MBoot User Guide

Revision History

Document Version	Date	Author	Comments
V 0.1	2009/04/16	Nick.Lai	Create
V 0.2	2009/05/05	Terry.Tsai	Update
V 0.3	2009/07/16	CM.Chen	Update
V 0.4	2010/05/31	CP.Hsu	Add Chunk Header Format
V 0.5	2011/01/14	CP.Hsu	Add Version Rule of SBoot & UBoot
V 0.6	2011/01/19	Cyber.Chang	Add SPI flash Layout
V 0.7	2011/03/26	Eric.Peng Cyber.Chang	Add MIU Setting / Environment Variable
V 0.8	2011/11/29	Cyber.Chang	Add Register Address Translation
V0.9	2012/02/08	ERIC.Wu	Update chunk header statu
V1.0	2012/10/8	Eric.Wu	New architecture for u-boot2011.06

Acronyms

Acronym	Description
MBoot	Mstar Bootloader
SBoot	Small Bootloader
UBoot	Universal Bootloader

Content

1. MBoot

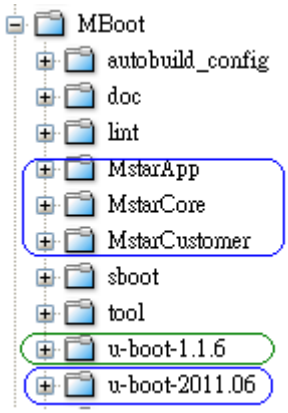
This chapter would provide an introduction of MBoot including overview, code tree, build flow, and setup environment

1.1 Overview of MBoot

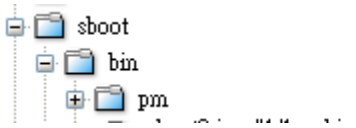
MBoot is the MStar boot loader which is composed of pm.bin, sboot.bin anduboot.binMBoot is used to boot up system, so it would initialize H/W setting and then load Linux kernel and applications from NAND flash to DRAM. Exept booting system, MBoot also has charge of software upgrade(oad/net/usb), displaying the booting logo and playing the booting music.

1.2 MBoot tree

MBoot source code is placed in “//DAILEO/MBoot”.



pm.bins are stored in this path.

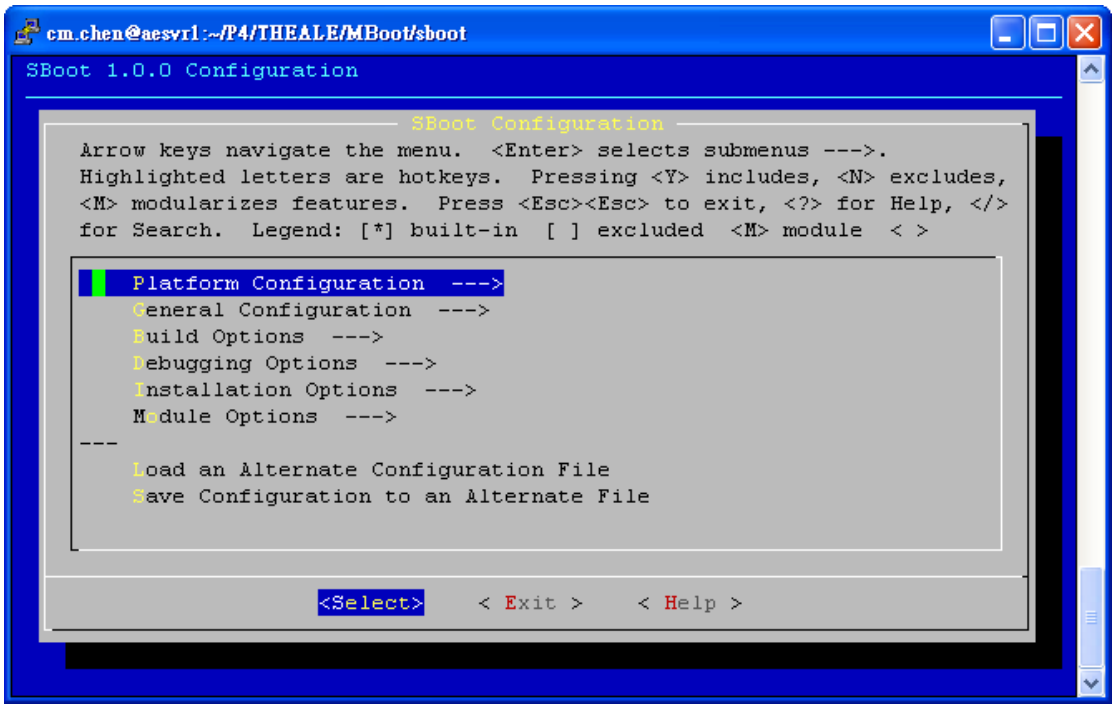


On this source tree, we can see the two u-boot related folders. “u-boot-1.1.6” is for Mstar’s old SOC and “u-boot-2011.06” is for Mstar’s new SOC. Because uboot is an open source code, we wish Mstar’s code can be separated from the open source codes, it can be easy to maintain. So we crated three folders “MstarApp”, “MstarCore” and “MstarCustomer” for u-boot-2011.06. In these three folders, all functions are designed from Mstar’s engineers, and these functions are all based on u-boot-2011.06. “sboot” folder is for sboot.bin. The main task for sboot.bin is low level hardware initialization. pm.bin exists in this source three in binary format. The binary files are all stored in “sboot/bin/pm”.

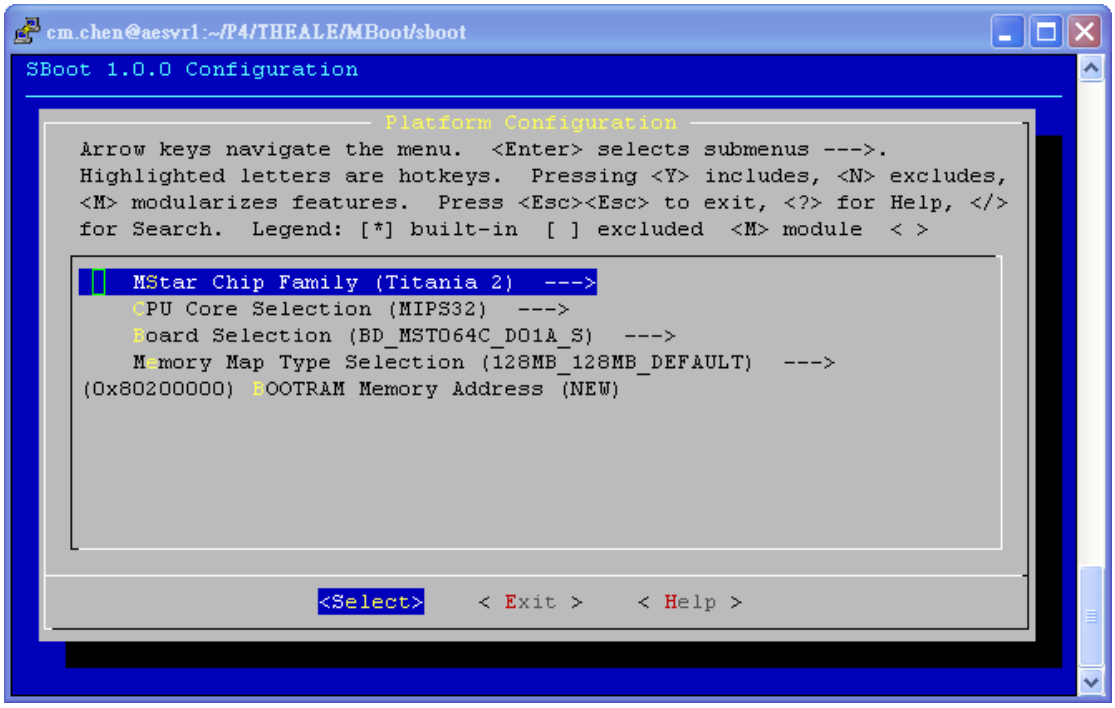
1.3 How to build MBoot

The following build flow of MBoot shows how to generate the MBoot bin file.

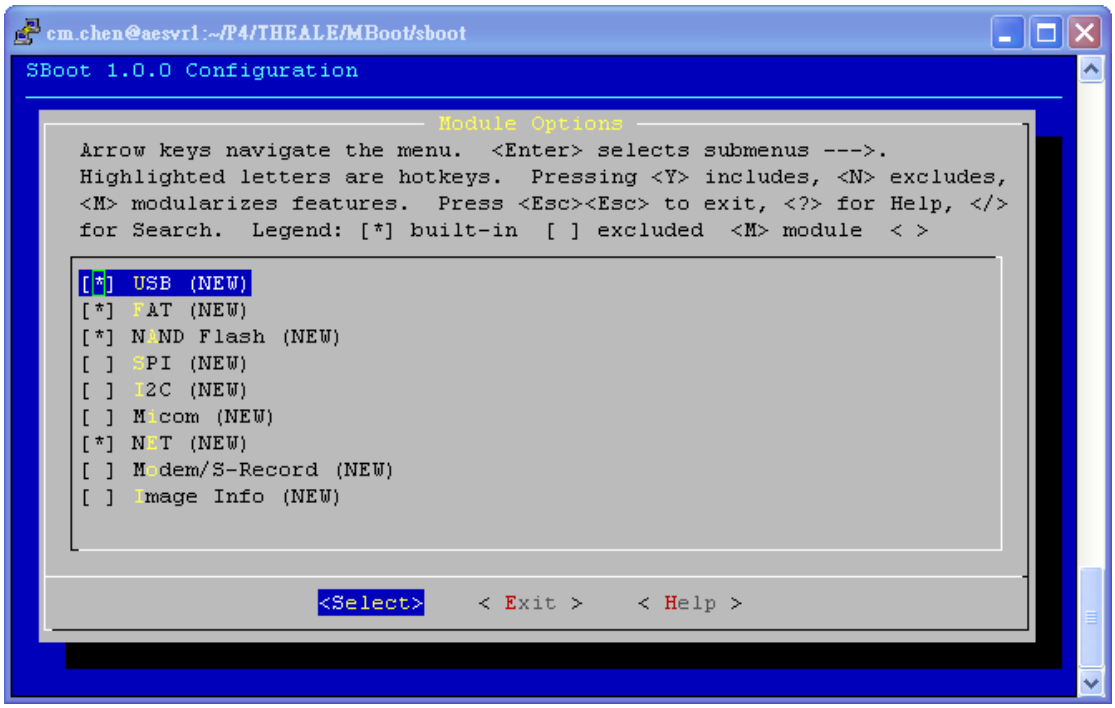
- Change working directory to **MBoot/sboot**
- "**make menuconfig**" to setup SBoot configuration menu



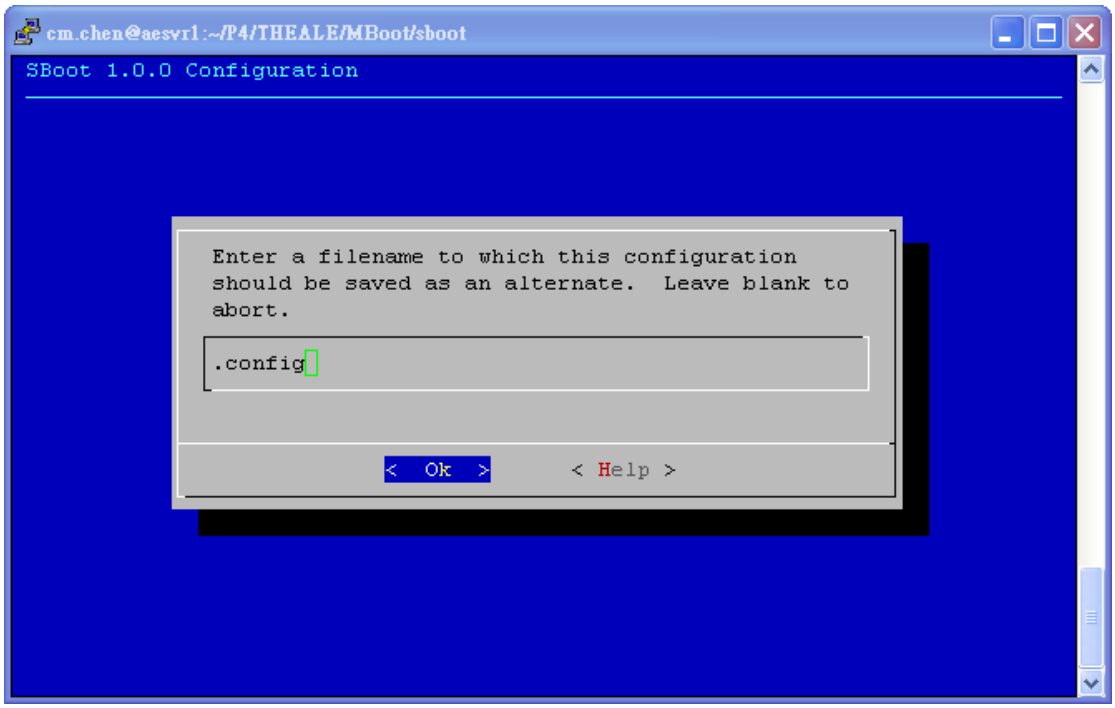
- Select and enter "Platform Configuration"
- Select Chip, Target board, and Memory Map type and then Exit



- Select and Enter “Module Options”
- Select which modules are included into UBoot



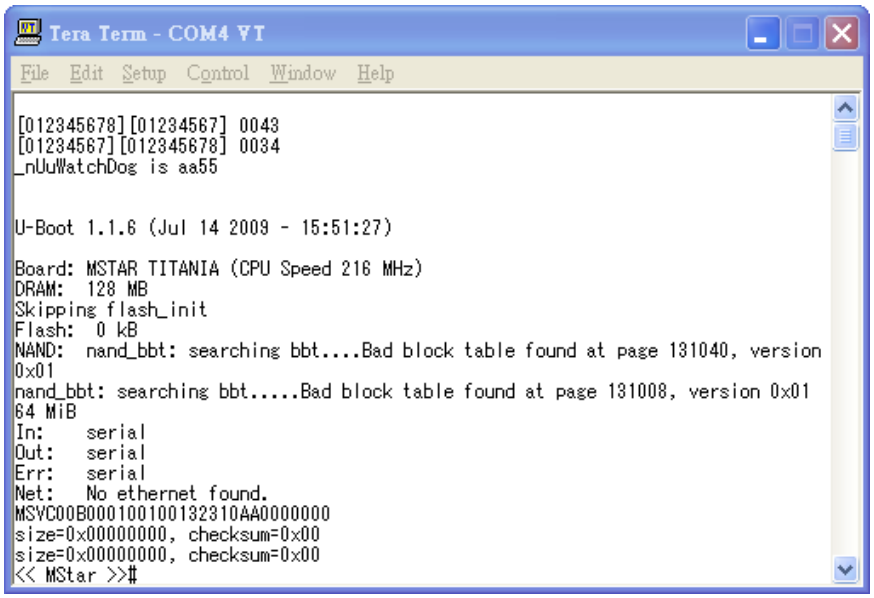
- Select and enter “Save Configuration to an Alternate File”
  ➔Save configuration file to “**.config**”



- Exit SBoot configuration menu
- "**make**" to compile UBoot and SBoot source code and then merge their bin files into MBoot bin file.
- The MBoot bin file would be generated as "**MBoot/sboot/out/mboot.bin**"
- Use ISP tool to burn-in the MBoot bin file into SPI Flash on the target board

1.4 How to use MBoot, and burn program through the tftp server

After the target board rebooted, the prompt "**<< MStar >>#**" would be displayed on console. At this moment, we can type some commands on console to interact with UBoot.



To load Linux kernel and application images through Ethernet interface (especially on developing/debugging stage), TFTP server is needed. The following environment variables should be set for LAN setting

- setenv ipaddr 172.16.89.199 (Based on your local IP address, and plus 1 on third fild.)
- setenv serverip 172.16.88.60 (your local IP address)
- macaddr 00 XX YY 00 00 01 (XXYY is your extension number.)
- saveenv (save environment into NAND Flash)

Type "**mstar**" command on console to download Linux kernel and applications to DRAM and then write into NAND flash.

1.5 SPI Flash Layout.

Now there are two booting modes on Mstar's TV sets, so we have two image layouts for SPI. This two booting modes are "Hosekeeping booting" and "PM51 booting".

Hosekeeping booting mode:

offset	Code	size
0x00000	sboot.bin	0x10000
0x10000	pm.bin	0x10000
0x20000	ChunkHeader	0x80
0x20400	u-boot.bin	
Last 7th bank	bootlog + boot music	0x40000
Last 3th bank	signature for nand image	0x10000
Last 2th bank	signature for nand image	0x10000
Last 1th bank	ENV1	0x10000
Last 0th bank	ENV2	0x10000

PM51 booting mode:

offset	Code	size
0x00000	pm.bin	0x10000
0x10000	sboot.bin	0x10000
0x20000	Customer Key bank	0x10000
0x30000	ChunkHeader	0x80
0x30400	u-boot.bin	
Last 7th bank	bootlog + boot music	0x40000
Last 3th bank	signature for nand image	0x10000
Last 2th bank	signature for nand image	0x10000
Last 1th bank	ENV1	0x10000
Last 0th bank	ENV2	0x10000

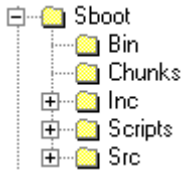
Actually, PM51 booting mode is only used on secure booting system, So, if you heard “secure booting”, the image layout is the same with “PM51 booting” mode. “Secure booting” is not a topic in this document. If you want to learn more about it, you have study other related documents.

2. SBoot

SBoot (Small Bootloader) is a system boot up entry that used to initialize the CPU, Cache, hardware registers, etc. It will jump to UBoot entry after all of settings are done.

2.1 Overview of SBoot

The directory sboot includes the following subdirectories: bin, inc, scripts, out and src.



bin：UBoot output bin file and memory map

inc：

1. board definition：Add or modify board definition in MBoot\sboot\inc\xxx\board

2. memory map：Add or modify mmap in MBoot\sboot\inc\xxx\board\mmap

scripts：Script files for menu configuration window

out：MBoot output bin file

src：SBoot source code files which are related with chip drivers

2.2 SBoot setup

- SBoot environment setup：
  Chip ➔ MBoot\sboot\inc\titania2\board\chip
  Board ➔ MBoot\sboot\inc\titania2\board
  Mmap ➔ MBoot\sboot\inc\titania2\board\mmap

- Boot entry and address in sboot.lds file
  - Entry(BOOT_Entry)
  - Start address: 0xBFC00000
  - boot.o should be put in the first section and its code size can't be larger than 3Kbytes

```
MEMORY
{
    boot :   ORIGIN = 0xBFC00000,          LENGTH = 3K
    rom :    ORIGIN = 0x94000000+0xC00,    LENGTH = 8K
    ram :    ORIGIN = 0x80200000,          LENGTH = 128K
    sram :   ORIGIN = 0x84000000,          LENGTH = 1K
}

SECTIONS
{
    .text1 :
    {
        KEEP(*boot.o(.text*))
    } > boot
    .text2 : AT ( LOADADDR(.text1) + SIZEOF(.text1) )
    ……
}
```

- Enable SBoot cache：
  bal BOOT_InitCache

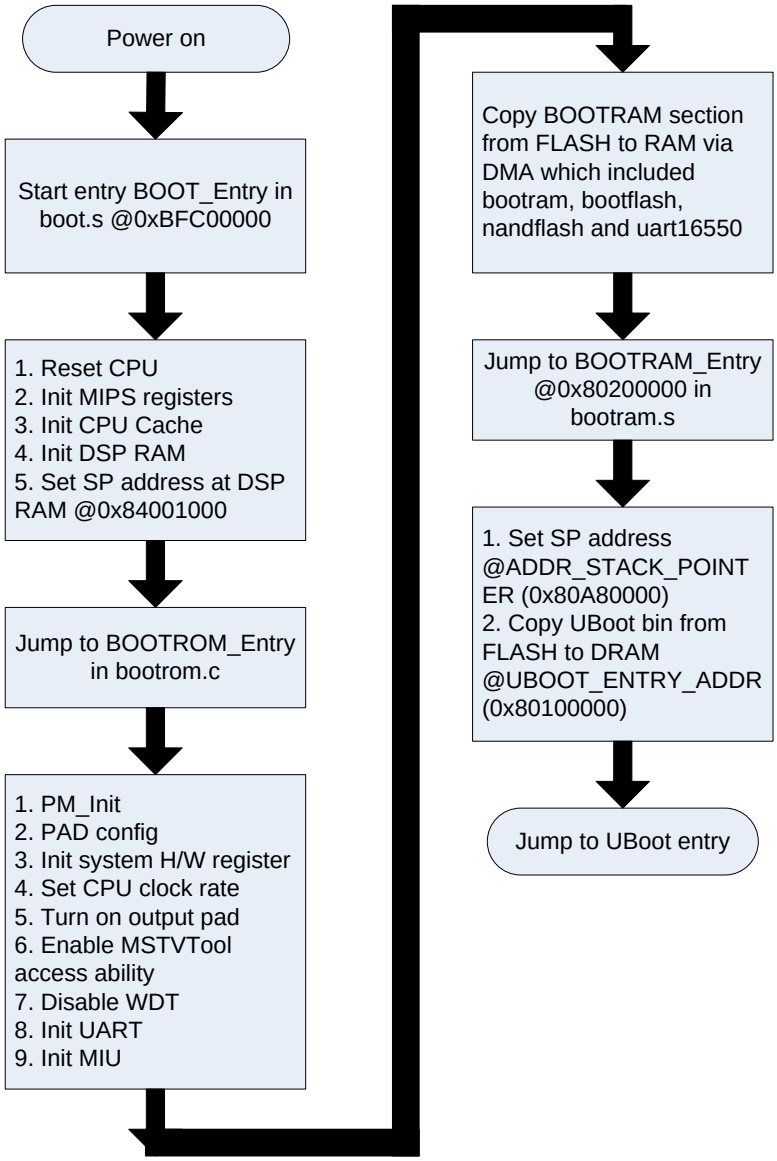
- Hardware register setup：
  MBoot\sboot\src\xxx\bootrom.c

- Setup CPU clock, UART baud rate：
  MBoot\sboot\src\xxx\boot.inc

- Jump to UBoot：
  MBoot\sboot\src\xxx\bootram.s
  ➔ BOOT_CopyBootRAM ➔ BOOTRAM_Entry ➔ UBoot entry

2.3 SBoot boot flow

SBoot flow chart is depicted as the following. SBoot starts from SPI Flash @0xBFC00000 to initialize H/W setting, execute some C code functions of H/W setting by using the DSPRAM space, copy bootram section from SPI Flash to DRAM to copy UBoot from SPI flash to DRAM, and finally jump to UBoot entry.



Version Rule of SBoot

- How to set SBoot version：

Set the change list number of SBoot into the following place before sboot.bin is built.

//MBoot/sboot/src/version.h

```
//-------------------------------------------------------------------------------------------------
// Version Control
//-------------------------------------------------------------------------------------------------
#define MSIF_TAG                {'M','S','V','C'}               // MSVC
#define MSIF_CLASS              {'0','0'}                       // DRV/API (DDI)
#define MSIF_CUS                {'0','0','S','3'}
#define MSIF_QUALITY            0
#define MSIF_MOD                {'S','B','T','_'}
#define MSIF_DATE               {'Y','Y','M','M','D','D'}
#define MSIF_SBT_CHANGELIST     {'0','0','2','8','4','7','1','2'}   //P4 ChangeList Number

#define SBT_VER             /* Character String for SBOOT version          */  \
    MSIF_TAG,               /* 'MSIF'                                      */  \
    MSIF_CLASS,             /* '00'                                        */  \
    MSIF_CUS,               /* '00S0'                                      */  \
    MSIF_QUALITY,           /* 0                                           */  \
    MSIF_MOD,               /* 'SBT_'                                      */  \
    MSIF_DATE,              /* 'TTMMDD'                                    */  \
    MSIF_SBT_CHANGELIST,    /* CL#                                         */  \
    {'0','0','0'}

typedef union _MSIF_Version
{
    struct _DDI
    {
        U8              tag[4];
        U8              type[2];
        U16             customer;
        U16             model;
        U16             chip;
        U8              cpu;
        U8              name[4];
        U8              version[2];
        U8              build[2];
        U8              change[8];
        U8              os;
    } DDI;
    struct _MW
    {
        U8              tag[4];
        U8              type[2];
        U16             customer;
        U16             mod;
```

```
        U16                     chip;
        U8                      cpu;
        U8                      name[4];
        U8                      version[2];
        U8                      build[2];
        U8                      changelist[8];
        U8                      os;
    } MW;
    struct _APP
    {
        U8                      tag[4];
        U8                      type[2];
        U8                      id[4];
        U8                      quality;
        U8                      version[4];
        U8                      time[6];
        U8                      changelist[8];
        U8                      reserve[3];
    } APP;
} MSIF_Version;

const MSIF_Version _sbt_version = {
    .APP = { SBT_VER }
};
```

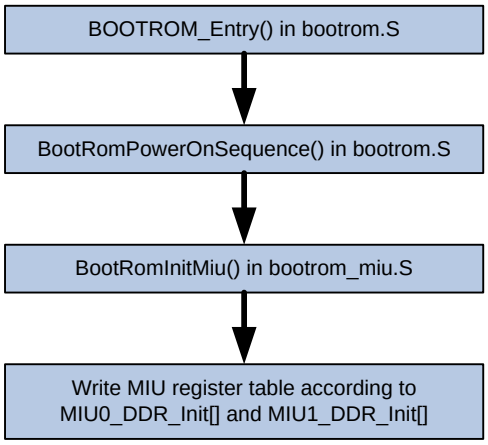
- How to get SBoot version in main application：

Read the last 32bytes in sboot.bin. (sboot.bin would be padded to be 64Kbytes, and SBoot version is kept in the last 32bytes of these 64Kbytes by off-line executable tool: pad_version.)

```
//MBoot/sboot/Makefile
   out/sboot.bin: out/sboot.elf
      $(Q)$(OBJCOPY) -O binary -I elf32-little $< $@
      $(Q)$(HOSTCC) -o ./scripts/pad_version ./scripts/pad_version.c
      $(Q)./scripts/pad_version $@
```

2.5 MIU Setting

- MIU init flow



You can configure 3 settings of MIU.

– Memory frequency

  MIU0

  ➔ make menuconfig → Platform Configuration → Memory Frequency Selection

  MIU1

  ➔ Modify MIU1_DRAM_FREQ in the board config. E.g. As in BD_MST008B_10ATX_10405.h

– Memory size

  ➔ make menuconfig → Platform Configuration → Memory Map Type Selection

  NOTE: If meets boot up problem, please consult CAE to check if needed to modify MIU0_DDR_Init[] or MIU1_DDR_Init[].

  E.g. _RV32_2(0x101202, 0x03a3) for MIU0, _RV32_2(0x100602, 0x02a2) for MIU1.

– MIU interface

  ➔ Set MIU_INTERFACE / MIU1_INTERFACE to DDR2_INTERFACE_BGA or DDR3_INTERFACE_BGA in the board config.

- An example to configure the memory setting at board define

ex: //MBoot/sboot/inc/titania13/board/BD_MST008B_10ATX_10405.h

You should adjust the MIU type (DDR3 or DDR2) and MIU frequency as following:

```
//------Memory Setting-------------------------------------------------------
#define BOOTUP_MIU_BIST                 1
#ifndef MEMORY_MAP
#define MEMORY_MAP
MMAP_128_128MB//MMAP_64MB
```

```
#endif
#define MIU_INTERFACE                   DDR3_INTERFACE_BGA
//DDR3_INTERFACE_BGA

#define MIU1_INTERFACE                  DDR2_INTERFACE_BGA
//DDR3_INTERFACE_BGA
#define MIU1_DRAM_FREQ                   800 //950 / 1066 / 1300 / 1600

#define ENABLE_AUTO_PHASE
//------Memory Setting-------------------------------------------------------
```

p.s. MMAP is depended on .config file, not here.

- Find the corresponding MIU header file of board：

ex: //MBoot/sboot/src/titania13/include/MIU_MST008B_10ATX_10405.h

Now you can take the miu setting table retrieve from CAE, and modify the value of register. Ex:

```
_RV32_2( 0x101204, 0x000A ),    //if I64Mode =0x8b else =0x0b
_RV32_2( 0x101206, 0x0434 ),    //refresh cycle=0x50
_RV32_2( 0x101208, 0x1899 ),    //reg_tRCD
_RV32_2( 0x10120a, 0x2155 ),    //reg_tRRD
_RV32_2( 0x10120c, 0x95a8 ),    //reg_tWL
_RV32_2( 0x10120e, 0x406b ),    //tRFC
_RV32_2( 0x101210, 0x1a50 ),    //MR0
```

2.6 Register Address Translation

- MIPS

[RIU offset address(16bits) X 2+ bank address] X 2 + 0XBF000000

- ARM

[RIU offset address(16bits) X 2+ bank address] X 2 + ARM RIU start address

- 8051

RIU offset address(16bits) X 2+ bank address

3. Chunk Header Format

sboot.bin uses some information that keep in the chunk header for jumping to next application program (Here the next application program means UBoot for Linux platform or means Chakra2 main application for Non-OS platform).

The chunk header format goes as follows.
The chunk header's base address depends on booting mode. If booting mode is "BOOTING_FROM_OTP_WITH_PM51" or "BOOTING_FROM_EXT_SPI_WITH_PM51", base address is 0x30000. If booting mode is "BOOTING_FROM_EXT_SPI_WITH_CPU", base address is 0x20000.

	0x00	0x04	0x08	0x0C
0x00	UBOOT_ROM_START	UBOOT_RAM_START	UBOOT_RAM_END	UBOOT_ROM_END
0x10	UBOOT_RAM_ENTRY	Reserved1	Reserved2	BINARY_ID
0x20	LOGO_ROM_OFFSET	LOGO_ SIZE	SBOOT_ROM_OFFSET	SBOOT_ SIZE
0x30	SBOOT_RAM_OFFSET	PM_ROM_OFFSET	PM_SIZE	PM_RAM_OFFSET
∈ 0x40	SECURITY_INFO _LOADER_ROM_OFFSET	SECURITY_INFO _LOADER_SIZE	SECURITY_INFO _LOADER_RAM_OFFSET	CUSTOMER_KEY_BANK_ ROM_OFFSET
0x50	CUSTOMER_KEY_BANK_S IZE	CUSTOMER_KEY_BANK_R AM_OFFSET	SECURITY_INFO _AP_ROM_OFFSET	SECURITY_INFO _AP_SIZE
0x60	UBOOT_ENVIRONMENT_R OM_OFFSET	UBOOT_ENVIRONMENT_SI ZE	DDR_BACKUP_TABLE_RO M_OFFSET	POWER_SEQUENCE_TAB LE_ROM_OFFSET
0x70	UBOOT_POOL_ROM_OFFS ET	UBOOT_POOLSIZE		
0x80				

4. UBoot

UBoot (Universal Boot loader) is a free open source under GPL which is developed by DENX.

4.1 Overview of u-boot-1.1.6

MStar UBoot is based on u-boot version 1.1.6 that used a simple command line interface (CLI) to interact with user, usually over a serial console port.

UBoot applies some features such as,

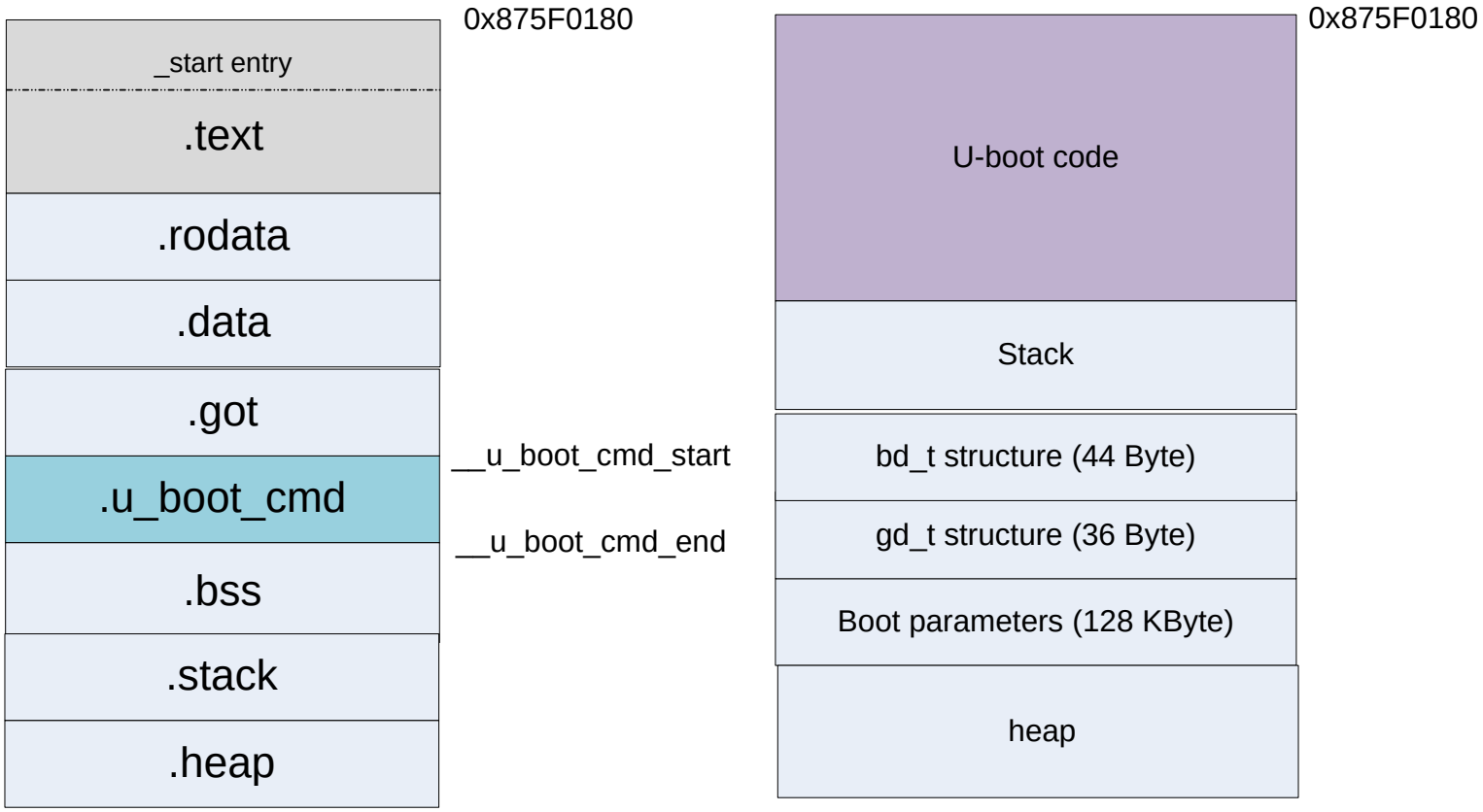
- Accept commands on console to setup environments and execute the system operation
- Initialize the system
- Load the kernel and application to DRAM through Ethernet interface
- Write the kernel and application into NAND Flash
- Set the kernel arguments
- Uncompress the kernel
- Pass the boot arguments to kernel to startup

The u-boot-1.1.6 directory includes the following subdirectories: board, common, cpu, disk, doc, drivers, fs, include, lib_generic, lib_mips, net, post, rtc, and tools.

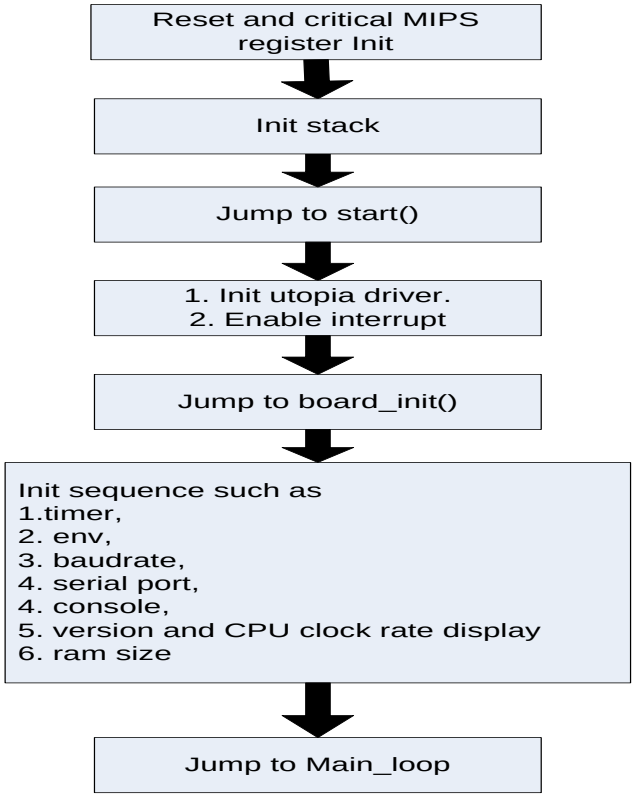
Directory	Description
Board	board define
common	Command and environment setup
Cpu	MIPS cpu start.s
Disk	Reports device info to the user
Doc	Documents
drivers	NAND driver, emac driver, usb driver…
Fs	ext2, fat, fdos, jffs2, …
include	header file
lib_generic	generic lib
lib_mips	MIPS lib
net	net, tftp, …
post	Power on self test
rtc	Real time clock

4.1.2 UBoot Image

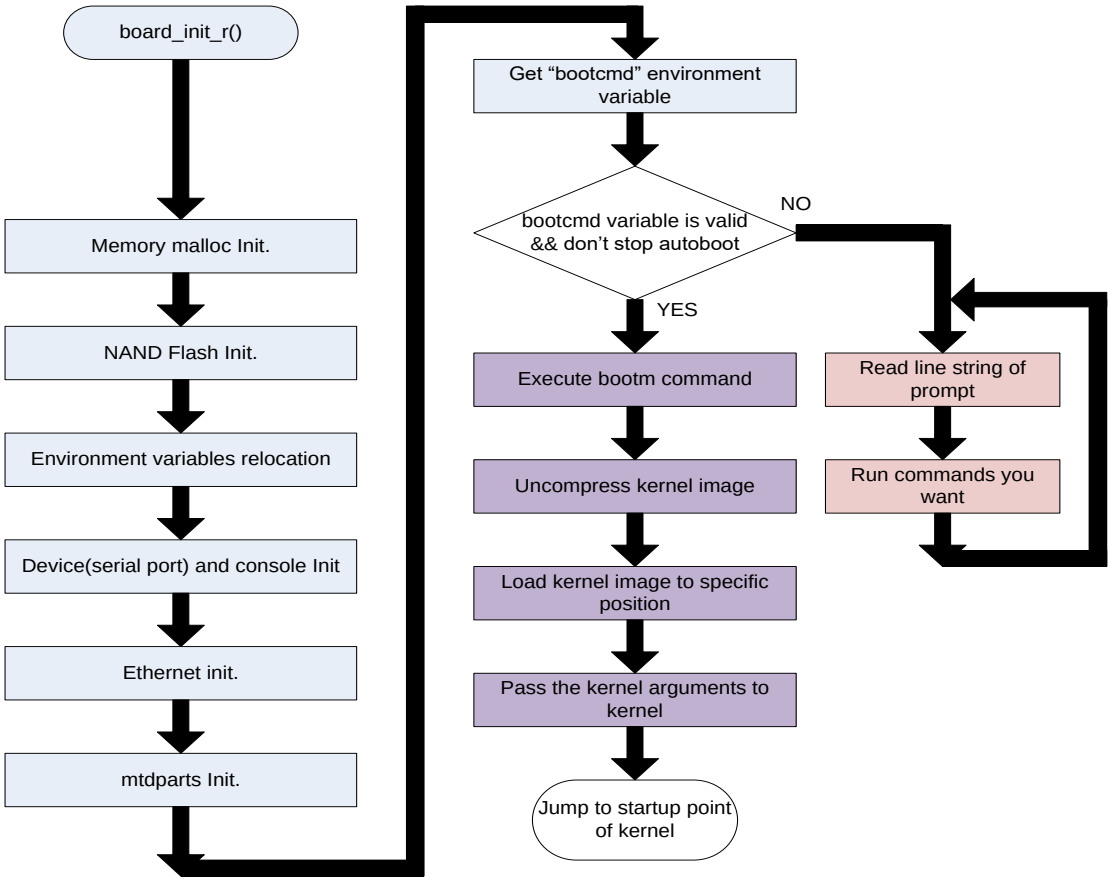
- Image layout:
  - Entry is _start @0x875F0180 in vectors.S
  - .u_boot_cmd: all the commands procedures are put in this section



4.1.3 UBoot startup



4.1.4 UBoot main flow



4.2 Overview of u-boot-2011.06

4.3 Auto boot sequence

Run command with “bootcmd”

- bootcmd=nand read.e 81000000 KL 300000; bootm 81000000;

  1. Read kernel image from NAND Flash
  2. Call the function - do_bootm()
     - ➔ Load image header from 0x81000000
     - ➔ Get Image real data address (entry - header offset)
     - ➔ Decompress image to the data load adder space (hdr->ih_load)
     - ➔ Read MStar bin from NAND Flash via command “cpmsbin”
  3. Call the function - do_bootm_linux()
     - ➔ Get the kernel entry point (hdr->ih_ep)
     - ➔ Initialize linux parameters:

       “Bootargs” are Linux kernel required parameters.

       These parameters are put in “Boot parameters section”.
     - ➔ Initialize environment set:

       Memory size, initrd address and size, flash address and size
     - ➔ Jump to kernel entry point to start Linux kernel code

4.4 Version Rule of UBoot

- How to set UBoot version：

Set the change list number of UBoot into the following place before mboot.bin is built.

//MBoot/u-boot-1.1.6/include/ms_ver.h

```
static char MBOOT_VBuf[32] = {'M', 'S', 'V', 'C', '0', '0',
                              'B', '0',
                              BuildNum0,
                              BuildNum1,
                              BuildNum2,
                              BuildNum3,
                              BuildNum4,
                              BuildNum5,
                              '0', '0', '2', '0', '8', '7', '6', '8',
                              'T', 'H', '0', '0', '0', '0', '0', '0', '0',
                              'T'};
```

- How to get UBoot version in main application：

Call prom_getenv("MBoot_version") to get UBoot version which is kept in environment variables.

//MBoot/u-boot-1.1.6/lib/lib_mips/mips_linux.c

```
    /*send MBoot version to linux kernel by seting environment*/
```

```
    /*please use prom_getenv("nev_name") function to get this information (e.g.
prom_getenv("MBoot_version");)*/
    /*prom_getenv() define in
\\RedLion\2.6.28.9\arch\mips\mips-boards\generic\init.c */
    memcpy(MBoot_Ver,MBOOT_VBuf,32);
    MBoot_Ver[32]='\0';
    linux_env_set ("MBoot_version", MBoot_Ver);
```

4.5 Environment Variable

You can use UBoot environment variable to store program settings. They would be stored in a non-volatile memory, SPI flash or NAND flash. You can configure it by: make menuconfig → Module Options → Env Config.

- UBoot commands to manage environment variable
  - printenv
    → Print values of all environment variables
  - setenv
    → Set environment variable 'name' to 'value ...'
  - saveenv
    → Update environment variables to the non-volatile memory.
- Environment variable – bootcmd

Defines the command to be executed in the end of UBoot stage. Mostly it is a command to bring up Linux kernel.

- Environment variable – bootargs

Defines the boot parameters passed to Linux kernel. Kernel would change its behaviors according to them. E.g. Modify console could enable/disable printing messages after kernel boot up.

- Enable printing message
  setenv bootargs console= ttyS0,115200 ubi.mtd=3,2048 root=ubi:RFS rootfstype=ubifs ro LX_MEM=0x2000000 EMAC_MEM=0x100000 DRAM_LEN=0x10000000 LX_MEM2=0x66381000,0x1C00000 BB_ADDR=0x7FFF000 mtdparts=edb64M-nand:256k(NPT),256k(KL_BP),5m(KL),121m(UBI),-(NA)
- Disable printing message
  setenv bootargs console=/dev/null ubi.mtd=3,2048 root=ubi:RFS rootfstype=ubifs ro LX_MEM=0x2000000 EMAC_MEM=0x100000 DRAM_LEN=0x10000000 LX_MEM2=0x66381000,0x1C00000 BB_ADDR=0x7FFF000 mtdparts=edb64M-nand:256k(NPT),256k(KL_BP),5m(KL),121m(UBI),-(NA)

- Environment variable – bootdelay

Defines the time (second) to wait before execute bootcmd.
E.g. setenv bootdelay 0
E.g. setenv bootdelay 3

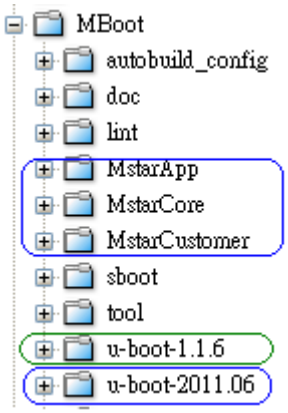
- Environment variable – UARTOnOff

Defines if uart on or off. (Default on)
E.g. setenv UARTOnOff on
E.g. setenv UARTOnOff off

5. New architecture for u-boot-2011.06

5.1 Description for three folders

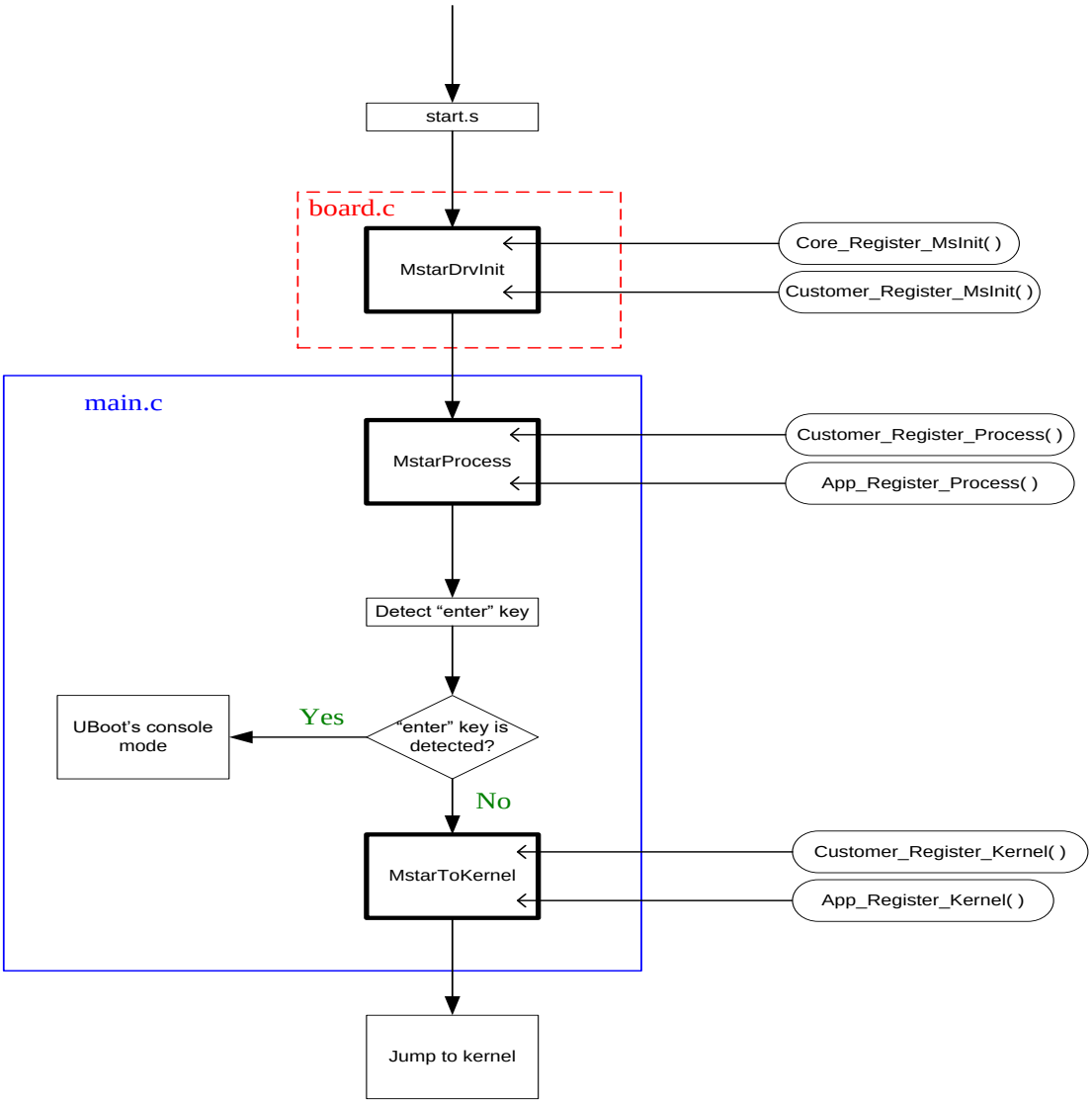
In order to maintain easily, we created three folders for Mstar's and Customer's functions.



"MstarCore" is for driver, "MstarApp is for application, ex: software upgrade, osd, display booting logo, and booting music..etc.
"MstarCustomer" is for customization. All functions in these three folders are based on u-boot-2011.06.

5.2 Entry point.

There are three entry points for adding our functions into uboot's booting flow.



All of mstar's or customer's function are added from these three points. "Core_xxx" means that it's for driver owner. "App_xxx" means that it's for application owner. "Customer_xxx" means that it's for customization.

5.2.1 Register command to cmdTable

We have entry points for different phase now, and we wish every developer can follow the Mstar's rules in this entry points. We don't use function call in these entry points, we only register command in it. For example:

```
void Customer_Register_Process(void)
{

    Add_Command_Table ("if_boot_to_pm" , 0, STAGE_PROCESS);
    //Register customter Process
    #if defined(CONFIG_ENV_IS_IN_SPIFLASH)&&defined(CONFIG_LOAD_ENV_FROM_SN)
    Add_Command_Table ("envload" , 0, STAGE_PROCESS);
    #endif
    #if defined (CONFIG_MBOOT_VERSION)
    Add_Command_Table ("mbootver_set" , 0, STAGE_PROCESS);
    #endif
    #if CONFIG_MSTAR_STR_ENABLE //checkstr always lowermost.
    Add_Command_Table ("checkstr" , 0, STAGE_PROCESS);
    #endif


}
```

We package the customization functions into different commands, and then register these commands through function "Add_CommandTable". If we follow this rule, we can get some advantages.

1. We can test customization behaviors throng commands on uboot's console mode.
2. We can use command "showtb" to double check which commands are registered. It's very good when we get a failed set.

5.3 Services

Now, we have already prepaed some command and functions for customization, like usb upgrade, oad upgrade, network upgrade..etc. We can use these basic services to help our customer developing their specific.

5.4 How to pacage a new command

It is very easy to add a new command on uboot's environment. We can refer to MsUtility.c and cmd_MsUtility.c, we can get many examples from this file.

```
int do_spi2usb ( cmd_tbl_t * cmdtp, int flag, int argc, char * const argv[])
{
    ulong u32SPIAddr=0;
```

```
U_BOOT_CMD(
    spi2usb, 4, 0, do_spi2usb,
    "Read data from spi to usb",
    "[spi offset] [length] [output file name]\n"
);
```

5.5 How to add a new file in these three folders.

1. Cerate a source file in specific folder.

   For example, If we want to add a new file in MstarCustomer, we have to create a new c file in /MstarCustomer/src

2. Edit /MstarCustomer/makefile

   Add our new file to COBJS

```
COBJS    =
#MstarCustomer:src
COBJS += ./src/MsCustomerRegister.o
COBJS += ./src/CusSystem.o
COBJS += ./src/CusUpgrade.o
COBJS += ./src/CusUpgradeUtility.o
COBJS += ./src/CusOsd.o
COBJS += ./src/CusPM.o
#MstarCustomer:cmd
COBJS += ./cmd/cmd_CusSystem.o
COBJS += ./cmd/cmd_CusUpgrade.o
COBJS += ./cmd/cmd_CusUpgradeUtility.o
COBJS += ./cmd/cmd_CusPM.o
```

3. If we have head file, place the haed file in /MstarCustomer/Include