U-Boot programming: A tutorial -- Part III

U-Boot's bring-up

Even though unnecessary in most cases, it's sometimes desired to modify U-Boot's own bring-up process, in particular for initializing custom hardware during early stages. This section explains the basics of this part of U-Boot.

U-Boot is one of the first things to run on the processor, and may be responsible for the most basic hardware initialization. On some platforms the processor's RAM isn't configured when U-Boot starts running, so the underlying assumption is that U-Boot may run directly from ROM (typically flash memory).

The bring-up process' key event is hence when U-Boot copies itself from where it runs in the beginning into RAM, from which it runs the more sophisticated tasks (handling boot commands in particular). This self-copy is referred to as "relocation".

Almost needless to say, the processor runs in "real mode": The MMU, if there is one, is off. There is no memory translation nor protection. U-Boot plays a few dirty tricks based on this.

In gross terms, the U-Boot loader runs through the following phases:

- Pre-relocation initialization (possibly directly from flash or other kind of ROM)
- Relocation: Copy the code to RAM.
- Post-relocation initialization (from proper RAM).
- Execution of commands: Through autoboot or console shell
- Passing control to the Linux kernel (or other target application)

Note that in several scenarios, U-Boot starts from proper RAM to begin with, and consequently there is no actual relocation taking place. The division into pre-relocation and post-relocation becomes somewhat artificial in these scenarios, yet this is the terminology.

A more detailed look

The sequence for the ARM architecture can be deduced from arch/arm/lib/crt0.S, which is the absolutely first thing that runs. This piece of assembly code calls functions as follows (along with some very low-level initializations):

- board_init_f() (defined in e.g. arch/arm/lib/board.c): Calls the functions listed in the init_sequence_f function pointer array (using initcall_run_list()), which is enlisted in this file with a lot of ifdefs. This function then runs various ifdef-dependent init snippets.
- relocate_code()
- coloured_LED_init() and red_led_on() are directly called by crt0.S. Defining these functions allow hooking visible indications of early boot progress.
- board_init_r() (also possibly defined in arch/arm/lib/board.c): Runs the initialization as a "normal" program running from RAM. This function never returns. Rather,
- board_init_r() loops on main_loop() (defined in common/main.c) forever. This is essentially the autoboot or execution of commands from input by the command parser (most likely hush).
- At some stage, a command in main_loop() gives the control to the Linux kernel (or whatever was loaded instead).

For those who wish to add some code during these init stages: CONFIG_BOARD_EARLY_INIT_F, CONFIG_BOARD_EARLY_INIT_R, CONFIG_BOARD_LATE_INIT and CONFIG_BOARD_POSTCLK_INIT can be defined to request a call to board_early_init_f(), board_early_init_r(), board_late_init() and board_postclk_init() in the respective stages. These can be used as simple hooks for whatever platform-specific functionality is needed.

SPL boot

The SPL (Secondary Program Loader) boot feature is irrelevant in most scenarios, but offers a solution As U-Boot itself is too large for the platform's boot sequence. For example, the ARM processor's hardware boot loader in Altera's SoC FPGAs can only handle a 60 kB image. A typical U-Boot ELF easily reaches 300 kB (after stripping).

The point with an SPL is to create a very small preloader, which loads the "full" U-Boot image. It's

Related pages

- U-Boot programming: A tutorial -- Part I
- U-Boot programming: A tutorial -- Part II
- Preparing a Uboot image for Altera's Cyclone V SoC FPGA

built from U-Boot's sources, but with a minimal set of code. So when U-Boot is built for a platform that requires SPL, it's typically done twice: Once for generating the SPL, and a second time for the full U-Boot.

The SPL build is done with the CONFIG_SPL_BUILD is defined. Only the pre-location phase runs on SPL builds. All it does is the minimal set of initializations, then loads the full U-Boot image, and passes control to it.

Further reading

- U-Boot programming: A tutorial -- Part I
- U-Boot programming: A tutorial -- Part II
- Preparing a Uboot image for Altera's Cyclone V SoC FPGA