LabBrick Digital Attenuator
GNU/Linux SDK for libusb 0.1 or libusb-compat-0.1

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. 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.

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.

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'.

Overall Strategy and API architecture

The API provides functions for identifying how many and what type of LabBrick digital attenuators are connected to the system, initializing the 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 each attenuator 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 digital attenuator 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 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 devices, an LDA-102 and an LDA-602.

The first step in talking to the devices 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.[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 LDAhid.h it's just an array of unsigned ints at this point. You may want to just 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 fnLDA_GetDevInfo(DEVID *ActiveDevices), which will fill in the array with the device ids for each connected digital 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.

---

[1] 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 the same number of devices are present, that they are not different devices.

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 frequency setting, sweep 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 frequency). The first argument is the device id of the attenuator, the second is the attenuation.

Frequency is specified in 0.25 dB increments, where:

attenuation = desired attenuation * 4

For example, to specify an output attenuation of 16 dB less than the input, attenuation = 16 * 4 or 64.

Note that the LabBrick attenuators have a maximum and minimum settable attenuation level. You can query the limits with calls to fnLDA_GetMaxAttenuation(DEVID deviceID) and fnLDA_GetMinAttenuation(DEVID deviceID). Both functions use the same encoding of the powerlevel as the SetAttenuation function.

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

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.

Functions – Setting up the environment & housekeeping

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* fnDA_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.

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 frequency step sweeps generated by the actual hardware, but it does simulate the behavior of the functions used to set the parameters for the stepped sweeps.

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 LDAhid.h This function can be used regardless of whether or not the attenuator has been initialized with the fnLDA_InitDevice function.

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_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 fnLDA_InitDevice(DEVID deviceID)

This function is used to open the device interface to the attenuator and initialize the dll'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.

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.

Functions – Setting parameters on the attenuator

LVSTATUS fnLDA_SetAttenuation(DEVID deviceID, int attenuation)

This function is used to set the output attenuation level. Attenuation is encoded as an integer number of 0.25 dB steps (reduced from full output):

attenuation = Attenuation (dB) * 4

For example, to specify an output frequency of 30 dB, attenuation = 120. The attenuation valuemust be within the range of the attached attenuator or an error will be returned.

LVSTATUS fnLDA_SetRampStart(DEVID deviceID, int rampstart)

This function sets the beginning value for a self-stepping ramp function. Encoding is in 0.25 dB increments as done in fnLDA_SetAttenuation.

LVSTATUS fnLDA_SetRampEnd(DEVID deviceID, int rampstop)

This function sets the ending or stop value for a self-stepping ramp function. Encoding is in 0.25 dB increments as done in fnLDA_SetAttenuation.

LVSTATUS fnLDA_SetAttenuationStep(DEVID deviceID, int attenuationstep)

This function sets the ramp step size in 0.25 dB units. The ramp will begin at the Start value, increase by Step value once every DwellTime millseconds until it hits the End or Stop value.

LVSTATUS fnLDA_SetDwellTime(DEVID deviceID, int dwelltime)

The length of time each attenuation step will last, specified in milliseconds.

LVSTATUS fnLDA_SetIdleTime(DEVID deviceID, int idletime)

When continuous ramping is selected, the Idle (or Wait) time specifies how long to pause between ramps, specified in milliseconds.

LVSTATUS fnLDA_SetRFOn(DEVID deviceID, bool on)

This function enables or disables the attenuator output. Set on=TRUE to enable the output or on=FALSE to disable. ??

LVSTATUS fnLDA_SetRampDirection(DEVID deviceID, bool up)

This function determines the ramp direction. Set up=TRUE to go from lower (less attenuation) to higher (more attenuation) values. ??

LVSTATUS fnLDA_SetRampMode(DEVID deviceID, bool mode)

This function sets the ramp function to be a continuous sequence of repeating ramps (mode=TRUE) or a single ramp (mode=FALSE) ??

LVSTATUS fnLDA_StartRamp(DEVID deviceID, bool go)

This function starts the automatic ramp in the mode you have previously selected. Set the start, end dwell, idle direction and modes first, then call this with go=TRUE to begin the ramp function.

LVSTATUS fnLDA_SaveSettings(DEVID deviceID)

The LabBrick 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.

Functions – Reading parameters from the attenuator

int fnLDA_GetAttenuation(DEVID deviceID)

This function reads back and returns the current attenuation value, expressed in 0.25 dB units. A return value of 63 would indicate signal attenuation of 63 / 4 or 15.75 dB.

int fnLDA_GetRampStart(DEVID deviceID)

This function returns the beginning value for ramp operations. The value expresses attenuation in 0.25 dB increments. A return value of 63 would indicate signal attenuation of 63 / 4 or 15.75 dB at the beginning of a ramp.

int fnLDA_GetRampEnd(DEVID deviceID)

This function returns the ending value for ramp operations. The value expresses attenuation in 0.25 dB increments. A return value of 63 would indicate signal attenuation of 63 / 4 or 15.75 dB at the end of a ramp.

int fnLDA_GetDwellTime(DEVID deviceID)

Expressed in milliseconds, this returns the amount of time each ramp step will hold before moving on to the next. A return value of 250 indicates that a ramp will composed of steps lasting 250 milliseconds each.

int fnLDA_GetIdleTime(DEVID deviceID)

Expressed in milliseconds, this returns the amount of time to pause before restarting a ramp when continuous operation is selected. A return value of 500 indicates that after a ramp completes, the brick will wait 500 milliseconds before repeating the ramp operation.

int fnLDA_GetAttenuationStep(DEVID deviceID)

Expressed in 0.25 dB increments, this function returns the additional attenuation for each new step in a ramp. If this function returns 2, a ramp will start at the Start value, and increment by 0.5 dB (2 * 0.25) once every Dwell Time milliseconds until it reaches End or Stop. If in continuous mode, it will then wait Idle Time milliseconds before starting over.

int fnLDA_GetRF_On(DEVID deviceID)

Returns a 0 if the output is disabled or a 1 if enabled.

int fnLDA_GetMaxAttenuation(DEVID deviceID)

Returns the maximum attenuation value of the brick, expressed in 0.25 dB units. A value of 252 indicates a maximum attenuation value of 252/4 or 63 dB.

int fnLDA_GetMinAttenuation(DEVID deviceID)

Returns the minimum attenuation value of the brick. All standard LDA brick devices will return 0, indicating a minimum attenuattion of 0 dB.

int fnLDA_GetDevResolution (DEVID deviceID)

Returns the minimum possible change in attenuation value of the brick. The resolution is reported in .25db increments, so a returned value of 2 corresponds to .5db resolution. The device will round down any attenuation value sent to it that exceeds it resolution.