DPF HACKING by L.CYFER 2012-01

- enabling & optimizing of the AX206 based “digital picture frame” for Windows 7 operation
- DPF is controlled by pure python code.
- if something does not work - try to consult this docu again
- in case of any failures - take my apologies, but dont blame me ;)

required: (= my environment for testing)		
Windows 7 32bit ultimate		
libusb-win32-bin-1.2.5.0	INF generator for DPF	http://sourceforge.net/apps/trac/libusb-win32/wiki [http://sourceforge.net/apps/trac/libusb-win32/wiki]
python 2.7	python interpreter	http://python.org/ftp/python/2.7.2/python-2.7.2.msi [http://python.org/ftp/python/2.7.2/python-2.7.2.msi]
pyusb-1.0.0a2	python lib to access USB	http://sourceforge.net/projects/pyusb/ [http://sourceforge.net/projects/pyusb/]
PIL-1.1.7.win32-py2.7.exe	python lib to handle images	http://www.pythonware.com/products/pil/ [http://www.pythonware.com/products/pil/]
a “hackfin” hacked DPF AppoTech_AX206		http://tech.section5.ch/news/?p=77 [http://tech.section5.ch/news/?p=77])

why windows?
After playing with dockstar & co successfully, I finally decided to use a “real PC” with Windows as a home server plattform. My MSI E350IS-E45 is running perfectly with approx. 10 watts @ idle. I use the Abyss Server Application and some Python scripts to monitor & control everything. One “fun” step was to integrate the hard & software to drive the Ax206 as a status display for the E350.

“pros” until now:

1. drivers for the E350 are optimized for Windows. only this plattform will lead to min. power consumption. example: without the appropriate vendor drivers, the system will have approx. 25 watts in idle - with it, approx. 10 watts
2. windows is running perfectly of the shelf. there is no need to spend hours for a linux setup
3. Abbys is free, light, fast & easy - just install and run the config via browser
4. normal CGI, FastCGI or WSGI with abbyss/python run seemlessly
5. Addons like Python, PHP, docuwiki are installed within minutes
6. specific hardware - e.g. RF protocol scanners - will run with native Windows driver - no hunt for linux drivers required
7. stable - my experience with debian @ dockstar was, that a reproducable power down & up is not possible

“cons” until now:

1. if you browse the inet for python docs, and run into “linux” stuff you could get “lost”. a lot of this sources are outdated or simply of chaotic nature. e.g. “FastCGI”

MODIFY FIRMWARE - step by step

hack your DPF

- forum discussion: e.g.: http://www.vdr-portal.de/board18-vdr-hardware/board11-lcds/109196-howto-pearl-dpf-easy-hacking/ [http://www.vdr-portal.de/board18-vdr-hardware/board11-lcds/109196-howto-pearl-dpf-easy-hacking/]
- windows tool for flashing: “ProgSPI.exe” (see forum discussion)
- use firmware file “Pearl DPF hackfin landscape 0.12devel firmware.bin”
- hint: there is no need to use linux and do all the complex software installation & compiling just to get the 2MB modded firmware to the DPF. Save your time!

1. Start “ProgSPI.exe” ⇒ Settings → check items “Program” & “Reset”
2. connect DPF to USB port
3. on DPF: press RESET & hold, press MENU, release RESET, release MENU ⇒ black screen ⇒ “channel 1” of “ProgSPI” must go green
4. “ProgSPI”: select firmware by “browse” and do the “update” ⇒ takes some minutes

this will add “custom” USB-commands to the DPF. So it can be controlled under windows (or any OS) by low level/bulk USB commands.

after this “update”:

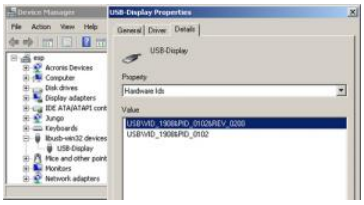
1. power on DPF.
2. hold MENU for some seconds, to activate “hackfin mode”.
3. windows will list an usb device without driver.
4. check “device manager” to lookup vendor & product IDs.

There is no technical difference between flashing the firmware by the known linux approaches or by Windows.
“linux flashed” devices can be controlled under Windows or vice-versa.
Most important is the flash-file itself. There are different versions floating through the internet. Dependant on the version, there may be different “commands” available for controlling! So, be sure to use the file above for the following setup.

WINDOWS DRIVER

In fact, there is no real driver needed for windows operation.
An easy solution is to generate a very small “bulk” driver for the DPF.
This can be done for any “driver-less” devices under Windows & just ensures the correct identification by Windows.

required steps:



1. get file “libusb-win32-bin-1.2.5.0” from http://sourceforge.net/apps/trac/libusb-win32/wiki [http://sourceforge.net/apps/trac/libusb-win32/wiki] & extract archive
2. read the docu (link above, section “Device Driver Installation”) or just proceed here
3. use the INF-Wizard to generate a INF file for the DPF (archive location: “bin/inf-wizard.exe”)
4. select the “unknown DPF” in windows device manager and “update driver” with the generated INF file
5. this “INF driver (+generic DLL/SYS) will make the DPF a known device for windows
6. lookup vendor and product IDs of your DPF (will be used in python code)

WINDOWS SOFTWARE CONTROL

Below you will find some basic code snippets to detect & control the Ax206 DPF.

This snippets will not run directly with python & the DPF - they are just to demonstrate the easiness of implementation on Windows machines.
To “see” something on the DPF ad hoc, you should download and run the demo files at the end of this section.

WinAx206_FindDPF.py

```
import usb.core # from pyusb-1.0.0a2
import usb.util # from pyusb-1.0.0a2
import sys

PSCRIPT = os.path.basename(__file__)
print PSCRIPT+ ': USB DPF CONTROL _/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/'

idV = 0x1908
idP = 0x0102

dev = None

while dev==None:

        print 'try to find device by idV= '+ hex(idV) + ' & idP= '+ hex(idP)

        dev = usb.core.find(idVendor=idV, idProduct=idP)

        if dev==None:
                print 'NO DEVICE FOUND.'
                print 'sleep 3 seconds...'
                time.sleep(3)
        else:
                str0 = ''
                print 'FOUND: str(dev) = ' + str(dev)
                for i0 in range(1,5): str0 += usb.util.get_string(dev, 100, i0)+', '
                print 'FOUND: args     = ' + str0

dev.set_configuration()
```

WinAx206_DPFblit.py

```
def dpf_screen_blit(buf0,rect):

        cmd0 = [        0xcd, 0, 0, 0, 0, 6, \
                        0x12, # USBCMD_BLIT \
                        WL(rect[0]),  WH(rect[0]),  WL(rect[1]),  WH(rect[1]), \
                        WLm(rect[2]), WHm(rect[2]), WLm(rect[3]), WHm(rect[3]), \
                        0 ]

        emulate_scsi(-1, cmd0 , 'CMD_AND_DATA', buf0)
```

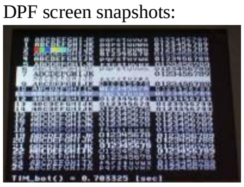
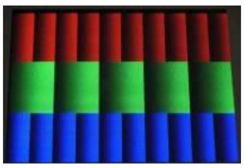
WinAx206_DPFcolorTestBars.py

```
# TEST PIC - COLOR BARS
# bits: 5xR 6xG 5xB = RRRRR GGGGGG BBBBB => 0..1F, 0..3F, 0..1f

buf0=[]
for y0 in range(ywid):
        for x0 in range(xwid):

                luma = x0 & 0x3f

                if y0>=0:       rgb = luma << 11
                if y0>=80:      rgb = luma << 5
                if y0>=160:     rgb = luma

                buf0 += [ (rgb>>8) & 0xff ]
                buf0 += [ (rgb>>0) & 0xff ]

dpf_screen_blit(buf0,[0,0,xwid,ywid])
```

Next, the full source code of a demo app for the Ax206 DPF.
This demo will run directly within a Windows cmd.exe terminal. (requirements see top of page)
Be sure, that the py-file-extension will refere to your actual python.exe interpreter. (normally, this is done by a registry entry by the python windows-installer.)
The App will try to detect the DPF every 3 seconds (infinite loop).
On success, it will display various demo screens on the DPF.

PYTHON demo with source: SRC_WinAX206_DemoShow_2012-01-28_14-31-02.zip
remark: if you read this docu as a pdf, all the source code will reside in the same archive like this PDF. (or “somebody” ripped the stuff apart….)

DPF screen snapshots:


DPF INSTANT STARTUP

Due to the nature of the modded firmware of “mr.hackfin”, entering the required command mode of the DPF hast to be done by holding key “MENU” for several seconds after every powerup. (e.g. reboot of the host PC). So, a feature to enter this specific mode directly after powerup is strongly required. After inspection of the 2MB 8052 assembler code (!), it came to my mind, that a pure hardware solution would be much more easy to implement.

Solution I: (solder time 5 min.)

1. remove & compensate the battery-pack (see picture) ⇒ this will prevent shutdown on low/bad bat
2. A simple, RC-delay generator + T to “emulate” the “M”-key after powerup (see picture)
    - drawbacks:

- 'M' will be “pushed” permanently.
- mo more manual operation possible
- on rebooting (not powerup) of the host PC there will be some USB communation by Windows that can cause the DPF to go to flash mode (black screen ⇒ dead end) - workaround: disable “USB legacy” in BIOS

3. rewire the “M-key” to enable “hackfin-menu control” (optional, see picture) - side effect: “M”-key will have reverse function, but work

Solution II: (solder + code time 10 min.)

1. use ATtiny to generate a “real” “M”-key-“OFF-ON-OFF” sequence at power-up
2. very simplistic approach
3. code soon ;) Fix Me!

snapshot of Solution I: