SYSTEM ONE

C LANGUAGE
FUNCTION REFERENCE

AUDIO PRECISION
MICROSOFT C 6.00
FUNCTION REFERENCE

November 27, 1991
Software Version 2.10A
Document Revision 2

This document describes function libraries for control of the Audio Precision System One hardware and is intended as a reference for experienced programmers who are familiar with the operation of System One.

P.O.Box 2209, Beaverton, Oregon 97075 U.S.A
Telephone (503) 627-0832
FAX (503) 641-8906
Telex 283957 AUDIO UR

TABLE OF CONTENTS

Microsoft C 6.00 Function Reference

NOTE TO USERS OF NON-DSP SYSTEMS

This upgrade of the Audio Precision System One C Library was done exclusively to support the DSP module. The code for non-DSP modules has not changed.

Since the last revision, the total code size has grown substantially and the libraries now only support Microsoft C in the Medium and Large models. (use of small code models is no longer practical). QuickC is not supported.

Because the DSP uses the file handling functions of the C standard library, other languages are not supported.

It is recommended that you upgrade to this version ONLY if you are using the DSP module

NOTE TO USERS OF SYSTEMS INCLUDING THE DSP

The DSP is the most complicated of the System One modules. Programming the DSP therefore involves more effort than for any of the other modules.

This manual does not attempt to describe the architecture of the DSP hardware. It is essential that the Audio Precision DSP User's Manual be read and understood before attempting to program the DSP.

It also highly recommended that new test setups be designed and verified using the S1.EXE software included with System One.

Note that since the last revision, the total code size has grown substantially and the libraries now only support Microsoft C in the Medium and Large models. (use of small code models is no longer practical). QuickC is not supported.

Because the DSP uses the file handling functions of the C standard library, other languages are not supported.

INTRODUCTION

This is a reference manual for programmers using the Audio Precision System One C language libraries. These libraries contain functions for support of the System One hardware.

This manual assumes that you are familiar with System One operation, your programming language, its compiler, and with MS-DOS.

The System One Function Libraries contain over 200 functions providing complete control over the System One hardware. The functions include hardware settings, readings, triggers, "reading ready" queries, and status queries. Also included is a sophisticated settling algorithm and the functions to control it. This algorithm allows the simultaneous settling of the various different readings available from the hardware, thus making quality data easy to obtain.

The libraries do not include graphics or support RS-232 ("remote" instruments) communications.

The functions are contained in libraries. The standard memory models are included and specifically support the following compilers:

```
Microsoft C Compiler Version 6.00
```

Information more up to date than is in this printed manual may be contained in the file README.DOC on the distribution diskettes.

DISTRIBUTION

You should find the following files on your distribution diskette:

Files	Description
DISK 1:	
README.DOC	Most up-to-date information
MAKE_AP.BAT	Batch file to compile & link with Microsoft C
AP.H	Header to be included with all C programs (Please read this header)
APDSP.H	Header for C programs that use the DSP (Please read this header)
MLIB_AP.LIB	Microsoft medium model library
LLIB_AP.LIB	Microsoft large model library
SAMPLE1.C	A simple frequency sweep
SAMPLE2.C	A sweep with limits added
SAMPFFTG.C	Sample using FFTGEN.DSP
SAMPFAST.C	Sample using FASTEST.DSP and ISO31.WAV
SAMPFFTS.C	Sample using FFTSLIDE.DSP
SAMPGENA.C	Sample using GENANLR.DSP
SAMPHARM.C	Sample using HARMONIC.DSP

(The .DSP and .WAV files are included with all DSP versions of System One)

INSTALLATION

There is no formal installation procedure included on the distribution diskettes. Installation is accomplished by simply copying the files you will be using onto your hard or floppy disk.

The .BAT file on the distribution disk assumes that the libraries are in your current working directory and that a PATH is specified to the compilers.

USAGE - GENERAL INFORMATION

There are two functions that must always be called in any program using the System One library.

First, before any other functions are called, the System One library must be initialized with a call to ap_setup.

Second, at the end of the program, ap_teardown must be called to reset the System One hardware and most importantly, to remove interrupt vectors that are installed by ap_setup.

> **Besides initializing the hardware settings, ap_setup modifies the timer interrupt vector INT8. This vector must be restored at program exit using the ap_teardown function.**
>
> **Failure to call ap_teardown at program exit will usually result in a system crash.**

The best way to guarantee that ap_teardown is called is to place explicit calls at all obvious program exit points and to use the language facilities available to trap abnormal exits.

CONTROL-BREAK exits are trapped with the signal function.

Other non-obvious program exit paths that may be possible are math errors and stack overflows. Math errors may be handled using the _control87 function. Stack overflows can only be avoided by making sure sufficient stack space is allocated during the linking phase.

Also, be sure to include the Audio Precision header files in your program. (AP.H and APDSP.H). These headers include the proper function declarations insuring that the correct argument types are passed and returned. They also include #defines that make programming System One much easier.

A sample program skeleton might appear as follows:

```
#include <stdio.h>
#include <signal.h>
#include <float.h>                  /* for _control87 declarations
   */
#include "ap.h"
#include "apdsp.h"                  /* if DSP module is used    */

main()
{
   extern int user_break();

   signal(SIGINT, user_break);      /* install break handler    */
   _control87( MCW_EM, MCW_EM);     /* mask floating point
exceptions  */

   ap_setup();                      /* initialize System One    */

   if( !ap_exist() || !gen_exist() || !lvf_exist() ||
!dsp_exist()) {
      printf( "\n\nSystem One not properly installed or not
powered on.\n");
      ap_teardown();                /* cleanup before program exit
   */
      exit( 1);                     /* abnormal exit            */
   }

   /* The body of your program                                  */

   ap_teardown();
   exit(0);                         /* normal exit              */
}

int user_break()
{
   ap_teardown();                   /* cleanup before program exit
   */
   printf( "\n\nProgram terminated by user.\n");
   exit(1);                         /* abnormal exit            */
}
```

USAGE - MICROSOFT C 6.00

C programs using the System One libraries should always include the header file AP.H. APDSP.H should be included if the DSP module is used. See the sample C files included on the distribution diskettes.

A sample .BAT file (MAKE_AP.BAT) is provided to compile and link with Microsoft C. Execute MAKE_AP.BAT with no arguments for usage information.

Using "cl" directly, a typical invocation is as follows:

```
cl  /AM  sample1.c  mlib_ap.lib
```

Note that the medium memory model is used as an example here. Although the large memory model may be used, with use of the "far" data keyword, the large memory model with its necessarily inefficient code is rarely needed.

Be sure to specify the proper System One library:

```
MLIB_AP.LIB    medium model
LLIB_AP.LIB         large model
```

The System One library must be searched first during linking. It makes use of other functions provided in the standard libraries included with the compiler. This is accomplished automatically when using the "cl" command as above. When using the linker separately, this is accomplished by specifying it before any other libraries on the linker command line.

Also the System One library uses floating point math. It was compiled using the /FPi option which generates in-line instructions for the 8087 or 80287 math coprocessor. The emulator library (mLIBCE.LIB) is used by default, but the 8087/80287 library (mLIBC7.LIB) may be specified instead at link time.

DIFFERENCES BETWEEN VERSIONS

Version 2.10:

Support for FASTEST.DSP version 2.10-07 has been added.

The APDSP.H file is updated to match version 2.10-07 of all DSP programs.

Only Microsoft C version 6.00 is supported by this version.

The lvf_read_stat_a, b and gainamp functions are no longer documented. The System One hardware returns only transient status information which makes these functions difficult to use.

Version 2.00:

Support for DSP has been added.

The dus_read_struct function is added to handle the now ten possible readings to be settled.

Only Microsoft C versions 5.00 and newer are supported by this version. This is because the DSP module contains file handling functions that are incompatable with other languages. Also, because of the library's code size, only Medium and Large memory models are supported.

Version 1.60C

GEN_READ_SETTLED was returning 0 for settled instead of 1. This has been fixed.

GEN_READ_FREQ was sometimes returning -1E34 when it should not have been. This has been fixed.

QuickBASIC was getting unresolved references when trying to make a .EXE file with MLIB_AP.LIB. A new library MLIB_APB.LIB was created to resolve these references.

Version 1.60:

All functions that previously returned floats now return doubles.

Support for the BUR-GEN (Burst, Noise, and Squarewave) hardware option has been added. The corresponding new functions are:

```
gen_set_binterval
gen_set_bofflvl
gen_set_bon
gen_set_bmode
gen_set_ntype
```

With the Burst waveform, the polarity capability of the LVF is now available with the following new functions:

```
lvf_rdy_polarity
lvf_read_polarity
```

Support for the DCX hardware has been added. The corresponding new functions are:

```
all  DCX  functions.

dus_set_dcvtol
dus_set_ohmtol
dus_set_dintol
dus_set_dcvfloor
dus_set_ohmfloor
dus_set_dinfloor
dus_set_dmmpts
dus_set_dinpts
```

A new set of functions has been added to determine the existence of the hardware. The corresponding new functions are:

```
ap_exist
gen_exist
lvf_exist
swi_exist
dcx_exist
```

Some existing functions have been enhanced to support the new BUR-GEN and DCX hardware:

```
ap_setup
ap_read_byte
ap_write_byte
ap_get_errcode
gen_init
gen_set_mode
```

```
dus_init
dus_read
```

Support for "A" version LVFs has been added. The following functions have been enhanced to support this:

```
lvf_measure
lvf_monitor_a
lvf_read_mode
lvf_set_mode
lvf_set_weight
```

Dus_set_slope was renamed to dus_set_shape to more properly describe this function.

The switcher functions did not always function correctly in version 1.41. This has been fixed.

Version 1.41:

Support for Switchers and Wow & Flutter hardware has been added.

The settling algorithm used on the Settling panel in S1.EXE is now available in the library.

NAME

ap_setup -- general initialize

SYNOPSIS

void ap_setup(void);

DESCRIPTION

This function does all the setup necessary to make any of the other function calls.

Specifically, it does the following things:

1. Calls ap_low_setup which:
   a. Installs a timer interrupt vector to support the timing functions.
   b. Verifies the existence of the PCI card.
   c. Calls ap.reset to reset the bus (APIB)

2. Calls gen_select to select gen number 1
3. Calls lvf_select to select lvf number 1
4. Calls dcx_select to select dcx number 1

5. Calls swi_init to init and set address = 14
   (this must be done before the other inits)

6. Calls dsp_init to init and set address = 3
7. Calls gen_init to init and set address = 2
8. Calls lvf_init to init and set address = 0
9. Calls dcx_init to init and set address = 11
10. Calls dus_init to init DUS parameters

Besides initializing the hardware settings, ap_setup modifies the timer interrupt vector INT8. This vector must be restored at program exit using the ap_teardown function.

This function MUST be called before any other calls are attempted.

If you need to setup more than one of each type of instrument, you will need to call the select and initialize functions individually.

See the sample program skeletons in the general usage section of this manual.

Since this function initializes all instruments, it causes most of the System One library to be included with your program. If you are not using some instruments (such as SWI and DCX), program size may be reduced by writing your own subset of ap_setup. The C language source for this function is included on the distribution diskettes as SETUP.C. See the sample C programs for a usage example.

SEE ALSO

ap_reset, gen_select, gen_init, lvf_select, lvf_init, swi_init, dus_init, dcx_init, ap_teardown

Sample program skeletons in the general usage section of this manual.

NAME

ap_teardown -- general clean up

SYNOPSIS

void ap_teardown(void);

DESCRIPTION

This function does all the cleanup necessary to safely exit the program.

Specifically, it does the following things:

1. Removes a timer interrupt vector that was supporting the timing functions.
2. Calls ap_reset to reset the bus

At the end of the program, ap_teardown must be called to reset the System One hardware and most importantly, to remove interrupt vectors that are installed by ap_setup.

Failure to call ap_teardown will usually result in a system crash.

The best way to guarantee that ap_teardown is called is to place explicit calls at all obvious program exit points and to use the language facilities available to trap abnormal exits.

CONTROL-BREAK exits are trapped with the signal function.

Other non-obvious program exit paths that may be possible are math errors and stack overflows. Math errors may be handled using the _control87 function. Stack overflows can only be avoided by making sure sufficient stack space is allocated during the linking phase.

See the sample program skeletons in the general usage section of this manual.

SEE ALSO

ap_setup, ap_reset

Sample program skeletons in the general usage section of this manual.

NAME

ap_reset -- general reset

SYNOPSIS

void ap_reset(void);

DESCRIPTION

This function pulses the reset port on the AP bus to force all instruments into a reset condition. The actual actions are dependent on each module but in general the instruments are reset to a condition with outputs off, inputs unterminated, and not returning readings (in safe states).

This function need be used only under several conditions.

1. When the Audio Precision Interface Bus has become disconnected and is then subsequently reconnected. ap_reset followed by the restore functions (such as gen_restore, etc.) will reestablish the hardware. The high level function UTIL RESTORE in S1.EXE is accomplished by this technique.

2. As an emergency shutdown of the System One chassis during conditions of operation with a faulty device under test.

3. To reset the hardware when exiting your program.

Ap_reset is called by ap_setup.

NAME

ap_exist -- determine the type of PCI card available.

SYNOPSIS

exist = ap_exist(void)

int exist; type of PCI card

DESCRIPTION

This function determines if a PCI card is installed and the type of PCI card.

AP_SETUP() must be called before this function is called.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

RETURNS

-1 if ap_setup has not been previously called.
0 if no PCI is installed
1 if PCI-1 is found
2 if PCI-2 is found at address 238 hex
3 if PCI-2 is found at address 298 hex
4 if PCI-2 is found at address 2B8 hex
5 if PCI-2 is found at address 2D8 hex

SEE ALSO

ap_setup

NAME

ap_get_errcode -- get and reset error code

SYNOPSIS

err = ap_get_errcode(void);

int err; returned error code

DESCRIPTION

This function returns and resets the System One error code. This error code will be set by the failure of many of the instrument settings functions.

See each individual function for how it uses the error code. It can be assumed that if the error code is non-zero, an error occurred.

```
The following list describes the possible states of the error
code:

0  = No error
3  = Minimum generator amplitude attempted or minimum burst level
attempted
4  = Maximum generator amplitude attempted or maximum burst level
attempted
5  = Minimum generator frequency attempted
6  = Maximum generator frequency attempted
44 = A string was passed that was not understood.
49 = Burst interval less than on cycles or on cycles greater than
interval attempted
52 = Maximum DC Volts output attempted
53 = Minimum DC Volts output attempted

69 = DSP is not returning readings
70 = DSP Host Vector not available
71 = DSP Transmit Register not available
72 = DSP receive Register not available
90 = DSP does not respond to reset

81 = Error loading DSP program
91 = File specified not a valid DSP file

82 = AES string format not correct
83 = Conflict with Minimum setting
84 = Conflict with Maximum setting
```

```
93 = Waveform buffer specification not understood
94 = Waveform transfer not supported by this DSP program

95 = Sampling rate not supported by this DSP program
96 = DSP reading unit selected must have input source from ANLR
97 = Ratio unit not supported for DSP readings from ANLR-A or
ANLR-B
```

RETURNS

This function returns the present value of the error code.

Note the error code is reset by this function so that each error will be returned only once.

NAME

ap_any_ready -- determine if any reading is available.

SYNOPSIS

ready = ap_any_ready(void);

int ready; 1 if any reading is ready, 0 if not

DESCRIPTION

This function determines if any reading is ready from the Audio Precision Interface Bus by polling a status bit on the PCI card. This bit is set by any module with a reading ready, thus allowing a single ready check to be made instead of polling everything on the bus.

Once an "any ready" is found, the individual ready functions may be used to determine which module has the reading available.

The status bit is reset by this function so that any given "ready" will be returned only once.

RETURNS

Returns a 1 if any reading is ready, 0 otherwise.

NAME

ap_read_byte -- input byte from AP bus port
ap_write_byte -- output byte to AP bus port

SYNOPSIS

```
data = ap_read_byte( address)
ap_write_byte(  data, address)

int address;            ap port address
int data;               data byte (upper bits ignored)
```

DESCRIPTION

These functions read or write a byte at the specified ap port address.

> **Normally, there is no need to read and write directly to AP bus ports. These functions are only needed when custom hardware is present.**

Addresses from 0 to 255 are allowed.

System One instruments are set by "module number". The module number multiplied by 16 is the actual base address for the instrument. Each instrument normally occupies a block of 16 addresses. (or multiples of 16)

The default module numbers and addresses for existing instruments are:

Module Name	Module Number	Addresses
LVF	0	00 - 15
DIS	1	16 - 31
GEN	2	32 - 47
DSP	3	48 - 63
DCX	11	176 - 191
SWI	14	224 - 255

RETURNS

ap_read_byte returns a byte read from the port as an int from 0 to 255

ap_write_byte returns nothing.

NAME

start_longtimer -- start a precision timer

fread_longtimer -- read the timer

SYNOPSIS

```
void start_longtimer( void);

time = fread_longtimer( void);

double time;    time in seconds since last start call
```

DESCRIPTION

The longtimer functions utilize both the standard PC timer interrupt and the high resolution timer present on the System One PCI card. As a result the longtimer has a resolution of 241 microseconds and usable timing period of 2730 days (7.47 years).

Note that because of other interrupts operating in the PC, do not expect accuracy better than about 1 millisecond.

RETURNS

fread_longtimer returns the time in seconds since the last start_longtimer call.

SEE ALSO

fdelay

NAME

fdelay -- delay for a precision time.

SYNOPSIS

void fdelay(msec);

double msec; time to delay in milliseconds

DESCRIPTION

This function simply waits the specified time then returns.

The fdelay function utilizes both the standard PC timer interrupt and the high resolution timer present on the System One PCI card. As a result the delay has a resolution of 241 microseconds.

The maximum delay allowed is 15000 milliseconds (15 seconds).

Note that because of other interrupts operating in the PC, do not expect accuracy better than about 1 millisecond.

RETURNS

None.

SEE ALSO

start_longtimer, fread_longtimer

NAME

gen_select -- select the active generator

SYNOPSIS

new_gennum = gen_select(gennum);

int gennum;	generator reference number
int new_gennum;	generator number actually set

DESCRIPTION

This function selects which generator will be used as the object of all other generator calls.

This function should be used before any other generator function calls, including gen_init, are used.

> **Note that gen_select is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one generator. Using multiple GENs in a system requires modification of the additional GEN circuit boards to change addresses. Contact Audio Precision for further information**

The software internally maintains 4 different sets of generator data. This function simply selects which set of data to use until gen_select is called again.

At present, generators 1 to 4 are supported. The passed value must be from 1 to 4. If it is out of this range, no new selection is made.

RETURNS

The generator number selected is returned.

SEE ALSO

gen_restore, gen_init, ap_setup

NAME

gen_init -- initialize generator

SYNOPSIS

new_modnum = gen_init(modnum);

int modnum;	module number
int new_modnum;	module number actually set

DESCRIPTION

This function initializes the generator to a known state. It must be used after power-up.

> **Note that gen_init is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one generator. Using multiple GENs in a system requires modification of the additional GEN circuit boards to change addresses. Contact Audio Precision for further information**

The selected generator data structure is set to default conditions, then the generator hardware is set to match using the gen_restore function.

This function also sets the module number of the selected generator. (see gen_select) The module number must be between 0 and 15, else the default of 2 is used.

> **The default module number of 2 should be used unless you have a system specifically modified to have multiple generators.**

Default initializations:

1. Set function to SINE.
2. Set outputs to 50 ohm balanced and off.
3. Set channel B polarity to normal.
4. Set output floating (ground off).
5. Set frequency to 1 kHz, no autocal. (see gen_autocal)
6. Set amplitude to 1 Volt.
7. Set monitor off.
8. Set IMD frequency to 60 Hz.
9. Set burst mode to normal.

10. Set burst interval to 3 cycles.
11. Set burst on to 1 cycle.
12. Set burst level to .0097 % (-80.25 dB) (this is the minimum allowed level).
13. Set noise to fast (pseudo random).

RETURNS

The module number used is returned.

SEE ALSO

gen_restore, gen_select, ap_setup

NAME

gen_restore -- restore generator hardware

SYNOPSIS

void gen_restore(void);

DESCRIPTION

This function restores the generator hardware to the present state of the software.

This function should be used if the hardware loses power or becomes disconnected from the computer.

This function is also used by the gen_init function.

SEE ALSO

gen_init

NAME

gen_exist -- determine if the GEN hardware is available.

SYNOPSIS

exist = gen_exist(void);

int exist; 1 if GEN exists, 0 if not

DESCRIPTION

This function determines if the selected generator is connected and powered on.

The generator status is updated each time gen_restore called. This function returns that status. Note that ap_setup and gen_init also call gen_restore and so update the status.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

If the generator is found not to exist (but should), prompt the operator to remedy the situation, then call gen_restore before calling gen_exist again.

RETURNS

Returns a 1 if the selected generator exists, 0 otherwise.

NAME

gen_autocal -- calibrate the generator frequency each time the frequency is set.

SYNOPSIS

void gen_autocal(on_off);

int on_off; 1 for autocal, 0 for not

DESCRIPTION

This function causes the generator to automatically perform a frequency calibration cycle each time the frequency is set.

This is identical to calling gen_freqcal after each gen_set_freq except that gen_set_freq will return the exact frequency passed if autocal is enabled.

Note that this function does not cause an immediate frequency calibration. The calibration will be done at the next call to gen_set_freq.

RETURNS

None

INITIAL STATE

Autocal is set to OFF by gen_init.

SEE ALSO

gen_freqcal, gen_set_freq

NAME

gen_freqcal -- calibrate generator freq from counter

SYNOPSIS

void gen_freqcal(void);

DESCRIPTION

This function reads the generator's onboard frequency counter then fine tunes to correct for any error found.

The frequency last passed to gen_set_freq is used as the target frequency for calibration.

RETURNS

None.

SEE ALSO

gen_autocal, gen_set_freq

NAME

gen_monitor_a -- set generator channel A to monitor.
gen_monitor_b -- set generator channel B to monitor.

SYNOPSIS

```
void gen_monitor_a( on_off);
void gen_monitor_b( on_off);

int on_off;                    1 for monitor, 0 for not
```

DESCRIPTION

These functions connect channel A or channel B to the internal monitor connection. It is generally necessary to also set the corresponding LVF channel to monitor.

RETURNS

None.

INITIAL STATE

Monitors are set to OFF by gen_init.

SEE ALSO

lvf_monitor_a, lvf_monitor_b

NAME

gen_trigger -- trigger a new reading with delay

SYNOPSIS

void gen_trigger(msec);

double msec; trigger delay time in milliseconds

DESCRIPTION

Causes the generator to wait x milliseconds then restart it's frequency readings cycle. If a reading is in progress, it is aborted. The gen_read_freq function, if called after this, will return the reading from this trigger.

This trigger causes a single reading to be made.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

SEE ALSO

gen_read_freq, gen_rdy_freq

NAME

gen_rdy_freq -- determine if a frequency reading is available.

SYNOPSIS

rdy = gen_rdy_freq(void);

int rdy; 1 if ready, 0 if not

DESCRIPTION

This function determines if a reading is ready from the frequency counter.

Because readings functions do not return until a reading is ready, this function may be used to avoid waiting for a reading.

The gen_read_freq function is guaranteed to return quickly if this function says a reading is ready.

Note that a reading will never be ready unless triggered first using gen_trigger.

RETURNS

Returns a 1 if a frequency reading is ready. Returns a 0 otherwise.

SEE ALSO

gen_read_freq, gen_trigger, gen_read_status

NAME

gen_read_freq -- read generator frequency

SYNOPSIS

freq = gen_read_freq(void);

double freq; frequency reading in Hertz

DESCRIPTION

This function reads the frequency from the generator's frequency counter.

If the counter has not been triggered previously to this function, a trigger will be done first.

If a trigger was made previously to this function, the reading from that trigger will be returned.

RETURNS

This function returns the most recent frequency reading. Note that any given reading will be returned only once.

The number -1E34 is returned if there is a hardware error encountered. This should normally not occur for this function, and may indicate that the hardware is disconnected or not powered on.

NAME

gen_read_ilimit -- read current limit status.

SYNOPSIS

limit = gen_read_ilimit(void);

int limit; current limit status flag

DESCRIPTION

Reads output current limit status.

RETURNS

Returns 1 if generator is current limited.
Returns 0 if not.

SEE ALSO

gen_read_status

NAME

gen_read_settled -- read generator settling status.

SYNOPSIS

settled = gen_read_settled(void);

int settled; current settling status

DESCRIPTION

Reads generator settling status.

After a generator frequency or amplitude change, there is a finite time required for the generator output to settle. This time varies with both amplitude and frequency, but is usually longest at low frequencies.

Normally, using the DUS algorithm, there is no need to check for generator settling. This functions is provided for when the DUS algorithm is not being used, or the test setup has some unusual requirements.

RETURNS

Returns 1 if generator is settled.
Returns 0 if not.

SEE ALSO

gen_read_status

NAME

gen_read_status -- read generator status byte

SYNOPSIS

status = gen_read_status(void);

int status; generator status byte

DESCRIPTION

This is the main status function for the generator.

The information available here is also available in other functions that return the individual status bits.

RETURNS

This function returns the generator status byte as an unsigned int from 0 - 255.

The bits returned are defined as follows:

Bit	Definition
1	Reserved for future options
2	Trigger/Gate Input - High when input is low
4	Reserved for future options
8	Sync output - High when the sync is high
16	Reserved for future options
32	Output current limit - Low when limited
64	Oscillator settling - High when settled
128	Frequency counter reading ready indicator. Low when a new reading is available. Reset high when the counter is read by the PC.

SEE ALSO

gen_read_ilimit, gen_read_settled, gen_read_sync, gen_rdy_freq

NAME

gen_read_sync -- read the sync signal.

SYNOPSIS

sync = gen_read_sync(void);

int sync; current sync logic level

DESCRIPTION

This provides a reproduction of the generator sync output signal.

Obviously this function only has value if sync is running at very low frequencies such as with tone-bursts.

RETURNS

Returns 1 if sync is at a high logic level.
Returns 0 if sync is at a low logic level.

SEE ALSO

gen_read_status

NAME

gen_set_amp -- set generator amplitude

SYNOPSIS

new_amp = gen_set_amp(amp);

double amp;	amplitude in Volts
double new_amp;	amplitude actually set in Volts

DESCRIPTION

This function sets the generator amplitude.

The generator output is set in "PEAK EQUIVALENT RMS VOLTS". This means that any waveform output by the generator will have the same peak voltage as would a sine waveform set in RMS Volts. Thus, when the generator waveform is set to sine, the amplitude is set in RMS Volts. This method of amplitude calibration ensures that a change in waveform will not clip the device under test.

Because of amplitude limitations and digital quantization, that actual amplitude set may not be exactly what was requested. Therefore, it is important to check the returned amplitude, and to use that value as the actual amplitude the generator is at.

RETURNS

This function returns the actual amplitude the generator is set to in "PEAK EQUIVALENT RMS VOLTS". (see above)

If the requested amplitude exceeds hardware limitations, an error code is set and the present amplitude is returned. See ap_get_errcode for definitions of the error codes.

INITIAL STATE

Amplitude is set to 1.0 Volts by gen_init.

SEE ALSO

Generator hardware specifications for amplitude limits.

ap_get_errcode for definitions of the error codes.

NAME

gen_set_bal -- set generator output to balanced or unbalanced configuration.

SYNOPSIS

void gen_set_bal(on_off)

int on_off; 1 for balanced, 0 for not

DESCRIPTION

This function sets both outputs to a balanced or unbalanced configuration.

Note that the output impedance may change between balanced and unbalanced.(see gen_set_zout).

It is possible for this function to cause an amplitude error since the maximum allowable amplitude in the unbalanced configurations is half that for the balanced configuration. In this case the error code is set but the output configuration is not changed. See ap_get_errcode for definitions of the error codes.

RETURNS

None.

INITIAL STATE

The output is set to BALANCED by gen_init

SEE ALSO

gen_set_zout, gen_set_cmode

System One Users manual, section 8.4, for a description of the output configurations.

ap_get_errcode for definitions of the error codes.

CAUTIONS

An unbalanced and common mode test configuration is an invalid combination. Normally this function should be used together with gen_set_cmode to configure the output.

It is possible for this function to cause an amplitude error. See explanation above.

EXAMPLES

```
gen_set_cmode( 0);    /* make sure output not in CMODE */
gen_set_bal( 0);      /* now set output unbalanced */
```

NAME

gen_set_binterval -- set generator burst interval

SYNOPSIS

new_cycles = gen_set_binterval(cycles);

int cycles;	burst interval in cycles
int new_cycles;	interval actually set in cycles

DESCRIPTION

This function sets the number of cycles for the burst interval. This number may be from 1 to 65535 cycles and must be greater than the number of on cycles. If then number of cycles attempted is not greater than the on cycles, the interval is not changed and the error code is set. See ap_get_errcode for definitions of the error codes.

Note that the interval will occur immediately when this function is called if the burst is running.

RETURNS

The number of cycles actually set is returned.

INITIAL STATE

The number of cycles in the burst interval is set to 3 by gen_init

SEE ALSO

gen_set_bon, gen_set_bofflvl, gen_set_bmode

ap_get_errcode for definitions of the error codes.

CAUTIONS

The BUR-GEN hardware option is required for any of the burst functions.

NAME

gen_set_bofflvl -- set generator burst off level

SYNOPSIS

new_percent = gen_set_bofflvl(percent);

double percent;	burst off level in percent
double new_percent;	level actually set in percent

DESCRIPTION

This function sets the amplitude of the generator during the burst 'off' time. This is as a percentage of the 'on' amplitude and may range from 100.0 percent to .009716280 percent (-80.25 dB).

RETURNS

The burst off level percentage actually set is returned.

If the requested percentage exceeds hardware limitations, an error code is set and the present percentage is returned. See ap_get_errcode for definitions of the error codes.

INITIAL STATE

The burst off level percentage is set to .009716280 % (80.17 dB) percent by gen_init

SEE ALSO

gen_set_binterval, gen_set_bon, gen_set_bmode

ap_get_errcode for definitions of the error codes.

CAUTIONS

The BUR-GEN hardware option is required for any of the burst functions.

NAME

gen_set_bon -- set generator burst on cycles

SYNOPSIS

new_cycles = gen_set_bon(cycles)

int cycles	burst on time in cycles
int new_cycles	on cycles actually set

DESCRIPTION

This function sets the number of cycles for the burst on time. This number may be from 1 to 65535 cycles and must be less than the number of interval cycles. If then number of cycles attempted is not less than the interval cycles, the on time is not changed and the error code is set. See ap_get_errcode for definitions of the error codes.

RETURNS

The number of cycles actually set is returned.

INITIAL STATE

The number of cycles in the burst on time is set to 1 by gen_init

SEE ALSO

gen_set_binterval, gen_set_bofflvl, gen_set_bmode

ap_get_errcode for definitions of the error codes.

CAUTIONS

The BUR-GEN hardware option is required for any of the burst functions.

NAME

gen_set_bmode -- set generator burst mode

SYNOPSIS

void gen_set_bmode(mode);

int mode; burst mode

DESCRIPTION

This function sets one of the following burst modes:

0 = normal burst
1 = gated burst
2 = triggered burst

Normal burst continuously outputs a burst of N cycles every M cycles where N is the 'on time' as set by gen_set_bon and M is the interval set by gen_set_binterval.

Gated burst ignores normal interval and on time settings. The output is at normal amplitude as long as the TRIGGER/GATE INPUT on the generator auxiliary signals panel is held high, and at the programmed off level when the input is low.

Triggered burst outputs a single burst of N cycles when the TRIGGER/GATE INPUT is changed from low to high, where N is the number of cycles set by gen_set_bon.

RETURNS

None.

INITIAL STATE

The burst mode is set to NORMAL by gen_init

SEE ALSO

gen_set_binterval, gen_set_bon, gen_set_bofflvl

CAUTIONS

The BUR-GEN hardware option is required for any of the burst functions.

NAME

gen_set_cmode -- set generator output to common mode configuration

SYNOPSIS

void gen_set_cmode(on_off);

int on_off; 1 for common mode, 0 for normal

DESCRIPTION

This function sets both outputs to a common mode test configuration.

It is possible for this function to cause an amplitude error since the maximum allowable amplitude in the common mode test configuration is half that for the normal configuration. In this case the error code is set but the output configuration is not changed. See ap_get_errcode for definitions of the error codes.

RETURNS

None.

INITIAL STATE

The output is set NORMAL by gen_init.

SEE ALSO

gen_set_bal

System One Users manual, section 8.4, for a description of the output configurations.

ap_get_errcode for definitions of the error codes.

CAUTIONS

An unbalanced and common mode test configuration is an invalid combination. Normally this function should be used together with gen_set_bal to configure the output.

It is possible for this function to cause an amplitude error. See explanation above.

EXAMPLES

```
gen_set_bal( 1);        /* set output balanced */
gen_set_cmode( 1);      /* now set common mode test */
```

NAME

gen_set_freq -- set generator frequency

SYNOPSIS

new_freq = gen_set_freq(freq);

double freq;	frequency in Hertz
double new_freq;	frequency actually set in Hertz

DESCRIPTION

This function sets the generator frequency.

If autocal (see gen_autocal) is turned off, the closest available frequency setting is made and returned.

If autocal is on then a calibration cycle is performed to fine tune the frequency to the frequency requested and that frequency is returned. Note that the calibration cycle adds to the time required to set the frequency.

Because of frequency limitations and digital quantization, that actual frequency set may not be exactly what was requested. Therefore, it is important to check the returned frequency, and to use that value as the actual frequency the generator is at.

It is possible for this function to cause an amplitude error since the maximum allowable amplitude changes with frequency. See ap_get_errcode for definitions of the error codes.

Note that even if autocal is off at the time this call is made, the exact frequency passed to this function will be stored in memory and will be used if a call to gen_freqcal is made.

RETURNS

This function returns the actual frequency the generator is set to in Hertz. (see above)

INITIAL STATE

The Frequency is set to 1 kHertz by gen_init.

SEE ALSO

gen_set_autocal, gen_freqcal

Generator hardware specifications for frequency limits and step sizes.

ap_get_errcode for definitions of the error codes.

CAUTIONS

It is possible for this function to cause an amplitude error. See explanation above.

NAME

gen_set_gnd -- set generator output grounded or floating.

SYNOPSIS

void gen_set_gnd(on_off);

int on_off; 1 for ground , 0 for floating

DESCRIPTION

This function sets both outputs to grounded or floating.

RETURNS

None.

INITIAL STATE

The output is set to FLOATING by gen_init.

SEE ALSO

System One Users manual, section 8.4, for a description of the output configurations.

NAME

gen_set_imfreq -- set generator intermodulation distortion frequency

SYNOPSIS

new_imfreq = gen_set_imfreq(imfreq);

double imfreq;	IMD frequency in Hertz
double new_imfreq;	IMD frequency actually set in Hertz

DESCRIPTION

This function sets the generator IMD frequency. The frequency passed is rounded to the closest available value.

Set the generator to an IMD waveform before calling this function in order to have the proper IMD frequency selected.

For a SMPTE mode waveform, this is the lower frequency tone. The following frequencies are available:

```
500Hz, 250Hz, 125Hz, 100Hz, 60Hz, 50Hz, 40Hz
```

For a CCIF mode waveform, this is the spacing between the two tones. The following frequencies are available:

```
1kHz, 500Hz, 250Hz, 200Hz, 120Hz, 100Hz, 80Hz
```

For a DIM mode waveform, this function has no effect. The frequencies are determined by the DIM mode selected. (see gen_set_mode)

Because of frequency limitations and digital quantization, that actual frequency set may not be exactly what was requested. Therefore, it is important to check the returned frequency, and to use that value as the actual frequency the generator is at.

RETURNS

This function returns the actual IMD frequency the generator is set to in Hertz. (see above)

INITIAL STATE

The IMD frequency is set to 60 Hertz by gen_init

SEE ALSO

Generator hardware specifications for any possible changes to the frequencies given above.

CAUTIONS

The IMD-GEN hardware option is required for any of the intermodulation functions.

NAME

gen_set_mode -- set generator waveform mode.

SYNOPSIS

void gen_set_mode(mode);

char *mode; mode string

DESCRIPTION

This function sets the generator waveform mode.

Available modes along with the minimal and recommended setting strings are:

minimal	recommended	mode
"S"	"SINE"	SINEWAVE

Additionally available with the IMD-GEN option:

"I"	"IMD-4"	IMD-SMPTE 4:1
"I1"	"IMD-1"	IMD-SMPTE 1:1
"C"	"CCIF"	IMD-CCIF
"D"	"DIM-A"	DIM-A (3.18kHz,100kHz BW)
"D3"	"DIM-30"	DIM-30 (3.18kHz,30kHz BW)
"DB"	"DIM-B"	DIM-B (2.96kHz,30kHz BW)

Additionally available with the BUR-GEN option:

"B"	"BURST"	BURSTED SINEWAVE
"NP"	"NOISE-PNK"	NOISE-PINK
"NW"	"NOISE-WHT"	NOISE-WHITE
"NB"	"NOISE-BP"	NOISE-BANDPASS
"SQ"	"SQUARE"	SQUAREWAVE

Additionally available with the DSP module installed:

"DS"	"DSP"	DSP A/D OUTPUT

The input string is searched for the first occurrence of a valid mode character, then for a second character if necessary. The case of letters are ignored.

If the mode string cannot be interpreted, the error code is set and nothing is done. See ap_get_errcode for definitions of the error codes.

RETURNS

None.

INITIAL STATE

The waveform mode is set to SINE by gen_init.

SEE ALSO

ap_get_errcode for definitions of the error codes.

CAUTIONS

The IMD-GEN hardware option is required for any of the intermodulation settings.

The BUR-GEN hardware option is required for any of the noise, burst, or squarewave settings

The DSP hardware option is required for the DSP setting

NAME

gen_set_ntype -- set generator noise type

SYNOPSIS

void gen_set_ntype(type);

int type; noise type

DESCRIPTION

This function sets one of the following noise modes:

0 = pseudo random noise
1 = true random noise

Pseudo random noise is normally used with System One. It consists of a sequence that repeats 4 times per second. This rate matches the slowest reading rate of the LVF and will allow readings to be taken without the 'digit bobble' normally associated with noise.

The length of the pseudo random noise sequence causes a 4 Hz frequency spacing between spectral lines. This is more than adequate for use with System One but may be a limitation if an external spectrum analyzer or very narrow filters are being used.

True random noise does not repeat and may be used when a closer spectral spacing is required.

RETURNS

None.

INITIAL STATE

The noise type is set to PSEUDO RANDOM by gen_init

SEE ALSO

gen_set_mode for other noise selections.

CAUTIONS

The BUR-GEN hardware option is required for this function.

NAME

gen_set_out_a -- set generator output A on or off
gen_set_out_b -- set generator output B on or off

SYNOPSIS

```
void gen_set_out_a( on_off);
void gen_set_out_b( on_off);

int on_off;                    1 for ON, 0 for OFF
```

DESCRIPTION

These functions set output A or output B to on or off.

RETURNS

None.

INITIAL STATE

Both outputs are set to OFF by gen_init.

NAME

gen_set_polarity -- set generator output B to normal or invert.

SYNOPSIS

void gen_set_polarity(polarity)

int polarity; 1 for normal, 0 for B inverted

DESCRIPTION

This function sets output B to normal polarity (in phase with channel A) or inverted polarity (180 degrees out of phase with channel A).

RETURNS

None.

INITIAL STATE

The polarity is set to NORMAL by gen_init.

NAME

gen_set_zout -- set generator output impedance

SYNOPSIS

new_range = gen_set_zout(range)

int range;	impedance range number
int new_range;	impedance range number actually set

DESCRIPTION

This function sets the output impedance of the generator. The impedance is chosen by range with ranges 1 through 3 available.

For normal systems, the impedances associated with the ranges are:

RANGE	OUTPUT MODE	
	BAL & CMTEST	UNBAL
1	50 ohms	25 ohms
2	150 ohms	80.2 ohms
3	600 ohms	600 ohms

For systems with the EURZ option installed, the impedances associated with the ranges are:

RANGE	OUTPUT MODE	
	BAL & CMTEST	UNBAL
1	36 ohms	18 ohms
2	200 ohms	114.4 ohms
3	600 ohms	600 ohms

RETURNS

The Range actually selected is returned.

NAME

lvf_select -- select the active LVF

SYNOPSIS

new_lvfnum = lvf_select(lvfnum);

int lvfnum;	LVF reference number
int new_lvfnum;	LVF number actually set

DESCRIPTION

This function selects which LVF will be used as the object of all other LVF calls.

This function should be used before any other LVF function calls, including lvf_init, are used.

> **Note that lvf_select is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one LVF. Using multiple LVFs in a system requires modification of the additional LVF circuit boards to change addresses. Contact Audio Precision for further information**

The software internally maintains 4 different sets of LVF data. This function simply selects which set of data to use until lvf_select is called again.

At present, LVFs 1 to 4 are supported. The passed value must be from 1 to 4. If it is out of this range, no new selection is made.

RETURNS

The LVF number used is returned.

SEE ALSO

lvf_init, lvf_restore, ap_setup

NAME

lvf_init -- initialize LVF

SYNOPSIS

new_modnum = lvf_init(modnum);

int modnum;	module number
int new_modnum;	module number actually set

DESCRIPTION

This function initializes the LVF to a known state. It must be used after power-up.

> **Note that lvf_init is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one LVF. Using multiple LVFs in a system requires modification of the additional LVF circuit boards to change addresses. Contact Audio Precision for further information**

The LVF data structures are set to default conditions, then the LVF hardware is set to match using the lvf_restore function.

This function also sets the module number of the selected LVF (see lvf_select) The module number must be between 0 and 15, else the default of 0 is used.

> **The default module number of 0 should be used unless you have a system specifically modified to have multiple LVFs.**

Default initializations:

1. Set measurement mode to AMPLITUDE.
2. Select channel A.
3. Set both inputs to autorange.
4. Set gainamp to autorange.
5. Highpass and low pass filters off.
6. Weighting filters off.
7. Set main voltmeter detector to RMS response.
8. Set detectors to longest averaging times.
9. Set reading rates to 4 per second.
10. Input terminations to HI-Z.

11. Do not monitor GEN
12. Set phase range to +-180 degrees.
13. Set bandpass/bandreject filter to autotune.
14. Set Wow & Flutter weighting to "WEIGHTED".
15. Restart 6805 micro-processor onboard the LVF

RETURNS

The module number used is returned.

SEE ALSO

lvf_restore, lvf_select, ap_setup

NAME

lvf_restore -- restore LVF hardware

SYNOPSIS

void lvf_restore(void);

DESCRIPTION

This function restores the LVF hardware to the present state of the software.

This function should be used if the hardware loses power or becomes disconnected from the computer.

This function is also used by the lvf_init function.

Because of time delays used in the restoring process, this function has an execution time of at least 10 milliseconds.

RETURNS

None

SEE ALSO

lvf_init

NAME

lvf_exist -- determine if the LVF hardware is available.

SYNOPSIS

exist = lvf_exist(void);

int exist; 1 if LVF exists, 0 if not

DESCRIPTION

This function determines if the selected LVF is connected and powered on.

The LVF status is updated each time lvf_restore called. This function returns that status. Note that ap_setup and lvf_init also call lvf_restore and so update the status.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

If the LVF is found not to exist (but should), prompt the operator to remedy the situation, then call lvf_restore before calling lvf_exist again.

RETURNS

Returns a 1 if the selected LVF exists, 0 otherwise.

NAME

lvf_channel -- select A or B channel.

SYNOPSIS

void lvf_channel(channel);

char *channel; channel string

DESCRIPTION

Selects channel A or channel B to be used for measurements.

The input string is searched for the first occurrence of 'A' or 'B' (either upper or lower case).

RETURNS

None.

INITIAL STATE

The measurement channel is set to A by lvf_init.

NAME

lvf_measure -- make a measurement from the LVF

SYNOPSIS

reading = lvf_measure(void);

double reading; reading (see below for unit)

DESCRIPTION

Make a measurement using the LVF's main voltmeter

The measurement is taken from the selected channel, using the selected mode, and using the units specified by that mode.

The available modes and their corresponding units are:

AMPLITUDE	- Volts
BANDPASS	- Volts
BANDREJECT	- Volts
THD	- Volts
THD %	- %
SMPTE IMD	- %
CCIF IMD	- %
DIM	- %
WOW & FLUTTER	- %

Additionally available with version "A" hardware:

"2 CHANNEL AMPLITUDE"	- Volts
"2 CHANNEL BANDPASS"	- Volts
"2 CHANNEL AMPLITUDE %"	- %
"2 CHANNEL BANDPASS %"	- %

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

RETURNS

The most recent reading is returned as above.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the hardware is over-ranged for any mode, or if the frequency is out of range for the Wow & Flutter modes. It may also indicate that the hardware is disconnected or not powered on.

SEE ALSO

lvf_trigger, lvf_rdy_measure, lvf_read_mode, lvf_set_mode

NAME

lvf_monitor_a -- set LVF input channel A to monitor.
lvf_monitor_b -- set LVF input channel B to monitor.

SYNOPSIS

```
void lvf_monitor_a( on_off);
void lvf_monitor_b( on_off);

int on_off;                    1 for monitor, 0 for not
```

Additionally with the version "A" hardware: may be set to 2 for Auxiliary Input on channel A

DESCRIPTION

These functions connect channel A or channel B to the internal monitor connection. It is generally necessary to also set the corresponding GEN channel to monitor.

RETURNS

None.

INITIAL STATE

Monitors are set to OFF by lvf_init.

SEE ALSO

gen_monitor_a, gen_monitor_b

NAME

lvf_trigger -- trigger a new reading with delay

SYNOPSIS

void lvf_trigger(msec);

double msec; trigger delay time in milliseconds

DESCRIPTION

Causes the LVF to wait x milliseconds then restart all of it's readings cycles. Readings in progress are aborted. Any readings function called after this will wait for it's appropriate new reading to be available.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

NAME

lvf_range_lock -- lock the LVF ranges for a DSP data acquisition.
lvf_range_release -- release the LVF ranges after a DSP acqusition.

SYNOPSIS

```
void lvf_range_lock( void);
void lvf_range_release( void);
```

DESCRIPTION

These functions were added to support DSP data acquisition for the FFT type DSP programs.

For the FFT type programs (fftgen and fftslide), the waveform data must be contiguous in time, therefore no ranging can occur while an acquisiton is in progress. Also, the waveform data needs to have the ranging information stored with it to allow proper scaling of the output data.

If the data is being acquired through the LVF, the ranges must be locked before data acquisition. This is accomplished be calling lvf_range_lock. The ranges may remain locked if you are sure that new ranges will not be need for the next acquisition.

Normally, the sequence is

```
lvf_range_lock();          /* Lock Ranges for Acquisition */
dsp_acquire_xform();       /* start Acquire and Transform */
dsp_wait();                /* loop calling dsp_read_status */
lvf_range_release();       /* Unlock ranges */
```

If the data is being acquired through the DSP direct inputs or through any of the digital inputs, these functions are not needed.

RETURNS

None

NAME

lvf_rdy_bpbrfreq -- determine if a BP/BR tuning frequency reading is ready.
lvf_rdy_freq -- determine if a frequency reading is ready.
lvf_rdy_involts -- determine if a level reading is ready.
lvf_rdy_measure -- determine if a main voltmeter reading is ready.
lvf_rdy_phase -- determine if a phase reading is ready.
lvf_rdy_polarity -- determine if a polarity reading is ready.

SYNOPSIS

```
rdy = lvf_rdy_bpbrfreq( void);
rdy = lvf_rdy_freq( void);
rdy = lvf_rdy_involts( void);
rdy = lvf_rdy_measure( void);
rdy = lvf_rdy_phase( void);
rdy = lvf_rdy_polarity( void);

int rdy;                        1 if ready, 0 if not
```

DESCRIPTION

This function determines if the appropriate reading is ready.

Because readings functions do not return until a reading is ready, these functions may be used to avoid waiting for a reading. These functions do NOT clear their respective ready statuses and so may be called any number of times. Only a call to the reading function will clear a ready status.

If the particular reading is found to be ready, then a subsequent call to the corresponding readings function will be guaranteed to return quickly.

Note that readings from the LVF will free run at the selected measurement rate and so will eventually become ready even without a call to lvf_trigger.

RETURNS

Returns a 1 if a reading is ready. Returns a 0 otherwise.

SEE ALSO

lvf_trigger, lvf_read_freq, lvf_read_involts, lvf_measure, lvf_read_bpbrfreq, lvf_read_phase, lvf_read_polarity

NAME

lvf_read_bpbrfreq -- read bandpass/bandreject frequency

SYNOPSIS

freq = lvf_read_bpbrfreq(void);

double freq; bandpass/bandreject frequency in Hertz

DESCRIPTION

Returns the frequency value that is used to tune the bandpass/bandreject filter.

If the filter is in auto tuning mode (see lvf_set_bpbrfreq) it will be at this frequency. Otherwise this is the frequency the filter would be tuned to if it were in auto tuning mode, not the actual filter frequency..

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

RETURNS

This function returns the bandpass/bandreject frequency in Hertz.

The number -1E34 is returned if there is a hardware error encountered. This should normally not occur for this function, and may indicate that the hardware is disconnected or not powered on.

SEE ALSO

lvf_set_bpbrfreq, lvf_rdy_bpbrfreq

NAME

lvf_read_freq -- read frequency from lvf

SYNOPSIS

freq = lvf_read_freq(void);

double freq; frequency reading in Hz.

DESCRIPTION

This function makes an LVF frequency reading from the selected input channel or the opposite input channel if the LVF is in a 2 channel mode. (See lvf_set_mode for a description of the 2 channel modes.)

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

RETURNS

This function returns frequency in Hertz.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the input amplitude is too low for a frequency reading. It may also indicate that the hardware is disconnected or not powered on.

SEE ALSO

lvf_rdy_freq

NAME

lvf_read_gainrange -- read gain amp gain

SYNOPSIS

gain = lvf_read_gainrange(void);

double gain; gain of gainamp

DESCRIPTION

This function reads the gain of the gainamp range in use.

A common use of these functions is in fixing the gainamp range by obtaining the gain and then using the gain to call lvf_set_gainrange.(see example below)

RETURNS

This function returns the present gain of the gainamp

SEE ALSO

lvf_set_gainrange, lvf_read_stat_gainamp, lvf_set_inrange_a, lvf_set_inrange_b, lvf_read_inrange_a, lvf_read_inrange_b, lvf_read_measrange.

EXAMPLES

```
double gain, lvf_read_gainrange();

gain = lvf_read_gainrange();            /*get the gain*/
lvf_set_gainrange( gain);                       /*fix the gainamp*/
```

NAME

lvf_read_inrange_a -- read channel A input range
lvf_read_inrange_b -- read channel B input range

SYNOPSIS

volts = lvf_read_inrange_a(void);
volts = lvf_read_inrange_b(void);

double volts; range in Volts

DESCRIPTION

These functions read the appropriate channel's input range and return nominal full scale of range in use.

The possible ranges are:

```
160V
80V
40V
20V
10V
5V
2.5V
1.2V
600mV
300mV
160mV
80mV
```

A common use of these functions is in fixing the input range by obtaining the range voltage and then using that voltage to call lvf_set_inrange_a. (or lvf_set_inrange_b)(see example below)

RETURNS

This function returns the full scale voltage of the appropriate input range in use

SEE ALSO

lvf_set_inrange_a, lvf_set_inrange_b, lvf_read_stat_a, lvf_read_stat_b, lvf_read_gainrange, lvf_set_gainrange.

EXAMPLES

```
double volts, lvf_read_inrange_a();

volts = lvf_read_inrange_a();          /*get the range*/
lvf_set_inrange_a( volts);                    /*fix the range*/
```

NAME

lvf_read_involts -- read input voltage (level reading) from LVF

SYNOPSIS

volts = lvf_read_involts(void);

double volts; input voltage

DESCRIPTION

Make a voltage reading from the selected input channel or the opposite input channel if the LVF is in a 2 channel mode. (See lvf_set_mode for a description of the 2 channel modes.)

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

NOTE: This is the voltmeter on the DIS-1 option and so is only available if this option is installed. This voltmeter functions independently of the reading mode selected.

RETURNS

This function returns the input voltage from the selected channel.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the hardware is over-ranged. It may also indicate that the hardware is disconnected or not powered on.

SEE ALSO

lvf_measure, lvf_rdy_involts.

CAUTIONS

The DIS-1 hardware module must be installed for this function to operate correctly.

The input voltmeter always uses an RMS detector and is not affected by the lvf_set_detector function.

NAME

lvf_read_measrange -- read the measurement range

SYNOPSIS

volts = lvf_read_measrange(void);

double volts; measurement range in Volts

DESCRIPTION

Returns the nominal full scale in Volts of the range of the measurement presently selected. This range is affected by the measurement mode, selected channel, its input range and the gain amp range.

This voltage corresponds to a 1.25 Volt output on the READING output on the front panel of the LVF. For instance, if the LVF was set to measure AMPLITUDE and the range returned by this function was 80 mVolt, then an 80 mVolt signal at the input to the LVF would appear as a 1.25 Volt signal at the READING output.

This function refers only to the range at the READING output. It always returns the scale in Volts, even if the measurement mode is set to return a reading in percentages (such as THD%).

RETURNS

Measurement range in Volts.

SEE ALSO

lvf_set_gainrange, lvf_set_inrange_a, lvf_set_inrange_b,
lvf_read_gainrange, lvf_read_inrange_a, lvf_read_inrange_b

NAME

lvf_read_mode -- read measurement mode string

SYNOPSIS

mode = lvf_read_mode(void);

int mode; number representing mode

DESCRIPTION

This function returns one of the following possible numbers that indicate which measurement mode the main voltmeter is in:

1 = AMPLITUDE
2 = BANDPASSED AMPLITUDE
3 = BANDREJECTED AMPLITUDE
4 = THD as an amplitude
5 = THD as a percent
6 = SMPTE IMD
7 = CCIF IMD
8 = DIM
9 = WOW AND FLUTTER

Additionally available with the version "A" hardware:

10 = 2 CHANNEL AMPLITUDE
11 = 2 CHANNEL AMPLITUDE (as a percent between channels)
12 = 2 CHANNEL BP AMPLITUDE
13 = 2 CHANNEL BP AMPLITUDE (as a percent between channels)

RETURNS

This function returns an int representing the mode.

SEE ALSO

lvf_set_mode, lvf_measure

NAME

lvf_read_phase -- read phase from lvf
lvf_read_hiphase -- read phase from lvf and set phase range

SYNOPSIS

```
phase = lvf_read_phase( void);
phase = lvf_read_hiphase( void);

double phase;                    phase in degrees
```

DESCRIPTION

Measures the phase between channels A and B.

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

lvf_read_hiphase functions the same as lvf_read_phase but it additionally makes a call to lvf_set_phase with the new reading. This keeps the phase range optimum for the readings being made.

RETURNS

This function returns the phase between channels A and B in degrees.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the input amplitude is too low for a phase reading. It may also indicate that the hardware is disconnected or not powered on.

SEE ALSO

lvf_set_phase, lvf_rdy_phase

CAUTIONS

The phase range must be properly set before taking the phase reading will be valid.(see lvf_set_phase). For most circumstances, lvf_set_hiphase takes care of this.

NAME

lvf_read_polarity -- read polarity from LVF

SYNOPSIS

polarity = lvf_read_polarity(void);

double polarity; polarity in degrees

DESCRIPTION

Measures the absolute polarity of the signal on the selected input channel.

Polarity can be either 0 or 180 degrees. A special asymmetric waveform is required to determine this. With System One this waveform is produced by the BUR-GEN option

Set the generator to:

```
BURST mode
FREQUENCY 1 kHz
1 CYCLE ON
3 CYCLES OFF
1 % BURST OFF LEVEL
```

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

RETURNS

This function returns the absolute polarity as either 0 or 180 degrees.

SEE ALSO

lvf_rdy_polarity

gen_set_mode, gen_set_bmode, gen_set_binterval, gen_set_bon, gen_set_bofflvl

NAME

lvf_read_term_a -- read channel A termination status
lvf_read_term_b -- read channel B termination status

SYNOPSIS

```
term = lvf_read_term_a( void);
term = lvf_read_term_b( void);

int term;                       1 if terminated, 0 if not
```

DESCRIPTION

Reads the appropriate channel's termination status.

The termination may remove itself from the LVF's inputs if an overload is detected. These functions provide a way of sensing this.

RETURNS

Returns 1 if terminated.
Returns 0 if not.

SEE ALSO

lvf_set_term_a, lvf_set_term_b

NAME

lvf_set_bpbrfreq; -- set bandpass/bandreject frequency

SSYNOPSIS

new_freq = lvf_set_bpbrfreq(freq);

double freq;	center frequency in Hertz
double new_freq;	frequency actually set in Hertz

DESCRIPTION

This function attempts to set the bandpass/bandreject filter to the frequency value passed.

0.0 sets the filter to autotuning.

RETURNS

Returns the frequency in Hertz actually set or 0.0 for autotune.

INITIAL STATE

The bandpass/bandreject filter is set to AUTOTUNE by lvf_init.

SEE ALSO

lvf_read_bpbrfreq

NAME

lvf_set_detector -- select detector type.

SYNOPSIS

void lvf_set_detector(type);

char *type; type string

DESCRIPTION

Selects detector type for main voltmeter.

Choices are as follows:

```
R    = RMS
A    = AVERAGE
P    = PEAK
Q    = QPEAK.
E    = PEAK-EQUIVALENT-SINE
```

The input string is searched for the first occurrence of 'R', 'A', 'P', 'Q' or 'E' (upper or lower case).

If the LVF is in Wow & Flutter mode, the choices mean the following:

```
R    = NAB-RMS
A    = JIS
P    = undefined
Q    = IEC-PK
E    = undefined
```

Refer to the System One Operator's manual sections 9.4 and 15.2 (Wow & Flutter) for a description of the various Wow & Flutter detectors.

RETURNS

None.

INITIAL STATE

The detector is set to RMS by lvf_init.

CAUTIONS

This functions affects only the main voltmeter. This is the voltmeter accessed by the lvf_measure function. The input voltmeter (accessed by lvf_read_involts) always uses an RMS detector.

NAME

lvf_set_detspeed -- set detector averaging speed
lvf_set_indetspeed -- set input detector averaging speed

SYNOPSIS

new_range = lvf_set_detspeed(range);
new_range = lvf_set_indetspeed(range);

int range;	range number
int new_range;	range number actually set

DESCRIPTION

These functions set the detector averaging times for the RMS and AVERAGE detectors.

> **Normally, these functions do not need to be called directly. Both lvf_set_measrate and the preferred lvf_set_response call these functions to set the optimum detector averaging time.**
>
> **If these functions are used, note that they must be called after lvf_set_measrate or lvf_set_response are used.**

lvf_set_detspeed sets the detector for the measurement functions.

lvf_set_indetspeed sets the detector for the input voltmeter. NOTE: This is for the detector on the distortion option and so is only available if this option is installed.

Ranges 1 through 4 are available. Any other setting attempted results in no action being taken.

These functions have no effect on the PEAK and QPEAK detectors.

There is an inherent relationship between the detector averaging time and the lowest frequency component of the measured signal. The combinations of detector time constant and reading rate will affect both low frequency accuracy and digit bobble.

For most applications, detector time constants should be ganged with reading rate, where the slowest time constant (range 1) is used for 4 readings/second, and the fastest (range 4) for 30 readings/sec.

(see also the discussion under lvf_set_measrate)

The relative time constants associated with the ranges are:

1 = SLOWEST
2
3
4 = FASTEST

RETURNS

These functions return the speed range their appropriate detectors are set to.

INITIAL STATE

Both detectors are set to their SLOWEST time constants by lvf_init.

SEE ALSO

lvf_set_measrate, lvf_set_response

NAME

lvf_set_gainrange -- set gain amp range

SYNOPSIS

new_gain = lvf_set_gainrange(gain);

double gain; gain
double new_gain; gain actually set

DESCRIPTION

Sets gain amp range. The range selected has a gain equal to or less than the requested gain.

If the gain specified is 0.0, autorange will be selected.

The following table shows the gains available and to what full-scale ranges they correspond to in the various measurement mode of the LVF:

GAIN	AMPLITUDE RANGE	THD & DIM RANGE	CCIF & SMPTE RANGE
0	AUTO	AUTO	AUTO
1	80 mV	100%	25%
4	20 mV	25%	6%
16	5 mV	6%	1.6%
64	1.2 mV	1.6%	0.4%
256	300 uV	0.4%	0.1%
1024	*	0.1%	0.025%

```
* a gain of 1024 is valid only for bandpass and
bandreject measurements and represents 75 uV.
```

Note that for the amplitude ranges, the input ranges (see set_inrange_a and b) must be set to the 80 mV range before the ranges here are valid.

Also, for amplitude ranges, the gain here should be set to 1 or AUTO if the input range (see set_inrange_a and b) is set to anything other than 80 mV.

The distortion ranges are independent of the input range settings.

This range must be reprogrammed if the measurement mode of the LVF is changed (see lvf_set_mode). Otherwise, the resulting range is not determinate.

A common use of these functions is in fixing the gainamp range by obtaining the gain and then using the gain to call lvf_set_gainrange.(see example below)

RETURNS

This function returns the gain the gain amp is actually set to, or 0.0 if the gainamp is set to autorange.

INITIAL STATE

The gain amp is set to AUTORANGE by lvf_init.

SEE ALSO

lvf_read_gainrange

EXAMPLES

```
double gain, lvf_read_gainrange(), lvf_set_gainrange();

gain = lvf_read_gainrange();          /*get the gain*/
gain = lvf_set_gainrange( gain);              /*fix the gainamp*/
```

NAME

lvf_set_hipass -- set high pass filter

SYNOPSIS

new_freq = lvf_set_hipass(freq);

double freq;	corner frequency in Hertz
double new_freq;	frequency actually set in Hertz

DESCRIPTION

This function sets the filter whose corner is equal to or below the input value. This is usually the best filter for the frequency of interest passed.

A zero selects no filtering.

The available high pass filter frequencies are:

```
22.4 Hz.
100 Hz.
400 Hz.
```

If this function causes the high pass filter selection to change, a 250 millisecond delay will be invoked to allow the hardware to settle.

RETURNS

Returns corner frequency of filter actually set in Hertz, or 0.0 if no filter is selected.

INITIAL STATE

The high pass filter is set to OFF by lvf_init.

NAME

lvf_set_inrange_a -- set channel A input range
lvf_set_inrange_b -- set channel B input range

SYNOPSIS

new_volts = lvf_set_inrange_a(volts);
new_volts = lvf_set_inrange_b(volts);

double volts;	range in Volts
double new_volts;	range actually set in Volts

DESCRIPTION

These functions set the appropriate channel's input range and return the nominal full scale of range in use.

The possible ranges are:

```
160 V
80 V
40 V
20 V
10 V
5 V
2.5 V
1.2 V
600 MV
300 mV
160 mV
80 mV
```

If the range specified is 0.0, autorange will be selected.

A common use of these functions is in fixing the input range by obtaining the range voltage and then using that voltage for this call.(see example below)

RETURNS

These functions return the full scale voltage of the appropriate input range in use, or 0.0 if the range is set to autorange.

INITIAL STATE

The input ranges are set to AUTORANGE by lvf_init.

SEE ALSO

lvf_read_inrange_a, lvf_read_inrange_b

EXAMPLES

```
double volts, lvf_read_inrange_a();

volts = lvf_read_inrange_a();          /*get the range*/
lvf_set_inrange_a( volts);             /*fix the range*/
```

NAME

lvf_set_lopass -- set low pass filter

SYNOPSIS

new_freq = lvf_set_lopass(freq);

double freq;	corner frequency to set filter
double new_freq;	corner frequency actually set

DESCRIPTION

This function sets the filter whose corner is equal to or above the input value. This is usually the best filter for the frequency of interest passed.

A zero selects no filtering.

The available low pass filter frequencies are:

```
80 kHz.
30 kHz.
22.4 kHz.
```

RETURNS

Returns corner frequency of filter actually set in Hertz, or 0 if no filter is selected.

INITIAL STATE

The low pass filter is set to OFF by lvf_init.

NAME

lvf_set_measrate -- set voltmeter reading rate

SYNOPSIS

new_rate = lvf_set_measrate(rate);

int rate;	rate in readings per second.
int new_rate;	rate actually set in readings per second.

DESCRIPTION

This function selects the measurement and frequency reading rate and detector averaging speeds closest to but above the input value.

Normally, this function does not need to be called directly. The preferred lvf_set_response calls this function to set the optimum measurement rate and detector averaging speeds.

If this function is used, note that it must be called after lvf_set_response is used.

This is a best estimate attempt. If a 0 is passed, no action taken and the present rate is returned.

Rates available are:

```
30, 15, 8, and 4 readings per second.
```

There is an inherent relationship between the fastest valid reading rate and the lowest frequency component of the measured signal. 30 readings per second should be used only for repetitive signals faster than approximately 80 Hz. Similarly, 15 readings/sec is valid for 40 HZ and faster, 8 readings/sec for 20 HZ and faster, and 4 readings/sec for 10 HZ and faster.

RETURNS

Returns the rate actually set in readings per second.

INITIAL STATE

The measurement rate is set to 4 per second by lvf_init.

SEE ALSO

lvf_set_response, lvf_set_detspeed

NAME

lvf_set_mode -- set measurement mode of main voltmeter

SYNOPSIS

void lvf_set_mode(mode);

char *mode; mode string;

DESCRIPTION

This function sets the measurement mode of the LVF's main voltmeter.

Available modes along with the minimal and recommended setting strings are:

minimal	recommended	mode
"A"	"AMPL"	AMPLITUDE
"BP"	"BANDPASS"	BANDPASSED AMPLITUDE
"BR"	"BANDREJECT"	BANDREJECTED AMPLITUDE
"T"	"THDAMPL"	THD as an amplitude
"T%"	"THD%"	THD as a percent
"S"	"SMPTE"	SMPTE IMD
"C"	"CCIF"	CCIF IMD
"D"	"DIM"	DIM
"W"	"W&F"	WOW and FLUTTER

Additionally available with the version "A" hardware:

"2A"	"2CH-AMPL"	2 CHANNEL AMPLITUDE
"2A%"	"2CH-AMPL%"	2 CHANNEL AMPLITUDE as % between channels
"2B"	"2CH-BP"	2 CHANNEL BP AMPLITUDE
"2B%"	"2CH-BP%"	2 CHANNEL BP AMPLITUDE as % between channels

The input string is searched for the first occurrence of the first letter of the mode.

For Thd, if a '%' is found, the mode is set to Thd%, otherwise it is set to Thdlevel.

Note that the 2 channel modes connect the input voltmeter and the frequency counter to the opposite input channel. For example, if channel "A" is selected (lvf_channel("A")) and the LVF is in one of the 2 channel modes, the input voltmeter and the frequency counter will be connected to channel "B".

RETURNS

None.

INITIAL STATE

The measurement mode is set to AMPLITUDE by lvf_init.

SEE ALSO

lvf_read_mode, lvf_measure

CAUTIONS

The DIS-1 hardware option is required for the "BANDPASS", "BANDREJECT", "THDAMPL", and "THD%" modes to operate properly.

The IMD-LVF hardware option is required for the "SMPTE", "CCIF", and "DIM" modes to operate properly.

The W&F-LVF hardware option is required for the "W&F" mode to operate properly

NAME

lvf_set_phase -- set phase measurement range

SYNOPSIS

new_phase = lvf_set_phase(phase);

int phase;	phase in degrees
int new_phase;	phase actually set in degrees

DESCRIPTION

This function sets the phase measurement range best for the input value.

Possible ranges are 0 to +360 degrees and -180 to +180 degrees.

The input value must be between -360 and 360 for any action to be taken.

RETURNS

Returns the center of the phase range selected, either 0 (-180 to +180 range) or 180 (0 to 360 range).

INITIAL STATE

The center of the phase range is set to 0 (the +-180 range) by lvf_init.

SEE ALSO

lvf_read_phase

NAME

lvf_set_response -- set response combinations

SYNOPSIS

new_freq = lvf_set_response(freq);

double freq;	frequency in Hertz
double new_freq;	frequency actually set in Hertz

DESCRIPTION

This function sets a pre-programmed combination of detector averaging speed and reading rate for both the measurement and input (distortion option)(if installed) detectors.

The combination is optimized for maximum speed possible within reasonable instrument accuracy for the given frequency of interest. It is by nature a compromise, but has proven to give good measurement results under most conditions.

Because the action of this function is dependent on the detector type in use, lvf_set_detector should be called before this function.

The following table describes the choices made by this function. Each row is in effect at or above the frequency given, with the appropriate columns selected by the main voltmeter detector type.

> or = FREQ	RMS and AVG		PEAK and QPEAK	
	DETSPEED RANGE	READING RATE	DETSPEED RANGE	READING RATE
65 Hz	4 (fast)	30/sec	NO EFFECT	30/sec
29 Hz	2	15/sec	NO EFFECT	30/sec
18 Hz	2	8/sec	NO EFFECT	30/sec
0 Hz	1 (slow)	4/sec	NO EFFECT	30/sec

Additional information:

1. The detector speeds are not affected if the detector type is PEAK or QPEAK.
2. The input voltmeter detector speed is set the same as the main voltmeter.
3. This performs the same function as the auto setting of the detector field on the S1.EXE analyzer panel.
4. The information in the table is subject to refinements by Audio Precision.

Unless a special application requires an unusual combination of averaging or reading rate, this function is an excellent starting point for setting up the LVF to make a measurement at any given frequency.

RETURNS

The frequency for the settings chosen.

SEE ALSO

lvf_set_detspeed, lvf_set_indetspeed, lvf_set_measrate, lvf_set_detector

EXAMPLES

```
lvf_set_detector("RMS");          /*set main voltmeter to RMS*/
lvf_set_response(40.0);           /*set response good at 40 Hz.*/
```

The above example would set the reading rate to 16/sec. and both detector time constants to range 3.

NAME

lvf_set_term_a -- set channel A termination.
lvf_set_term_b -- set channel B termination.

SYNOPSIS

void lvf_set_term_a(term);
void lvf_set_term_b(term);

int term; termination number

DESCRIPTION

This function selects one of the available terminations impedances for the LVF input.

Terminations 1 - 3 are available. Any other setting attempted results in Hi-Z being selected.

The available terminations are:

1 = 150 ohms
2 = 600 ohms
3 = Hi-Z

The termination may remove itself from the LVF's inputs if an overload is detected. The lvf_read_term_a and lvf_read_term_b functions provide a way of sensing this.

RETURNS

None.

INITIAL STATE

Both inputs set to HI-Z by lvf_init.

SEE ALSO

lvf_read_term_a, lvf_read_term_b

NAME

lvf_set_weight -- set weighting filter

SYNOPSIS

new_weight = lvf_set_weight(weight);

int weight;	weighting filter number
int new_weight;	weighting filter number actually set

DESCRIPTION

This function selects one or none of the available weighting filters.

The weighting filters are optional filters that plug into the LVF (internally), so the exact function of each of the selections here is dependent on filter installation.

Weighting numbers from 0 to 8 are available as follows:

0 = no weighting
1 = weighting slot #1
2 = weighting slot #2
3 = weighting slot #3
4 = weighting slot #4

7 = weighting slot #1 with a gain of 2.092 (for CCIR filter)
8 = weighting slot #1 with a gain of 4.0 (for CCIR filter)

Additionally available with the version "A" hardware:

5 = weighting slot #5
6 = external weighting filter

Any other setting attempted results in no weighting being selected (weighting 0).

SPECIAL NOTES FOR SLOT #1:

Normally weighting slot #1 has special gain selections for use with the "CCIR" weighting filter.

Because of the design of that filter, the reading will be corrected internally in the software whenever 7 or 8 is selected. The normal selection for CCIR is 8 with a correction gain of 4.0 (causing the CCIR filter to cross 0 dB at 1 kHz.).

As a second choice for CCIR, Selecting 7 causes slot #1 to be used with a correction factor of 2.09 (causing the "CCIR" filter to cross 0 dB at 2 kHz for DOLBY-ARM measurements).

By convention, weighting slot #2 is normally reserved for "A" weighting, although there are no gain corrections, etc. here.

RETURNS

Returns the number of the weighting filter actually set.

INITIAL STATE

Weighting set to NONE (0) by lvf_init.

CAUTIONS

Weighting does not affect Wow & Flutter measurements. (see lvf_set_wfweight)

Weighting slot #1 has a software gain correction, See the note above.

Selecting an empty filter slot will produce erroneous readings.

NAME

lvf_set_wfweight -- set wow & flutter weighting

SYNOPSIS

void lvf_set_wfweight(weight);

char *weight; weighting string;

DESCRIPTION

Selects weighting filter for Wow & Flutter only.

Available weighting filters along with the minimal and recommended setting strings are:

minimal	recommended	mode
"WEI"	"WEIGHTED"	WEIGHTED
"WEIHB"	"WEIGHTED-HB"	WEIGHTED-HIGH BAND
"UNW"	"UNWEIGHTED"	UNWEIGHTED
"UNWHB"	"UNWEIGHTED-HB"	UNWEIGHTED-HIGH BAND
"WID"	"WIDEBAND"	WIDEBAND
"SCR"	"SCRAPE"	SCRAPE-FLUTTER

The input string is searched for the first occurrence of the minimal 3 letters as above. (upper or lower case).

If a "WEI" or "UNW" is found, the string is then searched for an "HB".

RETURNS

None.

INITIAL STATE

Wow & Flutter weighting set to "WEIGHTED" by lvf_init.

CAUTIONS

The W&F-LVF hardware option is required for this function to operate correctly.

DUS (DELAY UNTIL SETTLED) FUNCTIONS OVERVIEW

The readings functions described in the previous sections all return raw unprocessed readings. While some applications require this, generally a reading should not be considered valid until both the device under test and the measurement instrument have settled to a steady state.

The following group of functions provide the capability of reading settled data.

The settling algorithm itself resides in the dus_read function. The other functions set the specifications for qualifying data and perform some other utility tasks.

The settling algorithm will return data only after the data has met the requirements for settling. As each reading is taken from the hardware, it is compared against previous readings to determine if it has settled.

The number of previous readings it is compared against is set by the various dus_set_XXXPTS functions.

The limits for each comparison are set by the dus_set_XXXTOL, dus_set_XXXFLOOR, and dus_set_shape functions.

Should a series of readings never settle, a time-out limit is provided by the dus_set_timeout function.

The dus_clear and dus_phase_clear functions erase previous data from the settling algorithm.

SETTLING ALGORITHM DESCRIPTION

Data is considered settled when a series of readings is within specified limits of each other. The comparison limit used is calculated by combining the specified tolerance, floor, shape, and the individual readings.

As each new reading arrives from the hardware, it is stored in an array. At any given time the array holds N readings, where N is the number of readings taken from the hardware since the last dus_clear. This array will be represented as:

```
A(0), A(1), A(2), A(3), A(4), A(5)...
```

where A(0) is the most recent reading.

A(0) is compared against A(1), then against A(2), etc. up to A(PTS 1) where PTS is the number set by the appropriate dus_PTS function.

For example: if PTS = 3 then A(0) will be compared against A(1) and A(2). PTS is the total number of readings involved in the comparisons.

For each comparison A(0) against A(X), the allowable difference D_x is calculated in the following sequence:

```
D = the maximum of ( A(0) * TOLERANCE ) or FLOOR
```

$D_x = D * (SHAPE^{(X-1)})$ (SHAPE to the power (X-1))

Note that SHAPE is set by the dus_shape function, TOLERANCE by the appropriate dus_TOL function and FLOOR by the appropriate dus_FLOOR function.

Some of the implications of this calculation are:

1. The allowable difference is never less than FLOOR.
2. The comparison of A(0) against A(1) is not affected by SHAPE.
3. If SHAPE = 1 all comparisons use the same difference value.
4. If SHAPE = 2 , the comparison of A(0) against A(2) allows twice the difference as against A(1).

The comparison process continues until either the number of points specified by PTS fall within the allowable differences or timeout occurs.

If timeout occurs, the returned reading will be the average of the last N readings. This is based on the assumption that the unsettled condition is due to noise and that averaging will help. (If averaged data is preferred. it also also possible to force the settling algorithm to always average. See the next section).

If settling occurs, the returned reading is the last reading taken from the hardware, no averaging occurs.

On the S1.EXE settling panel:

```
FLAT                - SHAPE = 1.0
EXPONENTIAL    - SHAPE = 2.0
```

NAME

dus_init -- initialize the DUS settings

SYNOPSIS

void dus_init(void);

DESCRIPTION

This function initializes the DUS settings to a known state. It must be called before dus_read can be used.

Note that dus_init is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be using more than one LVF module.

The tolerances are set as follows:

AMPL	1.0 %
THD 3.0 %	
IMD	3.0 %
W&F	5.0 %
INVOLTS	1.0 %
FREQ	0.5 %
PHASE	(there is no tolerance for phase)
DCV	0.2 %
OHMS	0.5 %
DIN	0.0 %
DSP0	1.0 %
DSP1	1.0 %
DSP2	1.0 %
DSP3	1.0 %

Note that DSP0 - DSP3 are also set by each DSP program loaded (see the dsp_restore() function)

The floors are set as follows:

AMPL	1.0E-7	Volts	(100 nV)
THD	7.0E-5	%	(0.00007 %)
IMD	3.0E-5	%	(0.00003 %)
W&F	2.0E-4	%	(0.0002 %)
INVOLTS	2.5E-5	Volts	(25 uV)
FREQ	2.0E-4	Hertz	(0.0002 Hz)
PHASE	0.5	degrees	
DCV	5.0E-4	Vdc	(500 uVdc)
OHMS	1.0E-1	Ohms	(100 mOhms)
DIN	0.0	Counts	
DSP0	1.0E-7	(units vary)	
DSP1	1.0E-7		
DSP2	1.0E-7		
DSP3	1.0E-7		

Note that DSP0 - DSP3 are also set by each DSP program loaded (see the dsp_restore() function). The unit of the floor is also program dependent.

The points are set as follows:

MEAS	3 POINTS
INVOLTS	3 POINTS
FREQ	3 POINTS
PHASE	2 POINTS
DMM	3 POINTS
DIN	1 POINT
DSP0	3 POINTS
DSP1	3 POINTS
DSP2	3 POINTS
DSP3	3 POINTS

Shape is set to 2.0 (same as EXPONENTIAL in S1.EXE)

Timeout is set to 4.0 seconds.

SEE ALSO

ap_setup

NAME

dus_read -- make settled measurements from the LVF and DCX

SYNOPSIS

timeout = dus_read(do_meas, do_involts, do_freq, do_phase, do_dmm, do_din, &meas, &involts, &freq, &phase, &dmm, &din)

int do_meas;	make a main voltmeter reading
int do_involts;	make a input voltage reading
int do_freq;	make a frequency reading
int do_phase;	make a phase (or polarity) reading
int do_dmm;	make a DMM reading (DCX)
int do_din;	make a digital input reading (DCX)

(any non-zero value causes the reading to be taken)

float meas;	main reading
float involts;	voltage reading
float freq;	frequency reading
float phase;	phase (or polarity) reading
float dmm;	DMM reading
float din;	digital reading

(NULL pointers may be sent for readings not being taken)

int timeout;	timeout flag

DESCRIPTION

This function delays returning any readings until settling has occurred.

> **DSP readings have not been added to this function because of the large number of arguments that would result. Instead, for DSP readings, the new function dus_read_struct has been added to the library.**

Since the hardware is capable of taking 4 measurements at once (more with DCX), this function is passed flags to indicate which measurements to do simultaneously. Each settling process is run independently of the others, and no readings are returned until all of the requested readings have settled or timeout occurs.

The do_meas, etc. flags are set to a non-zero value to cause their appropriate reading to be taken. 0 causes the reading not to be taken.

The units for each of the returned values is the same as for their raw readings functions, That is:

```
MEAS      same as lvf_measure()
INVOLTS   same as lvf_read_involts()(always Volts)
FREQ      same as lvf_read_freq()(always Hertz)
PHASE     same as lvf_read_phase() or
lvf_read_polarity()
          (always degrees)
DMM       same as dcx_read_dmm()(Vdc or Ohms)
DIN       same as dcx_read_din()
```

The readings are all taken from the selected input channel except when the LVF is in a 2 channel mode. 2 channel mode causes the input voltmeter and the frequency counter to be connected to the opposite channel. For example, if channel "A" is selected (lvf_channel("A")) and the LVF is in one of the 2 channel modes, the input voltmeter and the frequency counter will be connected to channel "B". (See lvf_set_mode for a description of the 2 channel modes.)

The comparison tolerances and floors are set the various lvf_set_TOL and lvf_set_FLOOR functions.

The timeout limit is set by lvf_set_timeout(). Any measurement that has not settled by timeout will have the average of its last 6 readings returned.

The ability to return polarity information instead of phase is controlled by the dus_polarity function. Normally, phase is returned. If dus_polarity() is called, then polarity information may be returned. See dus_polarity for further information.

Note: A special service is performed for phase measurements. Since the hardware aliases every 360 degrees, dus_read compares its settled phase reading against the settled phase reading returned during a previous call. If the difference is more than 180 degrees, dus_read added or subtracts 360 degrees from the reading in an attempt to correct for that aliasing. This means that the phase reading can be much more than 360 degrees. dus_phase_clear resets this correction to 0, and can be called before each dus_read to prevent any correction.

RETURNS

Returns a 1 if timeout occurred, 0 otherwise.

SEE ALSO

dus_clear, dus_phase_clear, and the many dus_set_PTS and dus_set_TOL functions

lvf_set_mode (to set the main measurement mode.)

dus_polarity (if phase or polarity is being used)

dus_read_struct

CAUTIONS

Pointers to floats must be passed for the readings, not the floats themselves.

NAME

dus_read_struct -- make settled measurements from all modules using a data structure.

SYNOPSIS

timeout = dus_read_struct(struct DUS_READ_STRUCT *dus)

struct DUS_READ_STRUCT *dus	pointer to struct as defined in AP.H
int timeout;	timeout flag

DESCRIPTION

This function performs the same tasks as dus_read with the difference that the data passing is handled in a structure instead of a long argument list.

The structure DUS_READ_STRUCT is defined in AP.H as follows:

```
struct DUS_READ_STRUCT {

   int    do_meas;
   int    do_lvl;
   int    do_freq;
   int    do_phs;
   int    do_dmm;
   int    do_din;
   int    do_dsp_read0;
   int    do_dsp_read1;
   int    do_dsp_read2;
   int    do_dsp_read3;

   float  meas;
   float  lvl;
   float  freq;
   float  phs;
   float  dmm;
   float  din;
   float  dsp_read0;
   float  dsp_read1;
   float  dsp_read2;
   float  dsp_read3;

};
```

The structure members replace the list of arguments used in dus_read above and perform the same function.

The default structure named "dus" is allocated by the library but may be replaced by user allocated structures by using the structure definition in AP.H and passing a pointer to the new structure.

RETURNS

Returns a 1 if timeout occurred, 0 otherwise.

SEE ALSO

dus_read for a complete functional description.

DSP program dependent functions section for a description of the DSP readings.

NAME

dus_clear -- clear all data from settling algorithm

SYNOPSIS

void dus_clear(void);

DESCRIPTION

This function causes all data to be purged from the settling algorithm and should normally be called at the beginning of a testing sequence such as a sweep.

Once the settling algorithm is cleared, N new readings will need to be taken before settling can occur, where N is the number of points set by the appropriate dus_set_PTS functions.

Normally, the algorithm is not cleared and it is possible for settling to occur on the first hardware reading taken by dus_read.

RETURNS

None.

NAME

dus_phase_clear -- clear phase correction

SYNOPSIS

void dus_phase_clear(void);

DESCRIPTION

This function causes the phase correction to be reset to 0.

Phase correction is a service is performed for phase measurements made with dus_read.

Since the hardware aliases every 360 degrees, dus_read compares its settled phase reading against the settled phase reading returned during a previous call. If the difference is more than 180 degrees, dus_read added or subtracts 360 degrees from the reading in an attempt to correct for that aliasing. This means that the phase reading can be much more than 360 degrees. dus_phase_clear resets this correction to 0, and can be called before each dus_read to prevent any correction.

This function should normally be called at the beginning of a testing sequence such as a sweep.

RETURNS

None.

NAME

dus_polarity -- cause dus_read to return polarity information instead of phase.

SYNOPSIS

void dus_polarity(on_off);

int on_off; 1 for polarity, 0 for phase

DESCRIPTION

This function sets whether dus_read returns phase or polarity information.

Phase is between channels A and B in degrees.

Polarity is the absolute polarity between channels A and B and can be either 0 or 180 degrees. A special asymmetric waveform is required to determine this. With System One this waveform is produced by the BUR-GEN option. See lvf_read_polarity for further information.

RETURNS

None

INITIAL STATE

dus_polarity is set to OFF (phase) by dus_init.

SEE ALSO

dus_read, lvf_read_polarity, lvf_read_phase

NAME

dus_set_ampltol -- set AMPL settling tolerance.
dus_set_thdtol -- set THD settling tolerance.
dus_set_imdtol -- set IMD settling tolerance.
dus_set_wftol -- set W&F settling tolerance.
dus_set_involtstol -- set input Volts settling tolerance.
dus_set_freqtol -- set frequency settling tolerance.

dus_set_dcvtol -- set DC VOLTS (DMM) settling tolerance.
dus_set_ohmtol -- set OHMS (DMM) settling tolerance.
dus_set_dintol -- set digital input settling tolerance.

dus_set_dsp0tol -- set DSP reading 0 settling tolerance.
dus_set_dsp1tol -- set DSP reading 1 settling tolerance.
dus_set_dsp2tol -- set DSP reading 2 settling tolerance.
dus_set_dsp3tol -- set DSP reading 3 settling tolerance.

SYNOPSIS

```
new_tol = dus_set_ampltol( tol);
new_tol = dus_set_thdtol( tol);
new_tol = dus_set_imdtol( tol);
new_tol = dus_set_wftol( tol);
new_tol = dus_set_involtstol( tol);
new_tol = dus_set_freqtol( tol);

new_tol = dus_set_dcvtol( tol);
new_tol = dus_set_ohmtol( tol);
new_tol = dus_set_dintol( tol);

new_tol = dus_set_dsp0tol( tol);
new_tol = dus_set_dsp1tol( tol);
new_tol = dus_set_dsp2tol( tol);
new_tol = dus_set_dsp3tol( tol);

double tol;          tolerance in % of reading
double new_tol;      tolerance actually set in % of reading
```

DESCRIPTION

These functions set the tolerances used in the settling algorithm.

Each tolerance is specified in % of reading.

Note that 4 of the functions affect the measurement from the main voltmeter. When a measurement is made from the main voltmeter, the appropriate tolerance is selected by dus_read based on the main voltmeters mode as follows:

mode	description	tolerance used
AMPL	AMPLITUDE	AMPL
BANDPASS	BANDPASSED AMPLITUDE	AMPL
BANDREJECT	BANDREJECTED AMPLITUDE	AMPL
THDAMPL	THD as an amplitude	AMPL
THD%	THD as a percent	THD
SMPTE	SMPTE IMD	IMD
CCIF	CCIF IMD	IMD
DIM	DIM	IMD
W&F	WOW and FLUTTER	WF

This method of setting tolerances associates the appropriate tolerance more with the type of measurement than with the voltmeter itself. Usually, the tolerances can be set once (or the defaults used) and forgotten.

Negative values are not accepted for tolerances.

RETURNS

Returns the tolerance actually set in %.

INITIAL STATE

The dus_init function sets the tolerances as follows:

AMPL	1.0 %
THD	3.0 %
IMD	3.0 %
W&F	5.0 %
INVOLTS	1.0 %
FREQ	0.5 %
DCV	0.2 %
OHMS	0.5 %
DIN	0.0 %

DSP0	1.0 %
DSP1	1.0 %
DSP2	1.0 %
DSP3	1.0 %

Note that DSP0 - DSP3 are also set by each DSP program loaded (see the dsp_restore() function)

NAME

dus_set_amplfloor -- set AMPL settling floor.
dus_set_thdfloor -- set THD settling floor.
dus_set_imdfloor -- set IMD settling floor.
dus_set_wffloor -- set W&F settling floor.
dus_set_involtsfloor -- set input voltmeter settling floor.
dus_set_freqfloor -- set frequency settling floor.
dus_set_phasefloor -- set phase settling floor.

dus_set_dcvfloor -- set DC VOLTS (DMM) settling floor.
dus_set_ohmfloor -- set OHMS (DMM) settling floor.
dus_set_dinfloor -- set digital input settling floor.

dus_set_dsp0floor -- set DSP reading 0 settling floor.
dus_set_dsp1floor -- set DSP reading 1 settling floor.
dus_set_dsp2floor -- set DSP reading 2 settling floor.
dus_set_dsp3floor -- set DSP reading 3 settling floor.

SYNOPSIS

```
new_floor = dus_set_amplfloor( floor);
new_floor = dus_set_thdfloor( floor);
new_floor = dus_set_imdfloor( floor);
new_floor = dus_set_wffloor( floor);
new_floor = dus_set_involtsfloor( floor);
new_floor = dus_set_freqfloor( floor);
new_floor = dus_set_phasefloor( floor);

new_floor = dus_set_dcvfloor( floor);
new_floor = dus_set_ohmfloor( floor);
new_floor = dus_set_dinfloor( floor);

new_floor = dus_set_dsp0floor( floor);
new_floor = dus_set_dsp1floor( floor);
new_floor = dus_set_dsp2floor( floor);
new_floor = dus_set_dsp3floor( floor);

double floor;          floor in units of reading
double new_floor;      floor actually set in units of reading
```

DESCRIPTION

These functions set the floors used in the settling algorithm.

Each floor is specified in the units of the measurement. The units used are:

AMPL	Volts
THD %	
IMD %	
W&F %	
INVOLTS	Volts
FREQ	Hertz
PHASE	degrees
DCV Vdc	
OHMS	Ohms
DIN	Counts
DSP0-3	Dependent on the DSP program loaded.

Note that 4 of the functions affect the measurement from the main voltmeter. When a measurement is made from the main voltmeter, the appropriate floor is selected by dus_read based on the main voltmeters mode as follows:

mode	description	floor used
AMPL	AMPLITUDE	AMPL
BANDPASS	BANDPASSED AMPLITUDE	AMPL
BANDREJECT	BANDREJECTED AMPLITUDE	AMPL
THDAMPL	THD as an amplitude	AMPL
THD%	THD as a percent	THD
SMPTE	SMPTE IMD	IMD
CCIF	CCIF IMD	IMD
DIM	DIM	IMD
W&F	WOW and FLUTTER	WF

Additionally available with the version "A" hardware:

2CH-AMPL	2 CHANNEL AMPLITUDE	AMPL
2CH-AMPL%	2 CHANNEL AMPLITUDE as % between channels	AMPL
2CH-BP	2 CHANNEL BP AMPLITUDE	AMPL
2CH-BP%	2 CHANNEL BP AMPLITUDE as % between channels	AMPL

This method of setting floors associates the appropriate floor more with the type of measurement than with the voltmeter itself. Usually, the floors can be set once (or the defaults used) and forgotten.

Negative values are not accepted for floors.

RETURNS

Returns the floor actually set in same units as the passed value.

INITIAL STATE

The dus_init function sets the floors as follows:

AMPL	1.0E-7	Volts	(100 nV)
THD	7.0E-5	%	(0.00007 %)
IMD	3.0E-5	%	(0.00003 %)
W&F	2.0E-4	%	(0.0002 %)
INVOLTS	2.5E-5	Volts	(25 uV)
FREQ	2.0E-4	Hertz	(0.0002 Hz)
PHASE	0.5	degrees	
DCV	5.0E-4	Vdc	(500 uVdc)
OHMS	1.0E-1	Ohms	(100 mOhms)
DIN	1.0	Counts	
DSP0-3	1.0E-7	Unit dependent on DSP program loaded	

Note that DSP0 - DSP3 are also set by each DSP program loaded (see the dsp_restore() function)

NAME

dus_set_measpts -- set main voltmeter settling points.
dus_set_involtspts -- set input voltmeter settling points.
dus_set_freqpts -- set frequency settling points.
dus_set_phasepts -- set phase settling points.

dus_set_dmmpts -- set DCV and OHMS settling points.
dus_set_dinpts -- set digital input settling points.

dus_set_dsp0pts -- set DSP reading 0 settling points.
dus_set_dsp1pts -- set DSP reading 1 settling points.
dus_set_dsp2pts -- set DSP reading 2 settling points.
dus_set_dsp3pts -- set DSP reading 3 settling points.

SYNOPSIS

```
new_points = dus_set_measpts( points);
new_points = dus_set_involtspts( points);
new_points = dus_set_freqpts( points);
new_points = dus_set_phasepts( points);

new_points = dus_set_dmmpts( points);
new_points = dus_set_dinpts( points);

new_points = dus_set_dsp0pts( points);
new_points = dus_set_dsp1pts( points);
new_points = dus_set_dsp2pts( points);
new_points = dus_set_dsp3pts( points);

int points;          number of readings for settling
int new_points;      number of readings actually set
```

DESCRIPTION

These functions set the number of points (readings) required for settling.

1 to 6 points are allowed, where 1 point implies that the settling algorithm is disabled.

RETURNS

Returns the number of points actually set.

INITIAL STATE

The dus_init function sets the points as follows:

MEAS	3 POINTS
INVOLTS	3 POINTS
FREQ	3 POINTS
PHASE	2 POINTS
DMM	3 POINTS
DIN	1 POINT
DSP0-3	3 POINTS

SEE ALSO

settling algorithm description

NAME

dus_set_shape -- set the shape of the settling comparisons

SYNOPSIS

new_shape = dus_set_shape(shape);

double shape;	shape of settling comparisons
double new_shape;	shape actually set

DESCRIPTION

This function sets the shape used in the calculation of the settling comparisons.

Any shape 0.0 or greater is allowed, although practically, shapes below 1.0 or above 5.0 are fairly worthless. A shape of 1.0 implies that all comparisons will be done to the same value.

RETURNS

Returns the shape actually set.

INITIAL STATE

The dus_init function sets the shape to 2.0. (same as EXPONENTIAL in S1.EXE)

SEE ALSO

The complete description of SHAPE in dus_read

NAME

dus_set_timeout -- set the timeout for the settling algorithm.

SYNOPSIS

new_timeout = dus_set_timeout(timeout);

double timeout;	timeout in seconds
double new_timeout;	timeout actually set in seconds

DESCRIPTION

This function sets the timeout used in the calculation of the settling comparisons.

Timeout values of 0 to 3000 seconds (50 minutes) are allowed.

Any measurement that has not settled by the timeout will have the average of its last 6 readings returned.

RETURNS

Returns the timeout actually set.

INITIAL STATE

The dus_init function sets the timeout to 4.0 seconds.

NAME

swi_init -- initialize the switchers

SYNOPSIS

new_modnum = swi_init(modnum);

int modnum;	module number
int new_modnum;	module number actually set

DESCRIPTION

This function initializes the SWI to a known state. It must be used after power-up.

Note that swi.init is called for you by ap_setup so that calling this function directly should normally not be necessary unless writing your own ap_setup procedure.

The SWI data structures are set to default conditions, then the SWI hardware is set to match using the swi_restore function.

This function also sets the module number of the swi_ The module number must be between 0 and 15, else the default of 14 is used.

The default module number of 14 should be used unless you have a system with modified switchers.

Default initializations:

All channels off

RETURNS

The module number used is returned.

SEE ALSO

swi_restore, ap_setup

NAME

swi_restore -- restore SWI hardware

SYNOPSIS

void swi_restore(void);

DESCRIPTION

This function restores the SWI hardware to the present state of the software.

This function should be used if the hardware loses power or becomes disconnected from the computer.

This function is also used by the swi_init function.

RETURNS

None

SEE ALSO

swi_init

NAME

swi_exist -- determine if the SWI hardware is available.

SYNOPSIS

exist = swi_exist(void);

int exist; 1 if SWI exists, 0 if not

DESCRIPTION

This function determines if any switcher is connected and powered on.

The switcher status is updated each time swi_restore called. This function returns that status. Note that ap_setup and swi_init also call swi_restore and so update the status.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

If the switchers are found not to exist (but should), prompt the operator to remedy the situation, then call swi_restore before calling swi_exist again.

RETURNS

Returns a 1 if any switcher exists, 0 otherwise.

NAME

swi_set_ain -- set switcher A INPUT channel
swi_set_bin -- set switcher B INPUT channel
swi_set_aout -- set switcher A OUTPUT channel
swi_set_bout -- set switcher B OUTPUT channel

SYNOPSIS

```
new_channel = swi_set_ain( channel);
new_channel = swi_set_bin( channel);
new_channel = swi_set_aout( channel);
new_channel = swi_set_bout( channel);

int channel;          channel number
int new_channel;      channel number actually set
```

DESCRIPTION

These functions set the switcher channels.

Channel numbers 0 to 192 are available, where 0 means all channels off. Any other number results in no action taken, but the present channel is returned.

As a special case, B OUTPUT allows -1 which sets it into the COMPLEMENT A mode, which selects all channels except the number requested. This is useful for cross-talk measurements.

RETURNS

Returns the channel actually set.

INITIAL STATE

The swi_init function sets all channels OFF.

NAME

dcx_select -- select the active DCX

SYNOPSIS

new_dcxnum = dcx_select(dcxnum);

int dcxnum;	DCX reference number
int new_dcxnum;	DCX number actually set

DESCRIPTION

This function selects which DCX will be used as the object of all other DCX calls.

This function should be used before any other DCX function calls, including dcx_init, are used.

> **Note that dcx_select is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one DCX. Using multiple DCXs in a system requires modification of the additional DCX circuit boards to change addresses. Contact Audio Precision for further information**

The software internally maintains 4 different sets of DCX data. This function simply selects which set of data to use until dcx_select is called again.

The passed value must be from 1 to 4. If it is out of this range, no new selection is made.

> **At present, the DCX hardware does not support the changing of addresses, so only 1 DCX may be used. This function is provided for consistency and for possible future expansion.**

RETURNS

The DCX number selected is returned.

SEE ALSO

dcx_restore, dcx_init, ap_setup

NAME

dcx_init -- initialize DCX

SYNOPSIS

new_modnum = dcx_init(modnum);

int modnum;	module number
int new_modnum;	module number actually set

DESCRIPTION

This function initializes the DCX to a known state. It must be used after power-up.

> **Note that dcx_init is called for you by ap_setup so that calling this function directly should only be necessary if you are going to be writing your own ap_setup procedure or using more than one DCX. Using multiple DCXs in a system requires modification of the additional DCX circuit boards to change addresses. Contact Audio Precision for further information**

The selected DCX data structure is set to default conditions, then the DCX hardware is set to match using the dcx_restore function.

This function also sets the module number of the selected DCX (see dcx_select) The module number must be between 0 and 15, else the default of 11 is used.

> **The default module number of 11 should be used unless you have a system specifically modified to have multiple DCXs.**

Default initializations:

1. Set DMM function to VOLTS
2. Set DMM to AUTORANGE, inputs enabled
3. Set DMM to 6 readings per second, freerunning
4. Set DCVOLTS 1 and 2 outputs enabled
5. Set DCVOLTS 1 and 2 to 0.0 Volts
6. Set DIN to 32 reading per second, internal strobe
7. Set DIN format to two's complement
8. Set DOUT to 0
9. Set DOUT format to two's complement
10. Set Ports A, B, C, and D to 0
11. Set GATE DELAY to 50 msec.

RETURNS

The module number used is returned.

SEE ALSO

dcx_restore, dcx_select, ap_setup

NAME

dcx_restore -- restore DCX hardware

SYNOPSIS

void dcx_restore(void);

DESCRIPTION

This function restores the DCX hardware to the present state of the software.

This function should be used if the hardware loses power or becomes disconnected from the computer.

This function is also used by the dcx_init function.

SEE ALSO

dcx_init

NAME

dcx_exist -- determine if the DCX hardware is available.

SYNOPSIS

exist = dcx_exist(void);

int exist; 1 if DCX exists, 0 if not

DESCRIPTION

This function determines if the selected DCX is connected and powered on.

The DCX status is updated each time dcx_restore called. This function returns that status. Note that ap_setup and dcx_init also call dcx_restore and so update the status.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

If the DCX is found not to exist (but should), prompt the operator to remedy the situation, then call dcx_restore before calling dcx_exist again.

RETURNS

Returns a 1 if the selected DCX exists, 0 otherwise.

NAME

dcx_data_pulse -- pulse the 'data acquired' output

SYNOPSIS

void dcx_data_pulse(void);

DESCRIPTION

This function outputs a single pulse on the 'data acquired' pin (pin # 3) of the DCX's program control output.

This is normally used to notify an external device that a reading has been taken, but may be used for other purposes.

RETURNS

None.

NAME

dcx_data_trigger -- pulse the 'trigger' output with delay

SYNOPSIS

void dcx_data_trigger(msec);

double msec; delay time in milliseconds

DESCRIPTION

Causes the DCX to wait x milliseconds then output a single pulse on the 'trigger' pin (pin # 4) of the DCX's program control output.

This is normally used to trigger an external device to take a reading, but may be used for other purposes.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

NAME

dcx_din_trigger -- trigger a new digital input reading with delay

SYNOPSIS

void dcx_din_trigger(msec);

double msec; trigger delay time in milliseconds

DESCRIPTION

Causes the DCX to wait x milliseconds then take a reading from the digital input.

Note that the digital input must have its readings latched by either an external strobe or an internal strobe of rate set by dcx_set_din_rate.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

NAME

dcx_dmm_trigger -- trigger a new DMM reading with delay

SYNOPSIS

void dcx_dmm_trigger(msec);

double msec; trigger delay time in milliseconds

DESCRIPTION

Causes the DCX to wait x milliseconds then restart the DMM readings cycle. If a reading is in progress, it is aborted. The dcx_read_dmm function, if called after this, will return the reading from this trigger.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

NAME

dcx_dmm_freerun -- set DMM readings to freerun

SYNOPSIS

void dcx_dmm_freerun(freerun);

int freerun; 1 for FREERUN, 0 for OFF

DESCRIPTION

The DMM's reading cycles are normally freerunning, which means reading will be returned at the reading rate without the need of explicit triggers.

Normally this is the desired mode of operation. A single trigger is made after a stimulus change, a new reading cycle is begun immediately, and then readings are taken until qualified as settled.

The DMM hardware is a bit different than the rest of System One in that it is not retriggerable. This means that any reading cycle must complete before a new cycle can begin. Therefore, there will be an undefined delay before a trigger can actually be accomplished. This delay depends on the reading rate and can be up to a complete reading cycle.

For time sensitive applications, it is fastest to stop the DMM from freerunning and trigger a single reading cycle whenever needed.

Note that if the DMM is not freerunning, you should never call dcx_read_dmm before dcx_rdy_dmm shows that a reading is ready.

RETURNS

None.

INITIAL STATE

The DMM is set to FREERUN by dcx_init.

CAUTIONS

Explicit triggers (dcx_dmm_trigger) must be used if the DMM is not freerunning. See above.

NAME

dcx_set_dmm_mode -- set DMM measurement mode.

SYNOPSIS

void dcx_set_dmm_mode(mode);

char *mode; mode string

DESCRIPTION

This function sets the DMM measurement mode.

Available modes along with the minimal and recommended setting strings are:

minimal	recommended	mode
"V"	"VOLTS"	DC VOLTS
"O"	"OHMS"	OHMS

The input string is searched for the first occurrence of a valid mode character. The case of letters are ignored.

If the mode string cannot be interpreted, the error code is set and nothing is done. See ap_get_errcode for definitions of the error codes.

RETURNS

None.

INITIAL STATE

The DMM measurement mode is set to VOLTS by dcx_init.

SEE ALSO

ap_get_errcode for definitions of the error codes.

NAME

dcx_set_dmm_on -- set DMM input on or off

SYNOPSIS

void dcx_set_dmm_on(on_off);

int on_off; 1 for ON, 0 for OFF

DESCRIPTION

This function connects or disconnects the DMM from the front panel jacks.

This allows the DMM to be wired to the circuit under test yet not be connected until needed. This is so that there is no possibility of the DMM input characteristics degrading the results of any other measurements being made by System One.

RETURNS

None.

INITIAL STATE

The input is set to ON by dcx_init.

NAME

dcx_set_dmm_rate -- set DMM reading rate

SYNOPSIS

new_rate = dcx_set_dmm_rate(rate);

int rate;	rate in readings per second.
int new_rate;	rate actually set in readings per second.

DESCRIPTION

This function selects the DMM reading rate closest to the input value.

Rates available are:

```
25 and 6 readings per second.
```

RETURNS

Returns the rate actually set in readings per second.

INITIAL STATE

The measurement rate is set to 6 per second by dcx_init.

NAME

dcx_set_dmm_range -- set DMM input range

SYNOPSIS

new_volts_ohms = dcx_set_dmm_range(volts_ohms);

double volts_ohms;	range in Volts or ohms
double new_volts_ohms;	range actually set in Volts or ohms

DESCRIPTION

This function sets the DMM's input range and returns the nominal full scale of range in use.

The possible ranges for DC Volts mode are:

```
500 V
200 V
20 V
2 V
0.2 V
```

The possible ranges for Ohms mode are:

```
2M Ohms
200k Ohms
20k Ohms
2k Ohms
200 Ohms
```

If the range specified is 0.0, autorange will be selected.

A common use of these functions is in fixing the input range by obtaining the range and then using that value for this call.(see example below)

RETURNS

This function returns the full scale volts or ohms of the appropriate input range in use, or 0.0 if the range is set to autorange.

INITIAL STATE

The DMM input range is set to AUTORANGE by dcx_init.

SEE ALSO

dcx_read_dmm_range

EXAMPLES

```
double volts, dcx_read_dmm_range();

volts = dcx_read_dmm_range();       /*get the range*/
dcx_set_dmm_range(volts);           /*fix the range*/
```

NAME

dcx_set_dcv1 -- set DC output 1 voltage
dcx_set_dcv2 -- set DC output 2 voltage

SYNOPSIS

```
new_volts = dcx_set_dcv1( volts);
new_volts = dcx_set_dcv2( volts);

double volts;          amplitude in DC Volts
double new_volts;      amplitude actually set in DC Volts
```

DESCRIPTION

These functions set the voltage at the DCX's DC outputs.

Because of amplitude limitations and digital quantization, that actual voltage set may not be exactly what was requested. Therefore, it is important to check the returned voltage, and to use that value as the actual voltage the outputs are at.

RETURNS

This function returns the actual voltage the appropriate output is set to.

If the requested voltage exceeds hardware limitations, an error code is set and the present voltage is returned. See ap_get_errcode for definitions of the error codes.

INITIAL STATE

Voltages are set to 0.0 by dcx_init.

SEE ALSO

DCX hardware specifications for amplitude limits.

ap_get_errcode for definitions of the error codes.

NAME

dcx_set_dcv1_out -- set DC Volts output 1 on or off
dcx_set_dcv2_out -- set DC Volts output 2 on or off

SYNOPSIS

```
void dcx_set_dcv1_out( on_off);
void dcx_set_dcv2_out( on_off);

int on_off;                    1 for ON, 0 for OFF
```

DESCRIPTION

These functions set DC Volts output 1 or 2 to ON or OFF.

The off state disconnects the outputs from the front panel.

RETURNS

None.

INITIAL STATE

Both outputs are set to OFF by dcx_init.

NAME

dcx_set_din_format -- set Digital Input format
dcx_set_dout_format -- set Digital Output format

SYNOPSIS

```
int dcx_set_din_format( format);
int dcx_set_dout_format( format);

int format;                    1 for BCD, 0 for TWO'S COMPLEMENT
```

DESCRIPTION

These functions set the format of the digital inputs and outputs.

The digital ports are 21 bits plus a sign bit.

The normal format is two's complement. This format combines the bits into a 22 bit word that follows normal two's complement conventions (1 is represented as 3FFFFF hex).

The BCD (Binary coded decimal) format is a signed magnitude representation (-1 is represented as 200001 hex, -10 is 200010 hex, etc.). As is normal in the BCD format, each decimal digit is represented by 4 bits.

RETURNS

The format actually set.

INITIAL STATE

Both digital input and output are set to two's complement by dcx_init.

SEE ALSO

dcx_set_dout, dcx_read_din

NAME

dcx_set_din_rate -- set digital input strobe rate

SYNOPSIS

new_rate = dcx_set_din_rate(rate);

int rate:	rate in readings per second.
int new_rate:	rate actually set in readings per second.

DESCRIPTION

This function selects the digital input strobe rate closest to but above the input value.

If a 0 is passed, EXTERNAL strobe is selected. This is available as a pin on the digital input connecter.

Internal strobe rates available are:

```
32, 16, 8, and 4 readings per second.
```

RETURNS

Returns the rate actually set in readings per second.

INITIAL STATE

The rate is set to 32 per second by dcx_init.

SEE ALSO

dcx_read_din

NAME

dcx_set_dout -- set digital output

SYNOPSIS

new_num = dcx_set_dout(num);

double num;	number to be output
double new_num;	number actually output

DESCRIPTION

This function sets the DCX's digital outputs.

The output format is either two's complement or BCD as set by dcx_set_dout_format

RETURNS

This function returns the actual number that the output is set to. This is the same as the number passed but truncated to 21 bits magnitude plus sign.

INITIAL STATE

The digital output is set to 0 by dcx_init.

SEE ALSO

dcx_set_dout_format

NAME

dcx_set_port_a -- set port A 8-bit output
dcx_set_port_b -- set port B 8-bit output
dcx_set_port_c -- set port C 8-bit output
dcx_set_port_d -- set port D 8-bit output

SYNOPSIS

```
void dcx_set_port_a( num);
void dcx_set_port_b( num);
void dcx_set_port_c( num);
void dcx_set_port_d( num);

int num;                    number to output in port
```

DESCRIPTION

These functions set the 8-bit output ports.

The number can be from 0 to 255. Larger numbers are truncated to 8 bits.

RETURNS

None.

INITIAL STATE

All 8-bit output ports are set to 0 by dcx_init.

NAME

dcx_set_swpgate -- set the Sweep Gate output

SYNOPSIS

void dcx_set_swpgate(sweep);

int sweep; 1 for SWEEPING, 0 for not

DESCRIPTION

This function sets the 'sweep gate' and 'delayed sweep gate' pins of the DCX's program control output.

The 'sweep gate' pin changes immediately, the 'delayed sweep gate' changes after the delay time set by dcx_set_gatedly.

These pins are normally used to notify an external device that a sweep is in progress and to cause some action during a sweep, but may be used for other purposes.

RETURNS

None.

INITIAL STATE

The sweep gate pins are set low by dcx_init.

SEE ALSO

dcx_set_gatedly

NAME

dcx_set_chlevel -- set Channel A/B output

SYNOPSIS

void dcx_set_chlevel(channel);

int channel; 0 for channel A, 1 for B

DESCRIPTION

This function sets the 'channel A/B' pin of the DCX's program control output.

This pin is normally used to notify an external device which channel is being used, but may be used for other purposes.

RETURNS

None.

INITIAL STATE

The channel A/B pin is set low by dcx_init.

NAME

dcx_set_gatedly -- set delay for delayed sweep gate output

SYNOPSIS

new_time = dcx_set_gatedly(time);

double time;	delay time in seconds
double new_time;	delay time actually set in seconds

DESCRIPTION

This function set the delay time for the 'delayed sweep gate' pin on the DCX's program control output port.

After a call to dcx_set_swpgate, the 'sweep gate' pin changes immediately, the 'delayed sweep gate' changes after the delay time set by dcx_set_gatedly.

Delays may range from 50 msec to 12.75 sec. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 50 msec and numbers greater then 12.75 are changed to 12.75).

RETURNS

Actual delay value used.

NAME

dcx_rdy_din -- determine if a digital input reading is ready.
dcx_rdy_dmm -- determine if a DMM reading is ready.
dcx_rdy_dmm_range -- determine if a DMM range reading is available.
dcx_rdy_key -- determine if a program control input key is ready.

SYNOPSIS

```
rdy = dcx_rdy_din( void);
rdy = dcx_rdy_dmm( void);
rdy = dcx_rdy_dmm_range( void);
rdy = dcx_rdy_key( void);

int rdy;                    1 if ready, 0 if not
```

DESCRIPTION

This function determines if the appropriate reading is ready.

Because readings functions do not return until a reading is ready, these functions may be used to avoid waiting for a reading. These functions do NOT clear their respective ready statuses and so may be called any number of times. Only a call to the reading function will clear a ready status.

If the particular reading is found to be ready, then a subsequent call to the corresponding readings function will be guaranteed to return quickly.

RETURNS

Returns a 1 if a reading is ready. Returns a 0 otherwise.

SEE ALSO

dcx_din_trigger, dcx_dmm_trigger, dcx_dmm_freerun, dcx_read_din, dcx_read_dmm, dcx_read_dmm_range, dcx_read_key

NAME

dcx_read_din -- make a DIGITAL INPUT reading

SYNOPSIS

reading = dcx_read_din(void);

double reading; reading

DESCRIPTION

This function reads the DCX's digital input.

See dcx_set_din_format for a description of the input formats.

If a reading is not ready when this function is called, it will wait for a reading to become available.

RETURNS

The most recent reading is returned as above.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the hardware is set for an external strobe which is not received by the time this function is called. It may also indicate that the hardware is disconnected or not powered on.

CAUTIONS

If external strobes are being used (see dcx_set_din_rate), dcx_rdy_din should always checked before this function is called.

SEE ALSO

dcx_set_din_format, dcx_rdy_din, dcx_set_din_rate

NAME

dcx_read_dmm -- make a DMM reading

SYNOPSIS

reading = dcx_read_dmm(void);

double reading; reading (see below for unit)

DESCRIPTION

This function makes a reading from the DMM.

The reading is taken using the selected measurement mode and the units specified by that mode.

The available modes and their corresponding units are:

DC VOLTS	- Volts
OHMS	- Ohms

If a reading is not ready when this function is called, it will wait for a reading to become available. Any particular reading will be returned only once.

RETURNS

The most recent reading is returned as above.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the hardware is over-ranged for any mode. It may also indicate that the hardware is disconnected or not powered on.

An error return will also occur if this function is called when the DMM is NOT freerunning and has not been triggered.

SEE ALSO

dcx_rdy_dmm, dcx_dmm_freerun

CAUTIONS

Explicit triggers (dcx_dmm_trigger) must be used if the DMM is not freerunning.

NAME

dcx_read_dmm_range -- make a DMM range reading

SYNOPSIS

reading = dcx_read_dmm_range(void);

double reading; reading (see below for unit)

DESCRIPTION

This function reads the DMM's input range and returns the nominal full scale of range in use.

The unit used are specified by the DMM's measurement mode.

The possible ranges for DC Volts mode are:

```
500 V
200 V
20 V
2 V
0.2 V
```

The possible ranges for Ohms mode are:

```
2M Ohms
200k Ohms
20k Ohms
2k Ohms
200 Ohms
```

If a reading is not ready when this function is called, it will wait for a reading to become available.

RETURNS

The most recent reading is returned as above.

The number -1E34 is returned if there is a hardware error encountered. This will occur if the hardware is disconnected or not powered on.

SEE ALSO

dcx_rdy_dmm_range

NAME

dcx_read_key -- make a PROGRAM CONTROL INPUT reading

SYNOPSIS

key = dcx_read_key(void);

int key; 8 bit key pattern

DESCRIPTION

This function reads the DMM's PROGRAM CONTROL INPUT.

This input port is designed for a simple keyboard consisting of 8 closures to ground. Any change at this port qualifies as a new reading. Keys are debounced and double key closures are not allowed.

Each key closure to ground is inverted and read as a 1 bit in the 8-bit input word. Thus the input will have one of the 8 following values:

1 = key 0
2 = key 1
4 = key 2
8 = key 3
16 = key 4
32 = key 5
64 = key 6
128 = key 7

If a new key is not ready, a 0 is returned immediately. This is in contrast to other readings functions that wait for a reading.

Any particular key combination will be returned only once.

RETURNS

The most recent key as above, or 0 if no new keys are ready.

SEE ALSO

dcx_rdy_key

DSP (Digital Signal Processor) FUNCTIONS OVERVIEW

The DSP is the most complicated of the System One modules. Programming the DSP therefore involves more effort than for any of the other modules.

> **This manual does not attempt to describe the architecture of the DSP hardware. It is essential that the Audio Precision DSP User's Manual be read and understood before attempting to program the DSP.**
>
> **It also highly recommended that new test setups be designed and verified using the S1.EXE software included with System One.**

The DSP module is unique in that it can change its functionality by loading different programs. At present, there are 5 such programs available:

FFTGEN.DSP	Signal generation and FFT analysis
FFTSLIDE.DSP	FFT analysis with pre and post trigger
FASTEST.DSP	Rapid Response, Distortion, Noise testing
GENANLR.DSP	Digital sinewave generation and distortion analysis
HARMONIC.DSP	Individual harmonic amplitude measurements
BITTEST.DSP	Digital data error measurements

These programs are distributed as files on the "DSP Tests and Programs" diskette provided with each System One that includes a DSP module.

> **Again, this manual does not attempt to describe the functionality of the various DSP programs. This information is in the Audio Precision DSP User's Manual**

The DSP functions are in two main groups, program dependent and program independent. Program dependent functions are used differently by each DSP program. These functions are:

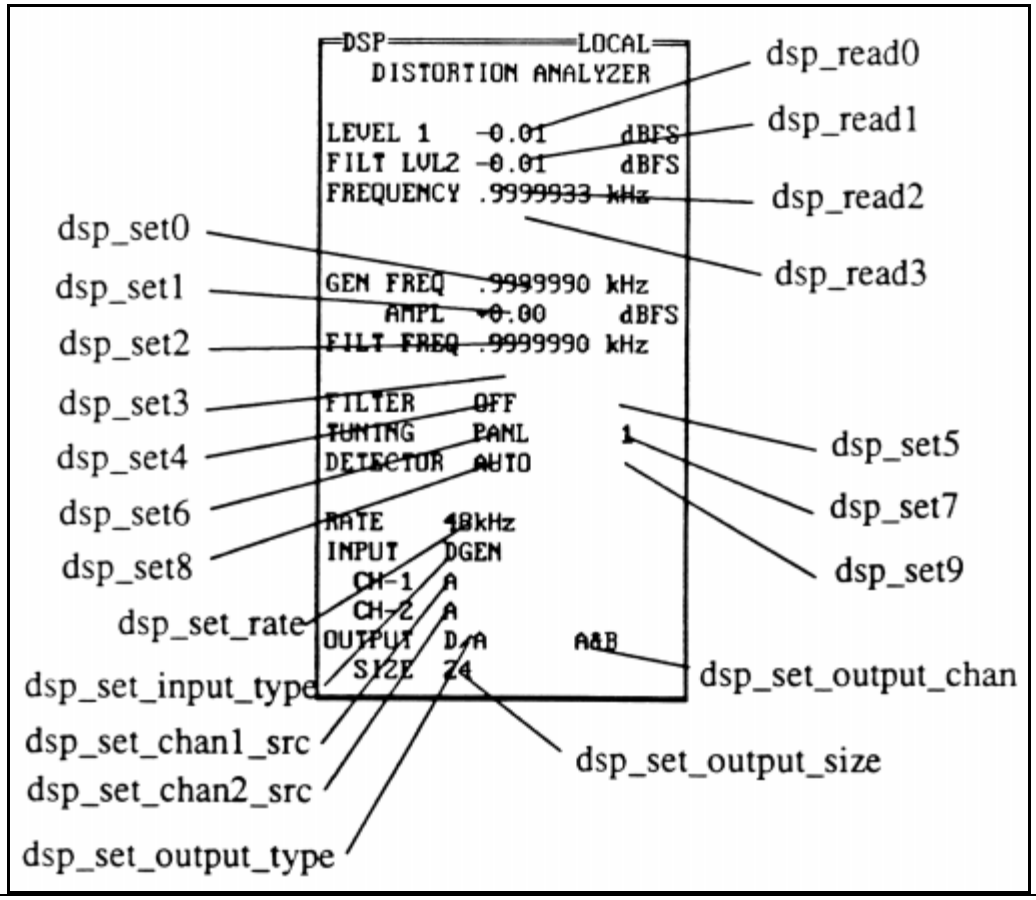
dsp_read0()	through	dsp_read4()
dsp_read0_unit()	through	
dsp_read3_unit()		
dsp_set0()	through	dsp_set9()

These functions are given names for each program (using defines) in the included C header file APDSP.H. It is highly recommended that these specific names be used instead of the "generic" names like dsp_read0. These names and other information is listed in the Program Dependent Information sections in this manual.

The dsp_rate command is program dependent only in that the choice of available rates is program dependent.

Program independent functions have the same functionality for all DSP programs. The rest of the DSP functions fall into this category.

Functionality of the various readings and settings is described in the DSP User's Manual. The following diagram shows how the S1.EXE panel fields map to the functions used here. The DSP readings and settings always correspond to the following fields:



Mapping of DSP function calls to S1.EXE DSP panel

DSP Operational States -- the dsp_opstate function

While powerful, the DSP can only do a finite number of tasks at once. For example, while doing FFT's, if the DSP had to service settings its performance would be degraded. Therefore, three different operational states exist, each optimized for a particular task.

The dsp_opstate function is used to set the operational states of the DSP. SETUP and READING are the most commonly used. A typical program for the DSP would use the SETUP state during setup and READING during the sweep.

The following is a description of these states.

DSP_OPSTATE SETUP.

This is the only state that allows multiple choice settings and should be the default state.

Also, some readings are altered for this mode. For example, with FFTGEN and FFTSLIDE, the readings return real time peak readings to allow you to determine if there is a valid input signal present.

The SETUP operational state should be used to make all initial settings to the DSP program.

The DSP program should be left in the SETUP state unless it needs to be in another state such as when sweeping (DSP_OPSTATE READING) or examining AES/EBU status bits (DSP_OPSTATE AES).

This state does not allow reading or setting the AES/EBU/SPDIF interface status bits.

DSP_OPSTATE READING

The READING state is used during sweeps. It produces optimum speed when processing and transforming data. In order to do this multiple choice settings are ignored.

Floating point settings (dsp_set0 through dsp_set3), which are necessary during a sweep, are still recognized.

This state does not allow reading or setting the AES/EBU/SPDIF interface status bits.

DSP_OPSTATE AES.

This state is only used when setting and reading the AES/EBU status bytes. These status bytes cannot be accessed in any other DSP operational state.

Note that these bytes are always transmitted and received whenever the AES/EBU or SPDIF interfaces are in use. However, they may only be read and changed when in this state.

For additional status information, see dsp_set_aes_channel_status and dsp_get_aes_channel_status.

Use of units for DSP Readings

The DSP readings are program dependent and so can have a variety of units used for the reading values. Also, each reading may have several units allowed.

> **Before a reading is made, the appropriate dsp_readX_unit function must be called to inform the DSP which unit to use. See the dsp_readX_unit description for more information.**

The following units are presently available and defined in APDSP.H:

DSP_UNIT_AMPL	(always in Volts)
DSP_UNIT_FFS	(0 to 1)
DSP_UNIT_PCT	
DSP_UNIT_DECIMAL	
DSP_UNIT_FREQ	(always in Hertz)
DSP_UNIT_SECONDS	
DSP_UNIT_DEGREES	

AMPL (always Volts):

This is standard unit for analog inputs. When this unit is used, all signal path ranging is calculated into the reading. For digital signals, full scale is equivalent to 1 volt.

FFS:

This is Fraction of Full Scale and ranges from 0 to 1. It always refers to the signal at the ADCs or digital inputs. Any ranging in an analog signal path is ignored making this unit most useful for digital inputs.

PCT (Percent):

This unit is used to express distortion and crosstalk values. In some programs, this unit may only be supported when the DSP input signal path is set to the LVF READING meter. Refer to the program specific sections of this and the DSP User's Manual for more information.

Decimal:

This is actually applied to unitless values such as the actual binary numbers received on a digital port. At present, it is used only for the BITTEST program

FREQ (always Hertz):

This is the unit used for all frequency readings.

Seconds:

This is the unit used for all time values.

Degrees:

This is the unit used for all phase values.

See also the appropriate .DSP program specific section in this manual and the DSP User's Manual for information on how each program uses these readings and units.

DSP Programs and Settling

Some DSP programs, for example FFTGEN, FFTSLIDE, and FASTEST return readings based on the processing of large data records previously acquired, therefore, these programs do not need the concept of settled readings.

FFTGEN, FFTSLIDE, and FASTEST ignore settling parameters.

Other DSP programs, for example HARMONIC and GENANLR, use settling in the same manner as the analog analyzer.

Each DSP programs special use of settling is noted in the individual DSP program sections in this manual.

FFTGEN.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for fftgen.dsp version 2.10-07:

Program Notes:

1. All rates are supported for Acquisition.
2. Digital sinewave generation not allowed at 176.4kHz and 192kHz.
3. Readings must use FFS unit when dsp_opstate is SETUP
4. Readings ignore settling when dsp_opstate is READING

DUS structure elements:

dsp_do_amp1	do_dsp_read0
dsp_do_amp2	do_dsp_read1
dsp_amp1	dsp_read0
dsp_amp2	dsp_read1

Readings, readings unit and readys (and allowable units):

dsp_fftgen_read_amp1()	dsp_read0()	(FFS, Pct, Volts)
dsp_fftgen_read_amp2()	dsp_read1()	(FFS, Pct, Volts)
dsp_fftgen_rdy_amp1()	dsp_rdy0()	
dsp_fftgen_rdy_amp2()	dsp_rdy1()	
dsp_fftgen_read_amp1_unit(a)	dsp_read0_unit(a)	(FFS, Pct, Volts)
dsp_fftgen_read_amp2_unit(a)	dsp_read1_unit(a)	(FFS, Pct, Volts)

Floating point settings:

dsp_fftgen_set_dgenfreq(a)	dsp_set0(a)	(Hz)
dsp_fftgen_set_dgenampl(a)	dsp_set1(a)	(FFS)
dsp_fftgen_set_freq(a)	dsp_set2(a)	(Hz)
dsp_fftgen_set_time(a)	dsp_set3(a)	(seconds)

Names for the DSP_SWEEP_SETUP first argument

DSP_FFTGEN_SWEEP_DGENFREQ
DSP_FFTGEN_SWEEP_DGENAMPL
DSP_FFTGEN_SWEEP_FREQ
DSP_FFTGEN_SWEEP_TIME

Integer settings and defined arguments:

dsp_fftgen_set_len(a)
DSP_FFTGEN_SET_LEN_MAX
DSP_FFTGEN_SET_LEN_4096
DSP_FFTGEN_SET_LEN_2048
DSP_FFTGEN_SET_LEN_1024
DSP_FFTGEN_SET_LEN_512
DSP_FFTGEN_SET_LEN_256

dsp_fftgen_set_win(a)
DSP_FFTGEN_SET_WIN_BH4
DSP_FFTGEN_SET_WIN_HANN
DSP_FFTGEN_SET_WIN_FLAT
DSP_FFTGEN_SET_WIN_NONE

dsp_fftgen_set_wave(a)
DSP_FFTGEN_SET_WAVE_INTERP
DSP_FFTGEN_SET_WAVE_NORMAL
DSP_FFTGEN_SET_WAVE_PEAK
DSP_FFTGEN_SET_WAVE_MAX

dsp_fftgen_set_waveavgsub(a)
DSP_FFTGEN_SET_WAVEAVGSUB_AVG
DSP_FFTGEN_SET_WAVEAVGSUB_OFF

dsp_fftgen_set_avg(a)
DSP_FFTGEN_SET_AVG_1
DSP_FFTGEN_SET_AVG_4
DSP_FFTGEN_SET_AVG_16
DSP_FFTGEN_SET_AVG_64
DSP_FFTGEN_SET_AVG_256
DSP_FFTGEN_SET_AVG_1024

dsp_fftgen_set_trig(a)
DSP_FFTGEN_SET_TRIG_OFF
DSP_FFTGEN_SET_TRIG_1
DSP_FFTGEN_SET_TRIG_2
DSP_FFTGEN_SET_TRIG_AUTO
DSP_FFTGEN_SET_TRIG_DGEN

dsp_set4(a)

dsp_set5(a)

dsp_set6(a)

dsp_set7(a)

dsp_set8(a)

dsp_set9(a)

FASTEST.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for fastest.dsp version 2.10-07:

Program Notes:

1. Rates supported are: 32k, 44.1k, and 48k only.
2. Readings must use FFS unit when dsp.opstate is SETUP
3. Readings ignore settling when dsp.opstate is READING

DUS structure elements:

dsp_do_amp1	do_dsp_read0
dsp_do_amp2	do_dsp_read1
dsp_do_phase1	do_dsp_read2
dsp_do_phase2	do_dsp_read3
dsp_amp1	dsp_read0
dsp_amp2	dsp_read1
dsp_phase1	dsp_read2
dsp_phase2	dsp_read3

Readings, readings unit and readys (and allowable units):

dsp_fastest_read_amp1()	dsp_read0()	(FFS, Pct, Volts)
dsp_fastest_read_amp2()	dsp_read1()	(FFS, Pct, Volts)
dsp_fastest_read_phase1()	dsp_read2()	(Deg)
dsp_fastest_read_phase2()	dsp_read3()	(Deg)
dsp_fastest_rdy_amp1()	dsp_rdy0()	
dsp_fastest_rdy_amp2()	dsp_rdy1()	
dsp_fastest_rdy_phase1()	dsp_rdy2()	
dsp_fastest_rdy_phase2()	dsp_rdy3()	
dsp_fastest_read_amp1_unit(a)	dsp_read0_unit(a)	(FFS, Pct, Volts)
dsp_fastest_read_amp2_unit(a)	dsp_read1_unit(a)	(FFS, Pct, Volts)
dsp_fastest_read_phase1_unit(a)	dsp_read2_unit(a)	(Deg)
dsp_fastest_read_phase2_unit(a)	dsp_read3_unit(a)	(Deg)

Floating point settings:

dsp_fastest_set_freqres(a)	dsp_set0(a)	(Hz)
dsp_fastest_set_dgenampl(a)	dsp_set1(a)	(FFS)
dsp_fastest_set_freq(a)	dsp_set2(a)	(Hz)
dsp_fastest_set_time(a)	dsp_set3(a)	(seconds)

Names for the DSP_SWEEP_SETUP first argument

DSP_FASTEST_SWEEP_FREQRES
DSP_FASTEST_SWEEP_DGENAMPL
DSP_FASTEST_SWEEP_FREQ
DSP_FASTEST_SWEEP_TIME

Integer settings and defined arguments:

dsp_fastest_set_len(a) dsp_set4(a)
DSP_FASTEST_SET_LEN_MAX
DSP_FASTEST_SET_LEN_4096
DSP_FASTEST_SET_LEN_2048
DSP_FASTEST_SET_LEN_1024
DSP_FASTEST_SET_LEN_512
DSP_FASTEST_SET_LEN_256

dsp_fastest_set_win(a) dsp_set5(a)
DSP_FASTEST_SET_WIN_NONE
DSP_FASTEST_SET_WIN_HANN

dsp_fastest_set_process(a) dsp_set6(a)
DSP_FASTEST_SET_PROCESS_NORMAL
DSP_FASTEST_SET_PROCESS_RESPWF
DSP_FASTEST_SET_PROCESS_DISTORT
DSP_FASTEST_SET_PROCESS_NOISE

dsp_fastest_set_phasediff(a) dsp_set7(a)
DSP_FASTEST_SET_PHASEDIFF_OFF
DSP_FASTEST_SET_PHASEDIFF_DIFF

dsp_fastest_set_trig(a) dsp_set9(a)
DSP_FASTEST_SET_TRIG_OFF
DSP_FASTEST_SET_TRIG_1
DSP_FASTEST_SET_TRIG_2
DSP_FASTEST_SET_TRIG_AUTO
DSP_FASTEST_SET_TRIG_DGEN

FFTSLIDE.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for fftslide.dsp version 2.10-07:

Program Notes:

1. All rates are supported.
2. Readings must use FFS unit when dsp_opstate is SETUP.
3. Readings ignore settling when dsp_opstate is READING

DUS structure elements:

dsp_do_amp1	do_dsp_read0
dsp_do_amp2	do_dsp_read1
dsp_amp1	dsp_read0
dsp_amp2	dsp_read1

Readings, readings unit and readys (and allowable units):

dsp_fftslide_read_amp1()	dsp_read0()	(FFS, Pct, Volts)
dsp_fftslide_read_amp2()	dsp_read1()	(FFS, Pct, Volts)
dsp_fftslide_rdy_amp1()	dsp_rdy0()	
dsp_fftslide_rdy_amp2()	dsp_rdy1()	
dsp_fftslide_read_amp1_unit(a)	dsp_read0_unit(a)	(FFS, Pct, Volts)
dsp_fftslide_read_amp2_unit(a)	dsp_read1_unit(a)	(FFS, Pct, Volts)

Floating point settings (and unit used):

dsp_fftslide_set_starttime(a)	dsp_set0(a)	(seconds)
dsp_fftslide_set_pretrigger(a)	dsp_set1(a)	(seconds)
dsp_fftslide_set_freq(a)	dsp_set2(a)	(Hz)
dsp_fftslide_set_time(a)	dsp_set3(a)	(seconds)

Names for the DSP_SWEEP_SETUP first argument

DSP_FFTSLIDE_SWEEP_STARTTIME
DSP_FFTSLIDE_SWEEP_PRETRIGGER
DSP_FFTSLIDE_SWEEP_FREQ
DSP_FFTSLIDE_SWEEP_TIME

Integer settings and defined arguments:

dsp_fftslide_set_len(a) dsp_set4(a)
DSP_FFTSLIDE_SET_LEN_MAX
DSP_FFTSLIDE_SET_LEN_4096
DSP_FFTSLIDE_SET_LEN_2048
DSP_FFTSLIDE_SET_LEN_1024
DSP_FFTSLIDE_SET_LEN_512
DSP_FFTSLIDE_SET_LEN_256

dsp_fftslide_set_win(a) dsp_set5(a)
DSP_FFTSLIDE_SET_WIN_BH4
DSP_FFTSLIDE_SET_WIN_HANN
DSP_FFTSLIDE_SET_WIN_FLAT
DSP_FFTSLIDE_SET_WIN_NONE

dsp_fftslide_set_wave(a) dsp_set6(a)
DSP_FFTSLIDE_SET_WAVE_INTERP
DSP_FFTSLIDE_SET_WAVE_NORMAL
DSP_FFTSLIDE_SET_WAVE_PEAK
DSP_FFTSLIDE_SET_WAVE_MAX

dsp_fftslide_set_waveavgsub(a) dsp_set7(a)
DSP_FFTSLIDE_SET_WAVEAVGSUB_AVG
DSP_FFTSLIDE_SET_WAVEAVGSUB_OFF

dsp_fftslide_set_trig(a) dsp_set8(a)
DSP_FFTSLIDE_SET_TRIG_ANLR_A
DSP_FFTSLIDE_SET_TRIG_ANLR_B
DSP_FFTSLIDE_SET_TRIG_DSP_A
DSP_FFTSLIDE_SET_TRIG_DSP_B
DSP_FFTSLIDE_SET_TRIG_RDNG
DSP_FFTSLIDE_SET_TRIG_GEN_SYNC
DSP_FFTSLIDE_SET_TRIG_LINE
DSP_FFTSLIDE_SET_TRIG_1
DSP_FFTSLIDE_SET_TRIG_2
DSP_FFTSLIDE_SET_TRIG_AUTO
DSP_FFTSLIDE_SET_TRIG_EXTERN

dsp_fftslide_set_trigpol(a) dsp_set9(a)
DSP_FFTSLIDE_SET_TRIGPOL_OFF
DSP_FFTSLIDE_SET_TRIGPOL_PLUS
DSP_FFTSLIDE_SET_TRIGPOL_INV

HARMONIC.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for harmonic.dsp version 2.10-07:

Program Notes:

1. Rates supported are: 48k and 192k only.
2. Pct may only be used when dsp_set_chanX_src() is set to LVFREAD.

DUS structure elements:

dsp_do_filtlvl1	do_dsp_read0
dsp_do_filtfreq	do_dsp_read1
dsp_filtlvl1	dsp_read0
dsp_filtfreq	dsp_read1

Readings, readings unit and readys (and allowable units):

dsp_harmonic_read_filtlvl1()	dsp_read0()	(Pct, Volts)
dsp_harmonic_read_filtfreq()	dsp_read1()	(Freq only)
dsp_harmonic_rdy_filtlvl1()	dsp_rdy0()	
dsp_harmonic_rdy_filtfreq()	dsp_rdy1()	
dsp_harmonic_read_filtlvl1_unit(a)	dsp_read0_unit(a)	(Pct, Volts)
dsp_harmonic_read_filtfreq_unit(a)	dsp_read1_unit(a)	(Freq only)

Floating point settings (and unit used):

dsp_harmonic_set_filtfreq(a)	dsp_set0(a)	(Hz)
dsp_harmonic_set_freqmult(a)	dsp_set1(a)	(*)
dsp_harmonic_set_freqoffset(a)	dsp_set2(a)	(Hz)

Names for the DSP_SWEEP_SETUP first argument

DSP_HARMONIC_SWEEP_FILTFREQ
DSP_HARMONIC_SWEEP_FREQMULT
DSP_HARMONIC_SWEEP_FREQOFFSET

Integer settings and defined arguments:

dsp_harmonic_set_tune(a) dsp_set4(a)
DSP_HARMONIC_SET_TUNE_DIRECT
DSP_HARMONIC_SET_TUNE_HARMONIC
DSP_HARMONIC_SET_TUNE_OFFSET

dsp_harmonic_set_tunesrc(a) dsp_set5(a)
DSP_HARMONIC_SET_TUNESRC_SET
DSP_HARMONIC_SET_TUNESRC_ANLR
DSP_HARMONIC_SET_TUNESRC_GEN

dsp_harmonic_set_filt(a) dsp_set6(a)
DSP_HARMONIC_SET_FILT_HP
DSP_HARMONIC_SET_FILT_BPWIDE
DSP_HARMONIC_SET_FILT_BPNARROW

dsp_harmonic_set_detspeed(a) dsp_set8(a)
DSP_HARMONIC_SET_DETSPEED_AUTO
DSP_HARMONIC_SET_DETSPEED_4
DSP_HARMONIC_SET_DETSPEED_8
DSP_HARMONIC_SET_DETSPEED_16
DSP_HARMONIC_SET_DETSPEED_32
DSP_HARMONIC_SET_DETSPEED_64

dsp_harmonic_set_detector(a) dsp_set9(a)
DSP_HARMONIC_SET_DETECTOR_RMS

GENANLR.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for genanlr.dsp version 2.10-07:

Program Notes:

1. Rates supported are: 32k, 44.1k and 48k only.

DUS structure elements:

dsp_do_lvl1	do_dsp_read0
dsp_do_filtlvl2	do_dsp_read1
dsp_do_freq	do_dsp_read2
dsp_lvl1	dsp_read0
dsp_filtlvl2	dsp_read1
dsp_freq	dsp_read2

Readings, readings unit and readys (and allowable units):

dsp_genanlr_read_lvl1()	dsp_read0()	(FFS only)
dsp_genanlr_read_filtlvl2()	dsp_read1()	(FFS, Pct)
dsp_genanlr_read_freq()	dsp_read2()	(Freq only)
dsp_genanlr_rdy_lvl1()	dsp_rdy0()	
dsp_genanlr_rdy_filtlvl2()	dsp_rdy1()	
dsp_genanlr_rdy_freq()	dsp_rdy2()	
dsp_genanlr_read_lvl1_unit(a)	dsp_read0_unit(a)	(FFS only)
dsp_genanlr_read_filtlvl2_unit(a)	dsp_read1_unit(a)	(FFS, Pct)
dsp_genanlr_read_freq_unit(a)	dsp_read2_unit(a)	(Freq only)

Floating point settings (and unit used):

dsp_genanlr_set_dgenfreq(a)	dsp_set0(a)	(Hz)
dsp_genanlr_set_dgenampl(a)	dsp_set1(a)	(FFS)
dsp_genanlr_set_filtfreq(a)	dsp_set2(a)	(Hz)

Names for the DSP_SWEEP_SETUP first argument

DSP_GENANLR_SWEEP_DGENFREQ
DSP_GENANLR_SWEEP_DGENAMPL
DSP_GENANLR_SWEEP_FILTFREQ

Integer settings and defined arguments:

dsp_genanlr_set_filt(a)
DSP_GENANLR_SET_FILT_OFF
DSP_GENANLR_SET_FILT_BPNARROW
DSP_GENANLR_SET_FILT_BR
DSP_GENANLR_SET_FILT_AWHT
DSP_GENANLR_SET_FILT_CCIR

dsp_genanlr_set_hipass(a)
DSP_GENANLR_SET_HIPASS_10
DSP_GENANLR_SET_HIPASS_224
DSP_GENANLR_SET_HIPASS_100
DSP_GENANLR_SET_HIPASS_400

dsp_genanlr_set_tunesrc(a)
DSP_GENANLR_SET_TUNESRC_SET
DSP_GENANLR_SET_TUNESRC_DGEN
DSP_GENANLR_SET_TUNESRC_FREQ
DSP_GENANLR_SET_TUNESRC_ANLR
DSP_GENANLR_SET_TUNESRC_GEN

dsp_genanlr_set_bpharmonic(a)
DSP_GENANLR_SET_BPHARMONIC_1
DSP_GENANLR_SET_BPHARMONIC_2
DSP_GENANLR_SET_BPHARMONIC_3
DSP_GENANLR_SET_BPHARMONIC_4
DSP_GENANLR_SET_BPHARMONIC_5

dsp_genanlr_set_detspeed(a)
DSP_GENANLR_SET_DETSPEED_AUTO
DSP_GENANLR_SET_DETSPEED_4
DSP_GENANLR_SET_DETSPEED_8
DSP_GENANLR_SET_DETSPEED_16
DSP_GENANLR_SET_DETSPEED_32
DSP_GENANLR_SET_DETSPEED_64
DSP_GENANLR_SET_DETSPEED_QPK

dsp_set4(a)

dsp_set5(a)

dsp_set6(a)

dsp_set7(a)

dsp_set8(a)

BITTEST.DSP Program Dependent information

The following are defined in APDSP.H and allow intelligible names for the program dependent functions for bittest.dsp version 2.10-07:

Program Notes:

1. Rates supported are: 32k, 44.1k and 48k only

DUS structure elements:

dsp_do_input1	do_dsp_read0
dsp_do_input2	do_dsp_read1
dsp_do_errors1	do_dsp_read2
dsp_do_errors2	do_dsp_read3

dsp_input1	dsp_read0
dsp_input2	dsp_read1
dsp_errors1	dsp_read2
dsp_errors2	dsp_read3

Readings, readings unit and readys (and allowable units):

dsp_bittest_read_input1()	dsp_read0()	(Decimal only)
dsp_bittest_read_input2()	dsp_read1()	(Decimal only)
dsp_bittest_read_errors1()	dsp_read2()	(Decimal only)
dsp_bittest_read_errors2()	dsp_read3()	(Decimal only)

dsp_bittest_rdy_input1()	dsp_rdy0()
dsp_bittest_rdy_input2()	dsp_rdy1()
dsp_bittest_rdy_errors1()	dsp_rdy2()
dsp_bittest_rdy_errors2()	dsp_rdy3()

dsp_bittest_read_input1_unit(a)	dsp_read0_unit(a)	(Decimal only)
dsp_bittest_read_input2_unit(a)	dsp_read1_unit(a)	(Decimal only)
dsp_bittest_read_errors1_unit(a)	dsp_read2_unit(a)	(Decimal only)
dsp_bittest_read_errors2_unit(a)	dsp_read3_unit(a)	(Decimal only)

Floating point settings (and unit used):

dsp_bittest_set_dgenfreq(a)	dsp_set0(a)	(Hz)
dsp_bittest_set_dgenampl(a)	dsp_set1(a)	(%FS)
dsp_bittest_set_dgenvalue(a)	dsp_set2(a)	(Decimal)

Names for the DSP_SWEEP_SETUP first argument

DSP_BITTEST_SWEEP_DGENFREQ
DSP_BITTEST_SWEEP_DGENAMPL
DSP_BITTEST_SWEEP_DGENVALUE

Integer settings and defined arguments:

dsp_bittest_set_wave(a) dsp_set4(a)
DSP_BITTEST_SET_WAVE_CONSTANT
DSP_BITTEST_SET_WAVE_RANDOM
DSP_BITTEST_SET_WAVE_WALKING1
DSP_BITTEST_SET_WAVE_WALKING0
DSP_BITTEST_SET_WAVE_SINE

dsp_bittest_set_valid(a) dsp_set5(a)
DSP_BITTEST_SET_VALID_VALID
DSP_BITTEST_SET_VALID_INVALID

dsp_bittest_set_inpdisp(a) dsp_set6(a)
DSP_BITTEST_SET_INPDISP_INTERP
DSP_BITTEST_SET_INPDISP_NORMAL
DSP_BITTEST_SET_INPDISP_PEAK
DSP_BITTEST_SET_INPDISP_MAX

dsp_bittest_set_readrate(a) dsp_set7(a)
DSP_BITTEST_SET_READRATE_AUTO
DSP_BITTEST_SET_READRATE_SLOW
DSP_BITTEST_SET_READRATE_FAST

dsp_bittest_set_errdisp(a) dsp_set8(a)
DSP_BITTEST_SET_ERRDISP_NORMAL
DSP_BITTEST_SET_ERRDISP_MAX
DSP_BITTEST_SET_ERRDISP_TOTAL

NAME

dsp_init -- initialize DSP module

SYNOPSIS

new_modnum = dsp_init(modnum);

int modnum;	module number
int new_modnum;	module number actually set

DESCRIPTION

This function initializes the DSP to a known state. It must be used after power-up.

Note that dsp_init is called for you by ap_setup so that calling this function directly should only be necessary if you are going to write your own version of ap_setup().

The DSP generator data structure is set to default conditions, then the DSP hardware is set to match using the dsp_restore function.

This function also sets the module number of the DSP module. The module number must be between 0 and 15, else the default of 3 is used.

The default module number of 3 should always be used.

Default initializations are valid only for the settings that are DSP program independent and are:

1. Set dsp_opstate to SETUP
2. Set rate to 48k
3. Set input_type to ADC
4. Set chan1_src to LVFA
5. Set chan2_src to LVFB
6. Set output_type to DAC
7. Set output_chan to AB
8. Set output_size to 24
9. Set serial to AESEBU
10. Set dither to TRIANGULAR
11. Set AES channel status bytes to "04 00 00 ..."

RETURNS

The module number used is returned.

SEE ALSO

dsp_restore, ap_setup

NAME

dsp_reset -- Reset the DSP.

SYNOPSIS

void dsp_reset(void);

DESCRIPTION

This function stops any program running in the DSP. The reset condition turns outputs off and will not return readings.

In order to restart the DSP, the dsp_restore command must be used with a pointer to an appropriate .DSP file.

RETURNS

None

SEE ALSO

dsp_init, dsp_restore

NAME

dsp_exist -- determine if the DSP hardware is available.

SYNOPSIS

exist = dsp_exist(void);

int exist; 1 if DSP exists, 0 if not

DESCRIPTION

This function determines if the DSP hardware is connected and powered on.

The DSP status is updated each time dsp_restore called. This function returns that status. Note that ap_setup and dsp_init also call dsp_restore and so update the status.

It is good practice to check that SYSTEM ONE is properly connected and powered on before beginning a test procedure.

If the DSP is found not to exist (but should), prompt the operator to remedy the situation, then call dsp_restore before calling dsp_exist again.

RETURNS

Returns a 1 if the DSP hardware exists, 0 otherwise.

NAME

dsp_restore -- restore DSP hardware and download DSP program

SYNOPSIS

success = dsp_restore(fptr);

FILE *fptr;	pointer to open binary .DSP file
int success;	1 if successful, 0 if not

DESCRIPTION

This function restores the DSP hardware to the present state of the software and downloads a DSP program.

The DSP needs a downloaded program to function, this function is different than the other restore functions in that it requires a file pointer to an open (in binary mode) .DSP file. The file will be rewound by dsp_restore before it is used.

The DSP will not be functional until dsp_restore is called.

The following statement correctly opens a .DSP file:

```
FILE *fptr;
fptr = fopen( "fftgen.dsp", "rb");
```

This function should be used if the hardware loses power or becomes disconnected from the computer.

This function is also used by the dsp_init function but dsp_init does not download a .DSP program.

RETURNS

1 if restore was successful, 0 if not

SEE ALSO

dsp_init

NAME

dsp_load_waveform -- load waveform into DSP

SYNOPSIS

success = dsp_load_waveform(fptr, bufnum);

int success;	1 if load was successful, 0 if not
FILE *fptr;	pointer to open rewound binary .WAV file
int bufnum;	destination buffer for waveform

DESCRIPTION

This function transfers a waveform to the appropriate DSP buffer from the PC.

The FILE pointed to by fptr must be opened in binary mode (using "rb").

The following buffer numbers are available and defined in APDSP.H:

```
WAVEBUF_CH1
WAVEBUF_CH2
WAVEBUF_CH1_T
WAVEBUF_CH2_T
WAVEBUF_CH1_G
WAVEBUF_CH2_G
```

The application of each buffer will be DSP program dependent but in general selections CH1 and CH2 are the acquisition buffers, CH1_T and CH2_T are the transform buffers and CH1_G and CH2_G are used for the digital generator waveform and for the FFT window functions.

RETURNS

1 if load was successful, 0 if not

SEE ALSO

dsp_store_waveform

NAME

dsp_store_waveform -- store waveform from DSP

SYNOPSIS

success = dsp_store_waveform(fptr, bufnum, cmtptr);

FILE *fptr;	pointer to open binary .WAV file
int bufnum;	source buffer for waveform
char *cmtptr;	pointer to a comment string (up to 128 bytes)
int success;	1 if store was successful, 0 if not

DESCRIPTION

This function transfers a waveform from the appropriate DSP buffer to the PC.

The FILE pointed to by fptr must be opened in binary mode (using "wb").

The comment string must be less than 128 bytes and NULL terminated. This string will be stored in the waveform file to aid in documentation of the waveform.

The following buffer numbers are available and defined in APDSP.H:

```
WAVEBUF_CH1
WAVEBUF_CH2
WAVEBUF_CH1_T
WAVEBUF_CH2_T
WAVEBUF_CH1_G
WAVEBUF_CH2_G
```

The application of each buffer will be DSP program dependent but in general selections CH1 and CH2 are the acquisition buffers, CH1_T and CH2_T are the transform buffers and CH1_G and CH2_G are used for the digital generator waveform and for the FFT window functions.

RETURNS

1 if store was successful, 0 if not

SEE ALSO

dsp_load_waveform

NAME

dsp_read0 -- program specific reading 0
dsp_read1 -- program specific reading 1
dsp_read2 -- program specific reading 2
dsp_read3 -- program specific reading 3

SYNOPSIS

reading = dsp_read0(void);
reading = dsp_read1(void);
reading = dsp_read2(void);
reading = dsp_read3(void);

double reading reading in various units (see below)

DESCRIPTION

These readings are DSP program dependent.

The unit used for each reading is also program dependent. Most programs have a choice of units for each reading

Before a reading is made, the appropriate dsp_readX_unit function must be called to inform the DSP which unit to use. See the dsp_readX_unit description for more information.

See also the appropriate .DSP program specific section in this manual and the DSP User's Manual for information on how each program uses these readings.

RETURNS

This function returns the most recent reading. Note that any given real-time reading will be returned only once.

The number -1E34 is returned if there is a hardware error encountered.

SEE ALSO

DSP Program Dependent information section.

NAME

dsp_read0_unit -- set unit to be used for dsp_read0
dsp_read1_unit -- set unit to be used for dsp_read1
dsp_read2_unit -- set unit to be used for dsp_read2
dsp_read3_unit -- set unit to be used for dsp_read3

SYNOPSIS

```
void dsp_read0_unit( unit);
void dsp_read1_unit( unit);
void dsp_read2_unit( unit);
void dsp_read3_unit( unit);

int unit;                    (see below)
```

DESCRIPTION

The DSP readings are program dependent and so can have a variety of units used for the reading values. Also, each reading may have several units allowed.

These functions are used to set the unit for the readings.

The following units are presently available and defined in APDSP.H:

DSP_UNIT_AMPL	(always in Volts)
DSP_UNIT_FFS	(0 to 1)
DSP_UNIT_PCT	
DSP_UNIT_DECIMAL	
DSP_UNIT_FREQ	(always in Hertz)
DSP_UNIT_SECONDS	
DSP_UNIT_DEGREES	

AMPL (always Volts):

This is standard unit for analog inputs. When this unit is used, all signal path ranging is calculated into the reading. For digital signals, full scale is equivalent to 1 volt.

FFS:

This is Fraction of Full Scale and ranges from 0 to 1. It always refers to the signal at the ADCs or digital inputs. Any ranging in an analog signal path is ignored making this unit most useful for digital inputs.

PCT (Percent):

This unit is used to express distortion and crosstalk values. In some programs, this unit may only be supported when the DSP input signal path is set to the LVF READING meter. Refer to the program specific sections of this and the DSP User's Manual for more information.

Decimal:

This is actually applied to unitless values such as the actual binary numbers received on a digital port. At present, it is used only for the BITTEST program

FREQ (always Hertz):

This is the unit used for all frequency readings.

Seconds:

This is the unit used for all time values.

Degrees:

This is the unit used for all phase values.

RETURNS

None

NAME

dsp_anyready -- determine if any DSP reading is ready.
dsp_rdy0 -- determine if DSP0 reading is ready.
dsp_rdy1 -- determine if DSP1 reading is ready.
dsp_rdy2 -- determine if DSP2 reading is ready.
dsp_rdy3 -- determine if DSP3 reading is ready.

SYNOPSIS

```
rdy = dsp_anyready( void);
rdy = dsp_rdy0( void);
rdy = dsp_rdy1( void);
rdy = dsp_rdy2( void);
rdy = dsp_rdy3( void);

int rdy;                    1 if ready, 0 if not
```

DESCRIPTION

These functions determine if the appropriate real-time reading is ready.

Because readings functions do not return until a reading is ready, these functions may be used to avoid waiting for a reading. These functions do NOT clear their respective ready statuses and so may be called any number of times. Only a call to the reading function will clear a ready status.

If the particular reading is found to be ready, then a subsequent call to the corresponding readings function will be guaranteed to return quickly.

> **For FFT type programs (fftslide.dsp and fftgen.dsp) the ready functions are only necessary when the dsp_opstate is SETUP and the programs are returning real-time readings**

RETURNS

Returns a 1 if a reading is ready. Returns a 0 otherwise.

SEE ALSO

dsp_read0, dsp_read1, dsp_read2, dsp_read3

NAME

dsp_set0 -- program specific setting 0
dsp_set1 -- program specific setting 1
dsp_set2 -- program specific setting 2
dsp_set3 -- program specific setting 3

SYNOPSIS

```
new_setting = dsp_set0( setting);
new_setting = dsp_set1( setting);
new_setting = dsp_set2( setting);
new_setting = dsp_set3( setting);

double setting;          setting (dependent on .DSP program)
double new_setting;      actual setting made
```

DESCRIPTION

These settings are DSP program dependent.

See the appropriate .DSP program specific section in this manual and the DSP User's Manual for information on how each program uses these settings.

RETURNS

Setting actually made.

SEE ALSO

DSP Program Dependent information section.

NAME

dsp_set4 -- program specific setting 4
dsp_set5 -- program specific setting 5
dsp_set6 -- program specific setting 6
dsp_set7 -- program specific setting 7
dsp_set8 -- program specific setting 8
dsp_set9 -- program specific setting 9

SYNOPSIS

```
new_setting = dsp_set4( setting);
new_setting = dsp_set5( setting);
new_setting = dsp_set6( setting);
new_setting = dsp_set7( setting);
new_setting = dsp_set8( setting);
new_setting = dsp_set9( setting);

int setting;          actual setting made
int new_setting;      setting (dependent on .DSP program)
```

DESCRIPTION

These settings are DSP program dependent.

See the appropriate .DSP program specific section in this manual and the DSP User's Manual for information on how each program uses these settings.

RETURNS

Setting actually made.

SEE ALSO

DSP Program Dependent information section.

NAME

dsp_set_rate -- Set the DSP sampling rate.

SYNOPSIS

new_rate = dsp_set_rate(rate);

int new_rate;	integer representing rate (see below)
int rate;	actual rate set

DESCRIPTION

This function controls the DSP sampling rate for A/D conversion and digital interfacing. The following rate are available and defined in APDSP.H:

OUTPUT_RATE_1K
OUTPUT_RATE_8K
OUTPUT_RATE_32K
OUTPUT_RATE_48K
OUTPUT_RATE_192K
OUTPUT_RATE_441K
OUTPUT_RATE_1764K

When some DSP programs are loaded, not all of the rate choices will be available.

The existing DSP programs allow the following rates in Hertz:

FFTGEN	1k, 8k, 32k, 48k, 192k, 44.1k, 176.4k
FFTSLIDE	1k, 8k, 32k, 48k, 192k, 44.1k, 176.4k
FASTEST	32k, 44.1k, 48k
HARMONIC	48k, 192k
GENANLR	32k, 44.1k, 48k
BITTEST	32k, 44.1k, 48k

RETURNS

The rate actually set.

INITIAL STATE

The rate is set to 48k by dsp_init.

SEE ALSO

NAME

dsp_set_input_type -- set part of input signal path of DSP.

SYNOPSIS

new_type = dsp_set_input_type(type);

int type;	integer representing type (see below)
int new_type;	actual type set

DESCRIPTION

This function used in conjunction with dsp_set_chan1_src and dsp_set_chan2_src determines from what source the DSP receives its data.

> **Note that the choice of input type affects the valid arguments for dsp_set_chan1_src and dsp_set_chan2_src.**

The following source types are available and defined in APDSP.H:

```
INPUT_TYPE_ADC
INPUT_TYPE_SERIAL
INPUT_TYPE_PARALLEL
IPUT_TYPE_DGEN
```

ADC selects the A/D converters which convert the analog signals into digital signals for processing.

SERIAL selects the serial ports (See dsp_set_serial to choose the specific serial port).

DGEN selects the digital signal generator which is sometimes not available, depending on the DSP program that is loaded.

PARALLEL selects the digital parallel input.

RETURNS

The input type actually set.

INITIAL STATE

The input type is set to ADC by dsp_init.

SEE ALSO

dsp_set_chan1_src, dsp_set_chan2_src, dsp_set_serial

NAME

dsp_set_chan1_src -- set DSP channel 1 signal source
dsp_set_chan2_src -- set DSP channel 2 signal source

SYNOPSIS

new_src = dsp_set_chan1_src(source);
new_src = dsp_set_chan1_src(source);

int source;	integer representing signal source (see below)
int new_src;	actual source set

DESCRIPTION

This command controls the input signal routing to the DSP measurement section.

The possible sources depend on the setting of dsp_set_input_type.

When dsp_set_input_type is set to INPUT_TYPE_ADC the possible choices are (as defined in APDSP.H):

```
INPUT_CHAN_LVFA
INPUT_CHAN_LVFB
INPUT_CHAN_LVFREAD
INPUT_CHAN_GENMON
INPUT_CHAN_DSPA
INPUT_CHAN_DSPB
INPUT_CHAN_ADC_OFF
```

LVFA and LVFB refer to channel A and B analog analyzer inputs following input range switching and AC coupling but before any other processing. This is equivalent to what the LVF's LEVEL meter sees.

LVFREAD accesses the signal read by the analog analyzers' main measurement meter, following all analog signal processing.

GEN monitors a fixed amplitude version of the signal from the analog generator. This is the same signal as appears on the MONITOR OUTPUT connection on the GENERATOR AUX SIGNALS panel.

DSPA and DSPB are the DSP panel inputs.

INPUT_CHAN_ADC_OFF disables the acquisition of signal into that channel of the DSP. This is useful when acquiring one channel of information only without disturbing a previously acquired signal.

When dsp_set_input_type is set to INPUT_TYPE_SERIAL, PARALLEL or DGEN the possible choices are (as defined in APDSP.H):

```
INPUT_CHAN_DIGA
INPUT_CHAN_DIGB
INPUT_CHAN_DIGOFF
```

DIGA and DIGB refer to the digital channel. INPUT_CHAN_DIGOFF disables the acquisition of signal into that channel of the DSP as noted above.

RETURNS

The signal source actually set.

INITIAL STATE

Channel 1 source is set to LVFA and channel 2 source to LVFB by dsp_init.

SEE ALSO

dsp_set_input_type

NAME

dsp_set_output_type -- set DSP output type

SYNOPSIS

new_type = dsp_set_output_type(type);

int type;	output type as describe below
int new_type;	actual type set

DESCRIPTION

This function sets the DSP output signal path.

The possible choices are (as defined in APDSP.H):

```
OUTPUT_TYPE_DAC
OUTPUT_TYPE_SERIAL
OUTPUT_TYPE_PARALLEL
```

DAC sends the digital signal through the D/A converter and out the DSP BNC labeled OUTPUT D/A. If used in conjunction with the gen_set_mode(DSP) function, the output from the DAC can also be found on the analog generator outputs.

SERIAL sends the digital signal to the serial port. (See dsp_set_serial for more on serial digital signal routing.)

PARALLEL will make the digital signal available on the parallel output.

Some DSP programs have no signal output capability. Therefore this command will have no effect on such programs.

RETURNS

Output type actually set

INITIAL STATE

The output type is set to DAC by dsp_init.

SEE ALSO

dsp_set_output_chan, dsp_set_serial

NAME

dsp_set_output_chan -- set DSP output channels ON or OFF.

SYNOPSIS

new_channel = dsp_set_output_chan(channel);

int channel;	channels to turn on (see below)
int new_channel;	actual channels set

DESCRIPTION

This function turns the DSP output channels on or off.

The possible choices are (as defined in APDSP.H):

```
OUTPUT_CHAN_OFF     both channels OFF
OUTPUT_CHAN_A       channel A only on
OUTPUT_CHAN_B       channel B only on
OUTPUT_CHAN_AB      both A and B on
```

If dsp_output_type is SERIAL or PARALLEL and both A and B are on, the data is two channel multiplexed on the serial or parallel port.

Since there is only a single output for the DAC, when dsp_output_type is set to DAC, A, B, and AB all perform the same function.

Some DSP programs have no signal output capability. Therefore this command will have no effect on such programs.

RETURNS

Channel actually set.

INITIAL STATE

The output channel is set to AB by dsp_init.

SEE ALSO

dsp_set_output_type

NAME

dsp_set_output_size -- set output word (and dither) size

SYNOPSIS

new_size = dsp_set_output_size(size);

int size;	size of word and dither in bits
int new_size;	actual size set

DESCRIPTION

This function sets the resolution of the digital signals and in most cases, controls the amplitude of dither added to the digital generator output.

Dither is noise combined with the signal to improve linearity, reduce distortion at low amplitudes, and extend the linear operating range below the theoretical minimum for undithered PCM signals of any particular resolution. The additional noise is introduced before quantizing and serves to randomize the quantization distortion and produce an undistorted signal with a slightly higher noise floor.

The digital generator generates a 24-bit resolution signal at all times. When the digital device under test has less than 24-bit resolution, only the higher (most significant) bits from the generator will be used. If an undithered signal is desired, use the dsp_set_dither function. To set dither at any desired bit level, enter that bit number in this function. For example, proper dither for a 16-bit system is obtained by calling dsp_set_dither(16);.

> **When the D/A converter is selected as the output port, the SIZE command is over-ridden and internally set to 16 bits, regardless of the SIZE setting.**

The DSP program BITTEST makes further use of the size. Please refer to the BITTEST specific program section of the DSP User's Manual for more information.

RETURNS

Size actually set.

INITIAL STATE

The output size is set to 24 bits by dsp_init.

SEE ALSO

dsp_set_dither

NAME

dsp_set_serial -- set the serial data format

SYNOPSIS

new_format = dsp_set_serial(format);

int format;	integer representing format (see below)
int new_format;	actual format set

DESCRIPTION

This function sets the format of the serial digital signal.

The possible choices are (as defined in APDSP.H):

```
DSP_AESEBU
DSP_SPDIF1
DSP_SERIAL
```

AESEBU selects an AES/EBU format on the front panel AES/EBU or SPDIF connectors.

SPDIF selects the consumer format on the front panel AES/EBU or SPDIF connectors. This format difference changes the function of the status bits in the interface but does not select between the front panel connectors.

SERIAL changes the format of the entire digital word and selects the 15 pin general purpose serial output on the rear panel connector.

RETURNS

Format actually set

INITIAL STATE

The serial format is set to AESEBU by dsp_init.

SEE ALSO

NAME

dsp_set_dither -- set DSP dither type

SYNOPSIS

new_type = dsp_set_dither(type);

int type;	integer representing dither type (see below)
int new_type;	actual type set

DESCRIPTION

This function sets the shape of the dither probability function and the shape of the dither spectrum.

Dither is noise combined with the signal to improve linearity, reduce distortion at low amplitudes, and extend the linear operating range below the theoretical minimum for undithered PCM signals of any particular resolution. The additional noise is introduced before quantizing and serves to randomize the quantization distortion and produce an undistorted signal with a slightly higher noise floor.

Dither is described further in the DSP User's manual.

The possible choices are (as defined in APDSP.H):

```
DSP_TRIANGULAR      triangular probability
     flat spectrum
DSP_RECTANGULAR     rectangular probability  flat
spectrum
DSP_SHAPED          triangular probability
     +6 dB/octave
DSP_NO_DITHER       none                none
```

RETURNS

Dither type actually set.

INITIAL STATE

The dither type is set to TRIANGULAR by dsp_init.

SEE ALSO

NAME

dsp_acquire_xform -- Acquire and transform waveform data.

SYNOPSIS

void dsp_acquire_xform(void);

DESCRIPTION

This function causes the DSP to acquire and transform waveform data.

If triggering is enabled, the DSP will wait for a trigger event. After the trigger is received, the DSP will acquire input data into a buffer. When the buffer is full, the DSP will perform whatever other processing is required, such as an FFT.

This function returns immediately. To track the progress of the acquisition and transformation, use the dsp_read_status function.

> **Note: While the DSP is working on the waveform data it will ignore settings. Use the dsp_status command to determine when the DSP processor is idle and ready to accept additional commands. The length of time will depend on the amount of DSP processing required.**

This performs the same function as the S1.EXE function key F9 without graphing the data.

> **If the data is being acquired through the LVF, the ranges must be locked before data acquisition. This is accomplished be calling lvf_range_lock. (see lvf_range_lock and release)**

RETURNS

None

SEE ALSO

lvf_range_lock, lvf_range_release, dsp_opstate, dsp_read_status

NAME

dsp_xform -- re-transform waveform data.
dsp_reprocess -- re-process waveform data.

SYNOPSIS

void dsp_xform(void);
void dsp_reprocess(void);

DESCRIPTION

These functions control the processing of data for the FFT type DSP programs.

dsp_xform:

Will re-transform existing waveform data in the acquisition buffer without re-acquiring the data.

This function is valuable when any of the settings that control transformation are changed and it is desired to work without new acquisition data. The acquisition buffer may have been loaded by a previous acquisition or by transferring a waveform to the DSP memory (see the dsp_load_waveform command).

This performs the same function as the System One key F6 without graphing the data.

This function returns immediately. To track the progress of the transformation, use the dsp_read_status function.

> **Note: While the DSP is working on the waveform data it will ignore settings. Use the dsp_status command to determine when the DSP processor is idle and ready to accept additional commands. The length of time will depend on the amount of DSP processing required.**

dsp_reprocess:

This function allows re-reading of acquired or transformed data using the setting and reading commands without performing a new acquisition or transformation.

> **Note that one of the functions:**

dsp_acquire_xform
dsp_xform
dsp_reprocess

must be called at the beginning of a sweep for the FFT type DSP programs (fftslide.dsp, fftgen.dsp, others in the future).

If a sweep is to be made again without any new acquisition or transformation of data, this function call is faster than using another dsp_xform call.

This performs the same function as the System One key ALT-F6 without graphing the data.

RETURNS

None

SEE ALSO

dsp_opstate, dsp_read_status, dsp_load_waveform

NAME

dsp_sweep_setup -- set parameters for a DSP sweep

SYNOPSIS

void dsp_sweep_setup(setnum, low, high, steps, step_type);

int setnum;	number of setting to be swept
double low;	low endpoint of sweep
double high;	high endpoint of sweep
int steps;	number of steps in sweep
int step_type;	type of sweep (see below)

DESCRIPTION

This function should be use at the beginning of every sweep.

This function informs the DSP of parameters to be swept. This is necessary to allow the DSP to intelligently process analyzed data based upon knowledge of how the data will be formatted. This may be used by the DSP program to prevent display aliasing in time domain displays and to adjust the spectral display resolution to guarantee that no components will be skipped over when the number of steps in a sweep is smaller than the number of points transformed. The operation of this command varies with the DSP program in use. For further information consult the documentation for the DSP program being used.

SETNUM:

This is the number (0-3) of the setting that will be used as the horizontal value (independent variable) for the sweep

LOW and HIGH:

These are the beginning and endpoints of the sweep in the same unit (volts, freq, etc.) as will be used for the horizontal value (the setting specified by SETNUM).

STEPS:

The number of steps that will be in the sweep. (The number of steps is one less than the number of points in the sweep.) This is not necessarily the same as the number acquired or transformed, and typically will be much smaller.

STEP_TYPE:

The type of the sweep which can a be LINEAR sweep (equally spaced steps), a LOG sweep (logarithmically spaced steps), or a TABLE sweep (arbitrarily spaced steps).

The possible choices are (as defined in APDSP.H):

DSP_STEP_TYPE_LIN
DSP_STEP_TYPE_LOG
DSP_STEP_TYPE_TABLE

For the present DSP programs, any data processing as described above is turned off during TABLE sweeps.

RETURNS

None

NAME

dsp_read_status -- Get DSP acquisition and transform status.

SYNOPSIS

status = dsp_read_status(void);

int status; integer representing DSP status

DESCRIPTION

This command provides DSP status information and is useful when acquiring and transforming data.

The returned integer represents the current state of the DSP processor. This command is useful when tracking progress of an dsp_acquire_xform or dsp_xform command. The values for the status (as defined in APDSP.H) are:

```
DSP_WAITING          (waiting for a trigger)
DSP_ACQUIRING        (acquiring the data)
DSP_TRANSFORMING     (transforming the data)
DSP_IDLE             (finished)
```

RETURNS

The DSP status as defined above.

SEE ALSO

dsp_acquire_xform, dsp_xform

NAME

dsp_trigger -- trigger new DSP readings with delay

SYNOPSIS

void dsp_trigger(msec);

double msec; trigger delay time in milliseconds

DESCRIPTION

This function will wait x milliseconds before accepting any valid DSP readings. When using real-time readings this command will make any DSP reading command wait for a new reading to become valid from the DSP before returning its response. This is useful when programming a series of parameter values and measuring after each new step. The delay may be used to prevent readings while the device under test is reaching steady-state operation.

This command has no effect on operation with acquired or transformed data readings.

Delays from 0 to 15000 milliseconds (15 seconds) are available. Delay values outside this range will be adjusted to the closest possible delay (ie. negatives are changed to 0 and numbers greater then 15000 are changed to 15000).

RETURNS

None.

NAME

dsp_opstate -- set the DSP operational state.

SYNOPSIS

void dsp_opstate(state);

int state state as described below

DESCRIPTION

While powerful, the DSP can only do a finite number of tasks at once. For example, while doing FFT's, if the DSP had to service settings its performance would be degraded. Therefore, several operational states exist each optimized for a particular task.

The dsp_opstate function is used to set the operational states of the DSP.

The possible choices are (as defined in APDSP.H):

DSP_OPSTATE_SETUP
DSP_OPSTATE_READING
DSP_OPSTATE_AES

SETUP and READING are the most commonly used. A typical program for the DSP would use the SETUP state during setup and READING during the sweep.

The following is a description of these states.

DSP_OPSTATE_SETUP.

This is the only state that allows multiple choice settings and should be the default state. Also, some readings are altered for this mode. For example, with FFTGEN and FFTSLIDE, the readings return real time peak readings to allow you to determine if there is a valid input signal present.

The SETUP operational state should be used to make all initial settings to the DSP program.

The DSP program should be left in the SETUP state unless it needs to be in another state such as when sweeping (DSP_OPSTATE READING) or examining AES/EBU status bits (DSP_OPSTATE AES).

DSP_OPSTATE_READING.

The READING state is used during sweeps. It produces optimum speed when processing and transforming data. In order to do this multiple choice settings are ignored.

Floating point settings, which are necessary during a sweep, are still recognized.

Also, for DSP programs that have an auto reading rate setting, for example GENANLR and HARMONIC, the READING state selects an optimal reading rate. How and what rate is selected is DSP program dependent.

For additional information, see the specific rate setting command for each program.

DSP_OPSTATE_AES.

This state is only used when setting and reading the AES/EBU status bytes. These status bytes cannot be accessed in any other DSP operational state.

For additional status information, see dsp_set_aes_channel_status and dsp_get_aes_channel_status.

DSP programs allow several different modes of operation that were originally optimized for efficient use of the DSP in the main functions of S1.EXE. For instance, many of the modes below ignore settings 4-9 so that the DSP may perform other tasks more quickly. The modes are:

RETURNS

None

INITIAL STATE

The opstate is set to SETUP by dsp_init.

SEE ALSO

NAME

dsp_set_response -- set reading response combinations

SYNOPSIS

new_freq = dsp_set_response(freq);

double freq;	frequency in Hertz
double new_freq;	frequency actually set in Hertz

DESCRIPTION

Only useful with DSP programs returning real-time readings (such as GENANLR)

This function sets a pre-programmed combination of detector averaging speed and reading rate for DSP real-time readings. The combination is optimized for maximum speed possible within reasonable instrument accuracy for the given frequency of interest.

This function is not presently used for any DSP programs.

RETURNS

Frequency setting actually used.

NAME

dsp_get_infobits -- Return low level DSP information flags.

SYNOPSIS

infoword = dsp_get_infobits(void);

long infoword; bits encodes in long integer

DESCRIPTION

This function returns several bits of DSP status information encoded as a long integer. The following bits (as defined in APDSP.H) are:

```
DSP_INFOBIT_CRC_A        0x000001L (bit 0 LSB)
DSP_INFOBIT_CRC_B        0x000002L (bit 1)
DSP_INFOBIT_VALIDITY_A   0x000004L (bit 2)
DSP_INFOBIT_VALIDITY_B   0x000008L (bit 3)
DSP_INFOBIT_PARITY_A          0x000010L (bit 4)
DSP_INFOBIT_PARITY_B          0x000020L (bit 5)
DSP_INFOBIT_RCVSYNC      0x000040L (bit 6)
DSP_INFOBIT_XMITSYNC     0x000080L (bit 7)
DSP_INFOBIT_MEMOPT       0x400000L (bit 22)
DSP_INFOBIT_DIOOPT       0x800000L (bit 24 MSB)
```

CRC_A and B. These bits are high when the appropriate channel received AES/EBU interface CRCC is correct. dsp_opstate should be set to DSP_AES before reading these bits.

VALIDITY_A and B. These bits are high when the appropriate channel received AES/EBU interface data is valid. dsp_opstate should be set to DSP_AES before reading these bits.

PARITY_A and B. These bits are high when the appropriate channel received AES/EBU interface parity checks. dsp_opstate should be set to DSP_AES before reading these bits.

XMITSYNC This bit is high when the AES/EBU transmit PLL is locked

MEMOPT This bit is high when the system is an SYS200 with memory option installed

DIOOPT This bit is high when the system is an SYS300 with digital I/O capability and memory

RETURNS

Long integer as described above

NAME

dsp_set_aes_channel_status -- set the transmitted AES/EBU channel status bytes

SYNOPSIS

void dsp_set_aes_channel_status(ptr);

unsigned char (*ptr)[24]; pointer to a 24 byte array

DESCRIPTION

This function will set the AES/EBU or SPDIF serial status bytes from a 24 byte array.

DSP_OPSTATE needs to be set to OPSTATE_AES before this function is called.

These are the status bytes that will be transmitted at the AES/EBU and SPDIF/EIAJ coaxial and optical output connectors of Dual Domain units.

The AES/EBU digital audio transmission standard (AES3-1985, also ANSI S4.40-1985) contains 24 8-bit status bytes in addition to two channels of digitized audio signals.

The AES/EBU standard defines the use of many of the status bytes. Some of those status bytes are also defined in the SPDIF/EIAJ consumer digital audio transmission standards. Use of bytes zero through three is critical to proper interfacing of professional digital audio devices.

Byte Zero:

Byte Zero defines (when transmitted) and displays (when received) a number of important parameters including use and type of emphasis, selected sample rate, and consumer vs professional use. In order to successfully transmit digital data to a device such as a digital recorder, first use dsp_get_aes_channel_status to determine the codes coming from the machine while in playback mode. Duplicating those codes via dsp_set_aes_channel_status will then normally cause the machine to accept the digital signal from System One Dual Domain. Changes from the received code will be necessary if, for example, the default playback mode uses emphasis but the user wishes to test without emphasis. Some of the common two-character hexadecimal codes for byte zero are shown in the following table:

Hex Code	Emph.	Pro/Consumer	Misc
00	no	consumer	copy inhibit
04	no	consumer	copy OK
08	50/15	consumer	copy inhibit
0C	50/15	consumer	copy OK
01	no	professional	rate n/i
0D	50/15	professional	rate n/i
81	no	professional	48 kHz
8D	50/15	professional	48 kHz
41	no	professional	44.1 kHz
4D	50/15	professional	44.1 kHz
C1	no	professional	32 kHz
CD	50/15	professional	32 kHz

The "rate n/i" entry under Misc means that the sample rate is not indicated in the status bytes. In practice, the receiver uses the transmitted signal's clock rate to set sample rate, since the rates must match to allow reception of data. However, some equipment requires the received clock rate to match the rate encoded in the status bits. The 50/15 entry under emphasis means that 50/15 microsecond pre-emphasis (CD type) is used during recording and matching deemphasis is used during playback.

Sample Address Code

The Dual Domain interface also generates a "sample address code" and transmits it on bytes 14-17. This code serves a similar function to a recording index counter on analog tape recorders. The code is reset to zero whenever a DRESET command is issued or a new DSP program is loaded. These bytes are not defined in any of the consumer versions of the interface.

CRCC Code

Byte 23 is a CRCC (cyclic redundancy check character) code. This code is computed by the Dual Domain interface for each status block and transmitted on byte 23. A receiving device could use this byte to detect errors occurring during transmission or recording and reproduction of the status bytes. This byte is not defined in the consumer versions of the interface.

Other Bytes

Any of the 24 bytes except bytes 14-17, which carry the automatically-generated sample address code, may have any hex value entered and thus transmitted until changed. Refer to the AES/EBU or EIAJ standards for the definition of these other bytes.

RETURNS

None

INITIAL STATE

Byte Zero is set to 0x04 by dsp_init. All other bytes are set to 0x00

SEE ALSO

dsp_get_aes_channel_status

NAME

dsp_get_aes_channel_status -- get the received AES/EBU channel status bytes

SYNOPSIS

void dsp_get_aes_channel_status(ptr);

unsigned char (*ptr)[24]; pointer to a 24 byte array

DESCRIPTION

This function will acquire the AES/EBU or SPDIF serial status bytes, then return it into a 24 byte array.

DSP_OPSTATE needs to be set to OPSTATE_AES before this function is called.

These are the status bytes being received at the AES/EBU and SPDIF/EIAJ coaxial and optical input connectors of Dual Domain units.

See the complete description of the status bytes in the dsp_set_aes_channel_status function above.

RETURNS

24 byte array at location pointed to by the passed argument

SEE ALSO

dsp_set_aes_channel_status