

VAUNIX TECHNOLOGY CORPORATION



Lab Brick® Programmable Attenuator

Linux - USB API User Manual

Revision B1

4/3/2024

NOTICE

Vaunix has prepared this manual for use by Vaunix Company personnel and customers as a guide for the customized programming of Lab Brick products. The drawings, specifications, and information contained herein are the property of Vaunix Technology Corporation, and any unauthorized use or disclosure of these drawings, specifications, and information is prohibited; they shall not be reproduced, copied, or used in whole or in part as the basis for manufacture or sale of the equipment or software programs without the prior written consent of Vaunix Technology Corporation.

Table of Contents

1. Overview	3
2. Setting up the SDK	3
3. Using the SDK	3
4. Programming.....	3
4.1 Overall Strategy and API Architecture	3
4.2 Status Codes.....	5
4.3 Functions – Setting up the environment	5
4.4 Functions – Selecting the Device	5
4.5 Functions – Setting parameters on the Attenuator.....	7
4.6 Functions – Reading parameters from the Attenuator	10

1. Overview

The LabBrick Digital Attenuator SDK for Linux supports developers who want to control LabBrick Digital Attenuators from Linux programs. For maximum compatibility, the SDK includes source code for C functions to find, initialize, and control the attenuators, 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.

2. Setting up 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. 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 LDAhidxx.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 (LDAhid.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 LDAhid.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, 'LDAhid.c'.

4. Programming

4.1 Overall Strategy and API Architecture

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

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 Lab Brick attenuator with them, but still want to be able to work on an applications program that uses the Lab Brick. 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 `fnLDA_Init()` and only needs to be done once.

Be sure to call `fnLDA_SetTestMode(FALSE)`, unless of course you want the API to operate in its test mode. In test mode there will be 2 attenuator devices.

The first step is to identify the attenuators connected to the system. Call the function `fnLDA_GetNumDevices()` to get the number of attenuators 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 `fnLDA_GetNumDevices()` to see if any new devices have been attached. Usually it is a good idea to call `fnLDA_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 they are different devices.

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 `LDAhid.h` it's just an array of unsigned integers at this point. You may want to simply allocate an array large enough to hold `MAXDEVICES` device ids, so that you do not have to handle the case where the number of attenuators increases.

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

The next step is to call `fnLDA_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 attenuator. Call `fnLDA_GetSerialNumber(DEVID deviceID)` to get the serial number of the attenuator. Based on that information, your program can determine which device to open.

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

To change one of the settings of the attenuator, use the corresponding set function. For example, to set the attenuation level, call `fnLDA_SetAttenuation(DEVID deviceID, int attenuation)`. The first argument is the device id of the attenuator, the second is the value of the attenuation you want to set. For this command, the attenuation is specified in .05 dB units, so 10 dB of attenuation is represented as 200, 6 dB of attenuation is represented as 120, and .1 dB, the minimum attenuation increment, is represented as 2.

When you are done with the device, call `fnLDA_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.

A separate function, `fnLDA_GetDeviceStatus(DEVID deviceID)` provides access to a set of status bits describing the operating state of the attenuator. 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 `LDAhid.h` header file.

4.3 Functions – Setting up the environment

`void fnLDA_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* fnLDA_perror(LVSTATUS status)`

Useful for debugging your user program, `fnLDA_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* fnLDA_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 fnLDA_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 attenuation ramps generated by the actual hardware, but it does simulate the behavior of the functions used to set the parameters for the ramps.

`int fnLDA_GetNumDevices()`

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

`int fnLDA_GetDevInfo(DEVID *ActiveDevices)`

This function fills in the `ActiveDevices` array with the device ids for the connected attenuators. Note that the array must be large enough to hold a device id for the number of devices returned by `fnLDA_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 `fnLDA_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 fnLDA_GetModelName(DEVID deviceID, char *modelName)
```

This function is used to get the model name of the attenuator. 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_atten.h This function can be used regardless of whether or not the attenuator has been initialized with the fnLDA_InitDevice function.

```
int fnLDA_InitDevice(DEVID deviceID)
```

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

For optimum performance you should open the device interface at the beginning of use of the device, and close it when your program has completed. Repeatedly opening and closing a device will incur a performance penalty.

```
int fnLDA_CloseDevice(DEVID deviceID)
```

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

```
int fnLDA_GetSerialNumber(DEVID deviceID)
```

This function is used to get the serial number of the attenuator. It can be called regardless of whether or not the attenuator has been initialized with the fnLDA_InitDevice function. If your system has multiple attenuators, 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 attenuators will typically be found in the same order, but there is no guarantee that this will occur.

```
int fnLDA_GetLibVersion()
```

This function returns the version of the library.

int fnLDA_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 or ramp activity of the device is not guaranteed to be available before the device is initialized.)

4.5 Functions – Setting parameters on the Attenuator

For multi-channel LDA devices, the functions act on the currently selected channel, except for fnLDA_SetAttenuationQ, which combines channel selection and setting the attenuation value in one function, and the fnLDA_StartRampMC and fnLDA_StartProfileMC functions which act on the channels selected in their channel selection masks.

LVSTATUS fnLDA_SetChannel(DEVID deviceID, int channel)

This function is used to set the channel to be controlled. The channel defaults to channel 1 in the absence of a setting.

LVSTATUS fnLDA_SetWorkingFrequency(DEVID deviceID, int frequency)

This function is used to set the midband working frequency of the attenuator to optimize attenuation accuracy. The frequency setting is encoded as an integer using 100 kHz units. The encoding is:

$$\text{Frequency (MHz)} / 10$$

For example, to specify a working frequency of 1500 MHz, frequency = 15,000

LVSTATUS fnLDA_SetAttenuation(DEVID deviceID, int attenuation)

This function is used to set the attenuation level of the programmable attenuator. The attenuation setting is encoded as an integer where each increment represents .05db of attenuation. The encoding is:

$$\text{attenuation} * .05\text{db} = \text{Attenuation in dB}$$

For example, attenuation = 100 for 5 dB of attenuation, 2 for .1 dB of attenuation, and 2400 for 120 dB attenuation.

LVSTATUS fnLDA_SetAttenuationQ(DEVID deviceID, int Attenuation, int Channel)

This function uses the same deviceID as all the other API functions, an attenuation value in .05 dB steps, and a channel number (1 to N). Note that this function will leave the channel set to whatever value it is called with, so if you want to use other functions on other channels you need to call the SetChannel function before doing so. The attenuation setting is encoded as an integer where each increment represents .05db of attenuation. The encoding is:

$$\text{attenuation} * .05\text{db} = \text{Attenuation in dB}$$

For example, attenuation = 100 for 5 dB of attenuation, 2 for .1 dB of attenuation, and 2400 for 120 dB attenuation.

LVSTATUS fnLDA_SetRampStart(DEVID deviceID, int rampstart)

This function sets the attenuation level at the beginning of an attenuation ramp or sweep. The encoding of rampstart, the attenuation level, is the same as the fnLDA_SetAttenuation function.

LVSTATUS fnLDA_SetRampEnd(DEVID deviceID, int rampstop)

This function sets the attenuation level at the end of an attenuation ramp or sweep. The encoding of rampstop, the attenuation level, is the same as the fnLDA_SetAttenuation function.

LVSTATUS fnLDA_SetAttenuationStep(DEVID deviceID, int attenuationstep)

This function sets the size of the attenuation step that will be used to generate the attenuation ramp or sweep. The encoding of attenuationstep, is the same as the fnLDA_SetAttenuation function.

LVSTATUS fnLDA_SetAttenuationStepTwo(DEVID deviceID, int attenuationstep2)

This function sets the size of the attenuation step that will be used to generate the attenuation ramp or sweep during the second phase of a bidirectional sweep. The encoding of attenuationstep2, is the same as the fnLDA_SetAttenuation function.

LVSTATUS fnLDA_SetDwellTime(DEVID deviceID, int dwelltime)

This function sets the length of time that the attenuator will dwell on each attenuation step while it is generating the attenuation ramp. The dwelltime variable is encoded as the number of milliseconds to dwell at each level. The minimum dwell time is 1 millisecond.

LVSTATUS fnLDA_SetDwellTimetwo(DEVID deviceID, int dwelltime2)

This function sets the length of time that the attenuator will dwell on each attenuation step while it is generating the attenuation ramp. The dwelltime2 variable is encoded as the number of milliseconds to dwell at each level. The minimum dwell time is 1 millisecond.

LVSTATUS fnLDA_SetIdleTime(DEVID deviceID, int idletime)

This function sets the length of time that the attenuator will wait at the end of an attenuation ramp before beginning the ramp again when the ramp mode is set to SWP_REPEAT. The idletime variable is encoded as the number of milliseconds to dwell at each level. The minimum idle time is 0 milliseconds.

LVSTATUS fnLDA_SetHoldTime(DEVID deviceID, int holdtime)

This function sets the time delay between the first and second phases of a ramp. The holdtime variable is encoded as the number of milliseconds between phases. The minimum hold time is 0 milliseconds.

LVSTATUS fnLDA_SetProfileElement(DEVID deviceID, int index, int attenuation)

This function sets the value of a profile element. The index runs from zero to the maximum profile length minus 1. The attenuation value is encoded in .05db steps.

LVSTATUS fnLDA_SetProfileCount(DEVID deviceID, int profilecount)

This function sets the number of elements in the profile that will be used. It must be greater than zero and less than PROFILE_MAX, the maximum profile length.

LVSTATUS fnLDA_SetProfileIdleTime(DEVID deviceID, int idletime)

This function sets the idle time after a profile is played before the profile is played again in repeating profile mode.

LVSTATUS fnLDA_SetProfileDwellTime(DEVID deviceID, int dwelltime)

This function sets the time duration of each element in the profile during playback. The dwelltime is specified in milliseconds.

LVSTATUS fnLDA_StartProfile(DEVID deviceID, int mode)

This function starts the playback of a profile. A mode value of 1 plays the profile once, a mode value of 2 plays the profile repeatedly.

LVSTATUS fnLDA_StartProfileMC(DEVID deviceID, int mode, int chmask, bool delayed)

This function starts the playback of a profile on one or more specified channels. The channels are selected based on the bits set in chmask, with channel 1 corresponding to the least significant bit.

For example, a chmask value of 0x000000001A will start profiles on channels 2, 4, and 5.

A mode value of 1 plays the profile once, a mode value of 2 plays the profile repeatedly.

LVSTATUS fnLDA_SetRFOn(DEVID deviceID, bool on)

This function allows rapid switching of the attenuator from its set value “on” (on = TRUE) to its maximum attenuation (on = FALSE).

LVSTATUS fnLDA_SetRampDirection(DEVID deviceID, bool up)

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

LVSTATUS fnLDA_SetRampMode(DEVID deviceID, bool mode)

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

LVSTATUS fnLDA_SetRampBidirectional(DEVID deviceID, bool bidir_enable)

This function selects bidirectional ramps. For a bidirectional ramp the attenuation changes from the start to end value in the first phase, and then back to the start value in the second phase.

LVSTATUS fnLDA_StartRamp(DEVID deviceID, bool go)

This function is used to start and stop the attenuation ramps. If go = TRUE the attenuator will begin sweeping, FALSE stops the sweep.

LVSTATUS fnLDA_StartRampMC(DEVID deviceID, int mode, int chmask, bool deferred)

This function is used to start and stop the attenuation ramps on the specified channel. The channels are selected based on the bits set in chmask, with channel 1 corresponding to the least significant bit.

For example, a chmask value of 0x000000001A will start profiles on channels 2, 4, and 5.

If go = TRUE the attenuator will begin ramping, FALSE stops the ramp.

LVSTATUS fnLDA_SaveSettings(DEVID deviceID)

The Lab Brick attenuators 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 Attenuator

int fnLDA_GetWorkingFrequency(DEVID deviceID)

This function returns the current working frequency of the selected device.

int fnLDA_GetMinWorkingFrequency(DEVID deviceID)

This function returns the Minimum working frequency of the selected device in 100 Khz units.

int fnLDA_GetMaxWorkingFrequency(DEVID deviceID)

This function returns the Maximum working frequency of the selected device in 100 KHz units.

int fnLDA_GetAttenuation(DEVID deviceID)

This function returns the current attenuation setting of the selected device. When an attenuation ramp is active this value will change dynamically to reflect the current setting of the device. The return value is in .05 dB units.

int fnLDA_GetRampStart(DEVID deviceID)

This function returns the current attenuation ramp start value setting of the selected device. The return value is in .05 dB units.

int fnLDA_GetRampEnd(DEVID deviceID)

This function returns the current attenuation ramp end setting of the selected device. The return value is in .05 dB units.

int fnLDA_GetDwellTime(DEVID deviceID)

This function returns the current dwell time for each step on the attenuation ramp in milliseconds. A one second dwell time, for example, would be returned as 1000.

int fnLDA_GetDwellTimeTwo(DEVID deviceID)

This function returns the current dwell time for each step on the second phase of a bidirectional attenuation ramp in milliseconds. A one second dwell time, for example, would be returned as 1000.

int fnLDA_GetIdleTime(DEVID deviceID)

This function returns the idle time, which is the delay between attenuation ramps when the device is in the repeating ramp mode, in milliseconds.

int fnLDA_GetHoldTime(DEVID deviceID)

This function returns the hold time, which is the delay between attenuation ramps when the device is in the bidirectional ramp mode, in milliseconds.

int fnLDA_GetAttenuationStep(DEVID deviceID)

This function returns the current attenuation step size setting of the selected device. The return value is in .05 dB units, so for example an attenuation step of 5db would be represented by a return value of 100.

int fnLDA_GetAttenuationStepTwo(DEVID deviceID)

This function returns the current attenuation step size setting of the selected device during the second phase of a bidirectional ramp. The return value is in .05 dB units, so for example an attenuation step of 5db would be represented by a return value of 100.

int fnLDA_GetRF_On(DEVID deviceID)

This function returns an integer value which is 1 when the attenuator is “on”, or 0 when the attenuator has been set “off” by the fnLDA_SetRFOn function. Note that the function does not attempt to interpret attenuation settings as either “on” or “off”, so if you set the attenuation level to 120 dB, (attenuation = 2400) the output signal level would be the same as if you had used the fnLDA_SetRFOn function with the on = FALSE, but this function would not return 0.

int fnLDA_GetProfileElement(DEVID deviceID, int index);

This function gets the value of a profile element. The index runs from zero to the maximum profile length minus 1. PROFILE_MAX is currently 100. The attenuation value is encoded in .05db steps.

int fnLDA_GetProfileCount(DEVID deviceID);

This function sets the number of elements in the profile that will be used. It must be greater than zero and less than PROFILE_MAX, the maximum profile length.

int fnLDA_GetProfileDwellTime(DEVID deviceID);

This function gets the time duration of each element in the profile during playback. The dwelltime is specified in milliseconds.

```
int fnLDA_GetProfileIdleTime(DEVID deviceID);
```

This function gets the idle time after a profile is played before the profile is played again in repeating profile mode.

```
int fnLDA_GetProfileIndex(DEVID deviceID);
```

This function sets the number of elements in the profile that will be used. It must be greater than zero and less than PROFILE_MAX, the maximum profile length.

```
int fnLDA_GetMaxAttenuation(DEVID deviceID)
```

This function returns the maximum attenuation value that the device can provide. For the LDA-802 series programmable attenuators this value is 120 dB, which is 2400 .05 dB units. Since future products may have different maximum attenuation capabilities your software should use this function to obtain the maximum attenuation possible.

```
int fnLDA_GetMinAttenuation(DEVID deviceID)
```

This function returns the minimum attenuation value that the device can provide. In general, this value is 0 dB for the programmable attenuators. Since future products may have different capabilities, your software should use this function to obtain the minimum attenuation possible.

```
int fnLDA_GetMinAttenuationStep(DEVID deviceID)
```

This function returns the Minimum attenuation step size that the device can provide. In general, the attenuation step size ranges from 0.1 dB to 2.0 dB. Since future products may have different minimum attenuation step size capabilities your software should use this function to obtain the minimum attenuation possible.

```
int fnLDA_GetFeatures(DEVID deviceID);
```

This function returns a bit vector with bits set to indicate the available features. See VNX_LDA_api.h for definitions. Legacy devices have a zero value for the feature vector.

```
int fnLDA_GetNumChannels(DEVID deviceID);
```

This function returns the number of attenuation channels available.

```
int fnLDA_GetProfileMaxLength(DEVID deviceID);
```

This function returns the maximum length profile available for the programmable attenuator.