurred while parsing a value.
- #define GEN_BINARY_ERESIZE -2

  An error occurred while resizing an array.

Typedefs

- typedef struct freqList freqList_type

  Holds all the information needed to describe a train of frequency pulses.
- typedef freqList_type * freqList_ptr

  Pointer to a freqList.

Functions

- freqList_ptr blankFreqList ()

  Allocates an empty freqList.
- void freeFreqList (freqList_ptr toFree)

  Frees all memory associated with the referenced freqList.
- int allocSubLists (freqList_ptr toSet, unsigned int nFreqs)

  Allocates new arrays of fixed size for the referenced freqList.
- int setFreqList (freqList_ptr toSet, const double start_f, const double stop_f)

  Sets the freqList::freqList to frequencies evenly spaced over an interval.
- int setFixedAmp (freqList_ptr toSet, const double amplitude)

  Sets the amplitude of every pulse.
- int setRandAmp (freqList_ptr toSet)

  Sets every pulse to a random amplitude.
- int setFixedDur (freqList_ptr toSet, const double duration)

Sets the duration of every pulse to the same value.

- unsigned int pointsToHalfCycle (double targetDuration, double pointInterval, double frequency)

  Find the number of points that finishes a half cycle closest to a target duration.

- unsigned int * pointCounts (const freqList_ptr freqList, const double pointInterval)

  Allocates an array holding the number of samples for every pulse.

- unsigned char * genPointList (const freqList_ptr freqList, const unsigned int *pointCounts, const double pointInterval, unsigned long *finalCount)

  Generates the full waveform, including continuity and length checks.

- unsigned char * genWavePts (double freq, double amp, unsigned int numPts, double pointInterval, unsigned char *startPtr)

  Generate the output samples for an individual pulse.

- ssize_t myGetLine (char **bufferPtr, size_t *bufferSize, FILE *fp)

  A custom, getLine implementation.

- int resizeFreqList (unsigned int newSize, freqList_ptr *toResize)

  Resizes all sublists of the pointed-to freqList to the specified length.

- freqList_ptr readSpecFile (const char *inPath)

  Parse the file at the passed path for a pulse train specification.

- int parseLine (char *lineBuf, freqList_ptr destList)

  Takes text specifying a frequency pulse and adds it to the end of a freqList.

- int writeToFile (const char *rootName, const unsigned char *ptsList, const unsigned long numPtrs, const double clockFreq)

  Writes byte stream suitable for transmission to the AWG to a file.

- int writeSummaryFile (const char *rootName, const freqList_ptr freqList, const unsigned int *pointCounts, const double clock_freq)

  Writes a human-readable text file describing the contents of the generated points file.

7.5.1 Detailed Description

Handles generating AWG command string from user-supplied options.

Device documentation is hosted at `exodus.poly.edu/~kurt/`:

- `User Manual (pdf)`
- `Programmer's Manual (pdf)`

7.5.2 Typedef Documentation

7.5.2.1 typedef struct freqList freqList_type

Holds all the information needed to describe a train of frequency pulses.

Stores pointers to arrays containing the frequencies, amplitudes, and durations of each pulse. The utilized and actual sizes of the arrays are actually stored. The arrays are in matched order, i.e. the first pulse represented by the 0th element of each array.

7.5.3 Function Documentation

7.5.3.1 int allocSubLists (freqList_ptr *toSet,* unsigned int *nFreqs*)

Allocates new arrays of fixed size for the referenced freqList.

Allocates all three freqList sub-arrays, and sets both freqCount and actualSize to that length.

Warning

Does not check if there are already arrays referenced, so check the pointers first. If they are non-null, calling this function is a memory leak.

Parameters

in,out	*toSet*	Pointer to the freqList whose arrays are being allocated.
in	*nFreqs*	The length of the arrays to allocate.

Returns

0 on success
-1 on failure.

7.5.3.2 freqList_ptr blankFreqList ( )

Allocates an empty freqList.

Default values are 0 or NULL, as appropriate.

Returns

A pointer to the newly allocated freqList
NULL pointer on failure.

7.5.3.3 void freeFreqList (freqList_ptr *toFree*)

Frees all memory associated with the referenced freqList.

Note

Cannot marker the pointer as NULL, the caller must do this.

Parameters

in	*toFree*	A pointer to the freqList to be freed.

7.5.3.4 unsigned char* genPointList (const freqList_ptr *freqList,* const unsigned int * *pointCounts,* const double *pointInterval,* unsigned long * *finalCount*)

Generates the full waveform, including continuity and length checks.

Takes the pulse description and other key parameters and generates all the output samples. Because we require continuity of the function and its derivative, we check for the final slope, and duplicate and flip the entire waveform if necessary.

Finally, the AWG itself requires waveforms to have a multiple of 32 samples. If this didn't already happen, we duplicate the entire waveform as many times as needed to meet this condition.

Warning

No check on total length fitting in memory is performed. However unlikely, if you exceed the total number of points allowed, I don't know what the AWG will do.

Parameters

in	*freqList*	A pointer to the the freqList describing the pulse train.
in	*pointCounts*	An array holding the length of each pulse in output samples.
in	*pointInterval*	The output sample period, in ns.
in,out	*finalCount*	Pointer to memory to hold the total number of points in the final waveform.

Returns

Pointer to the array holding all of the output waveform's points

7.5.3.5 unsigned char* genWavePts (double *freq*, double *amp*, unsigned int *numPts*, double *pointInterval*, unsigned char * *startPtr*)

Generate the output samples for an individual pulse.

Fills the next numPts unsigned chars starting at startPtr with a sin wave with the passed parameters.

Uses AWG_ZERO_VAL to set the offset of the waveform from zero.

Warning

No bounds checking for the array is performed internally, because the function doesn't have access to the information it needs to do that.

Parameters

in	*freq*	The frequency of the pulse, in MHz
in	*amp*	The amplitude of the pulse, should be in the range [-127.0, 127.0]
in	*numPts*	The number of samples to output
in	*pointInterval*	The output sample period, in ns.
in	*startPtr*	The first location to put a point in.

Returns

A pointer to the position in the array *after* the last one it filled.

7.5.3.6 ssize_t myGetLine (char ** *bufferPtr*, size_t * *bufferSize*, FILE * *fp*)

A custom, getLine implementation.

See GNU Getline Documentation

Reads a line from the specified file into the provided buffer. Will automatically resize the buffer for lines that will not fit.

Parameters

in,out	*bufferPtr*	Pointer to the buffer containing the line that was read
in,out	*bufferSize*	Total size, in bytes, of the buffer at bufferPtr.
in,out	*fp*	An open file pointer to the file we're reading from.

Returns

The number of characters we read into the buffer, including the newline (if present), but not the terminating NULL character.
-1 on error, including EOF.

7.5.3.7 int parseLine (char * *lineBuf*, freqList_ptr *destList*)

Takes text specifying a frequency pulse and adds it to the end of a freqList.

The format is described in templateStr in templateContents.h

Parameters

in,out	*lineBuf*	The buffer containing the line to be parsed
in,out	*destList*	The freqList we'll add any specified new pulse.

Returns

0 on success
GEN_BINARY_ERESIZE if we failed to make room for the new pulse.
GEN_BINARY_EPARSE if we couldn't parse the line (likely because it was malformed)

7.5.3.8 unsigned int* pointCounts (const freqList_ptr *freqList*, const double *pointInterval*)

Allocates an array holding the number of samples for every pulse.

Utilizes information in freqList to allocate an array, the fills it via calls to pointsToHalfCycle().

Parameters

in	*freqList*	A pointer to the the freqList describing the pulse train.
in	*pointInterval*	The output sample period being used, in ns.

Returns

A pointer to the array of points on success
NULL on failure.

7.5.3.9 unsigned int pointsToHalfCycle (double *targetDuration*, double *pointInterval*, double *frequency*)

Find the number of points that finishes a half cycle closest to a target duration.

This allows us to always change frequencies at a zero crossing. This makes phase continuity is easy to enforce, even with varying amplitudes. Additionally, the last point will always be before the zero crossing, so the first point of the next pulse will never result in discontinuity.

Parameters

in	*targetDuration*	How long we'd like the pulse to last, in ns.
in	*pointInterval*	The duration of an individual output sample, in ns.
in	*frequency*	The frequency for this pulse, in MHz.

Returns

The number of output samples to complete a half-cycle nearest the targetDuration.

7.5.3.10 freqList_ptr readSpecFile (const char * *inPath*)

Parse the file at the passed path for a pulse train specification.

Reads the file line by line via myGetLine() and parses the contents via parseLine(), attempting to build the pulse train specification.

Parameters

in	*inPath*	A string containing the path to the file to read the spec's from.

Returns

A pointer to the freqList from parsing the file
NULL on failures:

- Unable to (re)allocate buffers
- No pulses specified in the file
- Error reading the file

7.5.3.11 int resizeFreqList (unsigned int *newSize*, freqList_ptr * *toResize*)

Resizes all sublists of the pointed-to freqList to the specified length.

Specifically freqList has array members freqList, ampList, and durList.

Possible reasons for returning an error value:

- Passing a NULL pointer, or a pointer to a NULL pointer
- realloc() call failed for any of the sub-lists.

Parameters

in	*newSize*	New length for all of the sub-lists
in,out	*toResize*	Points to a freqList_ptr of the freqList for resizing.

Returns

0 on success
-1 on error

7.5.3.12 int setFixedAmp (freqList_ptr *toSet*, const double *amplitude*)

Sets the amplitude of every pulse.

Number of pulses is pulled from toSet's freqList::freqCount member.

Parameters

in,out	*toSet*	Pointer to the freqList to change.
in	*amplitude*	The value of the amplitude to use.

Returns

0, always.

7.5.3.13 int setFixedDur (freqList_ptr *toSet,* const double *duration*)

Sets the duration of every pulse to the same value.

Number of pulses is pulled from toSet's freqList::freqCount member.

Parameters

in,out	*toSet*	Pointer to the freqList to change.
in	*duration*	Duration of each pulse, in ns.

Returns

0, always

7.5.3.14 int setFreqList (freqList_ptr *toSet,* const double *start_f,* const double *stop_f*)

Sets the freqList::freqList to frequencies evenly spaced over an interval.

Uses the freqCount memeber of the referenced freqList for the number of frequencies, and determines the bounds from the passed parameters.

Parameters

in,out	*toSet*	Pointer to the freqList to change.
in	*start_f*	The first frequency in the pulse train
in	*stop_f*	The final frequency in the pulse train.

Returns

0, always.

7.5.3.15 int setRandAmp (freqList_ptr *toSet*)

Sets every pulse to a random amplitude.

Number of pulses is pulled from toSet's freqList::freqCount member.

Amplitudes are chosen on the interval [0.1, 1.0] (or [13, 127]). Uses rand() and calls srand(time(NULL)) for random number generation.

Parameters

in,out	*toSet*	Pointer to the freqList to change.

Returns

0, always

7.5.3.16 int writeSummaryFile (const char * *rootName,* const freqList_ptr *freqList,* const unsigned int * *pointCounts,* const double *clock_freq*)

Writes a human-readable text file describing the contents of the generated points file.

File will be output as "\<rootName\>_desc.txt" E.g. a rootName of "test" would result in a file "test_desc.txt"

The file contains the frequency, amplitude, duration, and number of samples for each pulse, in order. It also lists the output sample frequency (and period) used.

See writeToFile() for actual contents of points file.

Parameters

in	*rootName*	The base of the filename we're saving to.
in	*freqList*	freqList describing the generated pulse train.
in	*pointCounts*	Total number of points for each pulse in the output waveform
in	*clock_freq*	The output sample frequency

Returns

0 on success
-1 on failure

7.5.3.17 int writeToFile (const char * *rootName,* const unsigned char * *ptsList,* const unsigned long *numPtrs,* const double *clockFreq*)

Writes byte stream suitable for transmission to the AWG to a file.

File will be output as "\<rootName\>_points". E.g. a rootName of "test" would result in a file "test_points"

See writeSummaryFile() for human-readable description.

Commands generated to save the waveform on the AWG as "GPIB.WFM", then set up the correct format for the transfer, send the points, set the correct clock frequency, and ask for confirmation information back.

Parameters

in	*rootName*	The base of the filename we're saving to.
in	*ptsList*	Pointer to the array of output samples, already stored in AWG format
in	*numPtrs*	Total number of points in the output waveform
in	*clockFreq*	The output sample frequency

Returns

0 on success
-1 on failure

Index