U-Boot - How to debug

Contents

Stable: 26.05.2020 - 13:42 | Revision: 26.05.2020 - 13:41

A quality version of this page, approved on *26 May 2020*, was based off this revision.

Contents

1 Debug with console

Trace and error are available on the console defined in the **chosen** node of the device tree by the **stdout-path** field, for example on serial0=uart4:

```
chosen {
        stdout-path = " serial0:115200n8";
};
aliases {
        serial0 = &uart4;
};
```

By default, the macros used by U-Boot (debug(), pr_debug()...) do not print any trace; to activate the debug traces on a specific file, you need to enable the **DEBUG** compilation flag and change the LOGLEVEL for the file:

- define DEBUG before any include in the <file>.c file

```
#define DEBUG
#undef CONFIG_LOGLEVEL
#define CONFIG_LOGLEVEL 8
```

- with a Makefile

```
CFLAGS_<file>.o+= -DDEBUG -DCONFIG_LOGLEVEL=8
```

For details, see doc/README.log .

If U-Boot fails before the console configuration (in the first stage of U-Boot execution), trace is not available.

In this case, you need to:

- debug with GDB (see the next chapter)

or,

- activate the debug UART feature:
  - add in defconfig of U-Boot configuration
    - CONFIG_DEBUG_UART
    - CONFIG_DEBUG_UART_STM32
  - adpat the function board_debug_uart_init(): that configures the required resources (pad, clock) before initialization by the U-Boot driver.
    This function needs to be adapted for your board.

2 Debug with GDB

With OpenSTLinux, you can directly use GDB script Setup.gdb:

- GDB#U-Boot_execution_phase
- GDB#U-Boot_boot_case

Or for manual GDB connection, you need to:

1. get the elf files for U-Boot and/or SPL
(u-boot and u-boot-spl available in the build directory)
2. connect GDB to the target
3. **reset with attach** the target with the gdb **"monitor reset halt"** command:
execution is stopped in ROM code or at the beginning of FSBL execution.
4. load the symbols of the binary to be debugged with commands available in next chapter:
#Load U-Boot symbol, #Load SPL symbol, #Load SPL code and debug
5. start execution with the **"continue"** command

2.1 Load U-Boot symbol

With U-Boot relocation, symbols are more difficult to load.

See https://www.denx.de/wiki/DULG/DebuggingUBoot

If you connect GDB on running target, you can load the debug symbols:

- Before relocation with **"symbol-file"** command:

```
(gdb) symbol-file u-boot
```

- After relocation with **"add-symbol-file"** command to relocate the symbol with the code offset = gd->relocaddr:

```
(gdb) symbol-file u-boot                              --> only for "gd_t" definition
(gdb) set $offset = ((gd_t *)$r9)->relocaddr          --> get relocation offset
(gdb) symbol-file                                     --> clear previous symbol
(gdb) add-symbol-file u-boot $offset
```

The following GDB example script automatically loads the U-Boot symbol before and after relocation for a programmed board, after **"monitor reset halt"** command:

```
(gdb) thbreak *0xC0100000
(gdb) commands
> symbol-file u-boot
> thbreak relocate_code
> commands
  > print "RELOCATE U-Boot..."
  > set $offset = ((gd_t *)$r9)->relocaddr
  > print $offset
  > symbol-file
```

```
    > add-symbol-file u-boot $offset
    > thbreak boot_jump_linux
    > continue
    > end
  > continue
  > end
```

This script uses a temporary hardware breakpoint **"thbreak"** to load the symbol when U-Boot code is loaded in DDR by FSBL = TF-A or SPL at the U-Boot entry point (CONFIG_SYS_TEXT_BASE = 0xC0100000).

It allows the symbol to be loaded only when code is executed to avoid DDR access before DDR initialization.

2.2 Load SPL symbol

To debug SPL with GDB on a Flashed device, ROM code loads the binary and the GDB script just loads the SPL symbols:

```
(gdb) symbol-file u-boot-spl
```

2.3 Load SPL code and debug

Sometimes you need to debug SPL execution on an unprogrammed target (for example for board bring-up), so you can use GDB to load the SPL code in embedded RAM and execute it.

When execution is stopped in ROM code, you need to execute the **"load"** commands, depending on the compilation flags defined in U-Boot device tree to load the SPL code and the associated device tree:

- CONFIG_OF_SEPARATE = dtb appended at the end of the code, not present in the elf file (default configuration)

```
(gdb) file u-boot-spl
(gdb) load
(gdb) set $dtb =  __bss_end
(gdb) restore spl/dt.dtb binary $dtb
```

- CONFIG_OF_EMBED = dtb embedded in the elf file (debug configuration)

```
(gdb) file u-boot-spl
(gdb) load
```

2.4 Debug the first SPL instructions

Sometime the SPL code execution is stopped by the gdb command "monitor reset halt" after the first instructions.

To debug this part, you can modify the code: add a infinite loop in SPL code to wait the gdb connection.

For example in arch/arm/mach-stm32mp/spl.c :

```
void board_init_f(ulong dummy)
{
        struct udevice *dev;
        int ret;

        /* volatile is needed to avoid gcc optimization */
```

```
        volatile int stop = 0;
        /* infinite debug loop */
        while ( !stop ) ;

        arch_cpu_init();
```

when gdb is attached and the SPL symbols are loaded, the infinite loop is interrupted by :

```
(gdb) set var stop=1
```

And you can debug the SPL first instruction by gdb commands.

Das U-Boot -- the Universal Boot Loader (see U-Boot_overview)

GNU dedugger, a portable debugger that runs on many Unix-like systems

Universal Asynchronous Receiver/Transmitter

Secondary Program Loader, *Also known as* ***U-Boot SPL***

Read Only Memory

First Stage Boot Loader

Doubledata rate (memory domain)

Trusted Firmware for Arm Cortex-A

Random Access Memory (Early computer memories generally hadserial access. Memories where any given address can be accessed when desired were then called "random access" to distinguish them from the memories where contents can only be accessed in a fixed order. The term is used today for volatile random-acces ssemiconductor memories.)