Vaunix Technology Corporation

Lab Brick® LMS Series of Signal Generators

Linux API User Manual

Revision A

Table of Contents

1.0 OVERVIEW

The LabBrick LMS Series Signal Generator SDK for Linux supports developers who want to control LabBrick LMS Series Signal Generator from Linux programs. For maximum compatibility, the SDK includes source code for C functions to find, initialize, and control the synthesizers, along with header files and an example C program which demonstrates the use of the API. These functions are written to use the ‘libusb’ library which comes with most Linux distributions or is easily installed. Many distributions which use a kernel 2.4 or newer already have this library installed. A more recent library, version 1.0 is available but has a different API. The LabBrick functions rely on the older naming conventions since the libusb folks (http://www.libusb.org) also provide a compatibility wrapper to allow accessing the newer library with the older style calls.

2.0 SETTING UP FOR THE SDK

Before you can use the SDK or try the sample program, you need to make sure you have libusb installed. You can retrieve source from the developer's site at http://www.libusb.org, or use your distribution's package installer. Look for a package that contains "libusb-dev" in the package name. For Debian and Ubuntu, "libusb-dev" should work. For Redhat and Fedora, look for "libusb-devel". If you have the library installed, "locate usb.h" should turn up an include file in some appropriate location (perhaps '/usr/include') and that file should have declarations for usb_init(), usb_set_debug(), and usb_find_devices() among others. Help forums exist for most distributions and someone on one of these forums can probably help you find the appropriate library. Contact us if you get stuck.
The SDK also uses the Posix thread functions found in the 'pthread' library. Again, most recent distributions will have this library preinstalled.

3.0 USING THE SDK

The SDK consists of source code for the SDK functions, a .H header file for your C program, a sample C program (test.c) and a Makefile which demonstrates how to build your code to use the functions. Untar the SDK into a convenient place on your hard disk (tar -xvf LMShidxx.tar), and then copy these files into the directory of the executable program you are creating. Start by trying to build the sample (make all). If the build is successful, you're ready to add these functions to your own program. Add the header file (LMShid.h) to your project, and include it with the other header files in your program. Modify the make file by replacing 'test' with your program name. Or simply compile your program with the command line "gcc -o test -lm -lpthread -lusb <yourprogram>.c LMShid.c" In this case, the compiler will send the final output to 'test', link with the math, thread and usb libraries, and for source will use your program and the SDK source file, 'LMShid.c'.

[2] Usually it is a good idea to call fnLMS_GetNumDevices() at around 1 second intervals. While a short interval reduces the chances, it is still possible that the user will remove one device and replace it with another however, so to completely handle all the cases which can result from users hot plugging devices your application needs to check to see not only if the number of devices is different, but if the same number of devices are present, that they are not different devices.

4.0 PROGRAMMING

4.1 Overall Strategy and API Achitecture

The API provides functions for identifying how many and what type of LabBrick LMS signal generators are connected to the system, initializing signal generators so that you can send them commands and read their state, functions to control the operation of the signal generators, and finally a function to close the software connection to the signal generator when you no longer need to communicate with it.

The API can be operated in a test mode, where the functions will simulate normal operation but will not actually communicate with the hardware devices. This feature is provided as a convenience to software developers who may not have a LabBrick signal generator with them, but still want to be able to work on an applications program that uses the LabBrick. Of course it is important to make sure that the API is in its normal mode in order to access the actual hardware!
Before you do anything else, you MUST clear the SDK's internal structures. This is simply a call to fnLMS_Init() and only needs to be done once.
Be sure to call fnLMS_SetTestMode(FALSE), unless of course you want the API to operate in its test mode. In test mode there will be 2 devices, an LMS-103 and an LMS-123.
The first step is to identify the synthesizers connected to the system. Call the function fnLMS_GetNumDevices() to get the number of synthesizers attached to the system. Note that USB devices can be attached and detached by users at any time. If you are writing a program which needs to handle the situation where devices are attached or detached while the program is operating, you should periodically call fnLMS_GetNumDevices() to see if any new devices have been attached.[1]

Allocate an array big enough to hold the device ids for the number of devices present. While you should use the DEVID type declared in VNX_fmsynth.h it's just an array of units at this point. You may want to allocate an array large enough to hold MAXDEVICES device ids, so that you do not have to handle the case where the number of attached devices increases.

Call fnLMS_GetDevInfo(DEVID *ActiveDevices), which will fill in the array with the device ids for each connected frequency synthesizer. The function returns an integer, which is the number of devices present on the machine.

The next step is to call fnLMS_GetModelName(DEVID deviceID, char *ModelName) with a null ModelName pointer to get the length of the model name, or just use a buffer that can hold MAX_MODELNAME chars. You can use the model name to identify the type of synthesizer. Call fnLMS_GetSerialNumber(DEVID deviceID) to get the serial number of the synthesizer. Based on that information, your program can determine which device to open.

Once you have identified the synthesizer you want to send commands to, call fnLMS_InitDevice(DEVID deviceID) to actually open the device and get its various parameters like frequency setting, frequency sweep parameters, etc. After the fnLMS_InitDevice function has completed you can use any of the get functions to read the settings of the synthesizer.

To change one of the settings of the synthesizer, use the corresponding set function. For example, to set the synthesizer frequency, call fnLMS_SetFrequency(DEVID deviceID, int frequency). The first argument is the device id of the synthesizer, the second is the desired output frequency. Frequency is specified in 10 Hz increments, where:

frequency = Frequency (Hz) / 10

For example, to specify an output frequency of 5.5 GHz, frequency = 550000000.

To set the output power level, call fnLMS_SetPowerLevel(DEVID deviceID, int powerlevel) with the output power level you want. The power level is encoded as the number of .25dB increments, with a resolution of .5dB. To set a power level of +5 dBm, for example, powerlevel would be 20. To set a power level of -20 dBm, powerlevel would be -80.

Note that the LabBrick signal generators have a maximum and minimum settable power level. You can query the limits with calls to fnLMS_GetMaxPwr(DEVID deviceID) and fnLMS_GetMinPwr(DEVID deviceID). Both functions use the same encoding of the powerlevel as the SetPowerLevel function.

When you are done with the device, call fnLMS_CloseDevice(DEVID deviceID).

4.2 Status Codes

All of the set functions return a status code indicating whether an error occurred. The get functions normally return an integer value, but in the event of an error they will return an error code. The error codes can be distinguished from normal data by their numeric value, since all error codes have their high bit set, and they are outside of the range of normal data.

Functions that return a floating point result use specific, negative numeric values to indicate if an error occurred.

A separate function, fnLMS_GetDeviceStatus(DEVID deviceID) provides access to a set of status bits describing the operating state of the synthesizer. This function can be used to check if a device is currently connected or open.

The values of the status codes are defined in the VNX_fmsynth.h header file.

4.3 Functions – Setting up the environment & housekeeping

void fnLMS_Init(void)

Must be called once at the beginning of the user program to clear out the SDK's data structures, and initialize the USB library functions.

char* fnLMS_perror(LVSTATUS status)

Useful for debugging your user program, fnLMS_perror() takes a returned LVSTATUS value from another function and returns a pointer to a descriptive string you can display on screen or log.

char* fnLMS_LibVersion(void)

Returns a string which contains the version number of the SDK. If possible, call this function once when your program starts so you know the version number – that way, if you have questions or problems, you can include this version information in your question to us.

4.4 Functions – Selecting the Device

void fnLMS_SetTestMode(bool testmode)

Set testmode to FALSE for normal operation. If testmode is TRUE the dll does not communicate with the actual hardware, but simulates the basic operation of the dll functions. It does not simulate the operation of frequency sweeps generated by the actual hardware, but it does simulate the behavior of the functions used to set the parameters for sweeps.

int fnLMS_GetNumDevices()

This function returns a count of the number of connected synthesizers.

int fnLMS_GetDevInfo(DEVID *ActiveDevices)

This function fills in the ActiveDevices array with the device ids for the connected synthesizers. Note that the array must be large enough to hold a device id for the number of devices returned by fnLMS_GetNumDevices. The function also returns the number of active devices, which can, under some circumstances, be less than the number of devices returned in the previous call to fnLMS_GetNumDevices.

The device ids are used to identify each device, and are used in the rest of the functions to select the device. Note that while the device ids may be small integers, and may, in some circumstances appear to be numerically related to the devices present, they should only be used as opaque handles.

int fnLMS_GetModelName(DEVID deviceID, char *ModelName)

This function is used to get the model name of the synthesizer. If the function is called with a null pointer, it returns just the length of the model name string. If the function is called with a non-null string pointer it copies the model name into the string and returns the length of the string. The string length will never be greater than the constant MAX_MODELNAME which is defined in VNX_fmsynth.h. This function can be used regardless of whether or not the synthesizer has been initialized with the fnLMS_InitDevice function.

int fnLMS_GetSerialNumber(DEVID deviceID)

This function is used to get the serial number of the synthesizer. It can be called regardless of whether or not the synthesizer has been initialized with the fnLMS_InitDevice function. If your system has multiple synthesizers, your software should use each device's serial number to keep track of each specific device. Do not rely upon the order in which the devices appear in the table of active devices. On a typical system the individual synthesizers will typically be found in the same order, but there is no guarantee that this will occur.

int fnLMS_GetDeviceStatus(DEVID deviceID)

This function can be used to obtain information about the status of a device, even before the device is initialized. (Note that information on the sweep activity of the device is not guaranteed to be available before the device is initialized.)

int fnLMS_InitDevice(DEVID deviceID)

This function is used to open the device interface to the synthesizer and initialize the dll's copy of the device's settings. If the fnLMS_InitDevice function succeeds, then you can use the various fnLMS_Get* functions to read the synthesizer's settings. This function will fail, and return an error code if the synthesizer has already been opened by another program.

int fnLMS_CloseDevice(DEVID deviceID)

This function closes the device interface to the synthesizer. It should be called when your program is done using the synthesizer.

4.5 Functions – Setting parameters on the Signal Generator

LVSTATUS fnLMS_SetFrequency(DEVID deviceID, int frequency)

This function is used to set the output frequency of the synthesizer. Frequency is encoded as an integer number of 10 Hz steps:

frequency = Frequency (Hz) / 10

For example, to specify an output frequency of 6 GHz, frequency = 6000000. The value of frequency must be within the range of the attached synthesizer or an error will be returned.

LVSTATUS fnLMS_SetPowerLevel(DEVID deviceID, int powerlevel);

This function is used to set the output power level of the programmable synthesizer. The power level is specified in .25dB units. The encoding is:

powerlevel = desired output power in dBm / .25dB

For example, if you want -7.5 dBm output power then you should set powerlevel to -30.

LVSTATUS fnLMS_SetStartFrequency(DEVID deviceID, int startfrequency)

This function sets the frequency at the beginning of a frequency sweep. The encoding of startfrequency is the same as the fnLMS_SetFrequency function. Note that the start frequency should be less than the end frequency when you want the frequency to step upwards during the sweep. For a sweep where the frequency decreases, then the start frequency should be larger than the end frequency.

LVSTATUS fnLMS_SetEndFrequency(DEVID deviceID, int endfrequency)

This function sets the frequency at the end of a frequency sweep. The encoding of endfrequency is the same as the fnLMS_SetFrequency function.

LVSTATUS fnLMS_SetSweepTime(DEVID deviceID, int sweeptime)

This function sets the time duration of the frequency sweep. The sweeptime variable is encoded as a number of milliseconds. The minimum sweep time is 1 millisecond.

LVSTATUS fnLMS_SetRFOn(DEVID deviceID, bool on)

This function turns the RF stages of the synthesizer on (on = TRUE) or off (on = FALSE).

LVSTATUS fnLMS_SetUseInternalRef(DEVID deviceID, bool internal);

This function configures the synthesizer to use the internal reference if internal = 1. If internal = 0, then the synthesizer is configured to use an external frequency reference.

LVSTATUS fnLMS_SetSweepDirection(DEVID deviceID, bool up)

This function is used to set the direction of the frequency sweep. To create a sweep with increasing frequency, set up = TRUE. Note that the sweep start frequency value must be less than the sweep end frequency value for a sweep with increasing frequency. For a sweep that decreases in frequency, the sweep start value must be greater than the sweep end value.

LVSTATUS fnLMS_SetSweepMode(DEVID deviceID, bool mode)

This function is used to select either a single frequency sweep, or a repeating series of sweeps. If mode = TRUE then the sweep will be repeated, if mode = FALSE the sweep will only happen once.

LVSTATUS fnLMS_SetSweepType(DEVID deviceID, bool swptype)

This function is used to select between a single directional frequency sweep, or a sweep which returns to its original frequency after each sweep. If swptype = TRUE then the sweep will be bidirectional, if swtype = FALSE the sweep will only go in one direction. For a bidirectional sweep a graph of frequency vs. time for a repeating sweep will appear like a triangle wave, for a non-bidirectional sweep, the graph of frequency vs. time will appear like a sawtooth wave.

LVSTATUS fnLMS_StartSweep(DEVID deviceID, bool go)

This function is used to start and stop the frequency sweeps. If go = TRUE the synthesizer will begin sweeping, FALSE stops the sweep. You must set the sweep parameters before calling this function to start the sweep.

LVSTATUS fnLMS_SetFastPulsedOutput(DEVID deviceID, float pulseontime, float pulsereptime, bool on)

This function is the preferred way to control the internal pulse modulation option. The pulseontime parameter is the length of the pulse on time in seconds. The pulsereptime parameter is the length of the repetition period in seconds. Both values can range from 100 nanoseconds (0.100e-6) to 1000 seconds (1.0e3). Set on = TRUE to start the pulsed output modulation.

LVSTATUS fnLMS_SetPulseOnTime(DEVID deviceID, float pulseontime)

This function is used to set the length of the RF pulse on time of the device's internal modulation for devices that support pulsed output modulation. The pulseontime parameter is the length of the pulse on time in seconds, with a 100 nanosecond minimum. This function is not recommended for general use. Instead use the fnLMS_SetFastPulsedOutput function.

LVSTATUS fnLMS_SetPulseOffTime(DEVID deviceID, float pulseofftime)

This function is used to set the length of the RF pulse off time of the device's internal modulation. The pulseofftime parameter is the length of the pulse off time in seconds, with a 100 nanosecond minimum. The repetition period of the pulse modulation is equal to pulseontime + pulseofftime. This function is not recommended for general use. Instead use the fnLMS_SetFastPulsedOutput function.

LVSTATUS fnLMS_EnableInternalPulseMod(DEVID deviceID, bool on)

This function is used to turn on and off the internal output modulation. If on = TRUE the synthesizer will pulse its RF output on and off according to the values set for the pulse on time and pulse off time using either the fnLMS_SetFastPulsedOutput function or the functions to set pulse on and off time directly . To stop the internal pulse modulation, set on = FALSE. Always disable internal pulse modulation before setting the pulse on and off time using the fnLMS_SetPulseOnTime and fnLMS_SetPulseOffTime functions.

LVSTATUS fnLMS_SetUseExternalPulseMod(DEVID deviceID, bool external);

This function configures the synthesizer to use the external pulse modulation input signal if external = TRUE. If external = FALSE, then the synthesizer is configured to use the internal pulse modulation. Not all hardware configurations support an external pulse modulation input. Both the internal and external pulse modulation can operate at the same time, allowing more complex modulation patterns.

LVSTATUS fnLMS_SaveSettings(DEVID deviceID)

The LabBrick synthesizers can save their settings, and then resume operating with the saved settings when they are powered up. Set the desired parameters, then use this function to save the settings.

4.6 Functions – Reading parameters from the Signal Generator

int fnLMS_GetFrequency(DEVID deviceID)

This function returns the current frequency setting of the selected device. When a sweep is active this value will change dynamically to reflect the current setting of the device. The return value is in 10 Hz units.

int fnLMS_GetStartFrequency (DEVID deviceID)

This function returns the current frequency sweep starting value setting of the selected device. The return value is in 10 Hz units.

int fnLMS_GetEndFrequency (DEVID deviceID)

This function returns the current frequency sweep end setting of the selected device. The return value is in 10 Hz units.

int fnLMS_GetSweepTime(DEVID deviceID)

This function returns the current frequency sweep time in milliseconds. A one second sweep time, for example, would be returned as 1000.

int fnLMS_GetPulseOnTime(DEVID deviceID)
This function returns the pulse on time, which is the length of time that RF output is enabled when internal pulse modulation is operating, in seconds.

int fnLMS_GetPulseOffTime(DEVID deviceID)
This function returns the pulse off time, which is the length of time that RF output is disabled when internal pulse modulation is operating, in seconds. The pulse repetition period is equal to the pulse on time added to the pulse off time.

int fnLMS_GetPulseMode(DEVID deviceID)
This function returns an integer value which is 1 when the synthesizer internal pulse modulation is active, or 0 when the internal pulse modulation is off.

int fnLMS_GetUseInternalPulseMod(DEVID deviceID)
This function returns an integer value which is 1 when the synthesizer is configured to use its internal pulse modulation , or 0 when the external pulse modulation input is selected to control the output.

int fnLMS_GetHasFastPulseMode(DEVID deviceID)
This function returns an integer value which is 1 when the synthesizer has the internal pulse modulation option, or 0 when the option is not installed.

int fnLMS_GetRF_On(DEVID deviceID)

This function returns an integer value which is 1 when the synthesizer is “on”, or 0 when the synthesizer has been set “off” by the fnLMS_SetRFOn function.

int fnLMS_GetUseInternalRef(DEVID deviceID);

This function returns an integer value which is 1 when the synthesizer is configured to use its internal frequency reference. It returns a value of 0 when the synthesizer is configured to use an external frequency reference.

int fnLMS_GetPowerLevel(DEVID deviceID);

This function returns the current power level setting as an integer number of .25 dB units. As an example, an output power level of +3 dBm would result in the value 12 being returned, while an output power level of +3.5 dBm would result in the value 14 being returned. The output power resolution is .5 dB.

int fnLMS_GetMaxPwr(DEVID deviceID);

This function returns the maximum output power level that the synthesizer can provide, encoded in the same format as the fnLMS_GetPowerLevel function. For a synthesizer with +10 dBm maximum output power level this function returns the integer value 40. This is a read only value.

int fnLMS_GetMinPwr(DEVID deviceID);

This function returns the minimum output power level that the synthesizer can provide, encoded in the same format as the fnLMS_GetPowerLevel function. Typically this value is a negative number. For example, a device with -45 dBm minimum output power would return an integer value of -180. This is a read only value.

int fnLMS_GetMaxFreq (DEVID deviceID)

This function returns the maximum output frequency that the device can provide. The value is represented in 10 Hz units.

int fnLMS_GetMinFreq(DEVID deviceID)

This function returns the minimum output frequency that the device can provide. The value is represented in 10 Hz units.