Bootloader

Zilogic Systems

1. Introduction

- U-Boot is an open souce bootloader, that supports ~30 different processor families.
- U-Boot is primarily a boot loader, but it also used for board bring-up and production testing.

2. Getting U-Boot

- Before Oct 2008, U-boot had a x.y.z release numbering convention.
- Releases now have timestamp based release numbering yyyy.mm, development releases are numbered yyyy.mm-rcN.
- U-Boot can be obtained from ftp://ftp.denx.de/pub/u-boot/
- Extract the tar ball using `tar`.

```
$ tar -jxf u-boot-1.3.2.tar.bz2
```

3. Configuration

- Since U-boot supports multiple boards, the exact board for which U-boot is to be built has to be specified.
- Make is invoked as shown below. Where `xyz` is to be replaced by the board name.

```
$ make xyz_config
```

- The list of available boards can be found by looking at `include/configs/`, which contains one header file for each supported board.
- `include/configs/` contains the header file `sam9l9260.h`, which corresponds to the Olimex board.
- To configure U-boot for the Olimex board,

```
$ make sam9l9260_config
```

- This provides a sane default configuration for the board.
- It is also possible to fine tune U-boot's configuration before building U-boot. Customizing U-boot configuration will be shown later on.

4. Building U-Boot

- U-boot can be built using

```
$ make all
```

- To cross-compile U-boot, the cross-compiler prefix should be specified, through the `CROSS_COMPILE` environment variable.

```
$ make all CROSS_COMPILE=arm-none-eabi-
```

- As a result of the compilation process `u-boot.bin` will be produced.

5. Installing U-Boot

- If U-boot or Linux is already available in the target, then U-boot can be updated from U-boot or Linux.
- On a virgin board, U-boot is usually installed using, a debugging tool.
- SAM-BA can be used to flash the U-boot image on to a virgin board.

6. Configuring U-Boot

- U-Boot's configuration can be customized by modifying the board header file in `include/configs/`.
- The header file has two classes of configuration macros.

Configuration **Options**	Selectable by the user and have names starting with `CONFIG_`. Example: Configuration options is available for different sets of commands - jffs2, nand, usb.
Configuration **Settings**	Hardware related settings, should be touched only by the developer and have names starting with `CFG_`. Newer version of U-Boot use `CONFIG_SYS_` instead. Example: A configuration setting is available to specify the base address of the NAND flash.

6.1. Selecting Commands

- Unlike the Linux kernel, U-boot does not have a menu based configuration interface yet.
- For each class of commands there is a separation configuration option macro. If `CONFIG_CMD_JFFS2` is defined then commands for manipulating JFFS2 filesystems will be built into u-boot.
- Instead of defining the macros in each board header, a default set of commands is defined in `include/config_cmd_default.h`.
- Commands not required can then be `#undef`ed and additional commands required can be `#define`d.
- The list of all command macros is available in `include/config_cmd_all.h`.

```
/*
 * Command line configuration.
 */
#include <config_cmd_default.h>
#undef CONFIG_CMD_BDI
#undef CONFIG_CMD_IMI
#undef CONFIG_CMD_AUTOSCRIPT
#undef CONFIG_CMD_FPGA
#undef CONFIG_CMD_LOADS
#undef CONFIG_CMD_IMLS

#define CONFIG_CMD_PING         1
#define CONFIG_CMD_DHCP         1
#define CONFIG_CMD_NAND         1
#define CONFIG_CMD_USB          1
```

6.2. Specifying Boot Device

- U-boot can be put up in DataFlash 0, DataFlash 1 or Nand Flash.
- To specify the device that contains U-boot, the appropriate macro has to be defined.

```
#define CFG_USE_DATAFLASH_CS0           1
#undef CFG_USE_DATAFLASH_CS1
#undef CFG_USE_NANDFLASH
```

- This is later on used to calculate values for other macros.

```
#if defined(CFG_USE_DATAFLASH_CS0)

/* bootstrap + u-boot + env + linux in dataflash on CS0 */
#define CFG_ENV_IS_IN_DATAFLASH 1
#define CFG_MONITOR_BASE        (CFG_DATAFLASH_LOGIC_ADDR_CS0 + 0x8400)
#define CONFIG_ENV_OFFSET       0x4200
#define CFG_ENV_ADDR            (CFG_DATAFLASH_LOGIC_ADDR_CS0 + CONFIG_ENV_OFFSET)
#define CFG_ENV_SIZE            0x4200
#define CONFIG_ENV_SIZE         0x4200
#define CONFIG_BOOTCOMMAND      "cp.b 0xC0042000 0x22000000 0x210000; bootm"
#define CONFIG_BOOTARGS         "console=ttyS0,115200 "                 \
                                "root=/dev/mtdblock0 "                  \
                                "mtdparts=at91_nand:-(root) "           \
                                "rw rootfstype=jffs2"
#else
```

7. DataFlash Logical Addresses

- The logical addresses for the DataFlash are specified using the following configuration macros.

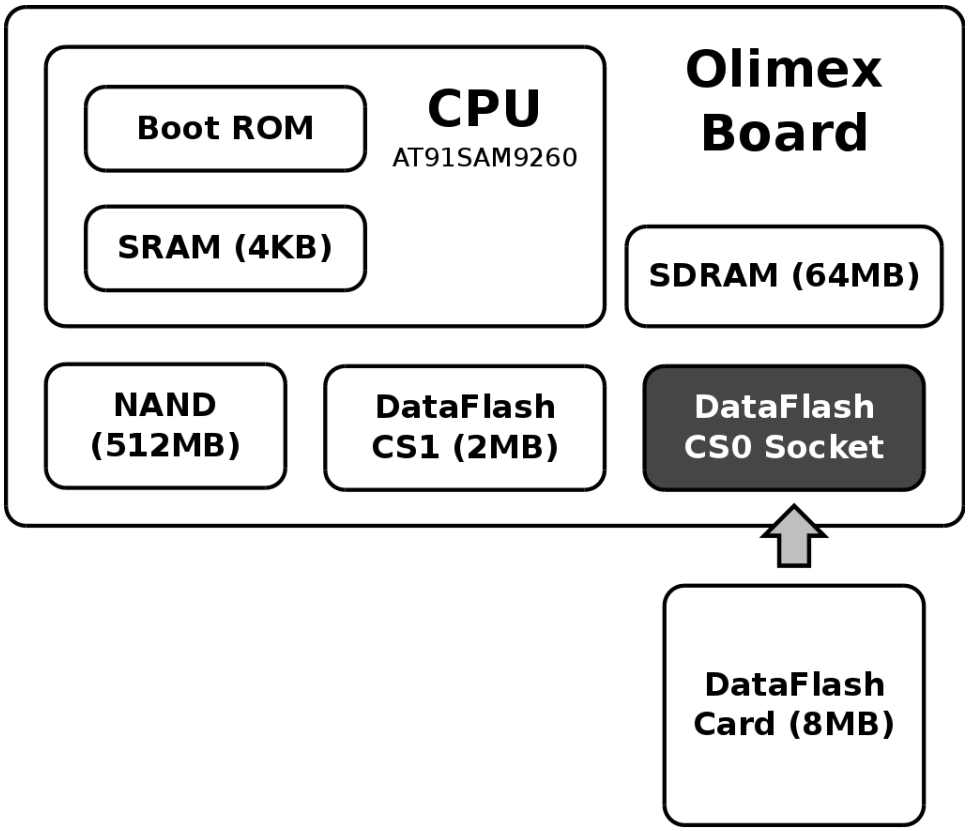
```
#define CFG_DATAFLASH_LOGIC_ADDR_CS0    0xC0000000      /* CS0 */
#define CFG_DATAFLASH_LOGIC_ADDR_CS1    0xD0000000      /* CS1 */
```

8. Shrinking U-boot

- If detailed help is not required, space can be saved by disabling `CFG_LONGHELP` option.
- If command history is not required, space can be saved by disabling `CONFIG_CMDLINE_EDITING` option.

9. Atmel Bootstrap

Figure 1. Memories Overview

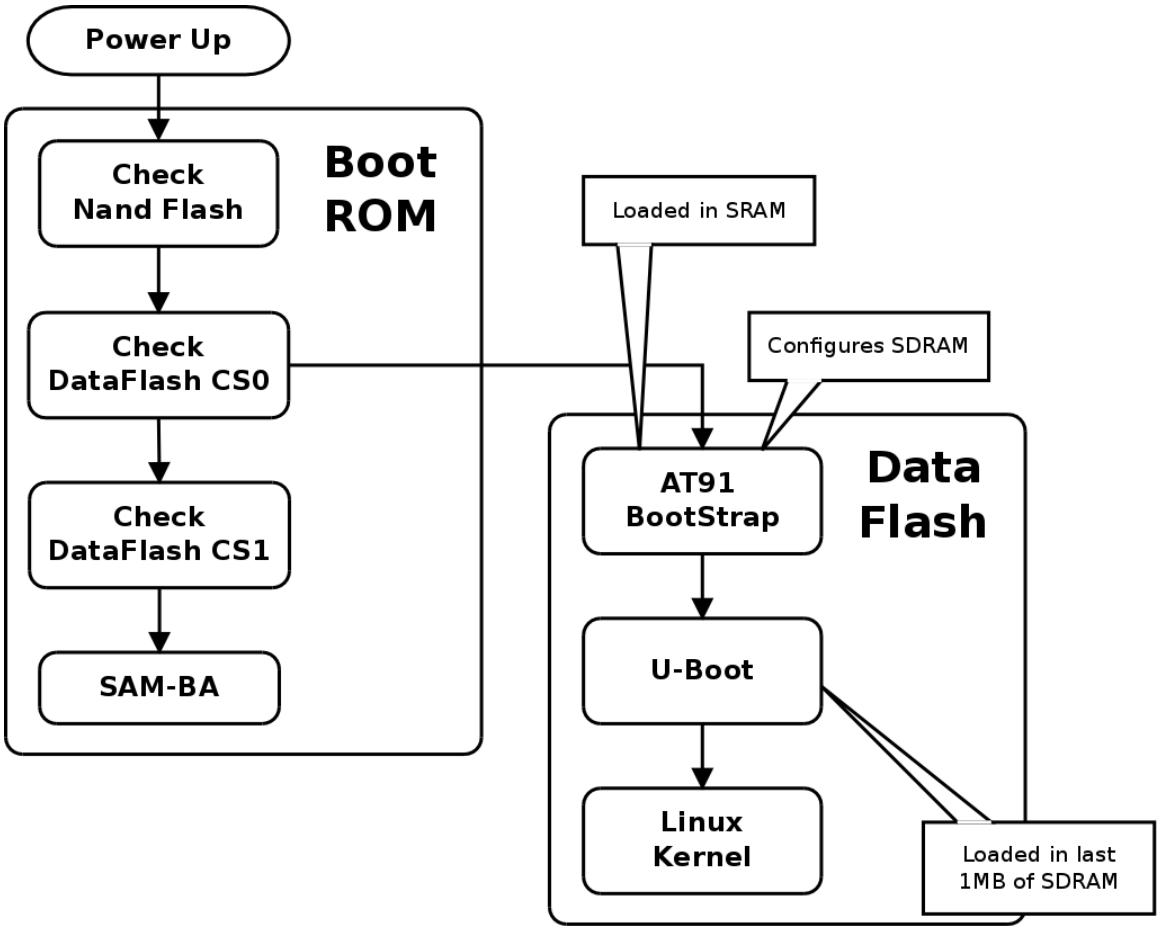


- When the processor boots up it, the boot code present in the ROM, checks to see if there is valid image in NAND Flash, DataFlash CS0, and DataFlash CS1.
- If valid images is present the image is copied to internal SRAM, and control is transfered to it.
- The size of internal SRAM is 4K, but U-boot image size is in the order of few hundred KB.

```
$ ls -lh u-boot.bin
-rwxr-xr-x 1 vijaykumar engineers 176K 2008-12-18 20:33 u-boot.bin
```

- The boot code cannot directly load and execute U-boot.
- A second stage boot loader is introduced, called the AT91 Bootstrap. The bootstrap is tiny and can be loaded into SRAM.
  - Initializes SDRAM
  - Loads U-boot from a hardcoded offset to SDRAM
  - Transfers control to U-boot

Figure 2. Boot Sequence



- The AT91 Bootstrap is available from ftp://www.linux4sam.org/pub/at91bootstrap/
- To compile the AT91 Bootstrap.
  1. go into the `board` directory
  2. the directory contains all supported boards, enter the `at91sam9260ek`.
  3. the directory contains one sub-directory for each media, enter `dataflash`.
  4. run `make` with `CROSS_COMPILE` set to the cross compiler prefix
- The bootstrap can be customized by modifying the header file present in the directory where `make` was invoked.

```
#define IMG_ADDRESS     0x8400          /* Image Address in DataFlash */
#define IMG_SIZE        0x30000         /* Image Size in DataFlash    */

#define MACH_TYPE       0x44B           /* AT91SAM9260-EK */
#define JUMP_ADDR       0x23F00000      /* Final Jump Address         */
```

10. Further Reading

- U-Boot Design Principles [http://www.denx.de/wiki/U-Boot/DesignPrinciples].
- U-Boot Design Requirements [http://www.denx.de/wiki/U-Boot/DesignRequirements].
- U-Boot Manual [http://www.denx.de/wiki/view/DULG/UBoot].
- U-Boot Release Cycle [http://www.denx.de/wiki/U-Boot/ReleaseCycle].

- `README` file in U-boot source tree has information on various configuration variables.
- AT91 Bootstrap Application Note [http://www.atmel.com/dyn/resources/prod_documents/doc6277.pdf].