GM8136

LINUX

User Guide
Rev.: 1.0
Issue Date: September 2014

REVISION HISTORY

GM8136 Linux User Guide

Date	Rev.	From	To
Sept. 2014	1.0	-	Original

Printed in Taiwan 2014

Grain Media, Inc.
5F, No. 5, Li-Hsin Road III, Hsinchu Science Park, Hsinchu City, Taiwan 300, R.O.C.

Grain Media's home page can be found at:
http://www.grain-media.com

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This chapter contains the following sections:

1.1 General Description

The hardware environment of GM8136 is a highly efficient RISC-based platform used to verify and evaluate the AMBA-based designs at the early development stage. This platform consists of a main board equipped with a GM8136 chip and an embedded FA6 CPU. This document is the Linux user guide for the GM8136 platform. Please refer to the GM8136 data sheet for further information.

Two Linux versions are involved in programming: **Linux** and **uClinux**. uClinux is a Linux version without the Memory Management Unit (MMU). RISC FA6 is an MMU-based CPU. Both FA6 and the Grand Media peripheral IP drivers have been ported to Linux. Grand Media provides the user guides for various IP drivers, in which the installation and usage of the drivers are described.

For further information regarding the USB device, Watchdog Timer (WDT), Real Time Clock (RTC), security engine, USB OTG, and other latest drivers, please refer to the related documents.

1.2 Requirements of Host Development Environment

The required development environments of the host system for developing Linux are as below:

Hardware:

- Intel x86 compatible PC
- Standard 16550 UART

Software:

- Standard Linux distribution (Fedora core 2.6.14-FC5 or above)
- FA6-based Linux distribution

1.3 Requirements of Common Platform Target System

The required target systems for developing the Linux kernel and device drivers are as below:

- GM8136:
  - 64MB onboard DDR
  - 8MB onboard Flash at least

Chapter 2

Linux Distribution Based on FA6

This chapter contains the following sections:

2.1 Introduction

This chapter introduces the architecture and implementation of the FA6-based Linux, which helps users easily and quickly understand and install the FA6-based Linux.

FA6-based Linux implements a Linux 3.3 software development environment for the FA6 processor and peripheral IPs. The information provided in this chapter will help users promptly install the FA6-based Linux on the GM8136 platform to implement the applications.

The following subsections include the information about the system requirements and how to install the FA6-Linux distribution.

2.1.1 FA6-Linux Distribution

The FA6-Linux distribution is a tar archive. An example of the file name is listed below:
arm-linux-3.3.tgz

Table 2-1. Disk Space Requirements for Linux Host after untar

Source Item	Disk Size
uClibc toolchain	150MB
Linux kernel	450MB
Ramdisk/Rootfs-cpio sample	2.5MB
Total	602.5MB

2.1.2 Install FA6-Linux Distribution

Install the FA6-Linux by extracting the tar archive with the sequence listed below:

1. Copy the file, "**arm-linux-3.3.tgz**", to the /usr/src directory by **# cp arm-linux-3.3.tgz /usr/src**
2. Extract the file, "**arm-linux-3.3.tgz**"
   # cd /usr/src
   # tar zxvf arm-linux-3.3.tgz

Install the ARM toolchain with the sequence listed below:

1. Login as "root user"
2. Copy "**uclibc_gnueabi-4.4.0_ARMv5TE.tgz**" to the **/usr/src/arm-linux-3.3** directory of the user PC Linux host system:
   "#cp uclibc_gnueabi-4.4.0_ARMv5TE.tgz /usr/src/arm-linux-3.3"
3. Decompress the toolchain:
   "#tar xvfz uclibc_gnueabi-4.4.0_ARMv5TE.tgz"
   into /usr/src/arm-linux-3.3
4. Include the toolchain path in the user path:
   The compile path is directly specified in "menu config". Users do not need to specify in the environment. This path is:
   "/usr/src/arm-linux-3.3/toolchain_gnueabi-4.4.0_ARMv5TE/usr/bin".
5. Test run the user toolchain:
   "#/usr/src/arm-linux-3.3/toolchain_gnueabi-4.4.0_ARMv5TE/usr/bin/arm-unknown-linux-uclibcgnueabi-gcc -v"

If the installation is correct, users will see the following messages:

```
Using built-in specs.
Configured with: … (some information)
Thread model: posix
gcc version 4.4.0 20100318 (experimental) (Buildroot 2012.02)
```

After completing the installation steps, users can build the FA6-Linux kernel or application.

2.1.3 Directory Structure of FA6-Linux

Assume that the top directory is **/usr/src/arm-linux-3.3/** when extracting the tar archive, such as "**arm-linux-3.3.tgz**", a set of subdirectories will be built on **/usr/src/arm-linux-3.3/**.

Table 2-2 lists the source directories and the files located in it. For example, if the user application is located in the **/usr/src/arm-linux-3.3/user/** directory, the customized ramdisk will be located in the **/usr/src/arm-linux-3.3/target/rootfs-cpio** directory; and the module driver will be located in the **/usr/src/arm-linux-3.3/module/** directory.

Please extract the target 8136.tgz first to establish the rootfs directory. Otherwise, an error message will pop when building the Linux kernel.

Table 2-2. FA6 Distribution Directories

Directory	Description
uclibc_gnueabi-4.4.0_ARMv5TE.tgz	Directory for the repository of the uClibc toolchain
rootfs.tgz	root filesystem
linux-3.3-fa/	Directory for the repository of the Linux kernel source
user/	Directory for the repository of the user application
target/rootfs-cpio	Directory for the repository of the ramdisk images
module/	Directory for the Linux 3.3 module
u-boot-2013.01	Directory for the ARM boot
nsboot	Directory for the SPI/NAND boot loader

Note: The toolchain includes GCC4.4.0, uClibc-0.9.33.2, and Binutil 2.19.

2.2 Build and Develop FA6-Linux

This section introduces how to configure and build an FA6-Linux kernel for the embedded system.

2.2.1 Kernel Tree

The Linux kernel is located in the following directory:

/usr/src/arm-linux-3.3/linux-3.3-fa/

The structure of the FA6-Linux subdirectory is identical to that of the standard Linux kernel, version 3.3. Table 2-3 lists and describes the Linux kernel subdirectory. **<TOPDIR>** stands for the **/usr/src/arm-linux-3.3/linux-3.3-fa** directory.

Table 2-3. Linux Kernel Subdirectory

Kernel Subdirectory	Description
<TOPDIR>/arch/arm	Architecture-dependent code for the ARM processors
<TOPDIR>/Documentation	Documentations for the Linux kernel
<TOPDIR>/drivers	Device drivers, which are divided into various subdirectories.
<TOPDIR>/fs	Various file systems supported by the Linux kernel
<TOPDIR>/include	Kernel head files
<TOPDIR>/init	Kernel startup functions
<TOPDIR>/ipc	Sources of system V IPC
<TOPDIR>/kernel	Kernel core sources
<TOPDIR>/lib	Standard C library sources
<TOPDIR>/mm	Kernel memory management
<TOPDIR>/net	Implementations of various network protocols

2.2.2 Build Kernel

This section introduces how to build the kernel image for the FA6 architecture. In addition to the Linux kernel, rootfs contains most of the Grain Media drivers and Grain Media libraries for the called user applications. Before compiling the kernel source code, users should first untar the corresponding rootfs.

2.2.2.1 Root File System

Grain Media prepares the root file system (Shorted as "rootfs"). Users should untar rootfs.tgz to arm-linux-3.3/target/rootfs-cpio and build the Linux image. Users can use the squash file system for the rootfs type selection.

As mentioned above, Linux uses the squash file system. Thus, users must use the **mksquash** tool to produce the squash rootfs. The source code is placed in arm-linux-3.3/user/squashfs4.2.tar.gz.
Users can put AP to arm-linux-3.3/target/rootfs-cpio/gm/bin and put only Grain Media AP to arm-linux-3.3/target/rootfs-cpio/gm/tools. If users do not need these files, these files can be deleted.
After modifying the root file system, users can enter arm-linux-3.3/target/ to use the **mksquash** tool. The usage is:
./mksquashfs rootfs-cpio rootfs-cpio.squashfs.img -comp xz

Or users can directly execute the script file:
./ mkrootfs.sh

Before booting Linux, users should first copy the built rootfs, rootfs-cpio.squashfs.img, to /dev/mtd4. When booting FA6, it will load rootfs from the mtd partition. For the firmware upgrade, please refer to the Flash user guide.

2.2.2.2 Configure Kernel

The first step to build the Linux kernel is to configure the kernel. The configuration file is located in **<TOPDIR>/.config** for the standard Linux.

In general, the purposes of reconfiguring FA6-Linux are summarized below:

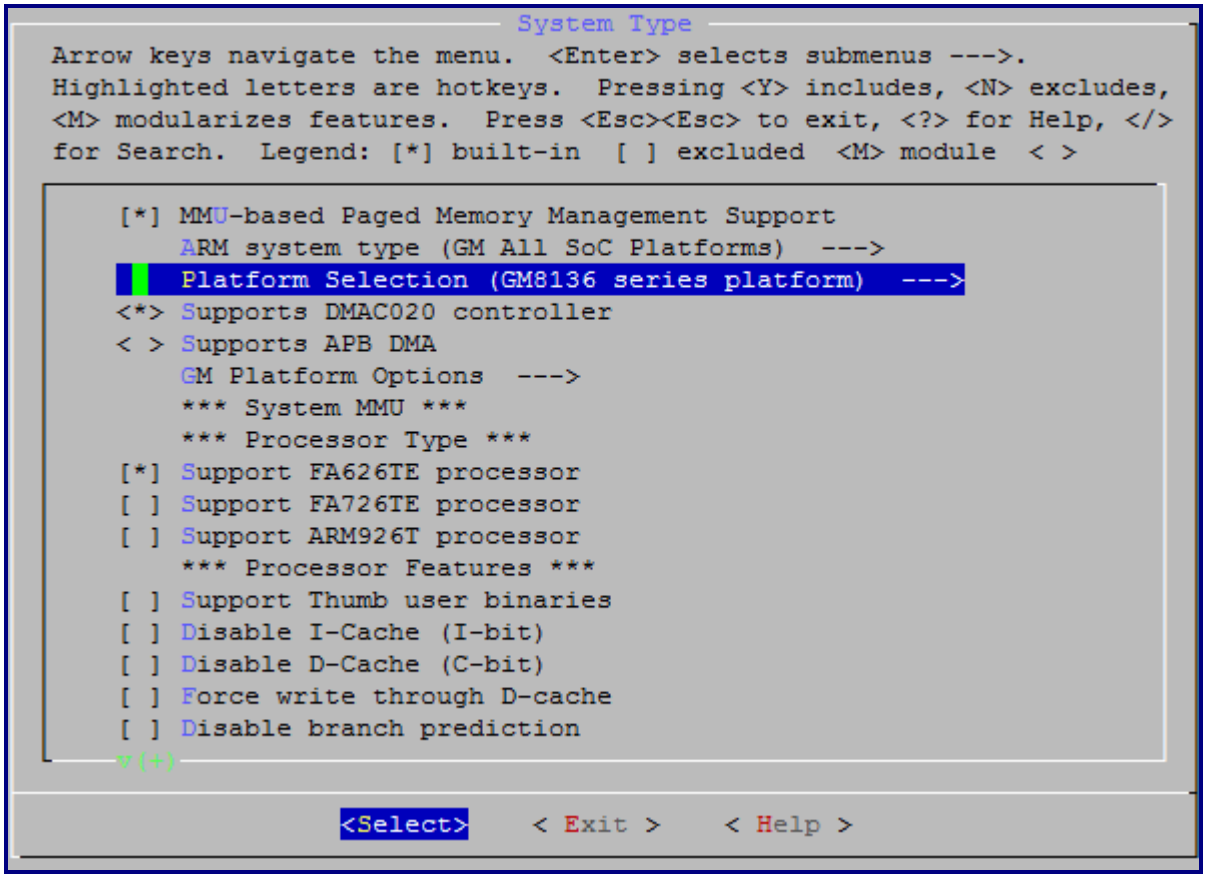
- Customize the processor and board functionalities: Modify the UART clock, system clock, and so on.
- Customize the hardware devices: Add or remove a particular device
- Customize the kernel functionalities: Add or remove a kernel feature, such as the network support

Users can copy "linux-3.3-fa/arch/arm/configs/GM8136_defconfig" to update ".config" and disable items that are not required. This file includes all necessary items. Users should decide the required function, such as the device drivers.

There are two ways for Linux kernel to configure the options as mentioned above:

- Change to the <TOPDIR> directory
  1. Use the menu display to select the options for configuration:
     # make menuconfig
  2. Use the GUI display to select the options for configuration:
     # make xconfig

Figure 2-1 shows how to use the console to configure the Linux kernel options.



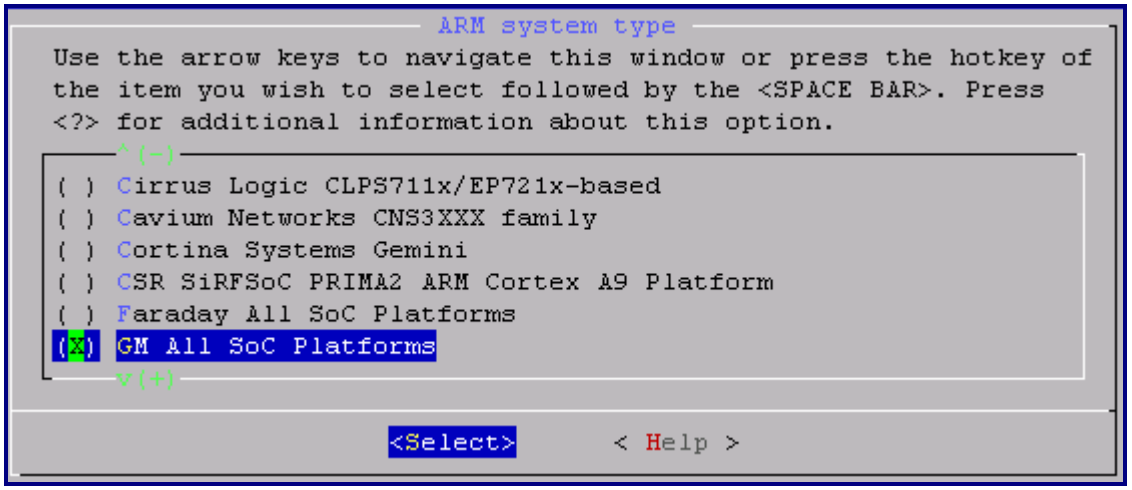


Figure 2-1. Linux Kernel Option Configuration by Using Console

Figure 2-2 shows how to use the menu display to select the configuration options of Linux kernel.

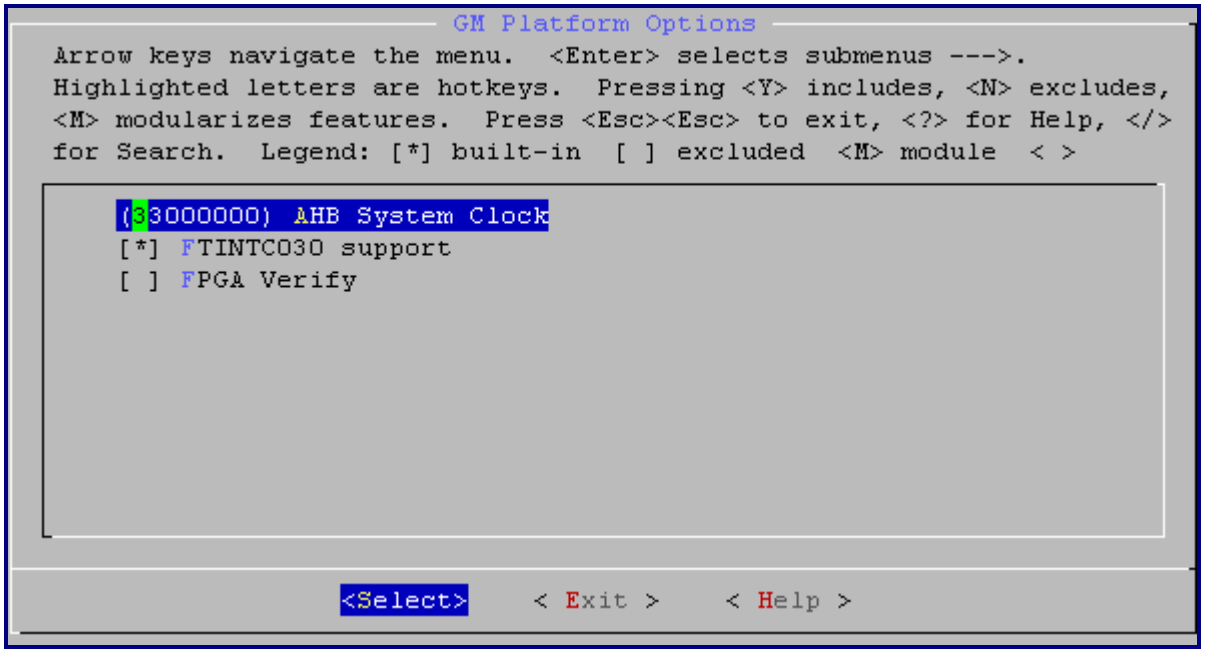
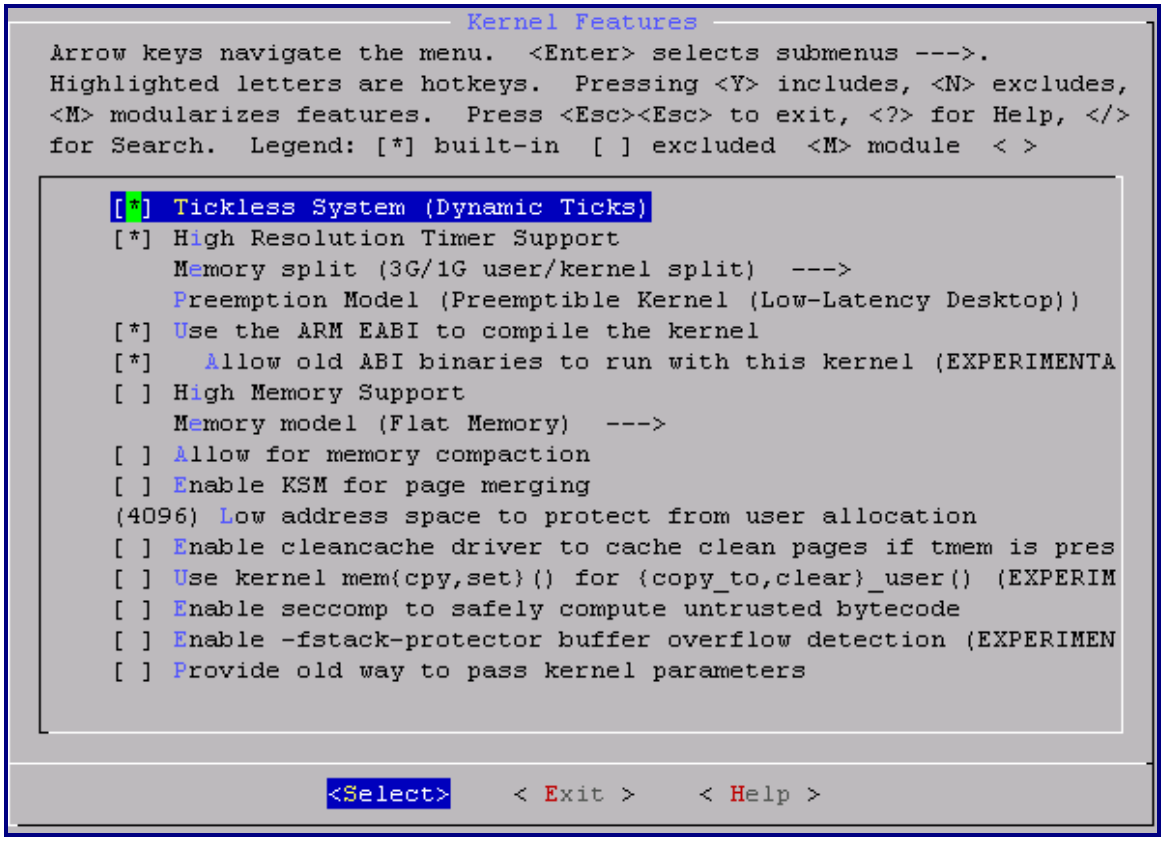


Figure 2-2. Linux Kernel Option Configuration by Using Menu Display

Note: Any fake value in the AHB System Clock is acceptable; it is almost obsolete.

Before compiling the kernel, users need to select the EABI options in the kernel configuration menu if the gcc-4.4.0 tool chain is used. The configuration path is "Kernel configuration Menu → Kernel Features → ARM EABI", as shown below.



Please refer to the FA6 data sheet for the detailed CPU functions.

2.2.2.3 Make Kernel

To make a Linux kernel, users need to clean all object files and recreate the dependency. The following is an example:

 # make clean

FA6-Linux provides the shell script, "build", for users to easily make the kernel.

 # ./build

If users want to use the U-BOOT transfer argument to Linux, please run the following shell script:

 # ./build_uImage_8136

If users do not want to use the U-BOOT transfer argument to Linux, please run the **./build** shell script in the linux-3.3-fa directory.

Notes:

1. build_uImage_8136 is only for the FA6 CPU core. For the FA6 CPU core, please run the **./build** shell script. For the explanation of build_uImage_8136, please refer to the U-BOOT user guide.
2. As mentioned before, please extract rootfs-cpio_8136.tgz first to the /usr/src/arm-linux-3.3/target directory.

It creates the final kernel image, **mbootpImage,** uImage, and the kernel ELF file, **vmlinux**. Users may modify "build" for the following reasons:

- Copy the output image to a specified directory: Modify the command, "cp <source> <target>", in the **build** file to meet the requirements.
- If users want to use U-BOOT to build the Linux image, the default address will be 0x20000000.

Note: This value may be changed due to the memory map change. This address means that kernel will execute from this address, which contains the uImage header.

```
[root@Harry linux-3.3-fa]# cat build_uImage_8136
rm -f ./usr/initramfs_data.cpio.gz
make
sudo cp arch/arm/boot/zImage /tftpboot/mbootpImage
sudo cp -f System.map /tftpboot/
sudo cp -f vmlinux /tftpboot/
sudo chown nobody:nobody /tftpboot/mbootpImage
sudo ./mkimage -A arm -O linux -T kernel -C none -a 2000000 -e 2000040 -n gm8136 -d
arch/arm/boot/zImage arch/arm/boot/uImage
sudo cp arch/arm/boot/uImage /tftpboot/uImage_8136
sudo chown nobody:nobody /tftpboot/uImage_8136
```

Figure 2-3. File Content of "build"

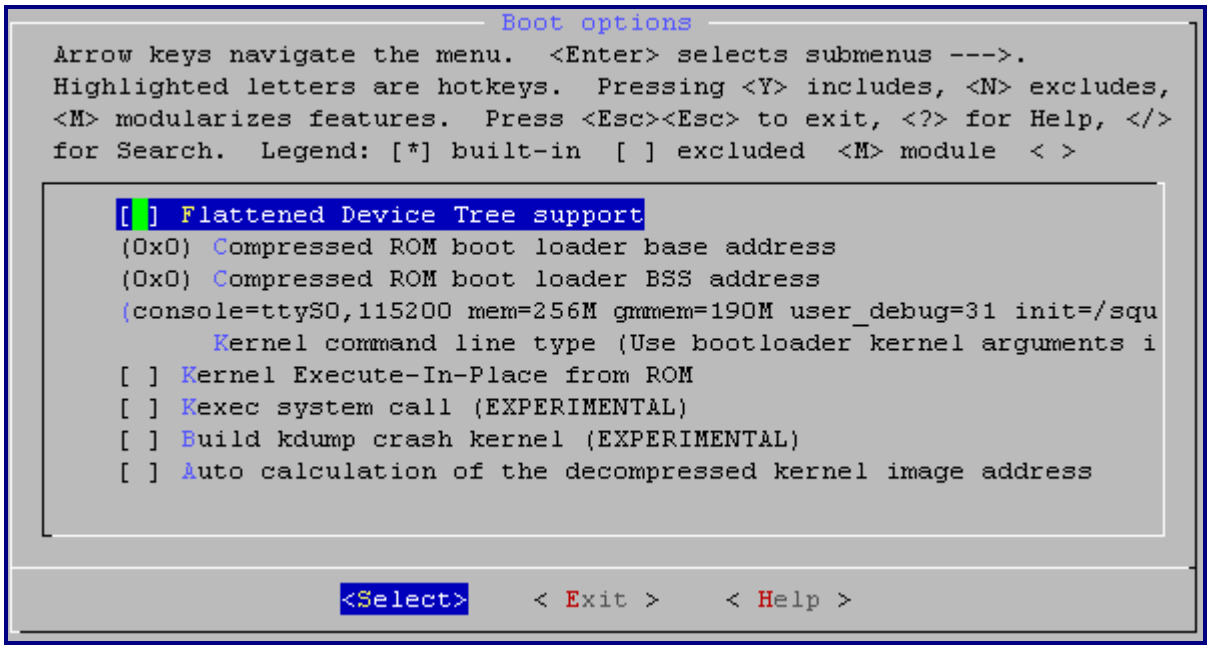


Figure 2-4. Configuring Boot Options

In this example, the boot command: console=ttyS0, 115200 mem=256M gmmem=190M user_debug=31 init=/squashfs_init root=/dev/mtdblock2 rootfstype=squashfs , may be changed due to the real memory map.

2.2.3 Builde U-BOOT

U-BOOT is used by FA6-Linux as an OS loader.

2.2.3.1 Configure U-BOOT

The FA6 U-BOOT maintains the configuration file, *GM.h*, to configure different hardware environments. The *GM.h* file is located at: **/usr/src/arm-linux-3.3/u-boot-2013.01/include/configs/GM8136.h**. Users can modify this file based on the circumstances.

In addition, users can modify the MAC address to download the Linux code to the MediaCreative platform. At this stage, users should modify the definition, CONFIG_ETHADDR, of the MAC address in **u-boot-2013.01/include/configs/GM8136.h** (As shown in Figure 2-5) and the variable, ftgmac100_mac_addr, in **u-boot-2013.01/driver/net/ftgmac100.c** (As shown in Figure 2-6).

```
#define CONFIG_ETHADDR                  00:42:70:00:30:22
#define CONFIG_NETMASK                  255.0.0.0
#define CONFIG_IPADDR                   10.0.1.52
#define CONFIG_SERVERIP                 10.0.1.51
#define CONFIG_GATEWAYIP                10.0.1.51
```

Figure 2-5. Modify MAC and IP for U-BOOT in config_GM8136.h

```
static char ftgmac100_mac_addr[] = {0x00, 0x42, 0x70, 0x00, 0x30, 0x52};
```

Figure 2-6. Modify MAC for U-BOOT in ftgmac100.c

2.2.3.2 Make U-BOOT

Once GMAC and IP are modified, users can build U-BOOT with the following commands:

 # cd /usr/src/arm-linux-3.3/u-boot-2013.01

 # ./make_8136

The above two commands will create the file, **u-boot.bin**, in the folder shown in the first line. Users should follow the instruction to burn U-BOOT into the GM8136 Flash and write the specific image, **u-boot.bin**, to the Flash address 0x140000 (For the NAND system). However, to upgrade SPI NOR or NAND Flash, the PCTOOL or SPI/NAND command should be used. Please refer to the Flash user guide for programming the SPI NOR or NAND Flash.

2.3 FA6-Linux OS Loader – U-BOOT

U-BOOT is a well-known OS loader in the Linux world for its capability of loading images from a terminal protocol (Such as **Kermit**) and booting the Linux kernel. It provides the Flash utilities and Ethernet TFTP transfer functions. Only three CPU cores in GM8136 and U-BOOT can serve the FA6-Linux image.

2.3.1 Run U-BOOT

The FA6-Linux distribution package provides the **U-BOOT** code to perform the following tasks:

- Program Flash
- Transfer data from PC to the target by UART (Kermit) or Ethernet (TFTP, please use the GMAC0 port)
- Load or branching the Linux kernel

Note: Users can run the **U-BOOT** code from Flash or via ICE.

2.3.1.1 Run U-BOOT from Flash

If the boot code (nsboot.bin) and the U-BOOT code are ready in a Flash, users can run U-BOOT from Flash by the keystroke, "ESC", at the reset time.

2.3.1.2 Run U-BOOT via ICE

Users may run U-BOOT via ICE by following the procedure below:

1. Connect the FA6 target to the user PC with JTAG ICE
2. Open the OPENice debugger and load **u-boot.bin** to the memory address, 0x0
3. Set PC to 0x0 and run

2.3.2 U-BOOT Environment Variables

U-BOOT maintains a number of environment variables for various functions. These environment variables can be displayed by using the following command:

=> printenv

Users can set the environment values by using the "setenv name value" command, where "name" denotes the name of the environment variable and "value" denotes the value to be set. The following command is an example of setting the IP address environment:

=> setenv ipaddr 192.168.68.48

Table 2-4 lists the environment variables used in this document.

Table 2-4. Environment Variables

Environment Variable	Description
Ipaddr	Target IP address
serverip	IP address of the TFTP server
ethaddr	MAC address value
netmask	Netmask value of the IP address

2.3.3 U-BOOT Command Reference

Some commonly used commands in U-BOOT are listed in Table 2-5. Users can type "help" on the U-BOOT terminal to display the command list.

Table 2-5. Commonly Used U-BOOT Commands

Command	Description
printenv	Print the environment variables of U-BOOT
setenv [env] [value]	Set the environment variable
go [address]	Jump to specific address, for zImage
bootm [address]	Jump to specific address, for uImage
tftp [address] [image]	TFTP transfer of images to specific address

2.4 Boot FA6-Linux

FA6-Linux can be booted on the target system by using one of the following two booting scenarios:

- Boot FA6-Linux via ICE
- Boot FA6-Linux from Flash

2.4.1 Boot FA6-Linux via ICE

Users can load the FA6-Linux kernel image to specific address via ICE and jump. The procedure is listed below:

1. Load **mbootpImage** to the address, 0x2000000 (Recommended), by the OPENice debugger (Please refer to Appendix B of this debuffer for the procedure.)
2. Set PC to **0x2000000** on the OPENice debugger and run
3. Before running Linux, the system should have completed the initialization.

2.4.2 Boot FA6-Linux from Flash

The Linux kernel can be automatically booted by users from Flash. To boot the Linux kernel from Flash, the specified boot code and U-BOOT code are required. Please follow the steps below to automatically boot FA6-Linux from Flash (In the case of 16-bit bus width):

1. Prepare the boot code: Write specific images, **nsboot.bin** and **rom.bin** (Optional), to Flash and the file name depends on the real image name.
2. Prepare the U-BOOT code: Write specific image, **u-boot.bin** (The file name depends on the real image name), to the Flash address by PCTOOL (If the system is SPI NOR or NAND, please use PCTOOL or other method to program Flash. Please refer to the Flash user guide).
3. Prepare the Linux kernel image: Write specific image, **mbootpImage**, to the Flash address and the address definition is not listed. If the system is SPI NOR or NAND, please use PCTOOL or other method to program Flash. Please refer to the Flash user guide.
4. For rootfs, currently, squashFS is used. Thus, rootfs must be updated by PCTOOL. Because when Linux boots up, it will find squashFS from Flash and mout it automatically. If Linux cannot find squashFS in Flash, the system will crash.
5. For the rootfs selection, users should select squashFS for using less memory.
6. Figure 2-7 is an example of PCTOOL for upgrading the images.

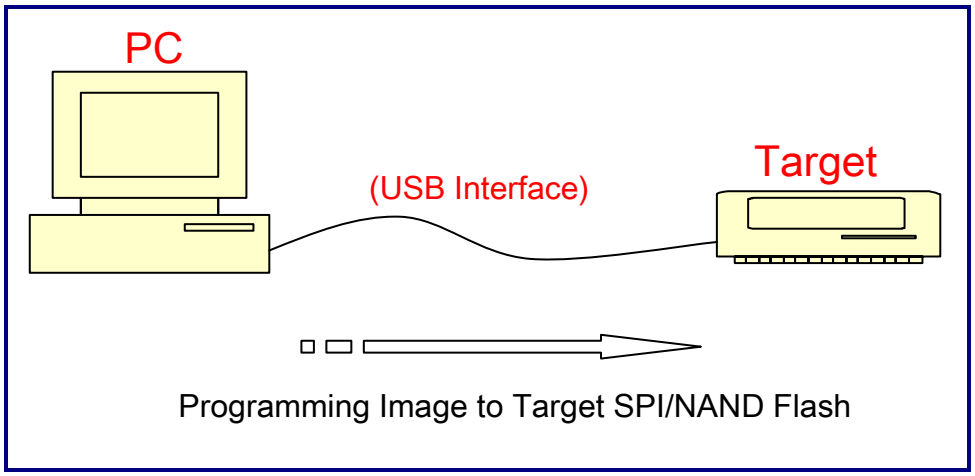


Figure 2-7. Upgrade Images by PCTOOL

2.4.3 Boot FA6-Linux by U-BOOT

At most parts of the development stage, users need to perform the iterations to modify the codes and download the codes until the results are satisfied. Under such circumstances, users need U-BOOT to download and run the code. The required procedure is as follows (Note: This example is only for the FA6 CPU core):

1. Set the tftp server on the user Linux host and /etc/xinetd.d/tftp, as shown in Figure 2-8.

```
Service tftp
{
    disable          = no
    socket_type      = dgram
    protocol         = udp
    wait             = yes
    user             = root
    server           = /usr/sbin/in.tftpd
    server_args      = -c -u nobody -s /tftpboot
    per_source       = 11
    cps              = 100 2
}
```

Figure 2-8. tftp Setting

2. Use the Linux making shell (build) in this package to compile the code, generate the Linux code, and place it to the /tftpboot folder. To run the Linux code, users should follow the steps below:
    1. Reset the GM8136 target
    2. Select the "ESC" item to enter U-BOOT
    3. Make sure that the IP addresses of the Linux host and GM8136 are correct (printenv), as shown in Figure 2-9.
    4. Type the **"tftp 0x2000000 mbootpImage"** command to download the code, as shown in Figure 2-10.
    5. Boot up Linux by using the **"go 0x2000000"** command. Users can see the Linux boot-up message on the screen, as shown in Figure 2-11.

6. Please be aware that CPU uses squqshFS as the rootfs, thus its rootfs cannot be downloaded through tftp. Instead, users should use flashcp or nandwrite command mentioned in the Flash user guide to burn it into Flash first. That is, while CPU is booting, its Linux kernel can find the rootfs indicated by the boot command line from Flash. If the rootfs is found, it will mount its rootfs as a squash filesystem.

```
GM # printenv
baudrate=115200
bootcmd=sf probe 0:0;run lm;bootm 0x2000000
bootdelay=3
cmd1=mem=64M gmmem=30M console=ttyS0,115200 user_debug=31 init=/squashfs_init
root=/dev/mtdblock2 rootfstype=squashfs
cmd2=mem=128M gmmem=90M console=ttyS0,115200 user_debug=31 init=/squashfs_init
root=/dev/mtdblock2 rootfstype=squashfs
cmd3=mem=256M gmmem=190M console=ttyS0,115200 user_debug=31 init=/squashfs_init
root=/dev/mtdblock2 rootfstype=squashfs
ethact=eth0
ethaddr=00:42:70:00:30:22
gatewayip=10.0.1.51
ipaddr=10.0.1.52
lm=sf read 0x02000000 z
netmask=255.0.0.0
serverip=10.0.1.51
stderr=serial
stdin=serial
stdout=serial

Environment size: 639/65532 bytes
```

Figure 2-9. Environment Settings of U-BOOT

```
8136
U-Boot 2013.01 (Aug 22 2013 - 11:26:26)

Warning: Only 256 of 512 MiB SDRAM is working
DRAM:  256 MiB
ROM CODE has enable I cache
```

```
SPI mode
SPI020 Revision:0x10001
SF: Got idcodes
00000000: ef 40 18 00    .@..


SF: Detected W25Q128 with page size 64 KiB, total 16 MiB
winbond chip
flash is 3byte mode


*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial

--------------------------------
ID: 8136100
AC: 200  HC: 200  P1: 810  P2: 600  P3: 540
C6: 810  DR:1080
J: 270   H1: 300
--------------------------------
Net:   GMAC set RMII modereset PHY
eth0
Hit any key to stop autoboot:  0
GM # setenv serverip 192.168.68.192;setenv ipaddr 192.168.68.191;tftp 0x2000000 mb
ootpImage;go 0x2000000
FULL
PHY_SPEED_100M
Using eth0 device
TFTP from server 192.168.68.192; our IP address is 192.168.68.191
Filename 'mbootpImage'.
Load address: 0x2000000
Loading: t RD_REQ, file: mbootpImage
#################################################################
         #################################################################
         #################################################################
```

```
         #########################
done
Bytes transferred = 3206240 (30ec60 hex)
## Starting application at 0x02000000 ...
```

Figure 2-10. Download Linux Code from U-BOOT

```
GM # bootm 0x2000000
## Booting kernel from Legacy Image at 02000000 ...
   Image Name:   gm8136
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    4253456 Bytes = 4.1 MiB
   Load Address: 02000000
   Entry Point:  02000040
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0
Linux version 3.3.0 (root@ftclab1) (gcc version 4.4.0 20100318 (experimental) (Buildroot
2012.02) ) #50 PREEMPT Fri Aug 29 11:08:51 CST 2014
CPU: FA6 [66056263] revision 3 (ARMv5TE), cr=0000397f
CPU VIPT aliasing data cache, unknown instruction cache
Machine: Grain-Media GM8136 series
Memory policy: ECC disabled, Data cache writeback
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 65024
Kernel command line: mem=256M gmmem=190M console=ttyS0,115200 user_debug=31
init=/squashfs_init root=/dev/mtdblock2 rootfstype=squashfs
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 256MB = 256MB total
Memory: 255588k/255588k available, 6556k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
```

```
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0x90800000 - 0xff000000   (1768 MB)
    lowmem  : 0x80000000 - 0x90000000   ( 256 MB)
    modules : 0x7f000000 - 0x80000000   (  16 MB)
      .text : 0x80008000 - 0x803d9858   (3911 kB)
      .init : 0x803da000 - 0x803f3000   ( 100 kB)
      .data : 0x803f4000 - 0x80411fe0   ( 120 kB)
       .bss : 0x80412004 - 0x80429f5c   (  96 kB)
NR_IRQS:64
gm_jiffies_init, system HZ: 100, pClk: 100000000
console [ttyS0] enabled
Calibrating delay loop... 806.91 BogoMIPS (lpj=4034560)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x2e2388 - 0x2e23d0
devtmpfs: initialized
FMEM: 48640 pages(0xbe00000 bytes) from bank0 are reserved for Frammap.
FMEM:    Logical    memory    ends    up    at    0x90000000,    init_mm:0x80004000(0x4000),
PAGE_OFFSET:0x80000000(0x0),
FMEM: FA726 Test and Debug Register: 0x0
NET: Registered protocol family 16
PMU: Mapped at 0xfe000000
IC: GM8136, version: 0x1
iotable: VA: 0xfe000000, PA: 0x90c00000, Length: 4096
iotable: VA: 0xfe001000, PA: 0x90700000, Length: 4096
iotable: VA: 0xfe002000, PA: 0x90800000, Length: 4096
iotable: VA: 0xfe003000, PA: 0x90900000, Length: 4096
iotable: VA: 0xfe004000, PA: 0x90d00000, Length: 4096
iotable: VA: 0xfe005000, PA: 0x96000000, Length: 4096
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
Switching to clocksource fttmr010:1
NET: Registered protocol family 2
IP route cache hash table entries: 2048 (order: 1, 8192 bytes)
```

```
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 3, 32768 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
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
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
Video Timer(timer3) Max 42000ms in 0xfa56ea00 HZ.
ftdmac020 ftdmac020.0: DMA engine driver: irq 1, mapped at 0x90804000
GM CPU frequency driver
CPUFREQ support for gm initialized
squashfs: version 4.0 (2009/01/31) Phillip Lougher
JFFS2 version 2.2. (NAND) ? 2001-2006 Red Hat, Inc.
msgmni has been set to 499
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
gpiochip_add: registered GPIOs 0 to 31 on device: ftgpio010.0
probe ftgpio010.0 OK, at 0x90862000
gpiochip_add: registered GPIOs 32 to 63 on device: ftgpio010.1
probe ftgpio010.1 OK, at 0x90864000
Serial: 8250/16550 driver, 3 ports, IRQ sharing disabled
serial8250: ttyS0 at I/O 0xfe001000 (irq = 21) is a 16550A
serial8250: ttyS1 at I/O 0xfe002000 (irq = 22) is a 16550A
serial8250: ttyS2 at I/O 0xfe003000 (irq = 25) is a 16550A
brd: module loaded
loop: module loaded
Not for SPI-NAND pin mux
SPI020 init
SPI020 uses AHB DMA mode
FTSPI020 enable DMA handshake 0x3
SPI020 gets DMA channel 0
ftspi020 ftspi020.0: Faraday FTSPI020 Controller at 0x92300000(0x90866000) irq 54.
spi spi0.0: setup: bpw 8 mode 0
```

```
CLK div field set 1
ERASE SECTOR 64K
SPI_FLASH spi0.0: w25q128bv (16384 Kbytes)
Creating 6 MTD partitions on "nor-flash":
0x000000010000-0x000000060000 : "UBOOT"
0x000000060000-0x000000400000 : "LINUX"
0x000000400000-0x000000600000 : "FS"
0x000000600000-0x000000700000 : "USER0"
0x000000700000-0x000000800000 : "USER1"
0x000000000000-0x000001000000 : "ALL"
Probe FTSPI020 SPI Controller at 0x92300000 (irq 54)
ftgmac: Loading version 0.1 ...
ftgmac100-0-mdio: probed
ftgmac100-0 ftgmac100-0.0: eth0: 1 tx queue used (max: 2)
ftgmac100-0 ftgmac100-0.0: eth0: 1 rx queue used (max: 1)
ftgmac100-0 ftgmac100-0.0: eth0: irq 3, mapped at 90868000
ftgmac100-0 ftgmac100-0.0: eth0: generated random MAC address 3e:ab:28:fa:fb:61
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
FOTG2XX Controller Initialization
Enter Device A
Drive Vbus because of ID pin shows Device A
fotg210 fotg210.0: FOTG2XX
fotg210 fotg210.0: new USB bus registered, assigned bus number 1
fotg210 fotg210.0: irq 9, io mem 0x93000000
fotg210 fotg210.0: USB 2.0 started, EHCI 1.00
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 1 port detected
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
i2c /dev entries driver
ftiic010 ftiic010.0: irq 18, mapped at 9086c000
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
mmc0: SDHCI controller on ftsdc021.0 [ftsdc021.0] using ADMA
sdhci-pltfm: SDHCI platform and OF driver helper
TCP cubic registered
NET: Registered protocol family 17
```

```
VFS: Mounted root (squashfs filesystem) readonly on device 31:2.
devtmpfs: mounted
Freeing init memory: 100K
busybox: /linuxrc: Read-only file system
Mounting root fs rw ...
Mounting other filesystems ...
Setting hostname ...
Bringing up interfaces ...
/bin/sh: run-parts: not found
Mounting user's MTD partion
Frammap: DDR0: memory base=0x3800000, memory size=0xbe00000, align_size = 4K.
Frammap: version 1.1.2, and the system has 1 DDR.
Frammap: fail to open /mnt/mtd/config_8139, /tmp/template_8139 is created.
----------------------------------------------------------------
ddr name: frammap0
base: 0x3800000
end: 0xf600000
size: 0xbe00000 bytes
memory allocated: 0x0 bytes
memory free: 0xbe00000 bytes
max available slice: 0xbe00000 bytes
memory allocate count: 0
clear address: 0x3800000
dirty pages: 0
clear pages: 48640
size alignment: 0x1000

   Press q -> ENTER to exit boot procedure? q


BusyBox v1.19.4 (2013-09-04 16:29:58 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ #
```

Figure 2-11. Linux Message during Booting-up

For booting the CPU core, the command line is:
setenv ipaddr 192.168.68.84;setenv serverip 192.168.68.234;tftp 0x2000000 mbootpImage; go 0x2000000;

Note: The above example is only for reference. Please change the settings according to the real environment.

In above booting message, "`Kernel command line: console=ttyS0, 115200 mem=256M gmmem=190M user_debug=31 init=/squashfs_init root=/dev/mtdblock2 rootfstype=squashfs`", the following is the brief description:

mem=256M: It means that the system has DDR0 with a size of 256Mbyte.
gmmem=190M: 190Mbytes memory will be reserved for Frammap usage from DDR0 during the system initialization. The system finally only has 256Mbytes ~ 190Mbytes left.
User_debug=31: User application debugging level. When the user application gets problem, such as crash, Linux kernel will dump some useful messages for debugging according to this level.
Init: First executed script file from rootfs
root: Which MTD partition the rootfs resides on
rootfstype: When kernel and rootfs are separated, the value indicates its root filesystem type.

2.5 FA6-Linux Internals

This section introduces the implementation of the FA6 Linux. The memory splits in the kernel space and user space is 2G/2G.

2.5.1 I/O Address Mapping

Because FA6-Linux uses the MMU-based Linux kernel, it requires the I/O memory mapping to access the I/O peripheral. Most of the I/O addresses of common IP mapping on FA6-Linux are defined in "**<TOPDIR>/**arch/arm/mach-GM/include/mach/platform-GM8136/platform_io.h", such as the GPIO, DDR controller, DMA controller, and interrupt controller.

Users may modify this file when:

- The peripheral device has been added or removed.
- The physical I/O address or size of the device has been changed.

2.5.2 Reserved Memory

Usually, the module drivers require large and contiguous physical memory. However, in the embedded system, it is difficult to prevent the memory from fragmentation. For this purpose, Grand Media has pre-allocated a large memory space and uses particular mechanisms to manage the memory. The memory manager of Grand Media is called "Frammap". Users can see the detailed behavior in the Frammap User Guide. The reserved memory size for the Frammap memory manager is defined in **<TOPDIR>/**arch/arm/mach-GM/include/mach/platform-GM8136/ memoy.h, which defines the size reserved from DDR for Frammap.

In *memory.h*, users will see:

```
#define DDR0_FMEM_SIZE     0x1000000
#define DDR1_FMEM_SIZE         -1
```

The above definition means that the system will reserve 0x1000000 DDR size from DDR0 and do best effort to allocate the entire memory from DDR1 for Frammap. Consequently, the Frammap module will manage two memory regions. Both the kernel drivers and user applications can allocate big memory block through the Frammap interface.

```
/ # cat /proc/frammap/ddr_info
----------------------------------------------------------------
ddr name: frammap0
base: 0x3800000
end: 0xb671000
size: 0x7e71000 bytes
memory allocated: 0x7e71000 bytes
memory free: 0x0 bytes
max available slice: 0x0 bytes
memory allocate count: 37
clear address: 0xb671000
dirty pages: 32369
clear pages: 0
size alignment: 0x1000
/ #
```

2.5.3 Virtual Memory Range

In *pgtable.h*, the start address is defined by a macro for the virtual memory.

#ifndef VMALLOC_START
#define VMALLOC_OFFSET (8*1024*1024)
#define VMALLOC_START (((unsigned long)high_memory + VMALLOC_OFFSET) & ~(VMALLOC_OFFSET-1))
#endif

The end address is defined in
<TOPDIR>/arch/arm/mach-GM/include/mach/platform-GM8136/vmalloc.h.
#define VMALLOC_END 0xFF000000

Note: The end address will be configurable if users want to expand the virtual memory range.

2.5.4 mbootpImage Flow

FA6-Linux provides **mbootpImage** to run the Linux kernel. **mbootpImage** comprises four components: Boots, Decompressor, Compressed Kernel, and Ramdisk, as shown in Figure 2-12.

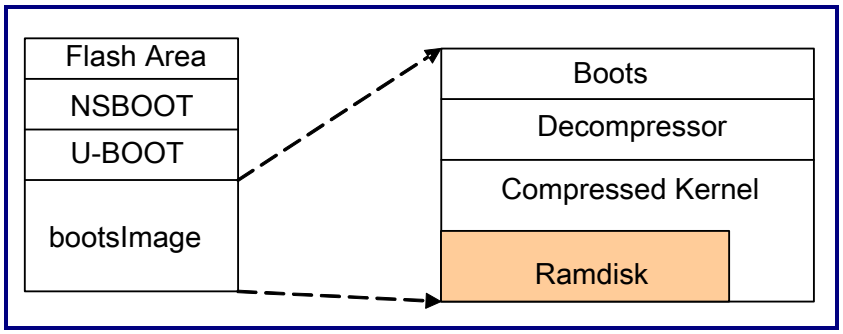


Figure 2-12. mbootpImage

- Boots: This component is in charge of copying bootsImage to the memory in the NOR system. However, this operation is not allowed in the SPI NOR/NAND system. The boot loader copies the Linux image to the memory.
- Decompressor: This component is in charge of decompressing "Compressed Kernel".
- Compressed Kernel: The image of the compressed kernel
- Ramdisk: The file system for the Linux kernel. If users choose squashFS, it will not be built with kernel. That kernel and its rootfs are separated. If initramfs is chosen, it will be a ramdisk built with kernel.

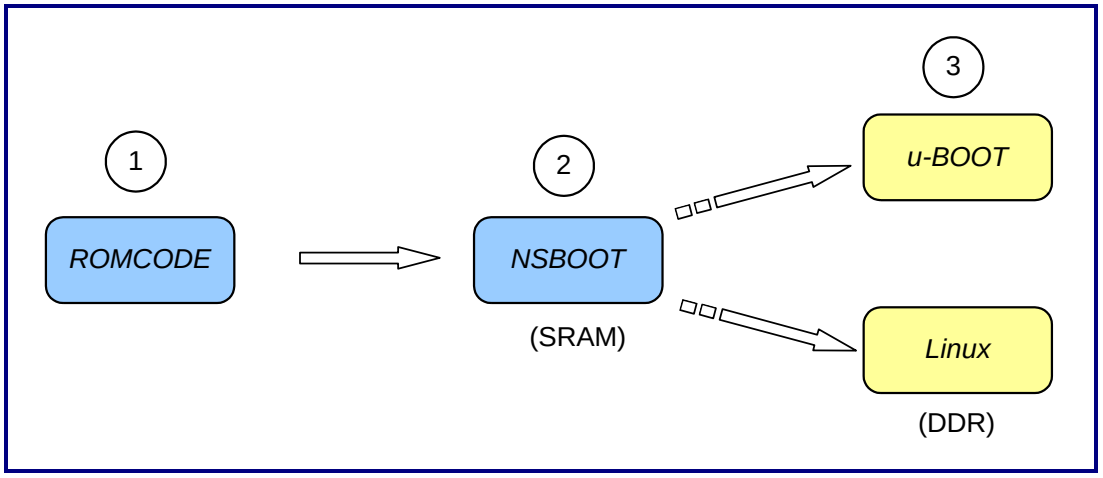
2.6 Boot Loader of SPI/SPI-NAND Flash

The content of the SPI/NAND Flash should not be read as a memory device. In order to read the boot image from the SPI/NAND Flash for booting, an embedded ROM code is responsible for loading the boot loader, which is called "nsboot", from the SPI/NAND Flash to SRAM for execution. SRAM is a readable/writeable memory in MCP100 of GM8136. GM8136 is equipped with an internal SRAM buffer to support the boot loader of the SPI/NAND Flash. When booting, the first page of the SPI/NAND Flash will be read by the embedded ROM code for verification purpose. The subsequent boot loader bodies stored in the SPI/NAND Flash memory will be fully loaded to SRAM for execution. For the current design, the image size of NSBOOT cannot exceed 24K bytes.

In general, the SPI/NAND boot code will copy the content of the SPI/NAND Flash to SDRAM. Once the copy is completed, the main program will be executed on SDRAM.

2.6.1 Boot Sequence

- When booting, the embedded ROM code will be first brought up and will read the jumper setting to know which Flash type will be accessed. Currently, there are two Flash types: SPI NOR and NAND. The embedded ROM code will then check the valid images in the Flash. ROMCODE will break the booting procedure and enter the firmware update mode if any false image is found.
- Once the images are verified, SPI/NAND will boot and execute the next block that will be loaded into SRAM. To simplify the process, the SPI boot and NAND boot will be combined as "NSBOOT".
- NSBOOT will read the information stored in the Flash to recognize the next image to be executed (Currently, it should be u-boot). NSBOOT will then copy the image body to DDR (SDRAM) according to the size of the image in the image header of the loaded image.
- Once the copy is completed, the loaded image will be executed on SDRAM.
  (The loaded image can be burn-in, U-BOOT, Linux, or any other customer image.)



2.7 Root File System: squash or initramfs

As mentioned before, squash rootfs is used for Kernel. Squash or Cramfs is a production rootfs instead of engineering rootfs. During mass production, squash or Cramfs rootfs is preferred because it can save more memory. But, it is inconvenient for the development stage. At the development stage, users want to directly load image through tftp. Thus, initramfs is suitable for this stage. Its disadvantage is to consume more memory for the whole ramdisk. For the pros and cons of rootfs, it is easy to find the data in internet.

If users would like to use initramfs at the development stage, please follow the steps below:

1) Enter menuconfig
2) General setup -> Initial RAM filesystem and RAM disk (initramfs/initrd) support
3) Please specify the rootfs location in Initramfs source file(s). Usually, "../target/rootfs-cpio" is used.
4) For the boot command line, it is not necessary to modify.
5) Please find the release file, such as rootfs-cpio.tgz for the rootfs and copy it to the specified location.
6) Build the kernel again

2.7.1 busybox

When squash rootfs is mounted, the first executed script file is squashfs_init.sh and the next will be busybox. In the busybox, Grain Media did some modifications for the environment variables. If users want to reconfigure busybox, please use the release files released by Grain Media. Its source code is in arm-linux-3.3/users/busybox-1.19.4.

Chapter 3

Device Driver

This chapter contains the following sections:

3.1 Timer Driver

For GM8136, all timing interrupts are IRQ. However, for Linux 3.3, IRQ provides high-resolution timing, including the system timing. GM8136 needs at least two timers: Timer0 and Timer1. The source file of the timers and the system timers is listed in Table 3-1.

Table 3-1. Timer Source File

File Name	Description
<TOPDIR>/arch/arm/mach-GM/fttmr010.c	This file implements the low-level timer functions.

3.2 Interrupt Driver

Table 3-2. Interrupt Source Files

File Name	Description
<TOPDIR>/arch/arm/kernel/irq.c	This file implements the kernel patch IRQ functions.
<TOPDIR>/arch/arm/mach-GM/ftintc030.c	This file implements the GM8136 low-level interrupt controller functions.

3.3 IRQ Setting

To use the Linux kernel IRQ setting, users can refer to the arch/arm/mach-GM/include/mach/platform-GM8136/platform-io.h IRQ string for this setting.

3.4 Serial Driver

3.4.1 Serial I/O Resource Overview

The Grain Media serial I/O resource and memory are listed in the GM8136 data sheet. Please refer to Table 3-3 for the UART clock setting. The UART clock is derived from PCIE CLK.

Table 3-3. GM8136 UART Clock Setting

UART clock	25 MHz

3.4.2 Source File Overview

FA6-Linux provides the UART driver to implement the tty/console driver. The related source files on Linux are shown in Table 3-4. Please note that <TOPDIR> is the top path of the kernel source tree.

Table 3-4. UART Source Files

File Name	Description
<TOPDIR>/drivers/tty/serial/8250/8250.c	This file implements the low-level UART functions.
<TOPDIR>/include/asm/arch-GM/include/mach/platform-GM8136/serial.h	This file is the header file of the serial port definition.

3.4.3 Serial Driver Guide

To activate the UART function, certain options, such as the serial port, setting the baud rate to 38400, and selecting UART0 as the kernel console port, should be selected for the kernel configuration. Figure 3-2 depicts the serial drivers (Device Drivers → Character Devices).

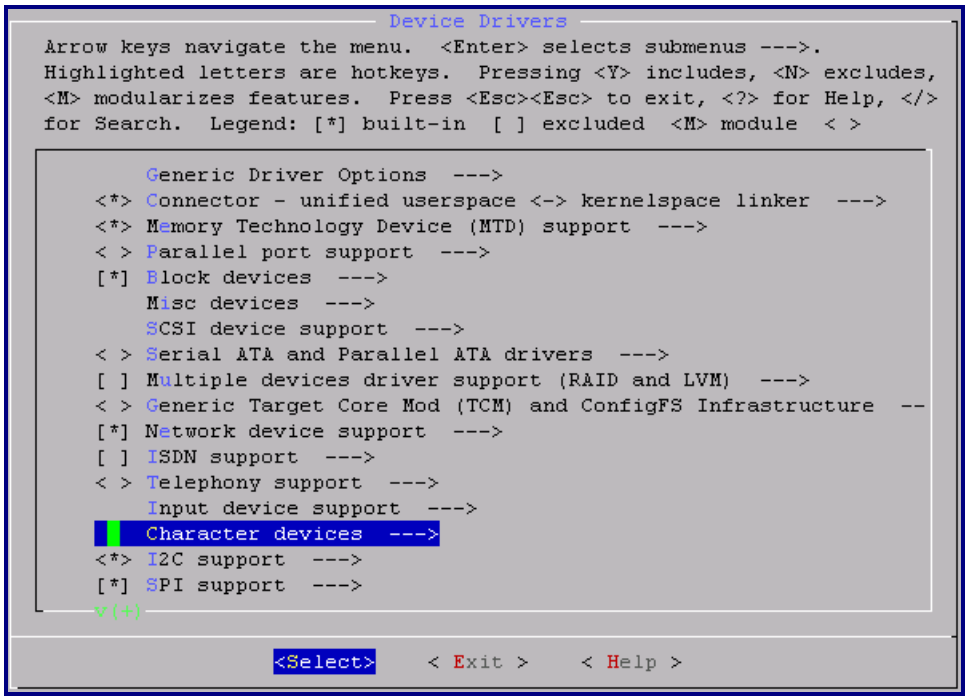


Figure 3-1. Kernel Configuration of Device Drivers

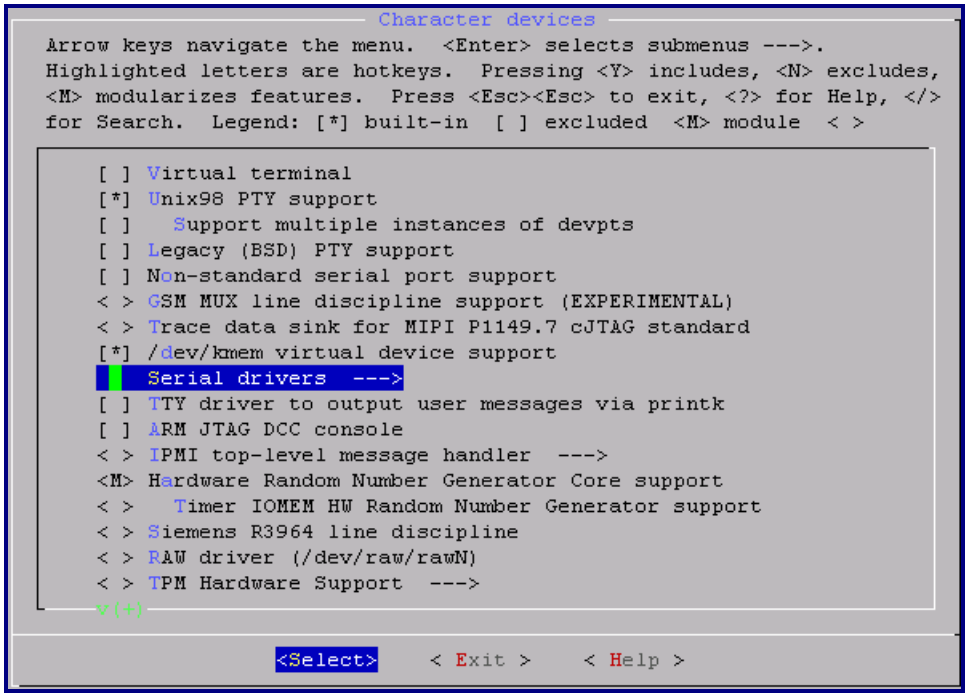


Figure 3-2. Kernel Configuration of Character Devices

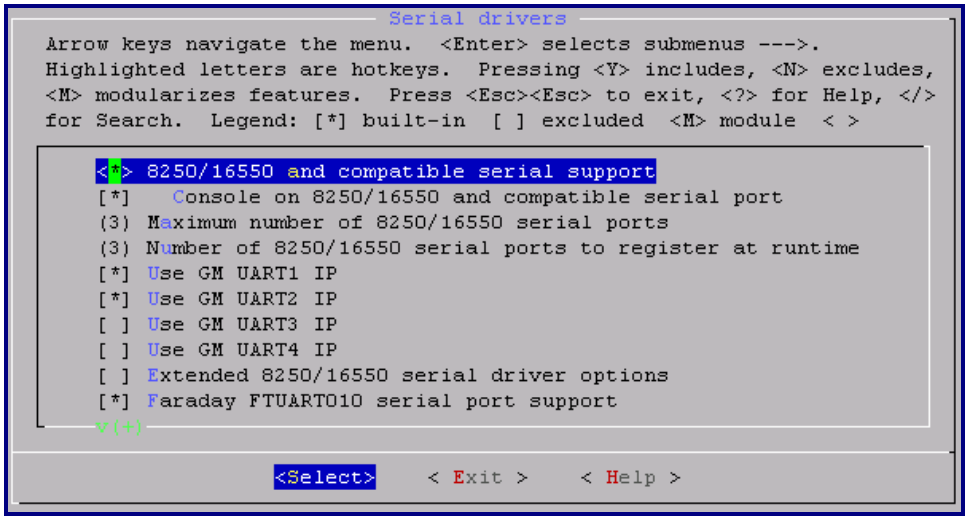


Figure 3-3. Kernel Configuration of CPE Serial Port Support

Some UART ports have the pin mux issue. Users can select the required UART ports. The default port, UART0, must be selected. If users select two items, the total UART ports will be three. The maximum number of the 8250/16550 serial ports must be set to 3 and the number of the 8250/16550 serial ports to the register at runtime should also be set to 3.

After compiling the Linux kernel with the options listed above, Linux will run and the booting message, as shown in Figure 3-4, will be shown by the terminal. Users will see the descriptions of UART in the booting message if the procedure is followed. Please note that the host terminal must be set to the same baud rate as the target.

```
Serial: 8250/16550 driver, 3 ports, IRQ sharing disabled
serial8250: ttyS0 at I/O 0xfe001000 (irq = 21) is a 16550A
serial8250: ttyS1 at I/O 0xfe002000 (irq = 22) is a 16550A
serial8250: ttyS2 at I/O 0xfe003000 (irq = 25) is a 16550A
```

Figure 3-4. Linux Booting Message: Serial

3.4.4 Driver Internals

This section describes the internal design of the Linux driver, including the features, architecture, and functions.

3.4.4.1 Driver Features

The Linux UART driver uses the device name, "**/dev/ttyS**". The major number of ttyS is 4 and the minor number starts from 64. Table 3-5 lists the device name and the device number.

Table 3-5. UART Device Number

Device Name	Major Number	Minor Number
/dev/ttyS0	4	64
/dev/ttyS1	4	65
/dev/ttyS2	4	66

The UART device driver does not implement any DMA feature; instead, it implements the interrupt routine to serve the Rx FIFO data.

3.4.4.2 UART and Image Mapping

The image uses UART0 to display the message.

3.4.5 References

- GM UART and IrDA Controller Specification
- Linux Device Drivers, 2nd Edition:
  Available at http://www.oreilly.com/catalog/linuxdrive2/chapter/book/index.html

3.5 CPU Frequency Scaling Driver

3.5.1 Driver Guide

Clock scaling allows users to change the clock speed of the operating CPUs and save the battery power. With lower clock speed, CPU will consume less power.

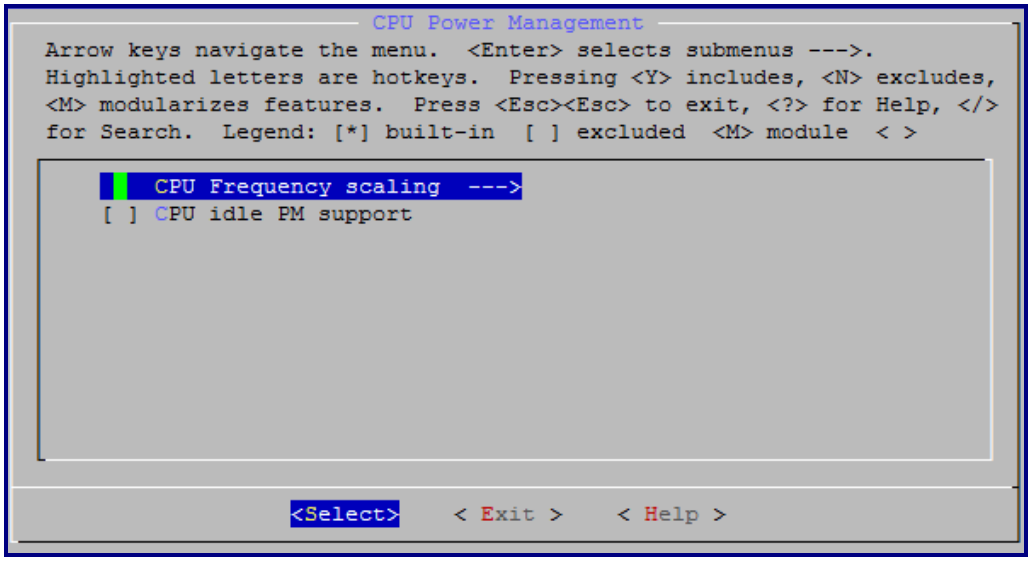


Figure 3-5. Kernel Configuration of CPU Power Management

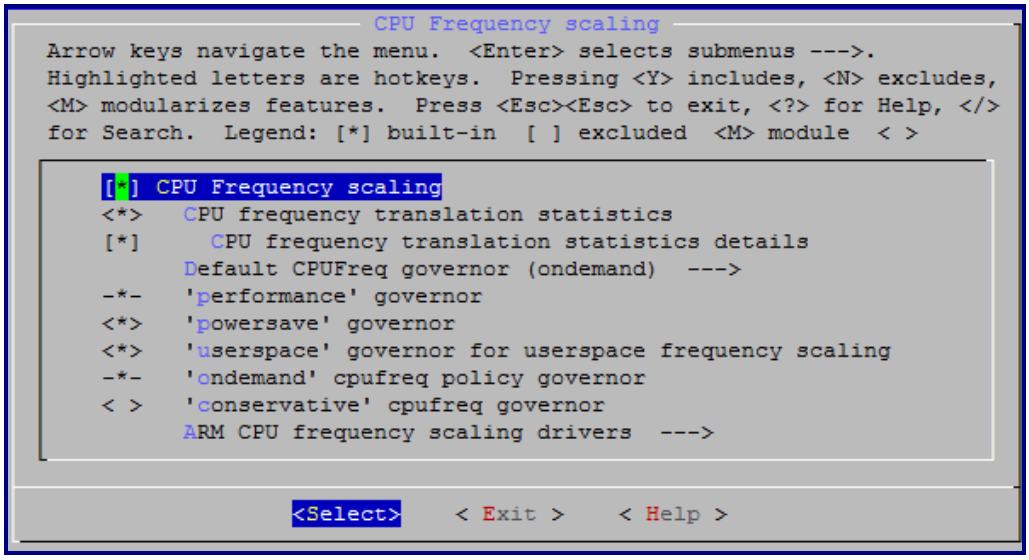


Figure 3-6. Kernel Configuration of CPU Frequency Scaling

3.5.2 Command

By default, users use the CPUFreq governor, "ondemand", to set CPU according to the current usage. CPU will very quickly switch the frequency. Grain Media provides some commands for the user tests

- Always set CPU to the highest speed

echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

- Return to the default setting:

echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor