FreeBSD ARM64: Porting on a new board

Emmanuel Vadot
manu@FreeBSD.org

BSDCan
Ottawa Canada
June 8 – 9, 2018

Who am I ?

- ARM Kernel Hacker for 2 and half year
- Self proclaimed maintainer for Allwinner SoCs (and now RockChip)
- Self proclaimed DTS Maintainer in FreeBSD
- U-Boot Maintainer
- Upstream guy in Linux for DTS and U-Boot

Agenda

- ARM/SoC/SBC
- Bootloader
- Serial
- First kernel boot
- Clocks and Resets
- Clock API

What is an SoC ?

- SoC == System On Chip
- ARM does not manufacture processor
- SoC vendor buys IP from ARM for the core
- Sometimes they also buy IP from other companies
- An SoC integrates a processor and peripherals

Single Board Computer

- SBC = Single Board Computer
- Generaly from another company than the SoC one
- Integrates SoC and other chips (PMU, PHY etc ...)
- Also adds GPIOs, SD/MMC, ethernet connectors etc ...

Pine64 Rock64

- RK3328 based SBC
- 1Gbps Ethernet
- USB3
- eMMC socket
- ...
- Donated by Pine64, Thank you TL Lim

Why Porting ?

- It's fun

Why Porting ?

- It's fun
- You learn a lot

Why Porting ?

- It's fun
- You learn a lot
- New arm/arm64 boards every month or so
  Having FreeBSD working on it expand our market

Why Porting ?

- It's fun
- You learn a lot
- New arm/arm64 boards every month or so
  Having FreeBSD working on it expand our market
- Porting to a new arch is hard, new SoC not that much

Bootloader requirement

- EFI aware

Bootloader requirement

- EFI aware
- FIT Image

Bootloader requirement

- EFI aware
- FIT Image
- AArch64 Linux Image

Bootloader requirement

- EFI aware
- FIT Image
- AArch64 Linux Image
- Convert kernel to kernel.bin

Bootloader requirement

- EFI aware
- FIT Image
- AArch64 Linux Image
- Convert kernel to kernel.bin
- But you want EFI aware

U-Boot life

- Mainline release every two months

U-Boot life

- Mainline release every two months
- SoC Vendor pick a release or a random commit

U-Boot life

- Mainline release every two months
- SoC Vendor pick a release or a random commit
- Stay on it and patch it

U-Boot life

- Mainline release every two months
- SoC Vendor pick a release or a random commit
- Stay on it and patch it
- SBC Vendor patch the SoC Vendor one

U-Boot life

- Mainline release every two months
- SoC Vendor pick a release or a random commit
- Stay on it and patch it
- SBC Vendor patch the SoC Vendor one
- Linux Distribution patch the SBC Vendor one

U-Boot life at RockChip

- Mainline release every two months

U-Boot life at RockChip

- Mainline release every two months
- SoC Vendor pick a release or a random commit
  Rockchip fork is based on 2017.09

U-Boot life at RockChip

- Mainline release every two months
- SoC Vendor pick a release or a random commit
  Rockchip fork is based on 2017.09
- Stay on it and patch it
  They do upstream some patches to mainline

U-Boot life at RockChip

- Mainline release every two months
- SoC Vendor pick a release or a random commit
  Rockchip fork is based on 2017.09
- Stay on it and patch it
  They do upstream some patches to mainline
- SBC Vendor patch the SoC Vendor one
  Few patches but none upstreamed

U-Boot life at RockChip

- Mainline release every two months
- SoC Vendor pick a release or a random commit
  Rockchip fork is based on 2017.09
- Stay on it and patch it
  They do upstream some patches to mainline
- SBC Vendor patch the SoC Vendor one
  Few patches but none upstreamed
- Linux Distribution patch the SBC Vendor one
  something like 100 patches, none upstreamed

U-Boot for Rock64

- Using vendor u-boot first

U-Boot for Rock64

- Using vendor u-boot first
- Decided to use the 'community' build by ayufan

U-Boot for Rock64

- Using vendor u-boot first
- Decided to use the 'community' build by ayufan
- Produce an sd card image for spi flash burning

U-Boot for Rock64

- Using vendor u-boot first
- Decided to use the 'community' build by ayufan
- Produce an sd card image for spi flash burning
- And it supports tftpboot

U-Boot for Rock64

- Using vendor u-boot first
- Decided to use the 'community' build by ayufan
- Produce an sd card image for spi flash burning
- And it supports tftpboot
- andreast@FreeBSD.Org updated to recent u-boot

Serial

- FreeBSD Kernel try to resolve the $/chosen/std\{in,out\}$

Serial

- FreeBSD Kernel try to resolve the */chosen/std*{*in,out*}
- Fallback on dtb node named *serial0*

Serial

- FreeBSD Kernel try to resolve the */chosen/std*{*in,out*}
- Fallback on dtb node named *serial0*
- Node need it's status to be != *disabled*

Serial

- FreeBSD Kernel try to resolve the */chosen/std*{*in,out*}
- Fallback on dtb node named *serial0*
- Node need it's status to be != *disabled*
- There is a good chance that the uart controller is already supported

First Boot

- uart + loader.efi = kernel booting

First Boot

- uart + loader.efi = kernel booting
- using a mfsroot can be handy

Device Driver

- Now you can write device drivers !!!

Device Driver

- Now you can write device drivers !!!
- Well no, you need clocks and resets support first

Clocks

Clocks

- 24Mhz oscilator on the board

Clocks

- 24Mhz oscilator on the board
- SoC derive some PLLs based on it

Clocks

- 24Mhz oscilator on the board
- SoC derive some PLLs based on it
- Peripherals clocks are derived from PLLs

Clocks

- 24Mhz oscilator on the board
- SoC derive some PLLs based on it
- Peripherals clocks are derived from PLLs
- Peripherals clocks can choose between multiples parent

Clocks

- 24Mhz oscilator on the board
- SoC derive some PLLs based on it
- Peripherals clocks are derived from PLLs
- Peripherals clocks can choose between multiples parent
- Each SoCs is different

Clocks

- 24Mhz oscilator on the board
- SoC derive some PLLs based on it
- Peripherals clocks are derived from PLLs
- Peripherals clocks can choose between multiples parent
- Each SoCs is different
- Most of the time Vendors reuse the clock models between SoCs

Resets

- Active/Deactivate the peripheral

Resets

- Active/Deactivate the peripheral
- Usually just a bit in one register

How to manage clocks

- Calling socname_clock_blah(uint64_t freq, bool enable)

How to manage clocks

- Calling socname_clock_blah(uint64_t freq, bool enable)
- It means a lot of if/else in driver code

  No generic way to manage clocks and clock/parent relationship

  No code reuse between SoCs (or just a little)

How to manage clocks

- Calling socname_clock_blah(uint64_t freq, bool enable)
- It means a lot of if/else in driver code
  No generic way to manage clocks and clock/parent relationship
  No code reuse between SoCs (or just a little)
- Right way is to use the clock api

Clock API

- First appeared in FreeBSD 11, work done by mmel@FreeBSD.Org

Clock API

- First appeared in FreeBSD 11, work done by mmel@FreeBSD.Org
- Used for Nvidia Tegra, Allwinner and RockChip SoCs

Clock API

- First appeared in FreeBSD 11, work done by mmel@FreeBSD.Org
- Used for Nvidia Tegra, Allwinner and RockChip SoCs
- Clock driver registers clocks

Clock API

- First appeared in FreeBSD 11, work done by mmel@FreeBSD.Org
- Used for Nvidia Tegra, Allwinner and RockChip SoCs
- Clock driver registers clocks
- Driver can enable/disable/change frequency of clock in a SoCs independant way.

Clock API

- First appeared in FreeBSD 11, work done by mmel@FreeBSD.Org
- Used for Nvidia Tegra, Allwinner and RockChip SoCs
- Clock driver registers clocks
- Driver can enable/disable/change frequency of clock in a SoCs independant way.
- Sadly no man pages

Clock API - Clock type

- basic clocks type exists

Clock API - Clock type

- basic clocks type exists
- clk_fixed : Either a fixed frequency or child of another clock + multiplier or divider

Clock API - Clock type

- basic clocks type exists
- clk_fixed : Either a fixed frequency or child of another clock + multiplier or divider
- clk_div : Support fractional divider or divider table

Clock API - Clock type

- basic clocks type exists
- clk_fixed : Either a fixed frequency or child of another clock + multiplier or divider
- clk_div : Support fractional divider or divider table
- clk_mux : Simple multiple parent clock

Clock API - Clock type

- basic clocks type exists
- clk_fixed : Either a fixed frequency or child of another clock + multiplier or divider
- clk_div : Support fractional divider or divider table
- clk_mux : Simple multiple parent clock
- All SoCs specific clock needs to be created

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)
- *clknode_init* is called during the clock registration
  Should init the parent(s)

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)
- *clknode_init* is called during the clock registration
  Should init the parent(s)
- *clknode_setgate* Enable/Disable the clock

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)
- *clknode_init* is called during the clock registration
  Should init the parent(s)
- *clknode_setgate* Enable/Disable the clock
- *clknode_setmux* Switch parent

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)
- *clknode_init* is called during the clock registration
  Should init the parent(s)
- *clknode_setgate* Enable/Disable the clock
- *clknode_setmux* Switch parent
- *clknode_recalc* Refresh the cached value of the current clock frequency

Clock API - Create a clock type

- Subclass the clknode_class (See clknode_if.m)
- *clknode_init* is called during the clock registration
  Should init the parent(s)
- *clknode_setgate* Enable/Disable the clock
- *clknode_setmux* Switch parent
- *clknode_recalc* Refresh the cached value of the current clock frequency
- *clknode_setfreq* Change the frequency of the clock

Clock API - Create a Clock Unit Driver

- 1) Device create a clock domain with *clkdom_create*

Clock API - Create a Clock Unit Driver

- 1) Device create a clock domain with *clkdom_create*
- 2) Create a clknode with *clknode_create*

Clock API - Create a Clock Unit Driver

- 1) Device create a clock domain with *clkdom_create*
- 2) Create a clknode with *clknode_create*
- 3) Register the clknode with *clknode_register*

Clock API - Create a Clock Unit Driver

- 1) Device create a clock domain with *clkdom_create*
- 2) Create a clknode with *clknode_create*
- 3) Register the clknode with *clknode_register*
- Repeat 2 and 3 for every clock on the SoC and finalize the clock domain with *clkdom_finit*

Clock API - Create a Clock Unit Driver

- 1) Device create a clock domain with *clkdom_create*
- 2) Create a clknode with *clknode_create*
- 3) Register the clknode with *clknode_register*
- Repeat 2 and 3 for every clock on the SoC and finalize the clock domain with *clkdom_finit*
- Use *clk_set_assigned* to parse the 'assigned-clock' properties

assigned-clock example (1)

```
assigned-clocks =
<&cru DCLK_LCDC>, <&cru SCLK_PDM>,
<&cru SCLK_RTC32K>, <&cru SCLK_UART0>,
<&cru SCLK_UART1>, <&cru SCLK_UART2>,
<&cru ACLK_BUS_PRE>, <&cru ACLK_PERI_PRE>,
<&cru ACLK_VIO_PRE>, <&cru ACLK_RGA_PRE>,
<&cru ACLK_VOP_PRE>, <&cru ACLK_RKVDEC_PRE>,
<&cru ACLK_RKVENC>, <&cru ACLK_VPU_PRE>,
<&cru SCLK_VDEC_CABAC>, <&cru SCLK_VDEC_CORE>,
<&cru SCLK_VENC_CORE>, <&cru SCLK_VENC_DSP>,
<&cru SCLK_SDIO>, <&cru SCLK_TSP>,
<&cru SCLK_WIFI>, <&cru ARMCLK>,
<&cru PLL_GPLL>, <&cru PLL_CPLL>,
<&cru ACLK_BUS_PRE>, <&cru HCLK_BUS_PRE>,
<&cru PCLK_BUS_PRE>, <&cru ACLK_PERI_PRE>,
<&cru HCLK_PERI>, <&cru PCLK_PERI>,
<&cru SCLK_RTC32K>;
```

assigned-clock example (2)

```
assigned-clock-parents =
<&cru HDMIPHY>, <&cru PLL_APLL>,
<&cru PLL_GPLL>, <&xin24m>,
<&xin24m>, <&xin24m>;
assigned-clock-rates =
<0>, <61440000>,
<0>, <24000000>,
<24000000>, <24000000>,
<15000000>, <15000000>,
<100000000>, <100000000>,
<100000000>, <100000000>,
<50000000>, <100000000>,
<100000000>, <100000000>,
<50000000>, <50000000>,
<50000000>, <50000000>,
<24000000>, <600000000>,
<491520000>, <1200000000>,
<150000000>, <75000000>,
<75000000>, <150000000>,
<75000000>, <75000000>,
<32768>;
```

Clock API - Create a Reset provider

- Usually same device as the clock unit

Clock API - Create a Reset provider

- Usually same device as the clock unit
- Two DEVMETHODs hwreset_assert and hwreset_is_asserted

Clock API - Create a Reset provider

- Usually same device as the clock unit
- Two DEVMETHODs hwreset_assert and hwreset_is_asserted
- Register as a reset provider with hwreset_register_ofw_provider

Clock API - Driver usage

- Clocks for devices are standardized

Clock API - Driver usage

- Clocks for devices are standardized
- Device driver get the clock using *clk_get_by_ofw_name* or *clk_get_by_ofw_index*

Clock API - Driver usage

- Clocks for devices are standardized
- Device driver get the clock using *clk_get_by_ofw_name* or *clk_get_by_ofw_index*
- Enable/Disable using *clk_enable,disable,stop*

Clock API - Driver usage

- Clocks for devices are standardized
- Device driver get the clock using *clk_get_by_ofw_name* or *clk_get_by_ofw_index*
- Enable/Disable using *clk_enable,disable,stop*
- Set/Get frequency using *clk_set,get_freq*

Clock API - Driver usage

- Clocks for devices are standardized
- Device driver get the clock using *clk_get_by_ofw_name* or *clk_get_by_ofw_index*
- Enable/Disable using *clk_enable,disable,stop*
- Set/Get frequency using *clk_set,get_freq*
- Free the clock using *clk_release*

Clock API - Advices

- Starts we a few clocks - Mandatory PLLs, AHB clocks etc ... Also start with no set_freq method

Clock API - Advices

- Starts we a few clocks - Mandatory PLLs, AHB clocks etc ... Also start with no set_freq method
- Use *clkdom_dump* after *clkdom_finit* under boot verbose

Clock API - Advices

- Starts we a few clocks - Mandatory PLLs, AHB clocks etc ... Also start with no set_freq method
- Use *clkdom_dump* after *clkdom_finit* under boot verbose
- Use the *hw.clock* sysctl

Clock API - Advices

- Starts we a few clocks - Mandatory PLLs, AHB clocks etc ... Also start with no set_freq method
- Use *clkdom_dump* after *clkdom_finit* under boot verbose
- Use the *hw.clock* sysctl
- Make sure that your clock is really working and that it is not a bootloader leftover

Device Driver

- Now you can write device drivers !!!

Device Driver

- Now you can write device drivers !!!
- Check if a driver already exists in the tree
  Some Vendor often use a common IP as a base

Device Driver

- Now you can write device drivers !!!
- Check if a driver already exists in the tree
  Some Vendor often use a common IP as a base
- Beware of docs, sometimes you need to read linux drivers ...

Questions ?

Emmanuel Vadot
manu@freebsd.org
Twitter: @manuvadot
Freelance contractor available for work