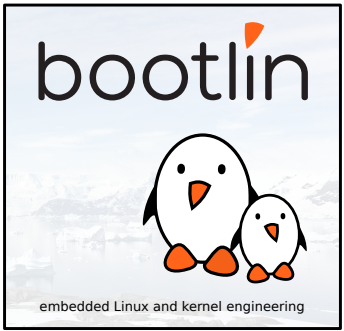
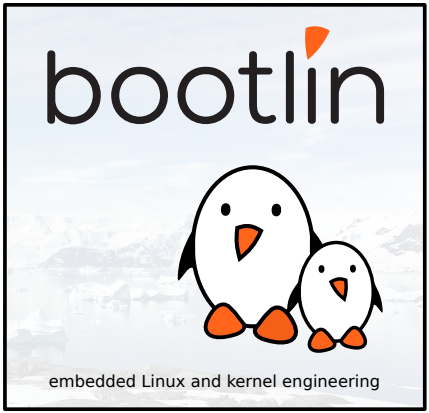
Porting U-Boot and Linux on new ARM boards: a step-by-step guide

Quentin Schulz
Bootlin
quentin.schulz@bootlin.com



Quentin Schulz

- Embedded Linux and kernel engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
  - Linux kernel contributors, ARM SoC support, kernel maintainers
- Added support in U-Boot and Linux kernel for an i.MX6 custom board,



Preamble

- Feedback from my journey to support a custom board in U-Boot and Linux,
- Examples for a board with a well-known SoC (i.MX6) and already supported IPs, almost no coding skill involved in this talk,
- More focused on the U-Boot part,

Golden rules

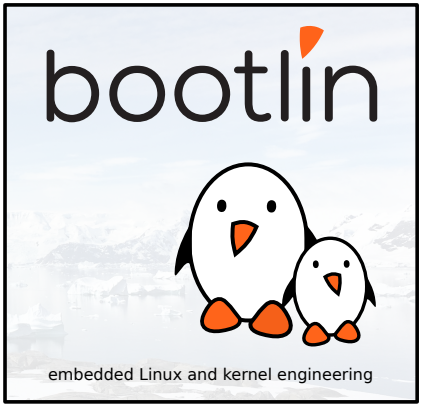
- If you have the sources of your BSP, compile and run the BSP to:
  1. Validate the IP you're working on works with some code,
  2. Have a reference code,
  3. Have a code that you can use to debug,
- Focus on correctly configuring RAM and UART only,
- Commit,
- One IP at a time,
- Commit,

Custom board presentation

- i.MX6-based module with an extension board,
- Ethernet, I2C, SPI, NAND, eMMC, SD Card reader, USB device, EEPROM, GPIO, UART, audio (I2S), HDMI, LVDS, PCIe, USB host, RTC, PMIC,

U-Boot porting

Quentin Schulz
quentin.schulz@bootlin.com



U-Boot status

- On-going migration from board header file defines to Kconfig options,
- On-going migration from *manual* drivers probing to Driver Model,

U-Boot directories

- *arch/*
  anything arch or platform related: DTS, CPU init, pinmux controller, DRAM, clocks, ...
- *board/*
  code board specific (init, pinmuxing configuration, etc), Kconfig file specifying board header file, board file, paths, Makefile for board file,
- *configs/*
  all boards' defconfigs
- *drivers/*
- *include/*
  all headers
- *include/configs/*
  all boards' header files
- ...

U-Boot new board support workflow

1. Create the board file,
2. Create the board Kconfig file,
3. Create the board Makefile,
4. Create the board defconfig,
5. Create the board header file,
6. Source board's Kconfig in the architecture's Kconfig,
7. Define the TARGET Kconfig option in its CPU's Kconfig,

- some platforms (e.g. sunxi (Allwinner)) share common files so only a defconfig would be required,

1. Create the board file

board/my_vendor/my_board/my_board.c

```
#include <...>

DECLARE_GLOBAL_DATA_PTR;

int dram_init(void)
{
  gd->ram_size = imx_ddr_size();
  return 0;
}

int board_init(void)
{
  return 0;
}
```

Global data

- `DECLARE_GLOBAL_DATA_PTR`,
- usable in code with `gd` global variable,
- on ARM, equals to hardware register *r9* for ARM32 and *x18* for ARM64,
- used to store info in *"some memory which is available very early after boot to allow for a minimum set of global variables during system initialization (until we have set up the memory controller so that we can use RAM)"*,
- `include/asm-generic/global_data.h` to find what kind of information it can store,

2. Create the board Kconfig file

board/my_vendor/my_board/Kconfig

```
if TARGET_MY_BOARD

config SYS_BOARD
        default "my_board"

config SYS_VENDOR
        default "my_vendor"

config SYS_CONFIG_NAME
        default "my_board"

endif
```

Kconfig options

- SYS_VENDOR and SYS_BOARD are used to identify the directory where make find the files it needs to compile,
  - if both are present,
    - board/SYS_VENDOR/SYS_BOARD/
  - if SYS_VENDOR is omitted,
    - board/SYS_BOARD/
  - if SYS_BOARD is omitted,
    - board/SYS_VENDOR/common/
- SYS_CONFIG_NAME is used to identify the board header file,
  - include/configs/SYS_CONFIG_NAME.h

3. Create the board Makefile

board/my_vendor/my_board/Makefile

```
obj-y  := my_board.o
```

4. Create the board defconfig

configs/my_board_defconfig

```
CONFIG_ARM=y
CONFIG_ARCH_MX6=y
CONFIG_TARGET_MY_BOARD=y
CONFIG_MXC_UART=y
```

4. Create the board defconfig

- put here anything that is selectable in Kconfig (menuconfig),
- drivers, features, U-Boot behaviour, libs, etc.

5. Create the board header file (minimal example for i.MX6 Solo)

include/configs/my_board.h

```
#ifndef __MY_BOARD_CONFIG_H__
#define __MY_BOARD_CONFIG_H__

#define CONFIG_MXC_UART_BASE           UART5_BASE

#include "mx6_common.h"

#define CONFIG_NR_DRAM_BANKS           1
#define CONFIG_SYS_MAX_FLASH_BANKS     1
#define CONFIG_SYS_MALLOC_LEN          (10 * SZ_1M)
#define CONFIG_SYS_FSL_ESDHC_ADDR      0
#define PHYS_SDRAM                     MMDC0_ARB_BASE_ADDR
#define CONFIG_SYS_SDRAM_BASE          PHYS_SDRAM
#define CONFIG_SYS_INIT_RAM_ADDR       IRAM_BASE_ADDR
#define CONFIG_SYS_INIT_RAM_SIZE       IRAM_SIZE
#define CONFIG_SYS_INIT_SP_OFFSET \
  (CONFIG_SYS_INIT_RAM_SIZE - GENERATED_GBL_DATA_SIZE)

#define CONFIG_SYS_INIT_SP_ADDR \
  (CONFIG_SYS_INIT_RAM_ADDR + CONFIG_SYS_INIT_SP_OFFSET)
#endif
```

6. Source board's Kconfig file

arch/arm/Kconfig or
arch/arm/mach-imx/mx6/Kconfig

```
...
source "board/imx31_phycore/Kconfig"
source "board/isee/igep003x/Kconfig"
source "board/my_vendor/my_board/Kconfig"
source "board/olimex/mx23_olinuxino/Kconfig"
source "board/phytec/pcm051/Kconfig"
...
```

7. Define board's TARGET Kconfig option

arch/arm/mach-imx/mx6/Kconfig

```
choice
...
config TARGET_MY_BOARD
    bool "My awesome board"
    select MX6S
...
endchoice
```

U-Boot init sequence

- U-Boot will run two lists of functions whose purpose is to init or configure specific IPs before the user have access to the console,
- the first list is defined in `common/board_f.c` in the `static init_fnc_t init_sequence_f[]` array,
- first list takes care of initialising DRAM, mapping it and relocating the bootloader code once it's working,
- the second list is defined in `common/board_r.c` in the `static init_fnc_t init_sequence_r[]` array,
- some functions are run only when a constant is defined (e.g. `CONFIG_BOARD_EARLY_INIT_F` defined to run `board_early_init_f()`),
- any function returning a non-zero value will stop the init sequence and make U-Boot fail to boot,
  - define `DEBUG` when having trouble with init sequence,
- not all "features" are available in all functions (i.e. no `udelay` in `board_early_init_f()`)

Driver selection

- take inspiration from boards with the same IP,
- inspect drivers in the appropriate subsystem,
  1. focus on the driver's behaviour,
  2. then check out the registers, bit offsets, masks, etc.
  3. check for undefined macros or constants,
  4. check for piece of code surrounded by `ifdef` blocks,
- look for the object file of this driver in the Makefile of the subsystem,

```
obj-$(CONFIG_MY_DRIVER) += my_driver.o
```

- `grep` for `CONFIG_MY_DRIVER`,
  - visible symbol in some Kconfig file => add to board defconfig,
  - non-visible symbol in some Kconfig file or not defined => board header file,
- make sure your driver is compiled (look for `my_driver.o`),

Driver selection - NAND example

- `drivers/mtd/nand/nand_mxs.c`,
- `CONFIG_NAND_MXS` for compiling the driver,
- `CONFIG_SYS_MAX_NAND_DEVICE` and `CONFIG_SYS_NAND_BASE` constants for configuring the device,

Driver selection - NAND example

configs/my_board_defconfig

```
CONFIG_ARM=y
CONFIG_ARCH_MX6=y
CONFIG_TARGET_MY_BOARD=y
CONFIG_MXC_UART=y
CONFIG_NAND_MXS=y
```

Driver selection - NAND example

include/configs/my_board.h

```
...
/* Define NAND settings */
/* Max number of NAND devices supported */
#define CONFIG_SYS_MAX_NAND_DEVICE     1
#define CONFIG_SYS_NAND_BASE           0x00112000
...
```

Driver selection - NAND example

board/my_vendor/my_board.c

```
...
static iomux_v3_cfg_t const nand_pads[] = {...};
...
int board_init(void)
{
  imx_iomux_v3_setup_multiple_pads(nand_pads, ARRAY_SIZE(nand_pads));

  return 0;
}
```

Note on Device Trees

- the migration to Device Trees started back in 2012 and the code is slowly migrated, driver by driver, subsystem by subsystem,
- need for Driver model to use device trees,
- most drivers have big `ifdef` blocks for `CONFIG_DM`,
  - you can't really chose on a per-driver basis to enable DM support,
- idem for subsystem core code,
- I didn't go really deep into it as we needed either no or full support for Device Trees and the NAND framework isn't migrated to the DM,

Effort needed to support U-Boot

- Ethernet, EEPROM, NAND, eMMC, SD Card reader, USB device, GPIO, UART, audio (I2S), PMIC,

```
$ wc -l board/my_vendor/my_board/* configs/my_board_defconfig include/configs/my_board.h
     15 board/my_vendor/my_board/Kconfig
      8 board/my_vendor/my_board/Makefile
    159 board/my_vendor/my_board/my_board.cfg #DCD conf for DDR controller and memory
    218 board/my_vendor/my_board/my_board.c
     39 configs/my_board_defconfig
    110 include/configs/my_board.h
    510 total
```

- +1 line in `arch/arm/Kconfig`,
- +4 lines in `arch/arm/cpu/armv7/mx6/Kconfig`,
- actually ~100 lines of "real" code (PHY and board init, mmc configuration),
- no modification of U-Boot source code otherwise,

Effort needed to update U-Boot

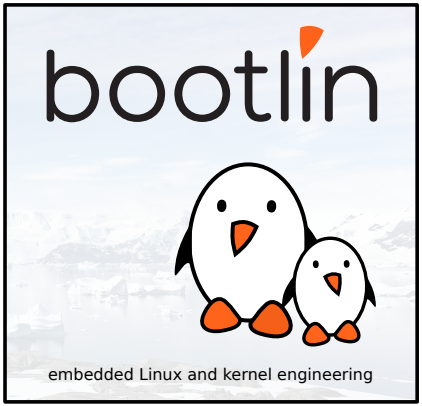
- got some weird bug with the RSA lib when checking fitImage signatures in U-Boot,
- update from 2017.03 to 2017.07 was very easy:
  - *port* board header file, board file, board Makefile, board Kconfig and update the Kconfig of the architecture,
  - make sure options defined in board header files are not Kconfig options now,
- problem solved in ~30min by updating,

Problems encountered

- board init sequence is toggling a few GPIOs with a given timing,
- all signals (even UART) go through the FPGA,
- failing board init sequence, no FPGA, no UART, no hair left on my head,
- no `udelay` in `board_early_init_f`,
- a workaround was to use a forloop with `cpurelax()`,

Linux kernel porting

Quentin Schulz
quentin.schulz@bootlin.com



Linux kernel new board support workflow

1. Create the board's Device Tree,
2. Add your board's DTB to the architecture DTS Makefile,
3. Create a defconfig for your board,

Device Tree

- a file in a special DTS (Device Tree Source) format,
- purely describes the hardware of your board,
- matches an IP with a driver thanks to `compatible` strings,
- documentation can be found in `Documentation/devicetree/bindings`,
- for a more in-depth explanation on what a Device Tree is and how to write it: `https://www.youtube.com/watch?v=m_NyYEBxfn8`,

1. Create the board's Device Tree

1. write a map of your IPs' relationships,
2. find the SoC DTSI that you'll include,
3. look for IPs' drivers in the correct subsystem,
   - `grep`ing the (code)name of the IP is usually a good start,
4. once found, look for the compatible string in `Documentation/devicetree/bindings`,
5. follow the documentation to add the correct binding,
6. some bindings are framework-wide defined so make sure to read the documentation of the framework involved,
7. for SoC IPs, correct binding is usually as simple as enabling them and setting the correct power supply and pinmuxing since most is already described in the SoC Device Tree,

1. Create the board's Device Tree

arch/arm/boot/dts/imx6s-my-board.dts

```
#include "imx6dl.dtsi"

/ {
  model = "MyVendor myBoard";
  compatible = "my_vendor,my_board";
};

...

&pcie {
  reset-gpio = <&gpio3 18 GPIO_ACTIVE_LOW>;
  vpcie-supply = <&reg_pcie>;
  status = "okay";
};

&uart5 {
  status = "okay";
};
```

2. Add the DTB to the arch DTS Makefile

arch/arm/boot/dts/Makefile

```
dtb-$(CONFIG_MACH_KIRKWOOD) += \
#...

dtb-$(CONFIG_SOC_IMX6Q) += \
#...
     imx6qp-sabresd.dtb \
     imx6s-my-board.dtb
#...
```

3. Create a defconfig for your board

1. start from the SoC family defconfig (e.g. `imx_v6_v7_defconfig`) or if there isn't one, from the architecture (e.g. `multi_v7_defconfig`),
2. strip the defconfig of useless SoC families, drivers and features,
3. add the `CONFIG` of the drivers you want to compile,
   - `grep`ing for the basename of the driver is the way to go,
   - most drivers depend on subsystems or other options, you have to add them to your defconfig as well if your driver doesn't select them,

Problems encountered

- PCIe driver was probing but not enumerating devices,
  - driver was working in BSP, found out missing support for regulator was the culprit,
  - quickly wrote a 40 lines patch and sent it upstream,
- Ethernet driver was missing a post reset delay for the PHY that was set in BSP,
  - quickly wrote a 20 lines patch and sent it upstream,

Effort needed to support Linux

- Ethernet, I2C, SPI, NAND, eMMC, SD Card reader, USB device, EEPROM, GPIO, UART, audio (I2S), HDMI, LVDS, PCIe, USB host, RTC, PMIC,

```
$ wc -l arch/arm/boot/dts/imx6s-my-board.dts
arch/arm/configs/my_board_defconfig
    606 arch/arm/boot/dts/imx6s-my-board.dts
    401 arch/arm/configs/my_board_defconfig
   1007 total
```

- +1 line in `arch/arm/boot/dts/Makefile`,
- +20 lines for regulator support in PCIe driver (now upstream),
- +40 lines for Ethernet PHY post reset delay (now upstream),
- no modification of Linux source code otherwise,

Effort needed to update Linux

- got some weird bug with dual display, the display driver would completely crash if both HDMI and LVDS outputs were enabled at the same time,
- update from 4.9 to 4.13 was very easy:
  - copy DTB from 4.9 to 4.13 and make sure the bindings haven't changed in-between those versions,
  - add the one-liner in `arch/arm/boot/dts/Makefile`,
  - patches were already upstream so nothing else to do,
- problem solved in ~30min by updating,

Questions? Suggestions? Comments?

Quentin Schulz
quentin.schulz@bootlin.com

`http://bootlin.com/pub/conferences/2017/elce/schulz-how-to-support-new-board-u-boot-linux`