Porting Linux to a new SoC

Who am I?

- PrasannaKumar Muralidharan
    - Linux kernel enthusiast
    - Contributed to a few open source projects
    - Contributed several patches to hwrng subsystem
    - Wrote CI20 PRNG driver
    - Implemented X1000 SoC support
- Email: prasannatsmkumar@gmail.com
- IRC Nick: prasannatsm
- Linkedin: https://www.linkedin.com/in/prasannakumartsm/

End result of porting

- Single CPU
- Early printk messages via UART
- Interrupts
- Timer
- Serial Port
- Boots tiny rootfs embedded in initramfs

Assumptions

- Boot loader is available, capable of booting Linux
- CPU architecture is supported by Linux
- SoC Programmer Manual is available or vendor provided Linux port is available
- Or info on similar SoC

What’s wrong with vendor code?

- Usually SoC vendor provide ancient version of Linux
- Does not use proper Linux provided infrastructure/framework
- Has major design issues and bugs
- Crashes if use case is different from vendor’s use case
- Security vulnerabilities
- Generic user space may not work
- Ugly, unreadable code

Vendor code has workaround for hardware bugs. It helps a lot while debugging issues.

What is Device Tree

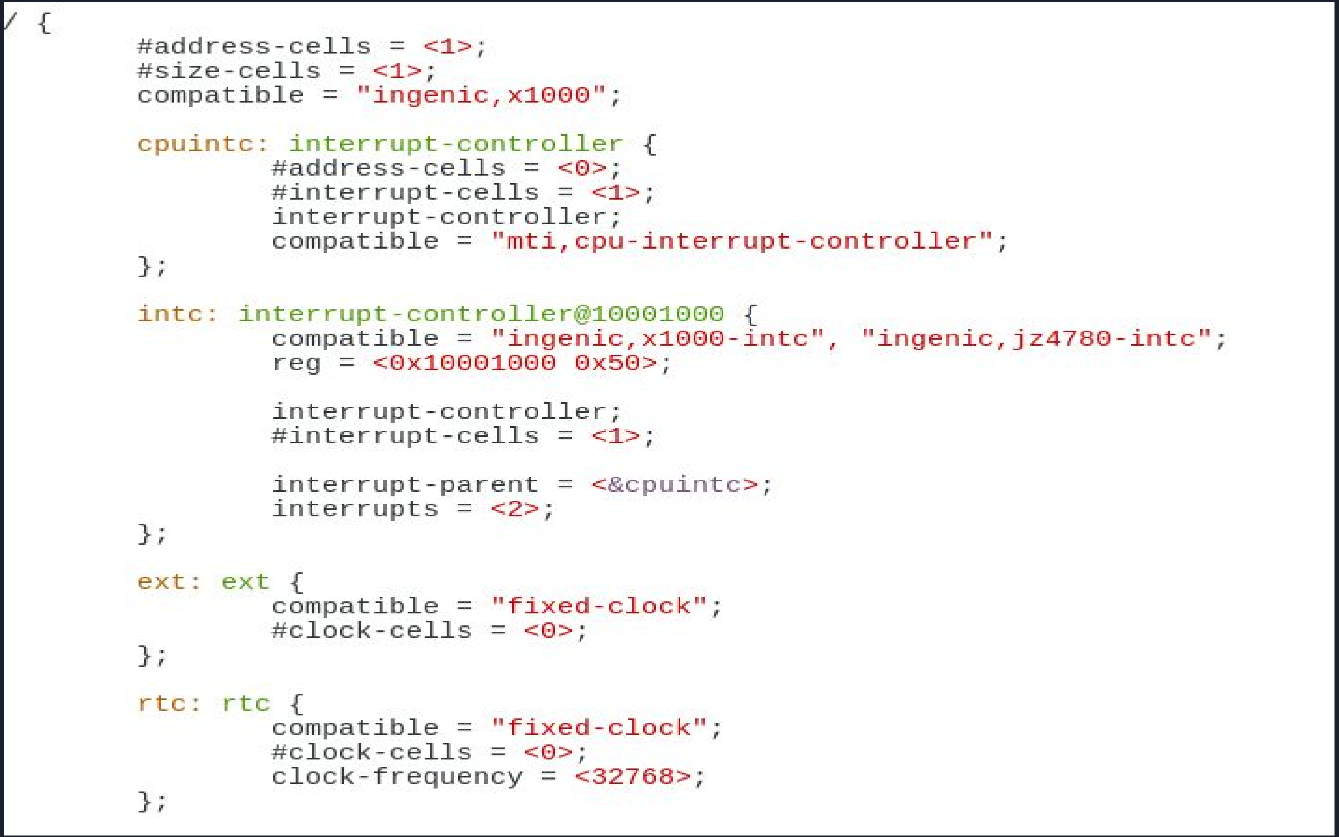
Device Tree provides hardware description

- Similar to what ACPI does but far better than ACPI
- Required for non discoverable devices
- Ideally should be provided by boot loader to Linux
  - Power, SPARC boot loaders does this
  - uboot does not provide its own device tree
- Device Tree Source (dts) is converted to Device Tree Blob (dtb)
- uboot takes dtb from disk or via tftp, passes it to Linux
- Or dtb is appended with kernel image (required for legacy uboot)

Device Tree

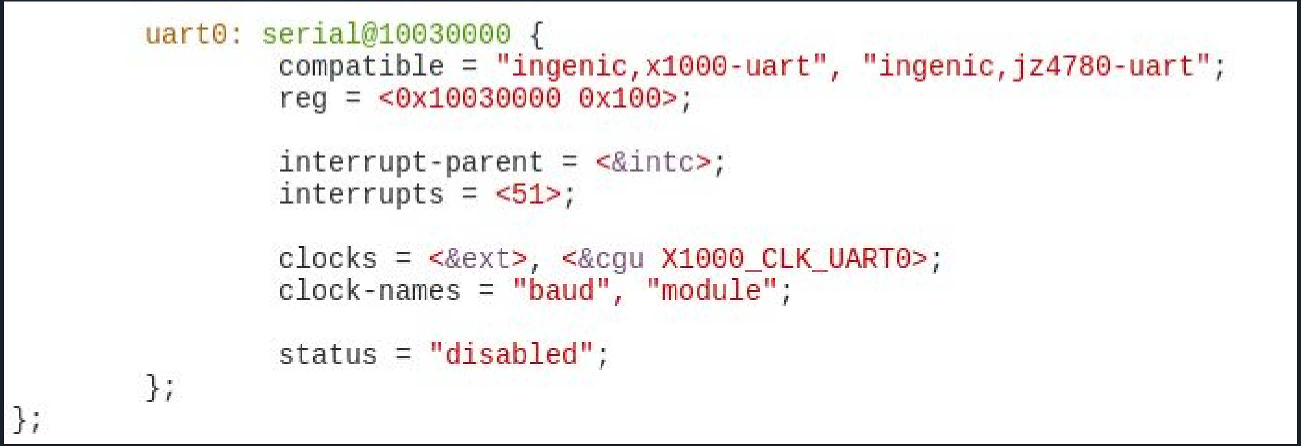
- Add one <soc>.dtsi file in arch/mips/boot/dts/<soc-vendor>/ that describes the SoC
- One <board>.dts for each board that uses the SoC
  - <soc>.dtsi is included in <board>.dts file
- Multiple boards can use the same SoC, each board gets one dts
- Add dtb-$(CONFIG_ARCH_<yourboard>) line to arch/mips/boot/dts/<soc-vendor>/Makefile for all boards. This will build dtb for the boards

X1000 SoC dtsi (x1000.dtsi)



```
/ {
	#address-cells = <1>;
	#size-cells = <1>;
	compatible = "ingenic,x1000";

	cpuintc: interrupt-controller {
		#address-cells = <0>;
		#interrupt-cells = <1>;
		interrupt-controller;
		compatible = "mti,cpu-interrupt-controller";
	};

	intc: interrupt-controller@10001000 {
		compatible = "ingenic,x1000-intc", "ingenic,jz4780-intc";
		reg = <0x10001000 0x50>;

		interrupt-controller;
		#interrupt-cells = <1>;

		interrupt-parent = <&cpuintc>;
		interrupts = <2>;
	};

	ext: ext {
		compatible = "fixed-clock";
		#clock-cells = <0>;
	};

	rtc: rtc {
		compatible = "fixed-clock";
		#clock-cells = <0>;
		clock-frequency = <32768>;
	};
```

X1000 SoC dtsi (x1000.dtsi)



```
        uart0: serial@10030000 {
                compatible = "ingenic,x1000-uart", "ingenic,jz4780-uart";
                reg = <0x10030000 0x100>;

                interrupt-parent = <&intc>;
                interrupts = <51>;

                clocks = <&ext>, <&cgu X1000_CLK_UART0>;
                clock-names = "baud", "module";

                status = "disabled";
        };
};
```

Basic code in arch/mips/<soc>/setup.c

Implements the following routines (for MIPS)

- plat_mem_setup - detects RAM size
- device_tree_init - unflattens device tree blob
- arch_init_irq - sets up interrupts
- Machine reset, halt

Usually very small and could be as simple as calling generic functions

Earlycon support

- Prints early kernel messages
- Polling based uart driver, interrupts not availble yet
- Must implement struct console’s write method
- Not necessary to implement uart read support
- Does not require kernel recompilation, can be enabled using command line
  - Earlyprintk support requires kernel recompilation

Interrupt controller support

- Should use irqchip framework
- No fixed number of IRQ (NR_IRQ)
- If the SoC uses CPU architecture’s generic interrupt controller, drivers are already available
- Other SoC from the same vendor can have same interrupt controller hardware, its driver could be reused
- If the SoC uses a custom interrupt controller then a new driver has to be implemented
- If irqchip framework is used, arch_init_irq should just call irqchip_init. irqchip_init scans device tree and call the interrupt controller driver’s init routine
- Instantiated from Device Tree .dtsi file

Timer driver

- Should use clocksource framework
- Timer driver must register
    - A clocksource device - a free running timer. It is used to keep track of passing time.
    - A clockevents device - a timer that can be programmed for one-shot or repeated events notified via interrupt
    - Note: Each CPU in SMP machine requires a clockevents device. Not going to go in detail about this.
- Driver must have a device tree binding, instantiated from dtsi file

Serial Port driver

- Common uart hardware drivers are available in kerenl. If the SoC uses any of these blocks the driver can be reused with few modification. Example 8250 compatible uart controller
- Custom UART controller requires a new driver that works with uart and console subsystems
- This is also instantiated from Device Tree .dtsi file

Ingenic X1000 SoC

- Reused Ingenic JZ4740 SoC’s
  - SoC setup code
  - Interrupt controller driver
  - Timer driver
  - Serial port driver
  - Parts of clock driver
- Writing a device tree source is enough to provide initial SoC support

Ready to submit

- All basic blocks are in place
- Linux can boot successfully in the SoC to a tiny rootfs embedded in initramfs
- Submit the basic SoC support code
- This provides a solid base for further development
- No disk, SMP or Power Management support yet

Clock driver

- Every digital circuit needs a clock
- Few clock sources are present - could be internal or external
- Every clock sent to a hardware block is derived from the clock source
- Common clock framework implements API required for device drivers to access the clock that is sent to a hardware block
- Clock provider driver has to be implemented
- Necessary for power management, dynamic CPU frequency scaling
- Instantiated via device tree

IO Pins with multiple functionality

- Modern SoCs can connect to several IO peripherals but have limited number of pins
- A set of pins is capable of working with multiple peripherals
- A particular functionality can be chosen from a set of supported functionality
    - A set of pins can work as USB bus, can take audio signals to a headphone jack etc
- The functionality is based on how a board uses the SoC, can be selected at runtime by software

Pinctrl driver

- Pinctrl subsystem provides a way to expose all functionalities a set of pins could have in a given SoC
- Pinctrl subsystem provides API to let the device drivers choose a particular functionality
- For example, a sound driver could request a pin configuration that supports audio data transport if the board is designed that way
- Pinctrl framework interacts with a SoC specific pinctrl driver to determine the available configuration, select a configuration etc
- Pin configuration can be described in device tree source

GPIO driver

- Stands for General Purpose IO
- Some pins are left for generic functionality
- Can connect any IO peripheral or electronic circuit
- Touch screen controller, LEDs etc can work with these pins
- GPIO framework exposes an API for drivers to claim GPIO pins and use it as they want
- GPIO pins can generate interrupts as well
- GPIO framework needs a SoC specific gpio driver to do hardware specific stuff
- Instantiated via device tree

Time to submit

- At this juncture basic infrastructure for the SoC is ready
- Time to submit it upstream and get it merged
- It’s time to work on drivers for peripherals

Next step

Next step would be to implement support for

- DMA controller
- SMP
- I2C, I2S, PCI, USB
- Disk
- Network (eth or wifi)
- Audio controller
- Touch controller
- Display
- Crypto hardware
- etc

Stuff that I am working on

- MIPS Creator CI20 pseudo random number generator driver
- CI20 efuse driver, exposing MAC address to dm9000 driver from CI20 efuse
- Ingenic X1000 SoC support

Things on my todo list

- Complete clock driver for X1000 SoC
- ADC driver for CI20
- SMP support on CI20
- CI20 display panel driver
- Upstreaming kernel patches carried by OpenWRT / LEDE project

Questions?

Thank You