Lab 25, network, part 2

By ADMIN | *Published:* FEBRUARY 23, 2014

After the Lab 24 when we prepared the layout I took serious study of the ways to implement the ethernet driver. And I found that there is actual convergence of 2 ways of implementation that I saw in previous lab. Comparing to implementation of ethernet driver in C (**ether.c**, **smsc.c**) it is enough to have code worked up to call of *kernelproxy()* which just bypass control to **etherusb.c** – send a "bind" command into the control file and bypass as arguments usb end point files for input and output data streams.

If so, then I have no need to implement the file-server support of "ether" device in limbo as it will handled by **etherusb.c** which we already have ready (of course to find this simple solution fisrt I ported even more C code to Limbo).

I took parts from Plan9 of **ether.c** and **smsc.c** and start converting into Limbo, which wasn't so complicated task. Actually I combined then into single file **ethersmsc.b**.

When I had attempt to start Raspberry with this new ethernet driver but to my wonder I found that Mac address is just zeros. Strange, but by asking experts and checking info, yes that's correct: eeprom of ethernet chip on Raspberry does not have the MAC there embedded. Instead it is kept in VideoCore chip and can be received by MailBox interface.

The communication with VideoCore is already implemeneted in **vcore.c** which we have from 9pi project. But my attempt to receive MAC showed something very strange – it didn't work at all. Channel 1 of mailbox interface – Framebuffer works perfect why Channel 8 – getting properties from chip does not work at all.

Almost two days of debugging, different trys, attempts while I realized that I probably have very old firmware. Oops. When I did try get new firware files (https://github.com/raspberrypi/firmware/tree/master/boot) I found another stumbling block: U-Boot no more working!

```
1 ** Unable to use mmc 0:1 for fatload **
```

Well, my U-Boot was from friend of mine and from 2012 (yep, old). As next I need to compile U-Boot myself. Version from repository didn't compile for some reason, so I tried the release 2013-10.

```
1 # emerge --ask crossdev
2 # crossdev -S -v -t armv6j-hardfloat-linux-gnueabi
3 # cd u-boot
4 # make ARCH=arm CROSS_COMPILE=armv6j-hardfloat-linux-gnueabi- rpi_b_config
5 # make ARCH=arm CROSS_COMPILE=armv6j-hardfloat-linux-gnueabi-
```

Complete!
But I as result I have elf file while I need an image file. For this special tool:

```
1 # emerge sys-boot/raspberrypi-mkimage
```

Next attempts to boot didn't work again. Ah, I need to process **u-boot.bin** but not just **u-boot** file:

```
1 # imagetool-uncompressed.py u-boot.bin /mnt/SD/u-boot.bin
```


Categories

- Blog
- Boost
- C++
- Cryptography
- Embedding
- Hybrids
- Inferno OS
- MacAppStore
- Misc
- Models
- Projects
- PyQt
- PySide
- Qt
- QtSpeech
- Raspberry Pi
- Research
- Ru
- TogMeg
- Trac
- TTS
- Tutorial
- Undo
- Usb
- Web

Finally! It boot! But wait:

```
## Executing script at 00000000
Unknown command 'usb' - try 'help'
```

Why? After searches I found that even up to current moment they haven't integrated Usb codes for raspberry into U-Boot 😕
I have to go back. The very recent but with Usb is the repository https://github.com/gonzoua/u-boot-pi.
Also looks from 2012 but I have hope.

Again, compile cycle and I got U-Boot with Usb finally and with new firmware!

```
In:    serial
Out:   lcd
Err:   lcd
mbox: Timeout waiting for response
bcm2835: Could not set USB power state
Net:   Net Initialization Skipped
No ethernet found.
Hit any key to stop autoboot:  0
reading uEnv.txt
** Unable to read file uEnv.txt **
reading boot.scr
247 bytes read in 8516 ms (0 Bytes/s)
Running bootscript from mmc0 ...
## Executing script at 00200000
(Re)start USB...
USB0:   Core Release: 2.80a
scanning bus 0 for devices... Error condition at line 653:  XACTERR
Error condition at line 653:  XACTERR
Error condition at line 653:  XACTERR
USB device descriptor short read (expected 18, got 0)
4 USB Device(s) found
       scanning usb for storage devices... 0 Storage Device(s) found
       scanning usb for ethernet devices... 1 Ethernet Device(s) found
Waiting for Ethernet connection... done.
BOOTP broadcast 1
*** Unhandled DHCP Option in OFFER/ACK: 95
*** Unhandled DHCP Option in OFFER/ACK: 95
DHCP client bound to address 10.0.55.107
Waiting for Ethernet connection... done.
Using sms0 device
TFTP from server 10.0.55.110; our IP address is 10.0.55.107
Filename 'irpi'.
Load address: 0x7fe0
```

And an attempt to get stored MAC Address with MailBox from VideoCore and success!

```
mac: b827ebd9eafd
```

To bypass it from kernel to Usb driver we will just create environment variable in **main.c** funcion *inito()*:

```
snprint(buf, sizeof(buf), "%s", getethermac());
ksetenv("ethermac", buf, 0);
```

Also bind "/env" as we haven't done it yet in **rpiinit.b**.
Testing and looks okay, I see the MAC in ifc

```
; cd /net/ether0
; ls
addr
clone
ifstats
stats
; cat addr
b827ebd9eafdx; cat stats
in: 470
link: 0
out: 6
crc errs: 0
overflows: 0
soft overflows: 0
framing errs: 0
buffer errs: 0
output errs: 0
```

```
prom: 0
mbps: 100
addr: b827ebd9eaf
```

Now time to get IP with DHCP. To do so I decided for now just put code into driver:

```
# default initialization:
# let's get ip with dhcp
confether() {
    fd: ref Sys->FD;
    ethername := "ether0";

    fd = sys->open("/net/ipifc/clone", sys->OWRITE);
    if(fd == nil) {
        sys->print("init: open /net/ipifc/clone: %r\n");
        return;
    }
    if(sys->fprint(fd, "bind ether %s", ethername) < 0) {
        sys->print("could not bind ether0 interface: %r\n");
        return;
    }

    fd = sys->open("/net/ipifc/0/ctl", Sys->OWRITE);
    if(fd == nil){
        sys->print("init: can't reopen /net/ipifc/0/ctl: %r\n");
        return;
    }

    dhcpclient = load Dhcpclient Dhcpclient->PATH;
    if(dhcpclient == nil){
        sys->print("can't load dhcpclient: %r\n");
        return;
    }
    dhcpclient->init();
    (nil, nil, err) := dhcpclient->dhcp("/net", fd, "/net/ether0/addr", nil,
nil);
    if(err != nil){
        sys->print("dhcp: %s\n", err);
        return;
    }
}
```

Rebooting again, and

```
; cd /net/ipifc/0
; ls
ctl
data
err
listen
local
remote
snoop
status
; cat local
10.0.55.105 -> 255.255.255.255 10.0.0.0 10.255.255.255 10.0.55.0 10.0.55.255
10.0.55.105
;
```

Great! Try to ping from both sides:

Linux:

```
# ping 10.0.55.105
PING 10.0.55.105 (10.0.55.105) 56(84) bytes of data.
64 bytes from 10.0.55.105: icmp_seq=1 ttl=255 time=0.843 ms
64 bytes from 10.0.55.105: icmp_seq=2 ttl=255 time=0.419 ms
64 bytes from 10.0.55.105: icmp_seq=3 ttl=255 time=0.389 ms
64 bytes from 10.0.55.105: icmp_seq=4 ttl=255 time=0.407 ms
```

Raspberry:

```
; ip/ping -n 4 10.0.55.110
sending 4 64 byte messages 1000 ms apart
0: rtt 20000 µs, avg rtt 20000 µs, ttl = 64
1: rtt 20000 µs, avg rtt 20000 µs, ttl = 64
2: rtt 20000 µs, avg rtt 20000 µs, ttl = 64
3: rtt 10000 µs, avg rtt 17500 µs, ttl = 64
;
```

Completed!

This entry was posted in *Blog*, *Inferno OS*, *Projects*, *Raspberry Pi*, *Research*, *Usb*. Bookmark the *permalink*. *Post a comment* or leave a trackback: *Trackback URL*.

« *Lab 24, network, part 1*