libtrue

You can’t handle the truth

http://libtrue.xyz/

GENERIC ARM

Andrew Turner
andrew@FreeBSD.org


BOOT
ON ALL THE THINGS
imgflip.com

About me

- Research Associate in the University of Cambridge Computer Laboratory
- Freelance Software Engineer

About me

- FreeBSD committer since 2010
- Focus on arm and arm64
- Major projects:
  - ARM EABI
  - arm64

History of FreeBSD/armv6

- FreeBSD on ARMv6 and ARMv7
- Started in August 2011
- Initial support for:
    - Marvell Armada XP
    - Ti OMAP3, OMAP4, AM335x
    - Nvidia Tegra 2

History of FreeBSD/armv6

- Imported into HEAD in August 2012
- Added support for more SoCs (System on Chip)
  - Broadcom BCM2835, BCM2836, (BCM2837)
  - NXP (Freescale) i.MX5, i.MX6, Vybrid
  - Samsung Exynos
  - Allwinner (too many to list)

History of FreeBSD/armv6

- Altera/Intel FPGA Cyclone5, Arria10
- Annarana
- Rockchip
- Xylinx Zync7
- QEMU
- gem5

What was needed for GENERIC?

What was needed for GENERIC?

- Early hardware configuration
- Device enumeration
- Handling singleton functions

Early hardware configuration

Early page tables

Problem: Each kernel had a hard coded physical address

Early page tables

Solution: Runtime detection of physical address, build page tables around it

Early page tables

1. Find where in the physical address space the kernel has been loaded

2. Build a page table around it

Early page tables

- How to find the kernels physical address?
- The program counter is just another register
  - Can use it as a source in e.g. a move, bit-clear, or load
- ARM has instructions to simplify pc-relative memory access

Early page tables

- Need to create two mappings:
  - Identity mapping, VA == PA — keep running when the MMU is enabled
  - Kernel mapping — keep running when branching to C code

PLATFORM

Problem: Everyone just copied the early boot code with minor changes.

PLATFORM

Solution: Pull out the common code, create new functions to handle per-SoC code

PLATFORM

Problem: Each SoC will need to implement the same functions

PLATFORM

Solution: Detect the SoC, use kobj to provide different implementations of these functions
(How PowerPC does it)

PLATFORM

- Used early in the boot to handle the hardware differences
- Started as part of my “specific_leg” project branch
- Proved a GENERIC kernel was possible
  - Almost the same kernel on a PANDABOARD and Raspberry Pi

FORM

Map device early memory

Start non-boot CPUs

- Minimal SMP example:

```
static platform_method_t virt_methods[] =
  PLATFORMMETHOD(platform_devmap_init,
    virt_devmap_init),
#ifdef SMP
  PLATFORMMETHOD(platform_mp_start_ap,
    virt_mp_start_ap),
  PLATFORMMETHOD(platform_mp_setmaxid,
    virt_mp_setmaxid),
#endif
  PLATFORMMETHOD_END,
};
FDT_PLATFORM_DEF(virt, "virt", 0
  "linux,dummy-virt", 1);
```

Tell the kernel how many
CPUs there are

PLATFORM

- Optional interfaces:
  - platform_attach — Called when probe was successful
  - platform_lastaddr — Returns the start of unusable kernel address space
  - platform_gpio_init — Called just before the console is ready, e.g. to configure GPIOs for the UART
  - platform_late_init — Called after the console is ready
  - platform_cpu_reset — Reboot the SoC

PLATFORM

Should only be needed by the bus code (FDT)

- Super optional interface:
    - platform_probe — Probe to see if the kernel is running on the supported hardware.

PLATFORM

- FDT_PLATFORM defines a platform that could be used

```
FDT_PLATFORM_DEF(virt, "virt", 0,
  "linux,dummy-virt", 1);
```

- virt — Variable name, e.g. will use `virt_methods`
- “virt” — Human readable name
- 0 — Unused (size of softc)
- “linux,dummy-virt” — FDT combatible string to match
- 1 — Number of iterations to busy wait in the early DELAY code

Device Enumeration

Flattened Device Tree

Problem (I): Memory mapped devices are non-enumerable

Flattened Device Tree

Problem (II): The kernel had a hardcoded list of these devices and their location

Flattened Device Tree

Solution: Have the firmware provide a hardware description to the kernel

Flattened Device Tree

- Added to arm before the armv6 project
- Is a requirement on armv6
- Also optionally used on AMD64, arm, arm64, i386, MIPS, PowerPC, and RISC-V

Flattened Device Tree

```
/dts-v1/;
/ {
  model = "My Board";
  compatible = “manufacturer,my_board","soc_vendor,my_soc";
  memory {
   reg = <0x10000000 0x20000000>;
  };
  soc {
    compatible = "simple-bus";
    #address-cells = <1>;
    #size-cells = <1>;
    my_device {
      compatible = "soc_vendor,my_device";
      reg = <0xf0000000 0x1000>;
    };
  };
};
```

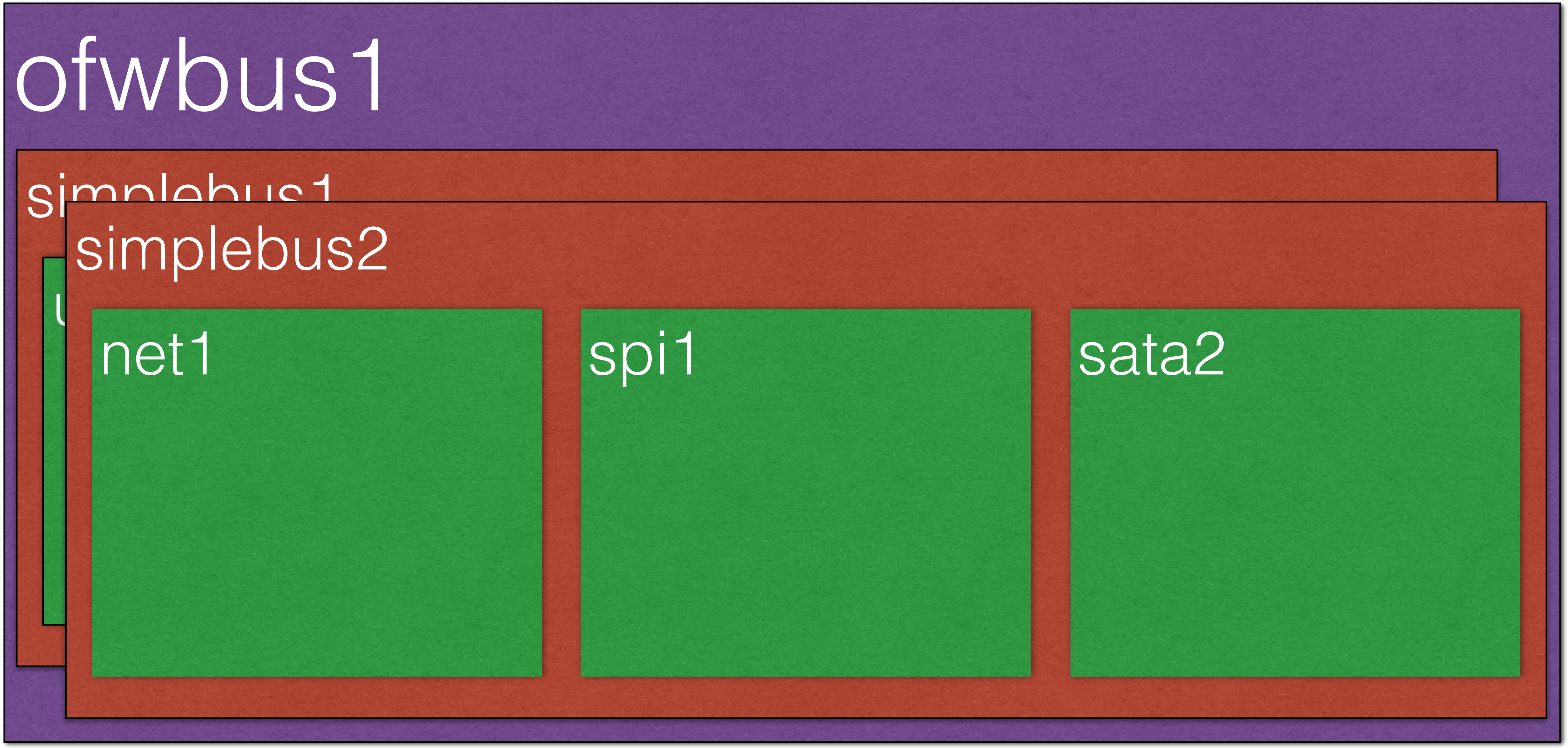
Unique board and SoC names

Board RAM

Device name, used by the probe function

Device memory range

Flattened Device Tree



Flattened Device Tree

- Probe function code:

```
static int
my_device_probe(device_t dev) {
 if (!ofw_bus_status_okay(dev))
  return (ENXIO);

 if (!ofw_bus_is_compatible(dev,
   “soc_vendor,my_device"))
  return (ENXIO);

 device_set_desc(dev, “My device”);
 return (BUS_PROBE_DEFAULT);
}
```

Handling singleton functions

MULTIDELAY

Problem: Each timer controller provided it’s own DELAY implementation

MULTIDELAY

Solution: Each timer provides a callback to handle the needed delay

MULTIDELAY

- The timer drivers register a callback to perform the delay:

```
arm_set_delay(pseudo_timer_delay, sc);
```

- Provides the callback, and an argument to pass

MULTIDELAY

- Example:

```
static void pseudo_timer_delay(int usec, void *arg) {
 struct pseudo_timer_softc *sc = arg;
 uint64_t first, last;
 uint32_t counts_per_usec;
 int32_t counts;

 counts_per_usec = (sc->timer_frequency / 1000000) + 1;
 counts = usec * counts_per_usec;
 first = pseudo_read_counter(sc);
 while (counts > 0) {
  last = pseudo_read_counter(sc);
  counts -= last - first;
  first = last;
 }
}
```

MULTIDELAY

- Default PLATFORM implementation:

```
static void
platform_delay(int usec
 int counts;
 for (; usec > 0; usec--
  for (counts =
   plat_obj->cls->delay_count;
   counts > 0; counts--)
    cpufunc_nullop();
}
```

From
FDT_PLATFORM_DEF

INTRNG

Problem: The interrupt handling code was only able to support a single interrupt controller

INTRNG

Solution: New framework to handle multiple interrupt controllers

INTRNG

- Started in 2012 as a Google Summer of Code project by Jakub Klama
- Worked on by Svatopluk Kraus and Ian Lepore
- Imported in the tree in 2015
- Optional on arm and mips, required on arm64

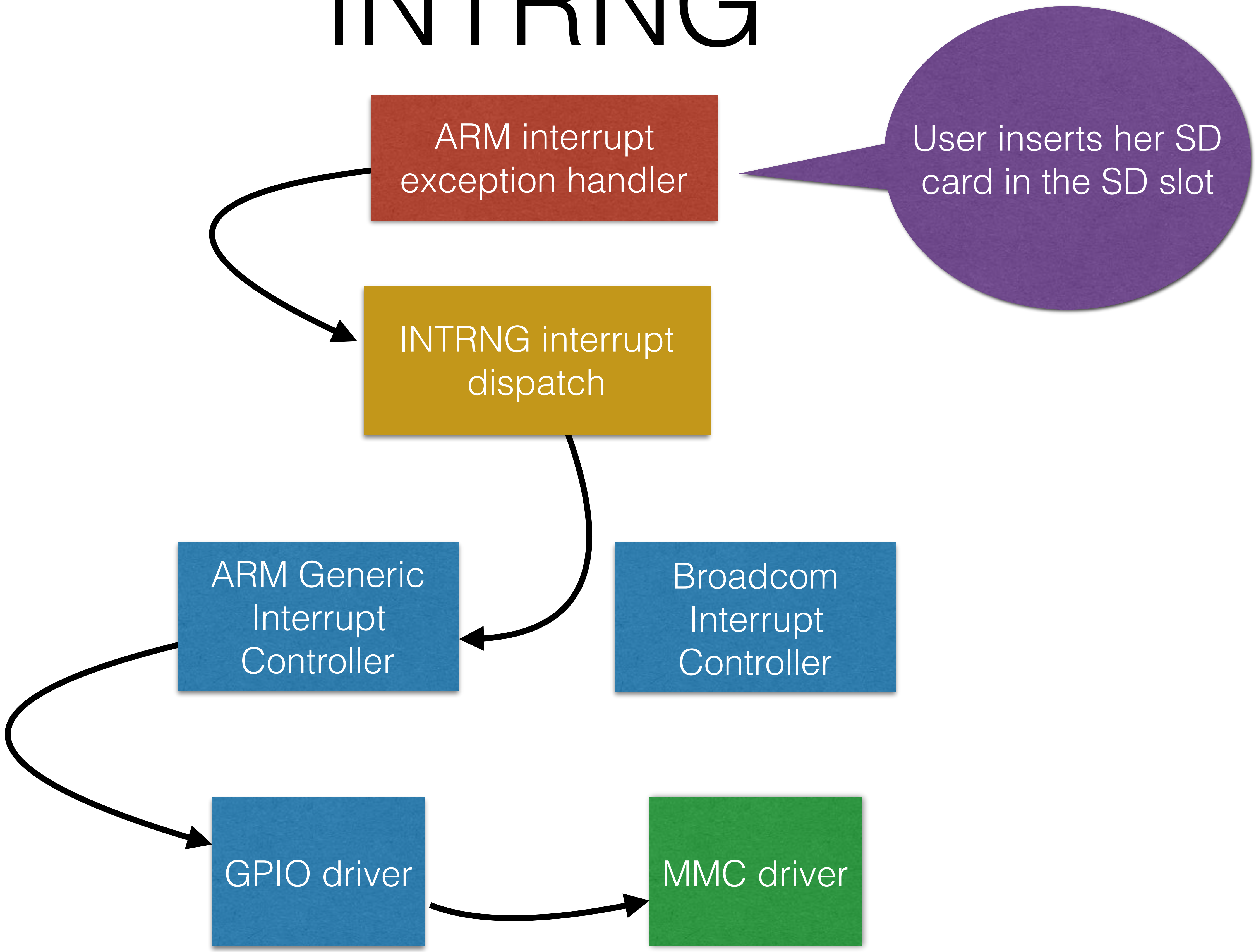
INTRNG

- Based on a tree of interrupt controllers
- Any driver could be a controller, e.g. a GPIO driver

INTRNG

- Creates a new newbus interface:
  - pic_bind_intr
  - pic_map_intr
  - pic_setup_intr, pic_teardown_intr
  - pic_post_filter
  - pic_pre_ithread, pic_post_ithread

INTRNG


ARM interrupt exception handler
User inserts her SD card in the SD slot
INTRNG interrupt dispatch
ARM Generic Interrupt Controller
Broadcom Interrupt Controller
GPIO driver
MMC driver

Putting it all together

Putting it together

- Create a test config
    - Merged the VIRT (qemu) and ALLWINNER kernel configs
- Test booting on both
    - Thanks to Emmanuel Vadot for testing on Allwinner

Putting it together

- Emmanuel gave a presentation on FreeBSD on Allwinner

1. Mentioned there is no GENERIC kernel for armv6 in subversion
2. I committed GENERIC
3. I pointed out his talk was out of date in the question section

Putting it together

- After EuroBSDCon more SoCs were added
- Now support:
  - All ARMv7 Allwinner (that FreeBSD supports)
  - Ti am335x and OMAP4
  - Raspberry Pi 2
  - Nvidia Tegra T124

Putting it together

- armv6 now requires INTRNG
    - Only 2 SoCs are missing
- Most armv6 configs use, or have patches for PLATFORM & PLATFORM_SMP
    - Except Marvell
- Many support MULTIDELAY

Putting it together

- The release scripts have been updated
  - Except BEAGLEBONE.conf and CUBIEBOARD.conf
- NanoBSD updated to use GENERIC for qemu
  - Needs an update for Raspberry Pi 2

Remaining issues

Remaining issues

- Not all kernel configurations have been converted
- Old versions of U-Boot don’t work well with GENERIC
  - Often assumes booting a kernel from a raw partition
  - Missing API or EFI support
- Many SoCs hardcode the CPU count
- pl310 needs a per-SoC function

Summary

Summary

- GENERIC on armv6 is possible
- Mostly engineering to fix replicated functions
- Need to support more SoCs
    - (patches welcome)

Questions?