i.MX6 U-Boot

imx6, u-boot, uboot, bootloader

Description

U-Boot is the bootloader for all i.MX6 devices. The bootloader first loads the devicetree, kernel and ramdisk, and then executes the kernel providing it with additional information such as the name of the device holding the root filesystem. U-Boot supports the ext2/3/4 filesystems as well as fat. Since our 2018.01 release, USB, SATA, eMMC and microSD are supported for both reading and writing.

General Operation:

The first thing that the i.MX6 does on power-on is load executable code from a predefined location. In terms of U-Boot this first piece of code is the SPL. Our devices can load it from either microSD, eMMC, SATA or SPI. This selection can be made by either fusing the SoM or adjusting the boot select jumpers if populated.

What happens next is decided by the SPL. It brings up the most important subsystems of the SoC, and then proceeds with loading the next stage of u-boot at predefined offsets from predefined storage. This is decided at compile-time.

Fallback:

The i.MX6 can also load code from USB using the Serial Download Protocol. When

- the SoM isn't fused and
- no bootable code was found

the SoM falls back to the SDP protocol on the USB OTG port automatically.

Furthermore when the SPL has been loaded and does not find a usable u-boot binary, it too will fall back to SDP on the OTG port allowing for recovery. This is also true for fused SoMs unless explicitly disabled at compile-time.

Success Indicator:

The Cubox-i has an LED that is off by default and turned on by u-boot. So if the LED turns red, U-Boot was loaded successfully.

Download Binaries

We are automatically building binaries whenever code is pushed to our U-Boot repository on github, currently tracking the **v2018.01-solidrun-imx6** branch. Please find the results at https://images.solid-build.xyz/IMX6/U-Boot/.

Installing to removable storage

microSD

This section assumes that you have a device running linux, and the target sdcard attached to it. This can be any device! Optionally create an MBR partition table, and any partitions you may want.

The Boot-ROM searches for the SPL after the first 1024 bytes. The SPL then looks for the full u-boot binary at both 69k and 42k. The dd command can be used for writing SPL and u-boot to these locations on your microSD card. Substitute sdX by the device node of your sdcard.

```
dd if=SPL of=/dev/sdX bs=1k seek=1 conv=sync
dd if=u-boot.img of=/dev/sdX bs=1k seek=69 conv=sync
```

Note - Take your time while identifying where your designated SD-Card is mapped on your linux system. Failure to do so can result in overwriting an arbitrary disk on your system!

Booting from USB (OTG)

An unfused i.MX6 SoM will fall back to booting from USB when it has not found any bootable code on the attached storage. Therefore it can be used to deploy software to a new system. Since the HummingBoard 2 there are also boot select jumpers that allow explicitly selecting USB as the boot source. The particulars can be found here on the HummingBoard page.

Identify the OTG port

To quote our developer Jon: “it is the top port next to the Ethernet jack” More formally it is the top port on the U5 header. This holds true of all our i.MX6 based boards.

Build an OTG cable

Essentially a straight Male to Male USB-A cable would accomplish the task. However it is highly recommended to build a custom cable with a large resistor on the VCC line to protect both your PC and your board from destruction. For this task some skills in electrical engineering are required! Hint: Both the voltage, and the maximum allowed current are defined by the USB standard.

imx_usb_loader

This application implements the Serial Download Protocol that the i.MX6 Boot-ROM uses to communicate. It is available on the github account of boundarydevices. Download and compile:

- git clone https://github.com/boundarydevices/imx_usb_loader
- cd imx_usb_loader

- make

The final binary is called imx_usb and can be executed in place.

load SPL

```
sudo ./imx_usb ../u-boot_imx6/SPL
```

On success, SPL should announce itself on the serial console:

```
U-Boot SPL 2018.01-00024-g457cdd60c3 (Aug 07 2018 - 21:16:13)
Trying to boot from USB SDP
SDP: initialize...
SDP: handle requests...
```

load U-Boot

```
sudo ./imx_usb ../u-boot_imx6/u-boot.img
```

On success, U-Boot should announce itself on the serial console:

```
Downloading file of size 329312 to 0x177fffc0... done
Jumping to header at 0x177fffc0
Header Tag is not an IMX image


U-Boot 2018.01-00024-g457cdd60c3 (Aug 07 2018 - 21:16:13 +0200)

CPU:   Freescale i.MX6Q rev1.5 996 MHz (running at 792 MHz)
CPU:   Extended Commercial temperature grade (-20C to 105C) at 41C
Reset cause: POR
Board: MX6 HummingBoard2 (som rev 1.5)
       Watchdog enabled
DRAM:  2 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
No panel detected: default to HDMI
Display: HDMI (1024x768)
In:    serial
Out:   serial
Err:   serial
Net:   FEC
Hit any key to stop autoboot:  0
```

At this point U-Boot has been loaded to RAM and is running. The next step is installing it to some internal storage such as eMMC or SPI Flash.

Installing to integrated storage

The most important thing to realize is that for copying data to integrated storage, the board has to already be running at least U-Boot, or even Linux. A fresh board where nothing has yet been installed can be brought up by the following methods:

- booting from a removable microSD
- booting from USB (OTG)

Please see the previous sections for more details.

The remainder of this section assumes that a recent version of U-Boot, at least 2018.01 is running.

There are actually three methods for installing the U-Boot binaries:

1. usb mass storage mode: The board presents itself as a usb drive to the host PC
2. DFU: The board listens for the DFU protocol on USB
3. the hard way: load and write files by hand using u-boot commands

We highly recommend using methods 1 or 2 as they are much easier to understand than the third!

USB Mass Storage:

This is probably the easiest option for most people. U-Boot has to be built from source with the ums command enabled:

```
diff --git a/configs/mx6cuboxi_defconfig b/configs/mx6cuboxi_defconfig
index 6c08e1d88e..664dba301f 100644
--- a/configs/mx6cuboxi_defconfig
+++ b/configs/mx6cuboxi_defconfig
@@ -67,3 +67,5 @@ CONFIG_OF_LIBFDT=y
 # CONFIG_CMD_BOOTEFI_HELLO_COMPILE is not set
 CONFIG_NET_RANDOM_ETHADDR=y
 CONFIG_NET_RANDOM_ETHADDR_OUI="d0:63:b4:00:00:00"
+CONFIG_CMD_USB_MASS_STORAGE=y
+CONFIG_USB_FUNCTION_MASS_STORAGE=y
diff --git a/include/configs/mx6cuboxi.h b/include/configs/mx6cuboxi.h
index 5e1fe5e760..75e3a14263 100644
--- a/include/configs/mx6cuboxi.h
+++ b/include/configs/mx6cuboxi.h
@@ -180,6 +180,8 @@

 #include <config_distro_bootcmd.h>

+#define CONFIG_USB_FUNCTION_MASS_STORAGE
+
 #else
 #define CONFIG_EXTRA_ENV_SETTINGS
```

```
 #endif /* CONFIG_SPL_BUILD */
```

The second step is to launch usb mass storage mode from the U-Boot console and specify what storage device to use.

For presenting the eMMC (mmc1) as a usb storage device, execute

```
ums 0 mmc 1
```

Now on a connected PC a new usb drive should have shown up. From this point onwards anything is possible! Partitioning, mounting, writing binaries in arbitrary locations, ... Note that this will also work for a microSD by using mmc0 instead.

Here is how U-Boot and SPL would be installed on this storage device:

```
dd if=SPL of=/dev/sdX bs=1k seek=1 conv=sync
dd if=u-boot.img of=/dev/sdX bs=1k seek=69 conv=sync
```

Note - Take your time while identifying where the board is mapped on your linux system. Failure to do so can result in overwriting an arbitrary disk on your system!

Using DFU:

DFU is not yet enabled by default, so it has to be built from source with two additional options:

```
CONFIG_CMD_DFU=y
CONFIG_DFU_MMC=y
```

In addition this patch is required to avoid a memory allocation problem!

DFU-Mode in U-Boot is configured and started on the serial console by first setting the dfu_alt_info environment variable, and then launching the dfu command. Below are samples targeting each eMMC, microSD and SPI Flash:

- eMMC:

```
setenv dfu_alt_info "spl.raw raw 0x2 0x66;u-boot.img.raw raw 0x8A 0x1000"
dfu 0 mmc 1
```

- microSD:

```
setenv dfu_alt_info "spl.raw raw 0x2 0x66;u-boot.img.raw raw 0x8A 0x1000"
dfu 0 mmc 0
```

- SPI

```
# Currently no SoMs with SPI Flash are available, so this section is empty
for now.
```

The syntax for dfu_alt_info can be looked up in the corresponding u-boot sources, namely drivers/dfu/dfu_mmc.c and drivers/dfu/dfu_sf.c. For (e)MMC the syntax is: <alt name> <access

method> <starting block> <size in blocks>

The dfu command takes 3 arguments: <usb controller number> <storage type> <device number> The controller number should always be 0 which means the OTG port is used!

The first command sets up two dfu names:

- spl.raw: space for SPL at offset 1kB
- u-boot.img.raw: space for u-boot.img at offset 69kB

The second command enables dfu mode on usb controller 0 (OTG port) for mmc 1 (eMMC).

Now dfu-util, which is available here on sourceforge, can be used on a PC to send both SPL and u-boot.img to the previously selected mmc 1:

```
sudo dfu-util -a spl.raw -D ../u-boot_imx6/SPL
sudo dfu-util -a u-boot.img.raw -D ../u-boot_imx6/u-boot.img
```

If everything went well the u-boot console should show

and the dfu commands on the pc should end with:

```
Copying data from PC to DFU device
Download        [=========================] 100%        52224 bytes
Download done.
state(7) = dfuMANIFEST, status(0) = No error condition is present
state(2) = dfuIDLE, status(0) = No error condition is present
Done!
```

The Hard Way:

On i.MX6 booting from eMMC works exactly like booting from an SD card: The Boot-ROM searches for executable code at 1K bytes into the data partition. The special purpose boot partitions are not supported.

If Linux is already running on the device, the procedure is identical to microSD. In this case the previous section can be used, where sdX is replaced with mmcblk1.

However it is more likely that neither Linux nor even U-Boot are available at this point. Refer to section Booting from USB (OTG) how to load u-boot to RAM.

Now that at least U-Boot is running on the target system, SPL and u-boot.img can be loaded from removable media or network, and then written to the eMMC. Below are the steps for loading binaries from a USB drive and writing them to eMMC:

```
# select eMMC (mmc1)
mmc dev 1

usb start
```

```
# expected output: 1 Storage Device(s) found

load usb 0:1 ${kernel_addr_r} SPL
# expected output: 52224 bytes read in 24 ms (2.1 MiB/s)
mmc write ${kernel_addr_r} 0x2 0x66
# expected output: MMC write: dev # 1, block # 2, count 102 ... 102 blocks
written: OK

load usb 0:1 ${kernel_addr_r} u-boot.img
# expected output: 329312 bytes read in 38 ms (8.3 MiB/s)
mmc write ${kernel_addr_r} 0x8A 0x284
# expected output: MMC write: dev # 1, block # 138, count 644 ... 644 blocks
written: OK
```

There are many magic numbers in this short script that require **attention**!:

- mmc dev 1: 0 is microSD, 1 is eMMC
- mmc write ${kernel_addr_r} 0x2 0x66:
  - 0x2 = destination block number in hexadecimal. Here one block is 512 bytes –> SPL goes to block 2 (1024 bytes)
  - 0x66: The size of SPL in blocks in hexadecimal. This is the ceiling of filesize divided by 512, in this example 52224 / 512 = 102 = 0x66
- mmc write ${kernel_addr_r} 0x8A 0x284
  - 0x8A: SPL is expected at 69kB –> 138 = 0x8A
  - 0x284: ceil(329312 / 512) = ceil(643,1875) = 644 = 0x284

It is highly recommended to write U-Boot from either Linux, or over the USB OTG port with DFU, which are both easier to use!

Boot Process after U-Boot has loaded

The way U-Boot searches for an operating system to load has recently undergone substantial changes towards a standard behaviour across all devices. This new standard is called distro support, and fully supported by our 2018.01 release. It automatically searches all attached storage for bootable files such as boot scripts, extlinux configuration or EFI applications. For documentation please refer to the official U-Boot documentation.

Loading Linux By Hand:

The manual way for loading Linux from U-Boot is very straight forward:

```
1. load the DeviceTree Binary
2. load the kernel image
3. load the initramfs (optional)
4. set boot commandline options
5. execute
```

Here is a sample for loading a kernel from microSD:

```
load mmc 0:1 ${fdt_addr_r}
load mmc 0:1 ${kernel_addr_r}
load mmc 0:1 ${ramdisk_addr_r}
setenv bootargs console=ttymxc0,115200n8 root=/dev/mmcblk0p1 rootfstype=auto
rootwait
bootz ${kernel_addr_r} ${ramdisk_addr_r}:${filesize} ${fdt_addr_r}
```

Transitioning from 2013.04 and earlier

We have developed a transitional boot-script that simulates the old behaviour to allow booting old system images that have not adopted support for distro boot: i.MX6 Legacy Boot-Script

Download and compile:

```
wget
https://gist.github.com/Josua-SR/feb0a32903154fab147c875efac749a7/raw/e17300
c31e0db6dc259a7568eaec5aa6b8077d09/imx6boot.txt
mkimage -A arm -O linux -T script -C none -a 0 -e 0 -d imx6boot.txt
imx6boot.scr
```

This mkimage'd boot-script should be placed at a well-defined location that u-boot can read from, under a name that does *not* equal boot.scr. We suggest calling it legacyboot.scr. Then the bootcmd environment variable has to be configured to force always running this script unconditionally. Here is a sample for when legacyboot.scr lives on the first partition on microSD:

```
setenv bootcmd 'load mmc 0:1 ${scriptaddr} legacyboot.scr; source
${scriptaddr}'
saveenv
```

This legacy script hasn't been thoroughly tested. There may be bugs and unexpected behaviour!

Compiling from source

This section assumes that you have git, make and a cross-compiler targeting 32-bit arm available on your system!

These are the instructions to fetch the code, and build a binary:

```
git clone --branch v2018.01-solidrun-imx6
https://github.com/SolidRun/u-boot.git u-boot-imx6
cd u-boot-imx6
export CROSS_COMPILE=<Set toolchain prefix to your toolchain>
# optionally add options to configs/mx6cuboxi_defconfig
make mx6cuboxi_defconfig
# optionally configure u-boot graphically
# make menuconfig
make
```

This will generate SPL and u-boot.img to be used when booting from eMMC. To target other boot media, set one of the following options in *configs/mx6cuboxi_defconfig* or through menuconfig:

- eMMC (default)

```
CONFIG_SPL_BOOT_DEVICE_MMC=y
```

- SD-Card

```
CONFIG_SPL_BOOT_DEVICE_SDHC=y
```

- mSATA SSD

```
CONFIG_SPL_BOOT_DEVICE_SATA=y
```

- SPI

```
CONFIG_SPL_BOOT_DEVICE_SPI_FLASH=y
```

Note: The resulting binaries are SPL and u-boot.img.

eMMC Tips

Size matters: The production line burns the image to emmc. the process runs at 5.6 MB/sec, which means that burning 8GB image of which 80% is free space is very wasteful. The best way to optimize for production is to create a single partition (as small as possible to contain all the data), and a “firstboot” script that will complete the image creation.

This is what I did to create a smaller image for burning: - mount a SD with the original image (2 mount points) - tar the contents of the 3rd partition and store the archive in /boot - create a first boot, oneshot systemd service that will:

- resize the rootfs partition
- resize the rootfs filesystem
- create and format the 2nd and 3rd partitions
- extract the archives from /boot into the relevant partitions
- delete the archives from /boot
- disable itself from systemd

- calculate the smallest image size that will contain your data

- run “du -s -B M /path/to/your/rootfs” to get the size of your data in megabytes
- add 10%
- partition size is at least 1MB smaller than the drive.

- create an image file with a single ext4 partition: and with the bootloader

- dd if=/dev/zero of=/path/to/vivi.img bs=1M count=$SIZE_MB
- losetup /dev/loop0 /path/to/vivi.img
- dd if=path/to/u-boot-imx6/SPL of=/dev/loop0 bs=1k seek=1 oflag=sync
- dd if=path/to/u-boot-imx6/u-boot.img of=/dev/loop0 bs=1k seek=42 oflag=sync
- sfdisk -f -uB /dev/loop0 «EOF

- 1024,${PARTSIZE}
- EOF
- sync
- losetup -d /dev/loop0

- populate the single partition with the rootfs, the archives of the other partitions, and the firstboot script

- losetup -o 1048576 /dev/loop0 /path/to/vivi.img # load the partition and not the entire disk image
- mkfs -t ext4 /dev/loop0
- mount /dev/loop0 /mnt
- do populate
- add the firstboot script: https://www.google.co.il/search?client=ubuntu&channel=fs&q=systemd+firstboot+script&ie=utf-8&oe=utf-8&gfe_rd=cr&ei=mWIPV-T8JLOo8wehw5GgCg
- umount /dev/loop0
- losetup -d /dev/loop0

Further reading

- Booting from network/PXE