TPL: SPL loading SPL

(and, SPL as just another U-Boot config)

Scott Wood
Freescale Semiconductor
October 2013

Background: What is SPL?

- Unlike NOR flash, many boot sources are not directly memory mapped.
- On-chip ROM or other mechanism loads a binary into an SRAM.
  - This SRAM is often very small, sometimes 4 KiB or less.
  - The ROM can't load us into main system RAM yet, since initialization is too complex and must be handled by U-Boot.
- SPL (Secondary Program Loader) is a small binary, generated from U-Boot source, that fits in the SRAM and loads the main U-Boot into system RAM.
- Configured by a parallel set of makefile config symbols – CONFIG_SPL_I2C_SUPPORT, CONFIG_SPL_NAND_SUPPORT, etc.
  - Normal CONFIG symbols also used, but not to control the differences between the SPL and the main U-Boot.
  - SPL also relies heavily on toolchain garbage collection.

What can we do in 4 KiB?

- Not much.
  - Hardcoded RAM initialization
    - Causes problems if a new revision of the board has different RAM, or if RAM is socketed
    - Code to use SPD to dynamically initialize RAM is way too large to fit.
  - Barely fits even then
    - Limited output capability (puts rather than printf)
    - No exception handling
    - Toolchain variations often cause breakage due to size differences

Running from a different SRAM

- Sometimes, there is more than one SRAM on the system, and the boot ROM loads us into the smaller one.
  - In particular, on some Freescale 85xx/QorIQ chips, we get loaded into a 4 KiB NAND I/O buffer, but L2 cache can be locked to provide a larger SRAM.
- ...but this time, it's the full U-Boot that needs to fit.
  - On Freescale 85xx/QorIQ:
    - Sometimes we have 512K, which is usually good.
    - Sometimes only 256K, which is not enough.

Tertiary Program Loader

- Tiny SPL loads moderate-size middle layer called TPL.
- TPL
  - Implementation by Ying Zhang, in v2013.10
  - Originally proposed in patch by Haiying Wang in 2011
  - does dynamic initialization of full RAM
  - loads the full U-Boot into RAM
  - uses existing SPL makefile infrastructure and symbols, with a separate autoconf.mk
  - Is essentially another instance of SPL
    - CONFIG_SPL_BUILD set for both TPL and SPL
    - CONFIG_TPL_BUILD set for TPL only

TPL configuration

```
#ifdef CONFIG_TPL_BUILD
#define CONFIG_SPL_NAND_BOOT
#define CONFIG_SPL_FLUSH_IMAGE
#define CONFIG_SPL_ENV_SUPPORT
#define CONFIG_SPL_NAND_INIT
#define CONFIG_SPL_SERIAL_SUPPORT
#define CONFIG_SPL_LIBGENERIC_SUPPORT
#define CONFIG_SPL_LIBCOMMON_SUPPORT
#define CONFIG_SPL_I2C_SUPPORT
#define CONFIG_SPL_NAND_SUPPORT
#define CONFIG_SPL_MPC8XXX_INIT_DDR_SUPPORT
#define CONFIG_SPL_COMMON_INIT_DDR
#define CONFIG_SPL_MAX_SIZE             (128 << 10)
#define CONFIG_SPL_TEXT_BASE            0xf8f81000
#define CONFIG_SYS_MPC85XX_NO_RESETVEC
#define CONFIG_SYS_NAND_U_BOOT_SIZE     (576 << 10)
#define CONFIG_SYS_NAND_U_BOOT_DST      (0x11000000)
#define CONFIG_SYS_NAND_U_BOOT_START    (0x11000000)
#define CONFIG_SYS_NAND_U_BOOT_OFFS     ((128 + 128) << 10)
#elif defined(CONFIG_SPL_BUILD)
#define CONFIG_SPL_INIT_MINIMAL
#define CONFIG_SPL_SERIAL_SUPPORT
#define CONFIG_SPL_NAND_SUPPORT
#define CONFIG_SPL_FLUSH_IMAGE
#define CONFIG_SPL_TEXT_BASE            0xff800000
#define CONFIG_SPL_MAX_SIZE             4096
#define CONFIG_SYS_NAND_U_BOOT_SIZE     (128 << 10)
#define CONFIG_SYS_NAND_U_BOOT_DST      0xf8f80000
#define CONFIG_SYS_NAND_U_BOOT_START    0xf8f80000
#define CONFIG_SYS_NAND_U_BOOT_OFFS     (128 << 10)
#endif
```

Another approach

- If we can use ifdefs in the board config file to differentiate SPL from TPL, why not to differentiate SPL from the main U-Boot?
- Patch from Joel Fernandes (in v2013.10) provides a separate autoconf.mk for SPL (which TPL builds on).
- TPL patch using similar concept by Matthew McClintock in 2011
- Would let us get away from having a separate, parallel build config system just for SPL
- Make config more fine grained in order to maintain ability to meet size requirements.
- Use kconfig to keep things manageable (we want this anyway).
  - This means our usage of kconfig would need to support multiple config instances.

Example (pre-kconfig)

```
#define CONFIG_CMD_NAND

…

#ifndef CONFIG_SPL_BUILD

#define CONFIG_CMD_NET

…

#endif /* non-SPL */
```

Example (pre-kconfig)

```
#define CONFIG_CMD_NAND
…
#if defined(CONFIG_SPL_BUILD) && !defined(CONFIG_TPL_BUILD)
#define CONFIG_SPL_INIT_MINIMAL
…
#endif /* first SPL */
#if defined(CONFIG_TPL_BUILD) || !defined(CONFIG_SPL_BUILD)
/* currently CONFIG_SPL_LIBGENERIC_SUPPORT */
#define CONFIG_LIBGENERIC
…
#endif /* TPL and main U-Boot */
#ifndef CONFIG_SPL_BUILD
#define CONFIG_CMD_NET
…
#endif /* main U-Boot */
```

Questions or comments?