**Uboot:**

1. Add Mfgtool config header file in : rpm\BUILD\u-boot\_xxx\_version\include\configs

         The file name should align with above config file (\_config).  Ex. mx51\_bbg\_mfg.h.  mx51\_bbg\_mfg here is prefix of mx51\_bbg\_mfg\_config.

         Generally, you can create it from existing config.

Following MACRO must be defined:

* + *#define CONFIG\_MFG*
  + *#define CONFIG\_CMDLINE\_TAG*
  + *#define CONFIG\_REVISION\_TAG*
  + *#define CONFIG\_SETUP\_MEMORY\_TAGS*
  + *#define CONFIG\_INITRD\_TAG*
  + *#define CONFIG\_BOOTDELAY 0*
  + *#define CONFIG\_BOOTARGS "console=ttymxc0,115200 "\  
                                                      "rdinit=/linuxrc"*
  + *#define CONFIG\_BOOTCOMMAND      "bootm $ {loadaddr}0x90800000"; 0x90800000 is initrd address, you should change it according to your platform memory configuration.*
  + *#define CONFIG\_ENV\_IS\_NOWHERE ;avoid read uboot command from storage*

If a new bsp arg need to be added, one can add it following the contents in CONFIG\_BOOTARGS, for instance:

* + *#define CONFIG\_BOOTARGS "console=ttymxc0,115200 "\  
                                                      "rdinit=/linuxrc arm\_freq=800"*

1. Reset USB when uboot runs to avoid host pc crash when multi boards download at the same time.

        Refence code: /board/freescale/mx51\_bbg/mx51\_bbg.c

* + *#ifdef CONFIG\_MFG*
  + */\* MFG firmware need reset usb to avoid host crash firstly \*/*
  + *#define USBCMD 0x140*
  + *int val = readl(OTG\_BASE\_ADDR + USBCMD);*
  + *val &= \~0x1; /\*RS bit\*/*
  + *writel(val, OTG\_BASE\_ADDR + USBCMD);*
  + *#endif*

**Kernel:**

Add new configuration for mfg firmware. Normally you can copy it from existing config.

1. Must build in USB and related driver, such as NAND,  SD Card ...
2. Must build in mass storage gadget class driver.
3. Must define CONFIG\_FSL\_UTP=y
4. Buit in initramfs support
5. Disable watchdog in i.mx50; For other platforms(i.mx25/51/53…), make sure watchdog is served

CONFIG\_WATCHDOG=y

CONFIG\_SOFT\_WATCHDOG=y

**LTIB:**

 Reference commit 07295302f85df2db74e5e1855e1fd8c6da2816fe

1. Configuration file:

         Add new configuration file in config\platform\imx, such as mx51\_bbg\_mfg\_config.

Please refer to config\platform\imx\mx51\_bbg\_mfg\_config as an example.

 Add config/platform/imx/imxXX\_updater.cf

* config/plaftorm/imx/main.lkc,   add uboot new configuration when PKG\_KERNEL\_UPDATER defined
* config/platform/imx/main.lkc,   add kernel new configuration when PKG\_KERNEL\_UPDATER defined
* you need choose "UUC" package.
* add new cf at config/platform/imx/preconfigs.lkc,  default imx51\_updater.cf if (PCF\_PLATFORM\_IMX51 && PCF\_UPDATER\_PROFILE) is selected.
* ./ltib, choose your platform and "mfg firmware profile",  then config/platform/imx/.config imxXX\_updater.cf

**Addition:**

**MFG Linux firmware Structure Demonstration**

**UTP Device Driver**

Kernel driver deals UTP protocol. It locates in:

linux-2.6-xxx/drivers/usb/gadget/file\_storage.c

linux-2.6- xxx /drivers/usb/gadget/fsl\_updater.c

utp\_handle\_message is the entry of each UTP command. The command is defined in file\_storage.c

It is a char (in fact, it is registered as misc driver) device driver, and combines with usb mass storage driver file\_storage. It mainly handles UTP messages according to freescale UTP protocol and kinds of message types. It supplies the read/write interface for user application to get the UTP message and returns its execution results.

There are two wait queues, utp\_context.wq and utp\_context.list\_full\_wq. The utp\_context.wq is used to sync read/write UTP message, and the utp\_context.list\_full\_wq is used to avoid there are so many unhandled UTP PUT( the data from the host to device) message.

At the very beginning of utp\_handle\_command defined in uu.c, a function utp\_can\_busy is called to decide if a command need a busy stage. If the command needs it, then UUC sends busy state to UTP device.

There are two double-linked lists for read/write UTP message, the UTP message will be listed at these two double-linked lists.

**UUC**

Uuc is responsible for command execution in a shell. It locates in: uuc-xxx-version/uu.c

It is the user program that decodes the UTP message and executes it. It is blocked when reading UTP message. The device driver will unblock the uuc process after it gets the UTP message from the host.

The uuc returns error message with non-zero value in UTP message to device driver.

Below is the work flow for UTP message handling, we take EXEC and PUT data as an example:

