Notes on the XenceLabs Quick Keys

Bought this off amazon the other day and it's an interesting device, so I'm documenting what I find about it.

Hardware info

USB: VID `0x28bd` PID `0x5202` - Manufacturer “HANVON UGEE”, advertises no version or serial number.

USB pcaps pending - currently on a mac machine and capturing USB traffic on a mac requires disabling system integrity protection.

Device has a rotary encoder, 10 buttons, a power slider switch, RGB light ring, green power LED, blue connectivity LED, power slider, internal battery (capacity appears to be 1950mAh from the FCC internal photos, chemistry unknown), OLED screen (low pixel density, white phosphor)

Device appears to transmit only signals (ie., button press and rotary wheel moving +/- steps); although it appears to be using HID, it does not present as a regular input device and requires the driver components for actioning anything.

The device seems to have an FCC ID - `2AYM6-K02A`. The firmware version indicates it as `K02-B`, making me wonder if this device is not actually compliant with the FCC as it does not display the regulatory mark anywhere.

Software info

Software seems to be a generic XenceLabs-branded application that works for both the QuickKeys and other products offered by the company.

The software is delivered as a DMG disk image, which contains an `xar` compressed pkg (on Mac). Within this, there is a set of preinstall and postinstall scripts (preinstall removes any existing copies of the software, and closes any running instances; postinstall runs a loop waiting for the driver and other software to be installed) and the actual software. The scripts and software are stored in gzipped `cpio` archives.

The software archive contains a `LaunchAgents` plist (`com.xencelabs.xencelabstablet.plist`) which registers a launch agent `com.ugee.XencelabsAgent` to be started during login and during a logged-in session, and launches `XencelabsAgent` with the command-line argument `/mini`. It registers the `com.ugee.Xencelabs` mach service.

The software archive also contains a set of applications in a tree:

- Xencelabs
    - .guide (contains a copy of LGPLv3, some images, and some plists)
        - Info.plist: `{“Mode”: “0”}`
        - com.xencelabs.config.plist: `{“guide”: 0}`

        - com.xencelabs.config_run.plist: `{“guide”: 1, “KeyBoard”: 0}`
    - Xencelabs.app
    - UninstallXencelabs.app
    - Driver
        - XencelabsAgent.app
            - Contents/MacOS/DriverUpdate.app
            - Contents/MacOS/FirmwareUpdate.app
        - XencelabsDriver.app
        - XencelabsGuide.app

All applications appear to be universal binaries with both arm64 and amd64 builds.

Going by the system permissions prompts that come up when various inputs from the device are received by the computer, `XencelabsDriver` handles scrolling and `XencelabsAgent` handles keypresses. This is known because both applications request the ability to control the system via accessibility features. Rudimentary binary analysis shows that if this is denied, the programs will execute `tccutil reset Accessibility` on themselves to ask again for the permission.

Firmware info

The device came loaded with firmware version `K02-B 20210824`, and the diagnostic tool states the “keyboard selection” is English.

Observations relating to the stock software and firmware

Scrolling with the rotary encoder appears to be handled inefficiently, either in-hardware or via the software. Turning the wheel too quickly will result in maybe scrolling up or down by one step, or may result in no scroll at all.

The OLED itself seems to receive its displayed data from the computer - there are references to functions for writing OLED data in the software. This may explain why the screen updates so slowly in normal use - I can see the screen update in blocks.

There's a small amount of travel between the rotary encoder's centre button and any actual resistance.

The green LED (power) fades in a “breathing” animation pattern when the battery is charging, but becomes a solid green when the battery is full. The blue LED (connectivity) flashes when the device is not connected, and fades in a breathing animation pattern when it is connected via USB, but is a solid blue when connected via the dongle. When connected and charging, both LEDs appear to be synchronised.

The device can be turned off while plugged in; it will continue charging while off, but will not operate. Plugging it in when turned off will turn it on. It's unclear what the utility of it being plugged in but off is - preventing screen-burn when charging?

When first turned on wirelessly, or when first plugged in, the device will not operate without the software installed. The software itself claims to include drivers, but this is a USB HID device, which notably does not use conventional drivers, so realistically the software could be reverse-engineered as open-source software. Plugging the device in or turning it on with any of the ten buttons held down does not appear to invoke any kind of alternate mode (eg. DFU).

The software allows you to input a name for the hardware, but the name is only stored locally.

Observations relating to the hardware

The eight buttons surrounding the screen are made of metal, as is the mode switch button below them. The button in the centre of the rotary encoder is part of the rotary dial itself.

The device seems very well-built, and has a rubber base to prevent slipping when in use - a nice addition, but over the course of at least a decade, it will eventually degrade and perish, and need to be removed.

The device has a kensington lock slot - I wonder if they intend this for businesses?

The bottom-right corner of the device has a slot that would normally be for a fingernail or something to pull the back panel off. I haven't tried to pull it off yet - there's little to no flex when trying with my fingers, which leads me to wonder if it's just an aesthetic touch.

The dongle

The device contains a battery so that it can be used wirelessly - my first thought was that this was via bluetooth, but this does not appear to be the case. It came with a dongle, which I would assume means it runs over normal 2.4GHz or perhaps 430/868/915MHz.

When first plugged in, the dongle appears to be identified by the system as a keyboard.

USB: VID `0x28bd` PID `0x5203` - Manufacturer “HANVON UGEE”, again advertising no serial number or version.

It has model number ACD12-A and FCC ID 2AYM6-ACD12A

With the dongle plugged in, the software allows you to manage up to two “pairings” for the dongle. The device appears to be pre-paired to slot 2, and the software lists it as having ID (address?) `f483b3LLNNLL` (last three octets anonymised but available on request) - this looks very much like a bluetooth MAC address, making me wonder if the dongle is simply bluetooth.

The FCC-published information about this device indicates that yes - this is simply bluetooth/BLE. The detailed emissions report includes a screenshot of nRFgo Studio, which suggests a Nordic Semiconductor nRF is being used for BLE transmission (which would make sense).

Diagnostic information in the software reports the dongle's firmware as `ACD12-B 20201214094214 15.3.0`

Linux Drivers

Somehow never occurred to me that the linux driver package might shed more light on how the software and hardware interact.

The Linux download contains packages for RPM and dpkg-based distros, as well as a tarball with installation scripts - the most obvious point for examination. It contains normal XDG stuff for startup, application registration and so on, but also contains udev rules:

10-xencelabs.rules

```
KERNEL=="uinput",MODE:="0666",OPTIONS+="static_node=uinput"
SUBSYSTEMS=="usb",ATTRS{idVendor}=="28bd",MODE:="0666"
```

It also contains the actual software and a set of Qt5 libraries. The application itself is invoked via a script which exports LD_LIBRARY_PATH beforehand.

The application binary is (predictably) a dynamically-linked ELF built for amd64.

Plans

Get USB packet captures of:

- Initial communication between the software and the device itself
- Initial communication between the software and the dongle
- Configuring the pairing for the dongle and the device
- Configuring the device
- Changing the sleep time and screen brightness

Check any network communications the software makes.

- Application has references to several hostnames/API servers, all ending with `/api`
  - `www,xencelabs,com`
  - `www,newtest,xencelabs,com,cn`
  - `www,xencelabs,com,cn`
  - `www,xencelabs,com/ctrad`
  - `www,xencelabs,com/{fr|de|jp|kr|it|es}`
  - `www,zs,xencelabs,cc`
- Checks for updates to itself by calling `{API server}/getLatestDriver?system_sign={mac|win|linux}&language={en|?}` - returns a JSON payload
- Checks for firmware updates by calling `{API server}/getLatestFirmware?project_sign={?}` - presumably returns a JSON payload, no idea what `project_sign` is though

Notes during reverse-engineering

Both the dongle and the direct USB connection expose a USB Usage with page FF0Ah

Not 100% on packet/HID report format, but..

- All writes are 32-byte packets
- Bytes 10-15 inclusive are the MAC address of the device, or zeroes if connected via USB instead of the dongle
- Byte 0 is always 0x02 - this is the endpoint I think
- Data is only sent from the device once it has been subscribed to:
- 02b410 - Subscribes to battery change events
- 02b004 - Subscribes to button press and wheel events
- 02b801 - Subscribes to dongle connection events

Commands/opcodes are as follows (an ampersand followed by a number indicates an argument/parameter)

- 02b40801&0 - Set the idle timeout in minutes between 0 and 255
- 02b4040101&0 - Set the velocity (poll rate?) of the wheel (5 = slowest, 1 = fastest)
- 02b10a01&0 - Set the display brightness (0 = off, 3 = highest)
- 02b1&0 - Set the display rotation (0 = 0º, 3 = 270º)
- 02b40101&0&1&2 - Set the RGB colour value of the ring around the wheel (0-255, 0-255, 0-255)
- 02b100&000&10000000000000000000000000000000&2… - Set the label for one of the eight macro keys (arg0 is the key number from 1 to 8, arg1 is the length of the label text * 2, arg2 is the label text encoded as unicode/utf16le) - the device seems able to take a label length up to 24 characters but i've not identified how it receives characters after the 8th.
- 02b1&0&100&2&3 - Display an “overlay” message (arg0 indicates if it is the first message, 5, or a continuation of the message, 6. arg1

Links

https://github.com/julusian/node-xencelabs-quick-keys - NodeJS implementation of the HID interface