i.MX28 Linux BSP

User’s Guide

Document Number: 924-76408
Rev. 2010.08
09/2010

About This Book

This document explains how to build and install the Freescale Linux board support package (BSP) to the i.MX28-EVK board.

Refer to the *i.MX Family Linux Software Development Kit Reference Manual* (to be released), for more information about installing the BSP and toolchain for the board, building zImage and root file system.

Audience

This document is intended for software, hardware, and system engineers who are planning to use the product. This document can also be used by any person who wants to understand more about the product.

Organization

This document contains the following chapters.

Chapter 1	Introduction
Chapter 2	Building the Linux Platform
Chapter 3	Configuring the Target Hardware
Chapter 4	Creating Boot Stream Image
Chapter 5	Booting the Target Hardware
Chapter 6	AGL

References

- i.MX Family Linux Software Development Kit Reference Manual

Chapter 1
Introduction

The i.MX Linux BSP is a collection of binary, code, and support files that can be used to create a Linux kernel image and a root file system for the i.MX board.

1.1 Flash Boot Loader

When the i.MX28 is reset, it executes the (Read-only memory) ROM. There is no alternative - no other code is permitted to handle the reset exception. The ROM reads the boot mode pins to detect the boot source (USB, SD/MMC, NAND flash, and so on.) and negotiates with that source in a device-dependent way to retrieve a “boot stream”. A boot stream is an executable collection of bytes in the Safe Boot (SB) format.

1.2 Boot Stream

NAND flashThe boot stream is an important concept for i.MX28.

The Linux release provides two boot stream images as follows:

- Linux kernel boot stream
- U-boot boot stream

Refer to Chapter 4 for more details about creating these boot stream images.

NOTE

i.MX28 supports U-boot instead of RedBoot.

A boot stream contains instructions that cause it to function as an extended boot loader for the ROM. Such a boot stream starts with a Load command that instructs the ROM to copy the executable into memory. A final Jump command instructs the ROM to transfer control to the executable that is loaded.

Another important command is Call. This command instructs the ROM to make a function call to a given address and continue processing the boot stream when the control returns. A Call command is usually preceded by a Load command that copies into memory the function to be called. Collectively, the Load command, the associated executable and the Call command are referred as a bootlet.

Figure 1 shows a boot stream that contains two bootlets, followed by the main executable:

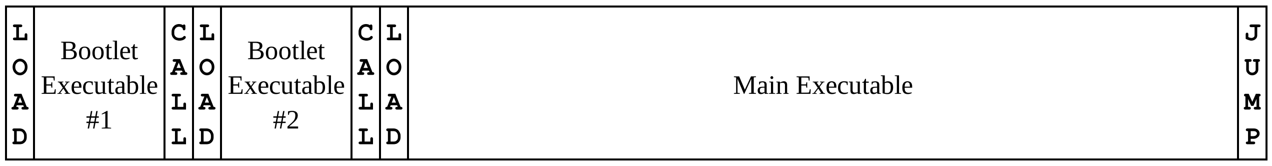

Figure 1. i.MX28 Boot Stream Outline

Each bootlet is an executable that is built separately, for a specific purpose, and may or may not know anything about the bootlets that precede or follow it.

The boot stream can instruct the ROM to call any number of executables before the final Jump, depending on the system needs. The i.MX28 Linux BSP boot streams contain the following bootlets:

- power_prep — This bootlet configures the power supply.
- boot_prep — This bootlet configures the clocks and SDRAM.
- linux_prep — This bootlet prepares to boot Linux

Figure 2 shows a kernel boot stream constructed with the i.MX28 Linux BSP:

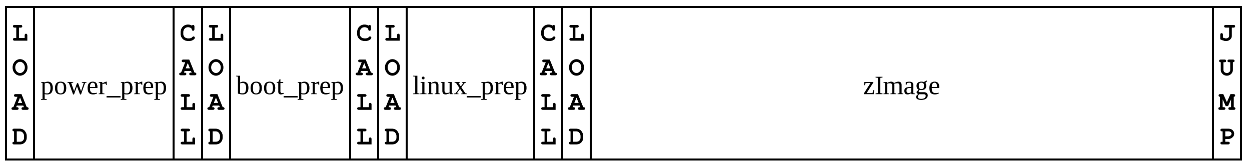

Figure 2. Example of i.MX28 Boot Stream Loading Linux Kernel

Figure 3 shows a U-Boot boot stream:

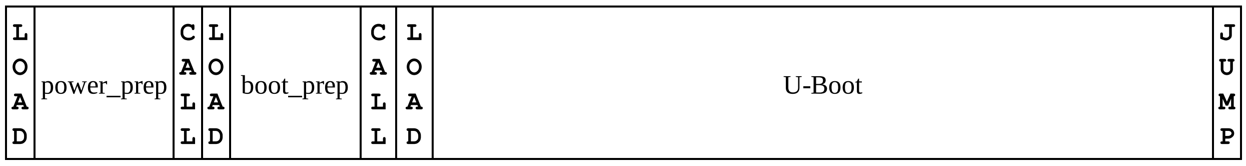

Figure 3. Example of i.MX28 Boot Stream Loading U-Boot

Refer to Chapter 4 for details about creating a boot stream image.

1.3 Linux Kernel and Driver

The Freescale BSP contains the Freescale Linux 2.6.31 kernel, driver source code, and a prebuilt kernel image. The kernel image is available in the following location:

```
imx28/zImage
```

```
imx28/uImage
```

```
imx28/imx28_linux.sb (combine boot steam and kernel image, HAB DISABLED boot
stream)

imx28/imx28_ivt_linux.sb (combine boot steam and kernel image, HAB ENABLED
boot stream)

imx28/imx28_uboot.sb (combine boot steam and uboot image, HAB DISABLED boot
stream)

imx28/imx28_ivt_uboot.sb (combine boot steam and uboot image, HAB ENABLED boot
stream)
```

NOTE

If the HAB_DISABLE bit HW_OCOTP_ROM7:0x8002C210:bit11 is “1”, then use boot streams without the name “`ivt`”.

If `the` HAB_DISABLE bit HW_OCOTP_ROM7:0x8002C210:bit11 is “0”, then use boot streams with the name “`ivt`”.

1.4 Root File System

The root file system package provides busybox, common libraries, and other fundamental elements. The Linux BSP contains the original root file system package as follows:

```
imx28/rootfs.jffs2

imx28/rootfs.ext2.gz
```

Chapter 2
Building the Linux Platform

This chapter explains how to setup the build environment, install and build the Linux Target Image Builder (LTIB), set rootfs for the Network file system (NFS), and setup the host environment.

2.1 Setting Up the Build Environment

Setting up the build environment includes installing Linux OS and LTIB.

2.1.1 Install Linux OS using Linux Builder

Install a Linux distribution such as Fedora 4/5, RedHat or Ubuntu 9.04, 9.10, or 10.04on a system.

2.2 Installing and Building LTIB

To install and build LTIB, perform the following steps:

NOTE

In some Linux systems, the following procedure must be performed with root permissions. However, these instructions are for performing the procedure not as root.

1. Install the LTIB package not as root:

```
tar zxf <ltib_release>.tar.gz

./<ltib_release>/install
```

   This command installs LTIB to a directory of your choice.

2. Build LTIB:

```
cd <your LTIB directory>

./ltib -m config
```

Select the platform as “Freescale iMX reference boards” and exit after saving the changes. In the next menu, select “mx28” as platform type and select a package profile. Profiles min, test, andfsl gnome have been tested. Exit after saving changes.

Then run the following command:

```
./ltib
```

When complete, LTIP will have produced the following in subdirectories of ./ltib:

- The kernel images `rootfs/boot/uImage` and `rootfs/boot/zImage`.
- The SB files of bootlets and kernel images `rootfs/boot/imx28_linux.sb`, `rootfs/boot/imx28_ivt_linux.sb`.
- The SB files of bootlets and uboot images `rootfs/boot/imx28_uboot.sb`, `rootfs/boot/imx28_ivt_linux.sb`.
- The Jffs2 rootfs image `rootfs.jffs2`

- **NOTE**

If you want an ext2 file system, execute `./ltib -c`, and change the option under the LTIB “Target Image Generation” menu from “JFFS2” to “EXT2”. After rebuilding,the EXT2 rootfs image can be found in `rootfs.ext2.gz`

2.3 Rootfs over NFS

2.3.1 Setting Network Parameters

The network parameters must be setup in LTIB to boot using the NFS. To set these parameters, execute:

```
./ltib -c
```

Set the network parameters in the following LTIB path:

Target System Configuration > Options > Network setup > IP address > netmask > broadcast address > gateway address > nameserver IP address

There are two places in the BSP to get the rootfs for NFS:

- Use the ext2 format rootfs package already provided in the distribution; or
- Use the rootfs that is created after making the build of the kernel

2.3.2 Using the rootfs Package in the Distribution

Use the following commands to set the `rootfs` directory for NFS (the user must be the root user for this operation):

```
mkdir /mnt/rootfs

cp imx28/rootfs.ext2.gz  /tools

cd /tools

gunzip rootfs.ext2.gz

mount -o loop -t ext2 rootfs.ext2 /mnt/rootfs

cp -r /mnt/rootfs .

export ROOTFS_DIR=/tools/rootfs
```

2.3.3 Using the rootfs Created After the kernel Build

Instead of using the `rootfs.ext2.gz`, use the root file system in `<your LTIB directory>`.

```
%export ROOTFS_DIR=/<your LTIB directory>/rootfs
```

Refer to the *i.MX Family Linux Software Development Kit Reference Manual,* for other ways to create a file system image file.

2.4 Setting up the Linux Host

To set up the Linux host system, perform the following steps:

1. Turn off the firewall to enable the `tftp` to work:
   ```
   iptables -F
   ```
   or, at the command line, type:
   ```
   setup
   ```
2. Install the **tftp** server.
3. Install the **nfs** server.
4. Create the `tftboot` directory.

   The kernel images (such as the **zImage** kernel image) and other things that need to be uploaded by `tftp` are stored in the `tftboot` directory.

   ```
   mkdir /tftpboot
   ```

5. Edit `/etc/xinetd.d/tftp` to enable tftp as follows:

```
{
disable  =  no
socket_type = dgram.
protocol = udp.
wait = yes
user = root
server = /usr/sbin/in.tftpd.
server_args = /tftpboot
}
```

6. Run the following command on the Linux host machine:

```
vi /etc/exports
```

   add this line in the file: `/tools/rootfs *(rw,sync,no_root_squash)`

7. Restart the **nfs** and **tftp** servers on the host:

```
/etc/init.d/xinetd restart
/etc/init.d/nfs restart
```

8. Copy `zImage` and `rootfs.jffs2` in the release package or LTIB to the tftp directory.

```
cp imx28/zImage /tftpboot
cp imx28/rootfs.jffs2 /tftpboot
```

   or

```
cp /<your LTIB directory>/rootfs/boot/zImage /tftpboot
cp /<your LTIB directory>/rootfs.jffs2 /tftpboot
```

NOTE

These instructions specify using an **nfs** server. Some Linux systems use **nfsserver**, rather than **nfs**. Use these instructions for either server type.

NOTE

A Windows tftp program "tftp.zip" is available in the LTIB release package `Common/` folder. This tftp program can be installed in the Windows OS to provide a Windows tftp server for downloading images.

2.5 Building the Manufacturing Firmware (To Be Supported)

Refer to Section 2.2 Installing and Building LTIB, to setup the ltib environment.

Configure the firmware build profile

```
./ltib --selectype
```

Choose the following:

--- Choose the platform type

Selection (**imx28**) --->

--- Choose the packages profile

Selection (**mfg firmware profile**) --->

After ltib completes the build, Updater.sb and updater_ivt.sb will have been created.

Chapter 3
Configuring the Target Hardware

This chapter details all hardware-specific configuration necessary to prepare the i.MX28-EVK development board for use with Linux.

3.1 External Cabling

Perform the following to setup external cabling:

- Plug the Linux host straight serial console cable into the UART DSUB9 connector on the i.MX28-EVK.
- Plug the Linux host USB A to mini-B USB cable into the mini-B USB connector on the i.MX28-EVK.

3.2 Board Configuration

The EVK board uses DIP switch S2 to select boot mode. Bits B0, B1, B2, and B3 are labeled on the board silkscreen. Table 1 gives the boot mode values.

Table 1. Boot Mode Values

B3	B2	B1	B0	Boot Mode
0	0	0	0	USB0
0	1	0	0	GPMI (NAND)
1	0	0	1	SSP0(SD0)
1	0	1	0	SSP1(SD1)

Refer to the *Hardware User Manual* for detailed EVK board configuration.

NOTE

The i.MX28 EVK board needs hardware rework for booting from SD1, Refer to the *EVK Hardware User Guide*.

Chapter 4
Creating Boot Stream Image

The i.MX28 SoC contains built-in ROM firmware that is capable of loading and executing binary images in special format from different locations, including an MMC/SD card and NAND flash. Such a binary image is called a “boot stream” and consists of a number of smaller bootable images (bootlets) and instructions for the ROM firmware to handle these bootlets (for example, load a bootlet to on-chip RAM and run it).

Refer to the *i.MX Family Linux Software Development Kit Reference Manual,* for kernel configuration and building.

4.1 Setting Up the Kernel Command Line

In LTIB, run the following command, then choose “Package list” and then set default and alternative kernel command lines under the “boot stream” option:

```
./ltib -m config
```

4.2 Building the boot stream image

In LTIB, to build a new Linux Kernel and U-Boot boot stream image, give the command:

```
./ltib -p boot_stream.spec -f
```

The output boot stream images are available in `rootfs/boot/` directory, named `imx28_linux.sb` and `imx28_uboot.sb` for HAB-disabled images, and `imx28_ivt_linux.sb` and `imx28_ivt_uboot.sb` for HAB-enabled images. The i.MX28 EVK is shipped with HAB enabled by default.

Chapter 5
Booting the Target Hardware

This chapter explains how to boot the I.MX28 development board for the first time. Linux kernel can be booted on the i.MX28 using the following ways:

- Boot from USB
- Boot from MMC/SD card
- Boot from NAND flash
- Boot from Ethernet (network boot)

All boot modes except network boot are supported by the i.MX28 built-in ROM firmware. The ROM code reads the boot stream image containing the Linux kernel from the first three sources. Network boot of the Linux kernel is performed by the U-Boot boot loader. U-Boot is loaded and started by the ROM firmware through the USB or MMC card or NAND flash.

The Linux SDK provides two boot stream images as follows:

- Linux kernel boot stream
- U-boot boot stream

Refer to Chapter 4 for more details on how to generate a new boot stream image.

The following sections describe how to prepare and boot the Linux kernel in each boot mode.

5.1 Target Preparation

5.1.1 Setting Up kernel command line

The kernel boot command line can be set in LTIB in the "boot steam" configuration menu under "Package list" -> "boot steam". These command line options include a default command line and three command lines selected by key-press during system start up. If the command line configuration file has less than four command lines, then the unused entries are replaced by the following default command line string:

```
console=ttyAM0,115200
```

To select the location of the root file system, the "root" command line variable must be configured. There may also be a need to set additional command line options based on the root file system storage type:

- Root file system located on a MMC card partition:

  ```
  root=/dev/mmcblk0p[N] rw rootwait
  ```

Where N is the number of the MMC card primary partition containing the root file system

- Root file system located on a NAND flash (Jffs2):

```
root=/dev/mtdblock1  rootfstype=jffs2
```

- Root file system on NFS over Ethernet link:

```
ip=dhcp/off/[Target IP] root=/dev/nfs nfsroot=/tools/rootfs
```

  Where:

  Host IP is the IP address of Ubuntu Linux host

  Target IP is the IP address assigned to the I.MX28 development board

- ENET MAC address

NOTE

On the i.MX28 EVK, the MAC address is stored in OTP fuses that have been pre-programmed. If a differentthe MAC address is to be used, then the following option can be added:

```
fec_mac=xx:xx:xx:xx:xx:x
```

Where:

’ xx:xx:xx:xx:xx:xx ‘ MAC address of ENET of the EVK board.

- gpmi or ssp1 selection

```
gpmi/ssp1
```

  Where:

  gpmi: initialize gpmi (i.e. NAND) interface

  ssp1: initialize ssp1 (i.e. SD Card 1) interface

  Either a NAND device can be used, or an SD device on SD slot 1 can be used, but not both. This is due to pin-sharing on the i.MX28. SD slot 0 is unaffected by this choice.

There are four preset command lines which allow booting the kernel with the root file system located on SD/MMC card, NFS, NAND flash, or RAM disk:

- Default command line for SD (no key press during booting):

```
noinitrd console=ttyAM0,115200 root=/dev/mmcblk0p3 rw rootwait gpmi
```

- Alternative command line 1 for NFS (press KEY1 during booting):

```
noinitrd console=ttyAM0,115200 ubi.mtd=1 root=ubi0:rootfs0
               rootfstype=ubifs rw gpmi
```

- Alternative command line 2 for NAND (press KEY2 during booting):

```
        noinitrd console=ttyAM0,115200 fec_mac=00:08:02:6B:A3:1A
                      root=/dev/nfs
                      nfsroot=10.193.100.213:/data/rootfs_home/rootfs_mx28
                      rw ip=dhcp rootwait gpmi
```

- Alternative command line 3 for RAM disk(press KEY3 during booting)

```
        noinitrd console=ttyAM0,115200 root=/dev/ram0 rdinit=/sbin/init
                      fec_mac=00:08:02:6B:A3:1A gpmi
```

5.1.2 Rebuilding the Linux image

If the default command lines are modified, then it is necessary to rebuild the release to get the Linux kernel boot stream image with those updated command lines. In LTIB, issue the command:

```
./ltib -p boot_stream -f
```

5.1.3 Writing the Boot Stream and rootfs to a Boot Medium

This section describes how to write the boot stream and rootfs a boot medium.

5.1.3.1 MMC Boot

The default kernel command line uses three primary partitions as follows:

1. File Allocation Table (FAT)
2. Boot stream partition of type 0x53 (OnTrack DM6 Aux3)
3. Linux rootfs such as ext2.

You can use the following steps to create such an SD/MMC card:

Create three primary partitions on MMC card in addition to the mandatory one that contains the boot stream image. The example below shows the fdisk -1 output, which is configured using the MMC/SD card (all these steps are done in the host Linux PC):

```
fdisk -l /dev/sdb

Disk /dev/sdb: 1967 MB, 1967128576 bytes

61 heads, 62 sectors/track, 1015 cylinders

Units = cylinders of 3782 * 512 = 1936384 bytes

Disk identifier: 0x00000000

Device Boot Start End   Blocks      Id    System
```

```
/dev/sdb1   *          1        974     1840896    b  W95 FAT32

/dev/sdb2            974        977        5339+  53  OnTrack DM6 Aux3

/dev/sdb3            977       1016       74572  83  Linux
```

- Create an MMC partition image that contains the boot stream in target using “sdimage”, which is in the “uuc” package in ltib.

    ```
    sdimage imx28_linux.sb /dev/sdb
    sync
    ```

- Format the second partition:

    ```
    mkfs.ext2 /dev/sdb2
    ```

- Mount the second MMC/SD partition:

    ```
    mount /dev/sdb2 /mnt/mmc
    ```

- Copy the Linux release development root file system to the directory where the MMC partition is mounted. Refer to Rootfs over NFS for retrieving the root file system.
- Add manual updates to the root file system in `/mnt/mmc`, if required
- Unmount the MMC partition:

    ```
    umount /mnt/mmc
    ```

As an alternative, the Windows tools called `cfimager` can create the partitions, and write the boot stream and rootfs**.**

```
Cfimager.exe -a -f imx28_linux.sb -e rootfs.ext2 --daul_boot -d <mass
                 storage disk, no ``:’’, such as H>
```

NOTE

The default rootfs file system released for SD is EXT2 format. EXT2 is not a journaling file system. Any disruption to the file system while syncing can cause a file system error, such as power lost, kernel panic and so on. To avoid such errors, either follow the normal power sequence or use an EXT3 file system. You can convert an EXT2 filesystem to an EXT3 filesystem by using the command `tune2fs -j rootfs.ext2`.

5.1.3.2 NAND Boot

A boot stream image can not be burned to NAND flash from the Linux host. It is necessary to first load the kernel from an MMC card or network boot. Then, after Linux is running on the board, it is possible to burn the boot stream image to NAND using the “kobs-ng” tool:

- Copy the boot stream to the root file system. For example, in the case of an NFS root, use:

```
cp <where the boostream lives>/iMX28_linux.sb /tools/rootfs
```

- Boot the target and log into it:
- On the target, burn the boot stream image to the flash:

```
#flash_eraseall /dev/mtd0
#kobs-ng init imx28_linux.sb
```

- Copy the jffs2 image to current root file system.

  For example, in the case of NFS root, run the following command on the Linux host. Note that the jffs2 image must match the type of flash device in use.

```
cp rootfs.jffs2 /tools/rootfs
```

- On the target, erase the MTD 1 partition:

```
flash_eraseall /dev/mtd1
```

- On the target, burn the jffs2 image from the rootfs to the flash:

```
nandwrite /dev/mtd1 rootfs.jffs2
```

5.1.3.3 Network Boot

Linux kernel network boot is implemented using the U-boot boot loader that is part of the Linux release for Freescale i.MX28. The U-boot boot stream is loaded from SD or USB by sb_loader.exe tool. Refer to USB Boot and Network Boot for more details.

5.2 Host preparation

This section describes preparation of the host computer.

5.2.1 Root File System on NFS Partition

Refer to 2.3 for more details.

5.3 Target Boot

This section describes how to boot the target EVK.

5.3.1 USB Boot

Perform the following to boot from USB:

- Select the USB boot mode (0000) on the DIP switch. Refer to Chapter 3 for more details.
- Plug in the power cord to the i.MX28 development board.

- Press the power key.
- Press KEY1/2/3 to select an alternative boot cmdline, hold the key until the bootlets have run. To use the default boot command line, do not press any key.
- After Windows recognizes the EVK as a USB HID device, run the following command in the Windows console:

```
z:\sb_loader.exe /f imx28_linux.sb
z:\sb_loader.exe /f imx28_uboot.sb
```

NOTE

Use the "ivt" bootstreams, if the chip is HAB-enabled.

5.3.2 MMC/NAND Boot

Perform the following to boot the MMC/NAND:

- Select a boot mode (SD/MMC:1001, NAND: 0100) using the DIP switch. Refer to Chapter 3 for more details.
- For SD, insert the SD card with i.MX28_linux.sb and rootfs.ext2 burned by cfimage.exe into the SD slot 0. For NAND, the boot stream and rootfs image should be burned into the flash as described previously.
- Apply power to the i.MX28 development board.
- Press the power key.
- Press KEY1/2/3 to select an alternative boot cmdline, hold the key until the bootlets have run. To use the default boot command line, do not press any key.
- The bootlets and kernel will run.

5.3.3 Network Boot

Perform the following to boot from the network:

- Connect the target and host using an Ethernet 10 Base-T cable
- Ensure that the Trivial File Transfer Protocol (TFTP) server is running on the Linux host.
- Copy the Linux kernel image to the `/tftpboot` directory:

```
cp rootfs/boot/uImage /tftpboot
```

- Ensure that the rootfs file is available in the ROOTFS_DIR folder.
- Insert SD card with i.MX28_uboot.sb burnt by cfimage.exe into SD slot0
- Power ON the i.MX28 development board
- Press the power key. Run Bootlets and uboot

- Press enter in the U-boot serial console (for example, using the minicom) to get the U-boot prompt
- Set the U-boot run-time variables as follows:

```
MX28 U-Boot > setenv bootargs 'console=ttyAM0,115200n8'
MX28 U-Boot > setenv bootcmd 'run bootcmd_net'
MX28 U-Boot > setenv bootdelay 2
MX28 U-Boot > setenv baudrate 115200
MX28 U-Boot > setenv serverip [Host IP]
MX28 U-Boot > setenv netmask 255.255.255.0
MX28 U-Boot > setenv bootfile uImage
MX28 U-Boot > setenv loadaddr 0x42000000
MX28 U-Boot > setenv nfsroot [ROOTFS_DIR]
MX28 U-Boot > setenv bootargs_nfs 'setenv bootargs ${bootargs}
              root=/dev/nfs ip=dhcp nfsroot=${serverip}:${nfsroot}
              fec_mac=[MAC address] gpmi’
MX28 U-Boot > setenv bootcmd_net 'run bootargs_nfs; dhcp; bootm'
MX28 U-Boot >setenv ethaddr [MAC address]
MX28 U-Boot >saveevn

Where:
'Host IP'      IP address of the Ubuntu Linux host
‘ROOTFS_DIR’   NFS folder path
‘MAC address’ MAC address of ethernet
```

- Load and start the Linux kernel:

```
boot
```

Chapter 6 AGL

6.1 AGL

AGL means Automative Grade Linux. Historically, Linux is a unixoid server operating system that emerged into the desktop and embedded world. While boot time is of less importance for server, it is very critical for embedded systems, especially when there is a user interface involved. For this reason, AGL implements a number of optimization to boot into user space as fast as possible, that is, in less than one second on the i.MX28-EVK platform.

6.2 BTCS

With a fast bootloader and some kernel optimizations, a Linux system can boot within several hundred milliseconds. However, some application requires system response in a much lower time. For example in the automotive area, an MCU should be able to serve incoming CAN messages within 60 milliseconds after power ON. That means the boot time critical services should be up and running before/while the Linux kernel is booting and executing.

Another reason t is that the intellectual properties inside the boot time critical services need not interfere with the GPL as used for the bootloader and the linux kernel. For this reason, AGL implements the boot time critical services architecture, called as BTCS.

6.3 AGL components

This section describes about the AGL components.

6.3.1 Boot Stream

The i.MX28-EVK AGL Linux delivery package contains U-Boot boot stream binary.

6.3.2 Linux Kernel image

This AGL BSP contains the Freescale Linux 2.6.31 kernel, driver source code, and a prebuilt kernel image `uImage_xip`.

6.3.3 BTCS

This AGL release contains one demo BTCS (can stack) binary: `btcs.bin`. It resides in its own memory space and interacts with the bootloader or the kernel. Refer to *BTCS driver introduction* manual for more details.

6.3.4 Root File System

The root file system package provides busybox, common libraries and other fundamental elements in min profile. The package contains the original root file system package, `rootfs.ext2.gz,` which can be flashed to MMC/SD card.

6.4 Installing and Building LTIB

Perform the following steps to install and build the LTIB:

NOTE

In some Linux systems, the following procedure must be done with root permissions. However, these instructions are for performing the procedure not as root.

1. Install the LTIB package not as root:

   ```
   tar zxf <ltib_release>.tar.gz
   ./<ltib_release>/install
   ```

   This command installs LTIB to the directory.

2. Build the LTIB:

   ```
   cd <your LTIB directory>
   ./ltib -m config
   ```

   Set platform choice as Freescale iMX reference boards and exit after saving the changes. In the next menu, select platform type (`imx28_agl`) and the min profile. Exit after saving the changes.

   Then run the following command:

   ```
   ./ltib
   ```

   When the build is complete, the boot stream is available at `rootfs/boot/imx28_uboot.sb`, the kernel image is available at `rootfs/boot/uImage_xip,` btcs demo application is available at `rootfs/boot/btcs.bin` and file system is available at `rootfs.ext2.gz.`

NOTE

If the host system does not have the packages that are required by the LTIB, during the execution, then the system would prompt to install additional packages.

6.5 Writing Boot Stream, kernel, BTCS and rootfs to the MMC/SD card

This section explains how to partition the MMC/SD card, and download boot stream, kernel image, btcs, and rootfs. .

6.5.1 MMC/SD Memory Map

i.MX28-EVK AGL's command line uses 2 primary partitions as follows:

- Boot stream partition of type 0x53(OnTrack DM6 Aux3)
- Linux rootfs such as ext2

Create a secondary primary partition on the MMC card in addition to the mandatory one that contains the boot stream image. The example below shows the fdisk -1 output, which is configured using the MMC/SD card (all these steps are done in the host PC):

```
Disk /dev/sdb: 3965 MB, 3965190144 bytes

122 heads, 62 sectors/track, 1023 cylinders

Units = cylinders of 7564 * 512 = 3872768 bytes

Disk identifier: 0xacfb286a



   Device Boot      Start         End      Blocks   Id  System

/dev/sdb1             512         768      971974   53  OnTrack DM6 Aux3

/dev/sdb2             769        1023      964410   83  Linux
```

Kernel and BTCS images are available in the MMC/SD card, with the offset 0x00100000 and 0x00500000 respectively.

NOTE

The rootfs can only be written to partition 2, sdb2, as in the default command line, the kernel tries to mount the rootfs from `/dev/mmcblk0p2`.

6.5.2 Writing Steps

This section describes the various writing procedures.

6.5.2.1 Writing the Boot Stream

Create an MMC partition image, which contains the boot stream in target by sdimage which is in uuc package in ltib.

```
sdimage imx28_uboot.sb /dev/sdb

sync
```

6.5.2.2 Writing the Kernel Image

The Kernel image should reside in the MMC/SD card with the offset 0x00100000.

```
sudo  dd if=uImage_xip of=/dev/sdb bs=1M seek=1
```

6.5.2.3 Writing the BTCS Image

The BTCS image should reside in the MMC/SD card with the offset 0x00500000.

```
sudo  dd if=btcs.bin of=/dev/sdb bs=1M seek=5
```

6.5.2.4 Writing the rootfs

The rootfs should be written to the second partition. Copy the Linux release development root file system to the directory where the MMC partition is mounted.

```
/* the src folder depends on the server */

sudo mount -t ext2 /dev/sdb2 /mnt/src

gunzip rootfs.ext2.gz

mkdir tmp

sudo mount -t ext2 -o loop rootfs.ext2 tmp

sudo mknod ./tmp/dev/ttyAM0 c 204 16

sudo cp -dR tmp/*  /mnt/src

sudo umount tmp

sudo umount /mnt/src
```

6.6 Running the Image from MMC/SD Card

Perform the following steps to boot the kernel from MMC/SD:

1. Set dip switch as MMC boot mode (SD: 1001).
2. Press the power key.

The logo is displayed within 1 second, kernel is ready in 1second, and the system switches to the user land within 3.78 seconds.

6.7 Running the BTCS Demo Application Console

This section describes on how to run the BTCS demo application console.

6.7.1 Building the BTCS Console Application from LTIB

```
  cd <your LTIB directory>
```

On the Freescale i.MX28-EVK Board with AGL page, enable the following:

- Leave the sources after building
- btcs-console in package list

save and Exit the menu configuration.

```
[*] Leave the sources after building

package list -->

     [*] btcs-console
```

Run the command to rebuild LTIB:

```
./ltib
```

Then the btcs-console application is generated on `rootfs/user/share/btcs/` folder. This application connects to the BTCS demo application, which runs on another i.MX28-EVK board standard and has the same functionality as that of the CAN test application.

6.7.2 Setting Up Hardware

Connect the two i.MX28-EVK board's CAN interfaces using a 1:1 9-Pin cable as shown in Figure 4.

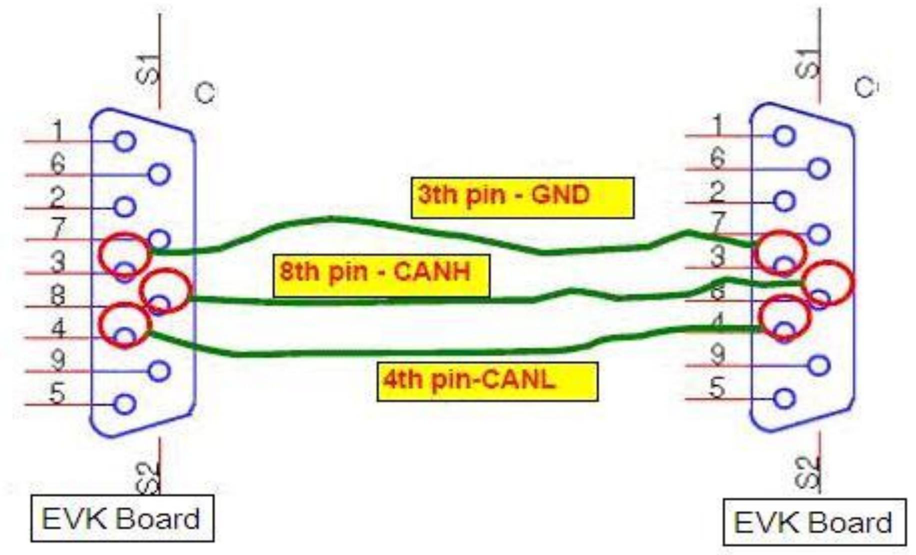

Figure 4. i.MX28 EVK CAN Interface

6.7.3 Setting Up Software

Perform the following steps to setup software:

1. AGL is written to target i.MX28-EVK board (target). Install btcs-console on another i.MX28-EVK board (host) with standard PDK released system.
2. On the host, launch the console application.

```
ifconfig can0 up
./btcs-console can0
```

3. Reset the target, greeter and the prompt of the BTCS demo application should be on the host console as the log shown below.

```
*** BTCS Demo Stack ***
>
```

4. Now display (d) and modify (m) the 10 BTCS demo register (0-9) as shown below.

```
Display register 1
```

```
>d 1

[1] 00000000

Modify register 1

>m 12345

Display register 1 again

>d 1

[1] 00012345
```

6.8 Running the BTCS Demo Application

This application displays and modifies BTCS demo registers locally. It runs on the target where the BTCS is active. Modified registers are propagated to a potentially connected btcs_console running on the host.

6.8.1 Building Latency Test Application from ltib

By default, the btcs_app application is built. The application `rootfs/boot/` folder of the ltib is available after the build.

6.8.2 Setting Up Hardware

Connect the two i.MX28-EVK board's CAN interfaces using a 1:1 9-Pin cable as shown in Figure 4.

6.8.3 Setting Up Software

Perform the following steps to setup software:

1. AGL is flashed to target i.MX28-EVK board (target).
2. Prepare another i.MX28-EVK board as host with standard i.MX28-EVK.

NOTE

The btcs application requires a device node, `/dev/btcs`, to access the BTCS. If `udev` is disabled in the i.MX28-EVK AGL, then it can be created with `mknod /dev/btcs c 10 63`.

3. On the i.MX28-EVK AGL board (target) , launch the btcs_app application:

```
mknod /dev/btcs c 10 63
/boot/btcs_app
```

The resulting log file should look as follows:

```
root@FSL /boot$ ./btcs_app
Usage: btcs [s|d <register>|m <register> <value>
  s = show all registers
  d = display a specific register
  m = modifiy a specific register
  o = observe register changes
```

4. Display (d) and modify (m) the 10 BTCS demo register (0-9), and show all the registers(s) or observe the register changes.

```
Display register 1
root@FSL /boot$ ./btcs_app  d 1
[1] 00000000
Modify register 1
root@FSL /boot$ ./btcs_app  m 12345
Display register 1 again
root@FSL /boot$ ./btcs_app d 1
[1] 00012345
Observe the changes
root@FSL /boot$ ./btcs_app o
Observe the changes
root@FSL /boot$ ./btcs_app s
[0] 00000000
[1] 00012345
[2] 00000000
[3] 00000000
[4] 00000000
[5] 00000000
[6] 00000000
[7] 00000000
[8] 00000000
[9] 00000000
```

6.9 Running SnoopSerial Test for Boot Time Benchmark

Snoopserial (sns) is a simple benchmark and time measurement tool. It captures log messages from the serial interface and for each line, it shows how much time has elapsed since the target reset (or optionally since the first line showed up) occurred and how much time has elapsed between two lines. This is very helpful to figure out which part of the target system boot process takes longer time. Together with sprinkling a Linux kernel with printks, sns can provides useful hints for optimization.

When sns is started, it sends a reset signal to the target MX28-EVK through the parallel port (data 0, pin 2). Then it listens to `/dev/ttyS0` (at 115200,8,n,1) and shows the time since /RST was released for each captured line, and the time gap between each line captured.

6.9.1 Building sns Application from ltib

Perform the following steps to build the sns application from ltib:

1. Run the following command in ltib folder:

```
./ltib -m prep -p sns
./ltib -m scbuild -p sns
./ltib -m scdeploy -p sns
```

2. The `sns` application generated on `./bin` folder in ltib.

6.9.2 Setting Up Hardware

Connect pin2 of parallel port on linux server to CN1/JTAG pin 15 (/rst) of the i.MX28-EVK board.

Boot the target (i.MX28-EVK AGL board) and ensure some output is displayed after power ON, that is, with minicom on linux server. Then it means that the hardware is setup. Exit from minicom.

6.9.3 Setting Up Software

Perform the following steps to setup software:

1. AGL is written to target i.MX28-EVK board (target). Copy the sns application to linux server and execute sns application on linux server as root to avoid permission problems.

```
root@shlx1$  ./sns
```

2. The output is displayed on the Linux server.

A sample output of sns measured on an demo image `/rootfs` is shown below:

The kernel copy takes 280 ms and the kernel takes 1130 ms to boot (7.100s-5.970s); The system fully startsup and enters into userland at about 3.78 seconds.

NOTE

After the RESET signal is released, it takes almost 5.5 s to reach the uboot execution stage. It is abnormal as it takes much lesser time, which seems within 1s, to reach the same stage when the RESET key is pressed on the EVK board. This has to be checked.

The kernel and rootfs take longer time to boot and mount. They should be optimized in the future.

```
[b04597@shlx1 snoopserial-reset]$ sudo ./sns-rst                                    .

                                                                [ 5.500  5.500 ]
U-Boot 2009.08-00018-g358235f (Aug 24 2010 - 20:15:46)          [ 0.000  5.500 ]
                                                                [ 0.000  5.500 ]
DRAM:  128 MB                                                   [ 0.000  5.500 ]
MMC:   IMX_SSP_MMC: 0                                           [ 0.010  5.510 ]
                                                                [ 0.000  5.510 ]
MMC read: dev # 0, block # 10240,count 30 ...                   [ 0.010  5.520 ]
30 blocks read: OK                                              [ 0.020  5.540 ]
--->Starting btcs ...                                           [ 0.000  5.540 ]
Hit any key to stop autoboot:  0                                [ 0.010  5.550 ]
                                                                [ 0.000  5.550 ]
MMC read: dev # 0, block # 2048,count 6144 ...                  [ 0.000  5.550 ]
6144 blocks read: OK                                            [ 0.280  5.830 ]
## Booting kernel from Legacy Image at 40007fc0 ...             [ 0.010  5.840 ]
   Image Name:   Linux-2.6.31                                   [ 0.000  5.840 ]
   Image Type:   ARM Linux Kernel Image (uncompressed)          [ 0.010  5.850 ]
  Data Size:    2938976 Bytes =  2.8 MB                         [ 0.000  5.850 ]
  Load Address: 40007fc0                                        [ 0.000  5.850 ]
  Entry Point:  40008000                                        [ 0.010  5.860 ]
  Verifying Checksum ... OK                                     [ 0.060  5.920 ]
  XIP Kernel Image ... OK                                       [ 0.000  5.920 ]
OK                                                              [ 0.000  5.920 ]
                                                                [ 0.000  5.920 ]
Starting kernel ...                                             [ 0.010  5.930 ]
                                                                [ 0.000  5.930 ]
Linux version 2.6.31-869-gde80e84-00018-g358235f (b04597@shl [ 0.040  5.970 ]
    CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=0005317      [0.010
                                                                       5.980]
CPU: VIVT data cache, VIVT instruction cache                    [ 0.000  5.980 ]
Machine: Freescale MX28EVK board                                [ 0.000  5.980 ]
Memory policy: ECC disabled, Data cache writeback               [ 0.010  5.990 ]
Size of 0x100000 sdram reserved for BTCS                        [ 0.000  5.990 ]
Built 1 zonelists in Zone order, mobility grouping on.  Tota    [ 0.010  6.000 ]
Kernel command line: console=ttyAM0,115200n8 root=/dev/mmcbl [ 0.010  6.010 ]
PID hash table entries: 512 (order: 9, 2048 bytes)              [ 0.000  6.010 ]
Dentry cache hash table entries: 16384 (order: 4, 65536 byte    [ 0.010  6.020 ]
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)    [ 0.000  6.020 ]
Memory: 128MB = 128MB total                                     [ 0.000  6.020 ]
Memory: 125680KB available (2616K code, 357K data, 104K init    [ 0.010  6.030 ]
    SLUB: Genslabs=11, HWalign=32, Order=0-3, MinObjects=0, CPUs           [
                                                                  0.010  6.040 ]
Hierarchical RCU implementation.                                [ 0.000  6.040 ]
NR_IRQS:288                                                     [ 0.000  6.040 ]
Console: colour dummy device 80x30                              [ 0.000  6.040 ]
console [ttyAM0] enabled                                        [ 0.010  6.050 ]
```

```
Calibrating delay loop (skipped) preset value.. 239.20 BogoM          [ 0.000  6.050 ]
Security Framework initialized                                         [ 0.010  6.060 ]
Mount-cache hash table entries: 512                                    [ 0.000  6.060 ]
CPU: Testing write buffer coherency: ok                                [ 0.010  6.070 ]
regulator: core version 0.5                                            [ 0.000  6.070 ]
NET: Registered protocol family 16                                     [ 0.010  6.080 ]
regulator: vddd: 800 <--> 1575 mV fast normal                          [ 0.010  6.090 ]
regulator: vddd_bo: 800 <--> 1575 mV fast normal                       [ 0.010  6.100 ]
regulator: vdda: 1500 <--> 2275 mV fast normal                         [ 0.010  6.110 ]
regulator: vddio: 2880 <--> 3680 mV fast normal                        [ 0.000  6.110 ]
regulator: overall_current: 0 <--> 2147483 mA fast normal              [ 0.010  6.120 ]
regulator: mxs-duart-1: 0 <--> 2147483 mA fast normal                  [ 0.000  6.120 ]
regulator: mxs-bl-1: 0 <--> 2147483 mA fast normal                     [ 0.010  6.130 ]
regulator: mxs-i2c-1: 0 <--> 2147483 mA fast normal                    [ 0.000  6.130 ]
regulator: mmc_ssp-1: 0 <--> 2147483 mA fast normal                    [ 0.010  6.140 ]
regulator: mmc_ssp-2: 0 <--> 2147483 mA fast normal                    [ 0.010  6.150 ]
regulator: charger-1: 0 <--> 2147483 mA fast normal                    [ 0.000  6.150 ]
regulator: power-test-1: 0 <--> 2147483 mA fast normal                 [ 0.010  6.160 ]
regulator: cpufreq-1: 0 <--> 2147483 mA fast normal                    [ 0.000  6.160 ]
i.MX IRAM pool: 124 KB@0xc8820000                                      [ 0.010  6.170 ]
audit: cannot initialize inotify handle                                [ 0.040  6.210 ]
bio: create slab <bio-0> at 0                                          [ 0.000  6.210 ]
Bus freq driver module loaded                                          [ 0.030  6.240 ]
mxs_cpu_init: cpufreq init finished                                    [ 0.020  6.260 ]
audit: initializing netlink socket (disabled)                          [ 0.010  6.270 ]
type=2000 audit(0.060:1): initialized                                  [ 0.000  6.270 ]
VFS: Disk quotas dquot_6.5.2                                           [ 0.070  6.340 ]
Dquot-cache hash table entries: 1024 (order 0, 4096 bytes)             [ 0.010  6.350 ]
msgmni has been set to 245                                             [ 0.010  6.360 ]
alg: No test for stdrng (krng)                                         [ 0.010  6.370 ]
cryptodev: driver loaded.                                              [ 0.000  6.370 ]
Block layer SCSI generic (bsg) driver version 0.4 loaded (ma           [ 0.010  6.380 ]
io scheduler noop registered                                           [ 0.000  6.380 ]
io scheduler anticipatory registered                                   [ 0.010  6.390 ]
io scheduler deadline registered                                       [ 0.000  6.390 ]
io scheduler cfq registered (default)                                  [ 0.000  6.390 ]
BTCS interface driver                                                  [ 0.360  6.750 ]
mxs-duart.0: ttyAM0 at MMIO 0x80074000 (irq = 47) is a Debug           [ 0.010  6.760 ]
mxs-auart.0: ttySP0 at MMIO 0x8006a000 (irq = 112) is a mxs-           [ 0.010  6.770 ]
Found APPUART 3.1.0                                                    [ 0.000  6.770 ]
mxs-auart.1: ttySP1 at MMIO 0x8006c000 (irq = 113) is a mxs-           [ 0.000  6.770 ]
Found APPUART 3.1.0                                                    [ 0.010  6.780 ]
mice: PS/2 mouse device common for all mice                            [ 0.000  6.780 ]
MXS RTC driver v1.0 hardware v2.3.0                                    [ 0.010  6.790 ]
mxs-rtc mxs-rtc.0: rtc core: registered mxs-rtc as rtc0                [ 0.000  6.790 ]
Linux video capture interface: v2.00                                   [ 0.010  6.800 ]
IRQ 6/mxs-battery: IRQF_DISABLED is not guaranteed on shared           [ 0.020  6.820 ]
mxs watchdog: initialized, heartbeat 19 sec                            [ 0.010  6.830 ]
mxs-mmc: MXS SSP Controller MMC Interface driver                       [ 0.000  6.830 ]
```

```
        mxs-mmc mxs-mmc.0: mmc0: MXS SSP MMC DMAIRQ 82 ERRIRQ 96      [
                                                                   0.110  6.940 ]
 Advanced Linux Sound Architecture Driver Version 1.0.20.        [ 0.010  6.950 ]
ALSA device list:                                                 [ 0.000  6.950 ]
  No soundcards found.                                            [ 0.010  6.960 ]
mxs-rtc mxs-rtc.0: setting system clock to 1970-01-01 00:04:      [ 0.010  6.970 ]
Waiting for root device /dev/mmcblk0p2...                         [ 0.000  6.970 ]
mmc0: new SDHC card at address d523                               [ 0.060  7.030 ]
mmcblk0: mmc0:d523 SD04G 3.69 GiB                                 [ 0.000  7.030 ]
 mmcblk0: p1 p2                                                   [ 0.010  7.040 ]
EXT2-fs warning: mounting unchecked fs, running e2fsck is re      [ 0.060  7.100 ]
VFS: Mounted root (ext2 filesystem) on device 179:2.              [ 0.000  7.100 ]
Freeing init memory: 104K                                         [ 0.000  7.100 ]
                                                                  [ 0.200  7.300 ]
init started: BusyBox v1.15.0 ()                                  [ 0.010  7.310 ]
                                                                  [ 0.000  7.310 ]
starting pid 343, tty '': '/etc/rc.d/rcS'                         [ 0.010  7.320 ]
Mounting /proc and /sys                                           [ 0.050  7.370 ]
Setting the hostname to FSL AGL                                   [ 0.050  7.420 ]
Mounting filesystems                                              [ 0.060  7.480 ]
mount: mounting shm on /dev/shm failed: No such file or dire      [ 0.020  7.500 ]
mount: mounting usbfs on /proc/bus/usb failed: No such file       [ 0.650  8.150 ]
                                                                  [ 0.100  8.250 ]
starting pid 368, tty '': '/sbin/getty -L ttyAM0 115200 vt10      [ 0.000  8.250 ]
                                                                  [ 1.020  9.270 ]
                                                                  [ 0.000  9.270 ]
arm-none-linux-gnueabi-gcc (GCC) 4.1.2                            [ 0.010  9.280 ]
                                                                  [ 0.000  9.280 ]
root filesystem built on Tue, 24 Aug 2010 16:31:58 +0800          [ 0.000  9.280 ]
                                                                  [ 0.000  9.280 ]
Freescale Semiconductor, Inc.                                     [ 0.000  9.280 ]
                                                                  [ 0.000  9.280 ]
                                                                  [ 0.000  9.280 ]
                                                                  [ 0.000  9.280 ]
```

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK