L4 / Fiasco.OC

Porting Guide

for Freescale i.MX6 Quad SABRE SD

2014. 03. 25 (Tues)

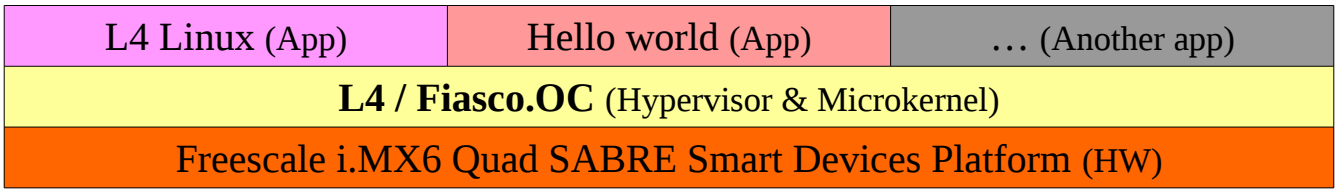
Taeung Song
treeze.taeung@gmail.com
taeung@xsinc.co.kr

L4 / Fiasco.OC – Porting Guide

1. The Goal & System Layer

This document's goal is booting "L4/Fiasco.OC" on Freeascale i.MX6q_sabre_sd.

The System layer of the goal is as follows...



And you need to set up SD card as example.

unallocated	u-boot.bin	unallocated	uImage	unallocated
0KB.....16384KB	431.6KB	2048KB	8.8MB	.....

2. Everything you prepare for booting

A. Packages and programs you need ,etc.

1) GNU C & C++ compiler, GNU Binutils, GNU Make, Perl, subversion

```
$ sudo apt-get install make gawk g++ binutils pkg-config subversion
```

2) 'ncurses' package for console like GUI

```
$ sudo apt-get install libncurses5 libncurses5-dev
```

3) u-boot-tools for 'mkimage'

```
$ sudo apt-get install u-boot-tools
```

4) ramdisk

```
$ sudo wget -c http://os.inf.tu-dresden.de/download/ramdisk-arm.rd
```

5) minicom

```
$ sudo apt-get install minicom
```

6) Librasry or tool for 32bit in 64bit development environment

If you don't need it , skip this.

```
$ sudo apt-get install libc6-dev-i386 lib32z1 lib32ncurses5 lib32bz2-1.0
```

B. Source code

1) L4re and Fiasco.OC

```
$ cd /somedir/tudos
$ svn cat http://svn.tudos.org/repos/oc/tudos/trunk/repomgr | perl – init
http://svn.tudos.org/repos/oc/tudos fiasco l4re l4linux_requirements
```

If you don't need source code related to 'L4 Linux', you can exclude 'l4linux_requirements'.

2) L4 Linux

```
$ svn co http://svn.tudos.org/repos/oc/l4linux/trunk l4linux
```

C. Tool chains (Cross Compiler)

You can choice one of tool chains.

1) Wiki Page

```
$ firefox http://tuxamito.com/wiki/index.php/Cross-compile_toolchains
```

2) apt-get

```
$ sudo apt-get install gcc-arm-linux-gnueabi
```

3) Download from Linaro.

```
Linaro: https://launchpad.net/linaro-toolchain-binaries
```

4) Download what freescale use.

Mine:https://www.dropbox.com/s/wuo9f2i2lde6hpl/gcc-4.6.2-glibc-2.13-linaro-multilib-2011.12.tar.gz

D. U-boot

You can download BSP files from Freescale home page, find u-boot.bin or source code. You will need to login it's home page.

1) Download BSP images including 'u-boot-mx6q-sabresd.bin' from Freescale home page.

https://www.freescale.com/webapp/Download?colCode=L3.0.35_4.1.0_DEMO_IMAGE_BSP&appType=license&location=null&fasp=1&WT_TYPE=Board%20Support%20Packages&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=gz&WT_ASSET=Downloads&Parent_nodeId=13378039778587110246 57&Parent_pageType=product&Parent_nodeId=133780397785871102 4657&Parent_pageType=product

2) Download it's source code.

https://www.freescale.com/webapp/Download?colCode=L3.0.35_4.1.0_ER_SOURCE_BSP&appType=license&location=null&fasp=1&WT_TYPE=Board%20Support%20Packages&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=gz&WT_ASSET=Downloads&Parent_nodeId=13378039778587110246 57&Parent_pageType=product

E. Board

Freescale i.MX6 Quad SABRE Smart Devices platform

F. SD card

G. MicroUSB cable

3. How to Build L4re ?

A. Modify source code about this board's UART serial part

```
diff --git a/l4/pkg/bootstrap/server/src/platform/imx.cc
           b/l4/pkg/bootstrap/server/src/platform/imx.cc
index e328deb..82ba849 100644
--- a/l4/pkg/bootstrap/server/src/platform/imx.cc
+++ b/l4/pkg/bootstrap/server/src/platform/imx.cc
@@ -124,8 +124,8 @@ class Platform_arm_imx : public Platform_single_region_ram

         /* End Neon Setup */

-    //static L4::Io_register_block_mmio r(0x02020000); // UART1
-    static L4::Io_register_block_mmio r(0x021e8000); // UART2
+    static L4::Io_register_block_mmio r(0x02020000); // UART1
+    //static L4::Io_register_block_mmio r(0x021e8000); // UART2
     static L4::Uart_imx6 _uart;
 #else
```

B. Create a build directory, configure and build

```
$ cd /somedir/tudos/src/l4
$ mkdir -p /path/to/obj/tudos
$ make O=/path/to/obj/tudos config

# Target Architecture (ARM architecture)  --->
# CPU variant (ARMv7A type CPU)  --->
# Platform Selection (Freescale i.MX6)  --->

$ make O=/path/to/obj/tudos
```

4. How to Build Fiasco ?

A. Modify source code about this board's UART serial part

```
diff --git a/kernel/fiasco/src/kern/arm/bsp/imx/mem_layout-arm-imx.cpp
           b/kernel/fiasco/src/kern/arm/bsp/imx/mem_layout-arm-imx.cpp
index 1b849b7..df03cfc 100644
--- a/kernel/fiasco/src/kern/arm/bsp/imx/mem_layout-arm-imx.cpp
+++ b/kernel/fiasco/src/kern/arm/bsp/imx/mem_layout-arm-imx.cpp
@@ -109,7 +109,7 @@ public:
     Watchdog_map_base    = Devices2_map_base + 0xbc000, // wdog1
     Gpt_map_base         = Devices2_map_base + 0x98000,
     Src_map_base         = Devices2_map_base + 0xd8000,
-    Uart_base            = Uart2_map_base,
+    Uart_base            = Uart1_map_base,
   };
```

```
diff --git a/kernel/fiasco/src/kern/arm/bsp/imx/uart-imx.cpp
           b/kernel/fiasco/src/kern/arm/bsp/imx/uart-imx.cpp
index 7f6e572..a704645 100644
--- a/kernel/fiasco/src/kern/arm/bsp/imx/uart-imx.cpp
+++ b/kernel/fiasco/src/kern/arm/bsp/imx/uart-imx.cpp
@@ -41,7 +41,9 @@ IMPLEMENTATION [imx6]:

 #include "uart_imx.h"

-IMPLEMENT int Uart::irq() const { return 59; }
+// uart-1: 58
+// uart-2: 59
+IMPLEMENT int Uart::irq() const { return 58; }

 IMPLEMENT L4::Uart *Uart::uart()
 {
```

B. Create a build directory, configure and build

```
$ cd /somedir/tudos/src/kernel/fiasco
$ make BUILDDIR=/path/to/obj/fiasco
$ cd /path/to/obj/fiasco
$ make menuconfig

# Target configuration ---> Architecture (ARM processor family)
# Platform (Freescale i.MX)
# Freescale i.MX (i.MX6)
# CPU (ARM Cortex-A9 CPU)

$ make
```

5. How to make ‘L4/Fiasco.OC uImage’ ?

Now, you are ready to make ‘L4/Fiasco.OC uImage’.
You need to three (A,B,C).

A. L4 Build tree

If you follow example equally, L4 Build tree is “/path/to/obj/tudos”.

B. Fiasco.OC that was built

If you’ve built following example equally, it is “/path/to/obj/fiasco”

C. Application of L4/Fiasco.OC

1) Hello world

If you have built L4re , you can know that ‘hello’ binary was already made in “<L4 Build tree>/bin/arm_armv7a/l4f”

2) L4 Linux – How to Build it ?

```
$ cd l4linux //This folder is what you got at step 2-B-2).
$ mkdir /path/to/obj/l4linux
$ make L4ARCH=arm CROSS_COMPILE=arm-linux- O=/path/to/obj/l4linux arm_defconfig
$ make L4ARCH=arm CROSS_COMPILE=arm-linux- O=/path/to/obj/l4linux menuconfig

# In the menu, in "L4Linux configuration", set "L4 tree build directory" to
/path/to/obj/tudos and “Build type” in “System type” to ARMv7
# System type -> ARM system type (L4 Linux)
# Build type (Use ARMv7)

$ make L4ARCH=arm CROSS_COMPILE=arm-linux- O=/path/to/obj/l4linux
```

And then, you can make ‘uImage’ as follows..

D. How to make uImage which is “Hello world + L4/Fiasco”

1) Make uImage

```
$ cd /path/to/obj/tudos
$ make uimage E=hello MODULE_SEARCH_PATH=/path/to/obj/fiasco/
```

E. How to make uImage which is “L4 Linux + L4/Fiasco”

1) Get requisite files and put its in

“<L4 Build tree path>/bin/arm_armva/l4f/”

```
$ cd src/l4/conf/examples
$ sudo cp arm-rv.io <L4 Build tree path>/bin/arm_armva/l4f/
$ sudo cp l4lx.cfg <L4 Build tree path>/bin/arm_armva/l4f/

$ cd <Download folder>
$ sudo cp ramdisk-arm.rd <L4 Build tree path>/bin/arm_armva/l4f/

# you can change path to your l4linux build folder
$ cd /path/to/obj/l4linux
$ sudo cp vmlinuz.arm <L4 Build tree path>/bin/arm_armva/l4f/
```

2) Make uImage

```
$ cd /path/to/obj/tudos
$ make uimage E="L4Linux ARM" MODULE_SEARCH_PATH=/path/to/obj/fiasco/
```

* Check module.list !!

When uImage is built, module.list in ‘src/l4/conf/’ is used.
‘E=’ keyword mean ‘entry’ and uImage is made according to it’s standard.

You can look into the part which we are use in module.list file. Follows as..

```
modaddr 0x01100000

default-kernel fiasco -serial_esc
default-bootstrap bootstrap

entry L4Linux ARM
roottask moe rom/l4lx.cfg
module l4re
module ned
module l4lx.cfg
module io
module arm-rv.io
module vmlinuz.arm
module ramdisk-arm.rd
```

or

```
entry hello
roottask moe --init=rom/hello
module l4re
module hello
```

6. How to set up SD-card for booting

You can get ‘bootstrap.uimage’ from “/path/to/obj/tudos/bin/arm_armv7a/”

```
$ sudo fdisk /dev/sdb
$ sudo mkfs.ext4 /dev/sdb1

$ cd <the path to folder containing u-boot.bin>
$ sudo dd if=u-boot-mx6q-sabresd.bin of=/dev/sdb bs=512 seek=2 skip=2 conv=fsync

$ cd <L4 Build tree>/bin/arm_armv7a/
$ sudo dd if=bootstrap.uimage of=/dev/sdb bs=512 seek=2048 conv=fsync
```

7. How to use ‘minicom’.

A. Download minicom.

B. How to configure in minicom.

```
$ sudo minicom -s

# Select menu ‘Serial port setup’
# Modify ‘A -    Serial Device’  such as ‘/dev/ttyUSB0’
# Modify ‘F - Hardware Flow Control’ such as ‘No’

# So, the result is..

    +-----------------------------------------------------------------------+
    | A -    Serial Device      : /dev/ttyUSB0                              |
    | B - Lockfile Location     : /var/lock                                 |
    | C -   Callin Program      :                                           |
    | D -  Callout Program      :                                           |
    | E -    Bps/Par/Bits       : 115200 8N1                                |
    | F - Hardware Flow Control : No                                        |
    | G - Software Flow Control : No                                        |

$ sudo minicom

# After running minicom , connect your board to PC by your microUSBcable.
# And then, power on your board.
```

8. How to configure in U-Boot.

Assume that the kernel image starts from the address 0x100000 byte (the block start address is 0x800). The kernel image size is less than 0x900000 byte. Enter the following commands in the U-Boot prompt:

```
MX6Q SABRESD U-Boot > setenv loadaddr 0x10800000

MX6Q SABRESD U-Boot > setenv bootargs_base 'setenv bootargs
console=ttymxc0,115200'

MX6Q SABRESD U-Boot > setenv bootargs_mmc 'setenv bootargs ${bootargs}
root=/dev/mmcblk1p1 rootwait rw video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24'

MX6Q SABRESD U-Boot > setenv bootcmd_mmc 'run bootargs_base bootargs_mmc;mmc dev
2;mmc read ${loadaddr} 0x800 0x4400;bootm'

MX6Q SABRESD U-Boot > setenv bootcmd 'run bootcmd_mmc'

MX6Q SABRESD U-Boot > saveenv
MX6Q SABRESD U-Boot > run bootcmd
```

9. Thank you.

I have had help from L4 Hackers.
So, I was able to do booting “L4 Linux + L4/Fiasco.OC” on this board.
I’m particularly grateful to *Tim Komarov* and *Ademd Ammar* , *Adam Lackorzynski*, *Matthias Lange*.

I don’t know how much this document will help some beginner. But I share this document because I have had a big help from them.

10. If you've succeed everything, you will see this log.

Congratulations !!

```
U-Boot 2009.08 (Mar 21 2014 - 14:41:10)

CPU: Freescale i.MX6 family TO1.2 at 792 MHz
Thermal sensor with ratio = 180
Temperature:   41 C, calibration data 0x5764ba7d
mx6q pll1: 792MHz
mx6q pll2: 528MHz
mx6q pll3: 480MHz
mx6q pll8: 50MHz
ipg clock     : 66000000Hz
ipg per clock : 66000000Hz
uart clock    : 80000000Hz
cspi clock    : 60000000Hz

ahb clock     : 132000000Hz

axi clock   : 264000000Hz

emi_slow clock: 132000000Hz

ddr clock     : 528000000Hz

usdhc1 clock  : 198000000Hz

usdhc2 clock  : 198000000Hz

usdhc3 clock  : 198000000Hz

usdhc4 clock  : 198000000Hz

nfc clock     : 24000000Hz

Board: i.MX6Q-SABRESD: unknown-board Board: 0x63012 [POR ]

Boot Device: SD

I2C:   ready

DRAM:   1 GB

MMC:   FSL_USDHC: 0,FSL_USDHC: 1,FSL_USDHC: 2,FSL_USDHC: 3

In:    serial

Out:   serial

Err:   serial

i2c: I2C3 SDA is low, start i2c recovery...

I2C3 Recovery success

Found PFUZE100! deviceid=10,revid=11

Net:   got MAC address from IIM: 00:04:9f:02:70:b5

FEC0 [PRIME]

Hit any key to stop autoboot:  0

mmc2 is current device


MMC read: dev # 2, block # 2048, count 17408 ... 17408 blocks read: OK

## Booting kernel from Legacy Image at 10800000 ...
```

```
   Image Name:   L4 Image #3
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    8843264 Bytes =  8.4 MB
   Load Address: 11000000
   Entry Point:  11000000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK
## Transferring control to Linux (at address 11000000) ...

Starting kernel ...
L4 Bootstrapper
  Build: #3 2014年  3月 24日 月曜日 20:57:09 JST, 4.6.2 20110630 (prerelease)
  Scanning up to 1024 MB RAM
  Memory size is 1024MB (10000000 - 4fffffff)
  RAM: 0000000010000000 - 000000004fffffff: 1048576kB
  Total RAM: 1024MB
  mod09: 1156f000-1186f000: ramdisk-arm.rd
  mod08: 11225000-1156e4b0: vmlinuz.arm
  mod07: 11224000-112240bd: arm-rv.io
  mod06: 1110e000-11223738: io
  mod05: 1110d000-1110d189: l4lx.cfg
  mod04: 110ba000-1110cc08: ned
  mod03: 110a0000-110b9490: l4re
  mod02: 1106e000-1109f658: moe
  mod01: 11064000-1106d374: sigma0
  mod00: 11015000-11063278: fiasco
  Moving up to 10 modules behind 11100000
  moving module 00 { 11015000-11063277 } -> { 1195a000-119a8277 } [320120]
  moving module 01 { 11064000-1106d373 } -> { 119a9000-119b2373 } [37748]
  moving module 02 { 1106e000-1109f657 } -> { 119b3000-119e4657 } [202328]
  moving module 03 { 110a0000-110b948f } -> { 1186f000-1188848f } [103568]
  moving module 04 { 110ba000-1110cc07 } -> { 11889000-118dbc07 } [338952]
  moving module 05 { 1110d000-1110d188 } -> { 11100000-11100188 } [393]
  moving module 06 { 1110e000-11223737 } -> { 11101000-11216737 } [1136440]
  moving module 07 { 11224000-112240bc } -> { 11217000-112170bc } [189]
  moving module 08 { 11225000-1156e4af } -> { 11218000-115614af } [3445936]
```

```
  moving module 09 { 1156f000-1186efff } -> { 11562000-11861fff } [3145728]
  moving module 03 { 1186f000-1188848f } -> { 11862000-1187b48f } [103568]
  moving module 04 { 11889000-118dbc07 } -> { 1187c000-118cec07 } [338952]
  Scanning fiasco -serial_esc
  Scanning sigma0
  Scanning moe rom/l4lx.cfg
  Relocated mbi to [0x1100e000-0x1100e15d]
  Loading fiasco
  Loading sigma0
  Loading moe
  find kernel info page...
  found kernel info page at 0x10002000
Regions of list 'regions'
    [ 10001000,  10001bff] {      c00} Kern   fiasco
    [ 10002000,  1005afff] {    59000} Kern   fiasco
    [ 10090000,  1009637b] {     637c} Sigma0 sigma0
    [ 10098000,  1009e17b] {     617c} Sigma0 sigma0
    [ 10140000,  1016cdfb] {    2cdfc} Root   moe
    [ 10170000,  10186f0f] {    16f10} Root   moe
    [ 11000000,  110143f7] {    143f8} Boot   bootstrap
    [ 1100a000,  1100a121] {      122} Boot   mbi
    [ 1100e000,  1100e25a] {      25b} Root   Multiboot info
    [ 1101404c,  110140a3] {       58} Boot   mbi
    [ 11100000,  118cec07] {   7cec08} Root   Module
  API Version: (87) experimental
  Sigma0 config    ip:10090100 sp:11013de4
  Roottask config  ip:101401e0 sp:00000000
  Starting kernel fiasco at 10001314
Hello from Startup::stage2
Number of IRQs available at this GIC: 160
Cache config: ON
ID_PFR[01]:  00001231 00000011 ID_[DA]FR0: 00010444 00000000
ID_MMFR[04]: 00100103 20000000 01230000 00102111
L2: ID=410000c7 Type=9e340340 Aux=02070000 WMask=ffff S=0
L2: Type L2C-310 Size = 1024kB
SERIAL ESC: allocated IRQ 58 for serial uart
Not using serial hack in slow timer handler.
Welcome to Fiasco.OC (arm)!
```

```
L4/Fiasco.OC arm microkernel (C) 1998-2013 TU Dresden
Rev: r62 compiled with gcc 4.6.2 for i.MX6    []
Build: #1 2014��年  3��月 24��日 ��月���日 13:45:58 JST
Minicom2.6.2Calibrating timer loop... done.
MDB: use page size: 20
MDB: use page size: 12
SIGMA0: Hello!
  KIP @ 10002000
  allocated 4KB for maintenance structures
SIGMA0: Dump of all resource maps
RAM:------------------------
[0:10000000;10000fff]
[0:1005b000;1008ffff]
[0:10097000;10097fff]
[0:1009f000;1013ffff]
[4:10140000;1016cfff]
[0:1016d000;1016ffff]
[4:10170000;10186fff]
[0:10187000;1100dfff]
[4:1100e000;1100efff]
[0:1100f000;110fffff]
[4:11100000;118cefff]
[0:118cf000;4effffff]
IOMEM:----------------------
[0:0;ffffffff]
[0:50000000;ffffffff]
MOE: Hello world
MOE: found 1023504 KByte free memory
MOE: found RAM from 10000000 to 4f000000
MOE: allocated 1008 KByte for the page array @0x10187000
MOE: virtual user address space [0-bfffffff]
MOE: rom name space cap -> [C:501000]
  BOOTFS: [11862000-1187b490] [C:503000] l4re
  BOOTFS: [1187c000-118cec08] [C:504000] ned
  BOOTFS: [11100000-11100189] [C:505000] l4lx.cfg
  BOOTFS: [11101000-11216738] [C:506000] io
  BOOTFS: [11217000-112170bd] [C:507000] arm-rv.io
```

```
  BOOTFS: [11218000-115614b0] [C:508000] vmlinuz.arm

  BOOTFS: [11562000-11862000] [C:509000] ramdisk-arm.rd

MOE: cmdline: moe rom/l4lx.cfg

MOE: Starting: rom/ned rom/l4lx.cfg

MOE: loading 'rom/ned'

Ned says: Hi World!

Ned: loading file: 'rom/l4lx.cfg'

l4linux | libio: Warning: Query of 'vbus' failed!

l4linux | PH  0 (t:        1) offs=00008000 vaddr=02000000 vend=023b10ac

l4linux |                     phys=02000000 ephys=023b10ac

l4linux |                     f_sz=0032c3b6 memsz=003b10ac flgs=rwx

l4linux | PH  1 (t:        4) offs=002f0db4 vaddr=022e8db4 vend=022e8dd8

l4linux |                     phys=022e8db4 ephys=022e8dd8

l4linux |                     f_sz=00000024 memsz=00000024 flgs=r-x

l4linux | PH  2 (t: 1685382481) offs=00000000 vaddr=00000000 vend=00000000

l4linux |                     phys=00000000 ephys=00000000

l4linux |                     f_sz=00000000 memsz=00000000 flgs=rwx

l4linux | Starting binary at 0x20002a8, argc=7 argv=0x80007f8c *argv=0xb1007ff0 argv0=rom/vmlinuz.arm

l4linux | External resolver is at 0xa8000750

l4linux | ======> L4Linux starting... <========

l4linux | Linux version 3.13.0-l4-svn46 (root@taeung-MS-7798) (gcc version 4.6.2 20110630 (prerelease) (Freescale
MAD -- Linaro 24
l4linux | Binary name: rom/vmlinuz.arm

l4linux |    This is an AEABI build.

l4linux | Linux kernel command line (6 args): mem=64M console=ttyLv0 l4x_rd=rom/ramdisk-arm.rd root=1:0
ramdisk_size=4000 init=/bh
l4linux | CPU mapping (l:p)[1]: 0:0

l4linux | Image: 02000000 - 02400000 [4096 KiB].

l4linux | Areas: Text:     02000000 - 02306000 [3096kB] (a bit longer)

l4linux |        Data:     02306000 - 023278a0 [134kB]

l4linux |        Initdata: 022e9000 - 02305e1c [115kB]

l4linux |        BSS:      0232c3b6 - 023b10ac [531kB]

l4linux | Device scan:

l4linux | Device scan done.

l4linux | l4lx_thread_create: Created thread 413 (cpu0) (u:b3000e00, v:b3000c00, sp:02307fa4)

l4linux | main thread will be 413

l4linux | l4x_register_pointer_section: addr = 02000000 size = 3874816

l4linux | section-with-init: Virt: 0x2000000 to 0x23b10ab [3780 KiB]

l4linux | section-with-init: Phys: 0x1030a000 to 0x106bb0ab, [3780 KiB]

l4linux | Main thread running, waiting...
```

```
l4linux | L4x: Memory size: 64MB
l4linux | L4x: Setting superpages for main memory
l4linux | L4x: Adjusted memory start: 02000000
l4linux |     Main memory: Virt: 0x2400000 to 0x63fffff [65536 KiB]
l4linux |     Main memory: Phys: 0x11900000 to 0x158fffff, [65536 KiB]
l4linux | l4x: vmalloc area: 06400000 - 0e400000
l4linux | l4x_register_pointer_section: addr = 02000000 size = 3874816
l4linux |            text: Virt: 0x2000000 to 0x23b10ab [3780 KiB]
l4linux |            text: Phys: 0x1030a000 to 0x106bb0ab, [3780 KiB]
l4linux | l4x_rd_path: rom/ramdisk-arm.rd
l4linux | Loading: rom/ramdisk-arm.rd
l4linux | INITRD: Size of RAMdisk is 3072KiB
l4linux | RAMdisk from 00002000 to 00302000 [3072KiB]
l4linux | l4lx_thread_create: Created thread 418 (timer) (u:b3000a00, v:00000000, sp:02349fa0)
Booting Linux on physical CPU 0x0
Linux version 3.13.0-l4-svn46 (root@taeung-MS-7798) (gcc version 4.6.2 20110630 (prerelease) (Freescale MAD --
Linaro 2011.07 -- 4
CPU: Fiasco [412fc09a] revision 10 (ARMv7), cr=00000000
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction cache
Machine: L4
Memory policy: Data cache writeback
CPU: All CPU(s) started in SVC mode.
INITRD: 00002000 - 00302000
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 17209
Kernel command line: mem=64M console=ttyLv0 l4x_rd=rom/ramdisk-arm.rd root=1:0 ramdisk_size=4000 init=/bin/sh
PID hash table entries: 512 (order: -1, 2048 bytes)
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 64804K/69636K available (2327K kernel code, 134K rwdata, 652K rodata, 115K init, 531K bss, 4832K reserved)
Virtual kernel memory layout:
    vector  : 0xbffff000 - 0xc0000000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0x06400000 - 0x0e400000   ( 128 MB)
    lowmem  : 0x00000000 - 0x06400000   ( 100 MB)
    modules : Virtual kernel memory layout:
    vector  : 0xbffff000 - 0xc0000000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0x06400000 - 0x0e400000   ( 128 MB)
```

```
    lowmem  : 0x00000000 - 0x06400000   ( 100 MB)
    moNR_IRQS:220
l4timer: Using IRQ210
sched_clock: 32 bits at 1000kHz, resolution 1000ns, wraps every 2147483648000ns
Console: colour dummy device 80x30
console [ttyLv0] enabled
Calibrating delay loop... 1574.50 BogoMIPS (lpj=7872512)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x222f310 - 0x222f310
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
bio: create slab <bio-0> at 0
Failed to create "l4x" debugfs directory: 1
Switched to clocksource l4kipclk
NET: Registered protocol family 2
TCP established hash table entries: 1024 (order: 0, 4096 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP: reno registered
UDP hash table entries: 256 (order: 0, 4096 bytes)
UDP-Lite hash table entries: 256 (order: 0, 4096 bytes)
NET: Registered protocol family 1
Trying to unpack rootfs image as initramfs...
rootfs image is not initramfs (junk in compressed archive); looks like an initrd
INITRD: Freeing memory.
l4x: Checks passed.
NetWinder Floating Point Emulator V0.97 (double precision)
msgmni has been set to 126
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
L4 serial driver
ttyLv0 at MMIO 0x1 (irq = 211, base_baud = 230400) is a L4
l4ser_shm: L4 shared mem serial driver
l4cdds: No name given, not starting.
brd: module loaded
```

```
l4bdds: No name given, not starting.

mousedev: PS/2 mouse device common for all mice

TCP: cubic registered

NET: Registered protocol family 17

Minicom2.6.2RAMDISK: ext2 filesystem found at block 0

RAMDISK: Loading 3072KiB [1 disk] into ram disk... done.

EXT4-fs (ram0): couldn't mount as ext3 due to feature incompatibilities

EXT4-fs (ram0): mounting ext2 file system using the ext4 subsystem

EXT4-fs (ram0): mounted filesystem without journal. Opts: (null)

VFS: Mounted root (ext2 filesystem) readonly on device 1:0.

Freeing unused kernel memory: 112K (022e9000 - 02305000)

/bin/sh: can't access tty; job control turned off
/ # Minicom2.6.2
/ # ls
bin         etc         lost+found  sbin        tmp
dev         linuxrc     proc        sys         usr
```