

FreeBSD/arm na DaVinci DMSoC

Jakub Klama

jceel@semihalf.com

MeetBSD 2010, Kraków

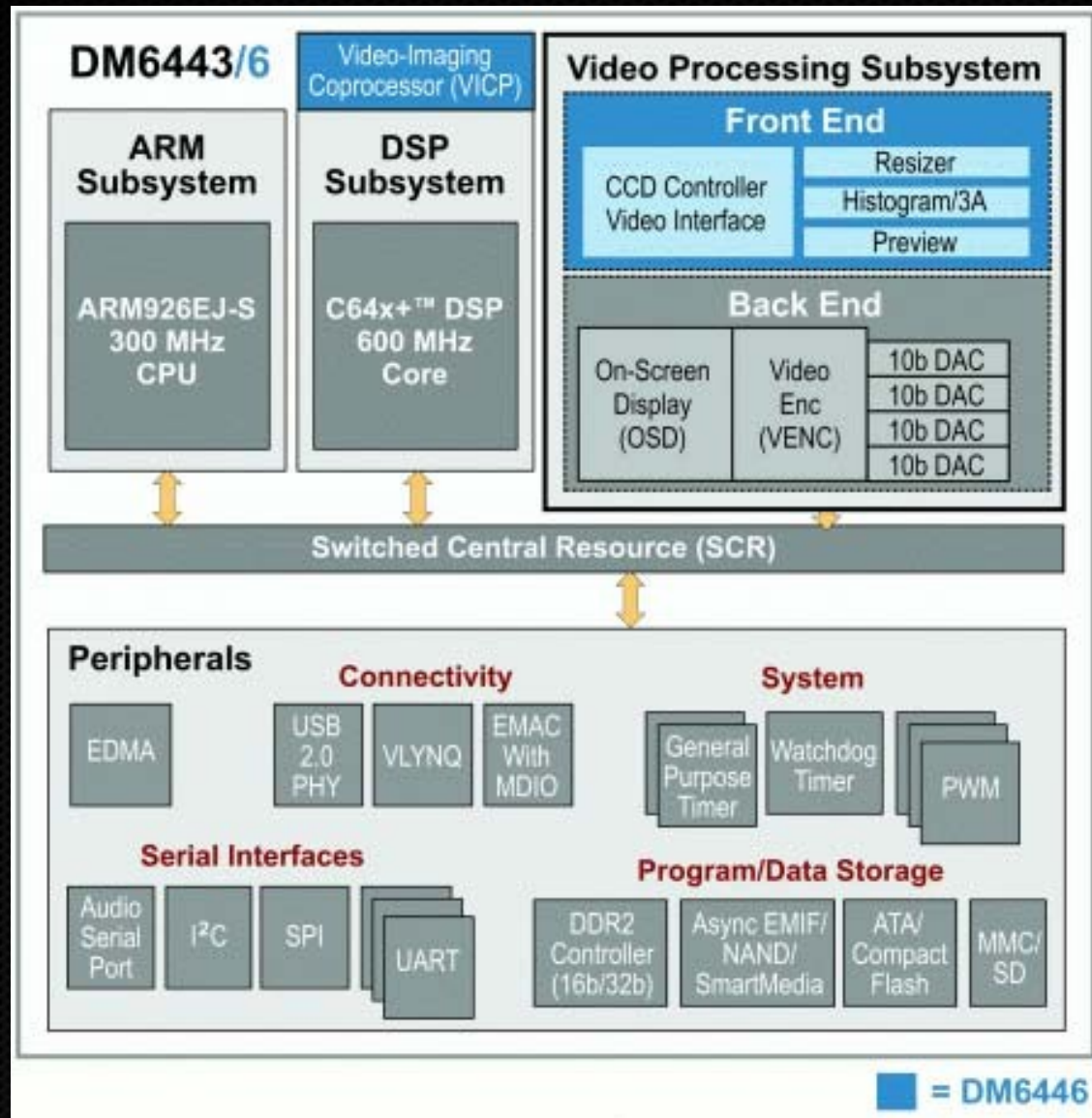
Szkic

- Wstęp, krótko o sprzęcie, wymagania początkowe, struktura portu
- Pierwsze uruchomienie kernela
 - machdep
 - wczesna konsola
- Podstawowe sterowniki
 - UART
 - Kontroler przerwań
 - Timer
- Wejście do userspace
- Dalszy rozwój: multiuser mode, ethernet, MMC/SD
- Pytania, uwagi

Wstęp

- **Struktura portu: core, SoC, platforma**
- **Texas Instruments TMS320DM6446**
 - Dwa rdzenie: ARMv5 i DSP
 - Rdzeń DSP nas chwilowo nie interesuje
 - SoC: DM644x
 - Platforma: Semihalf HASE-1

TI DaVinci

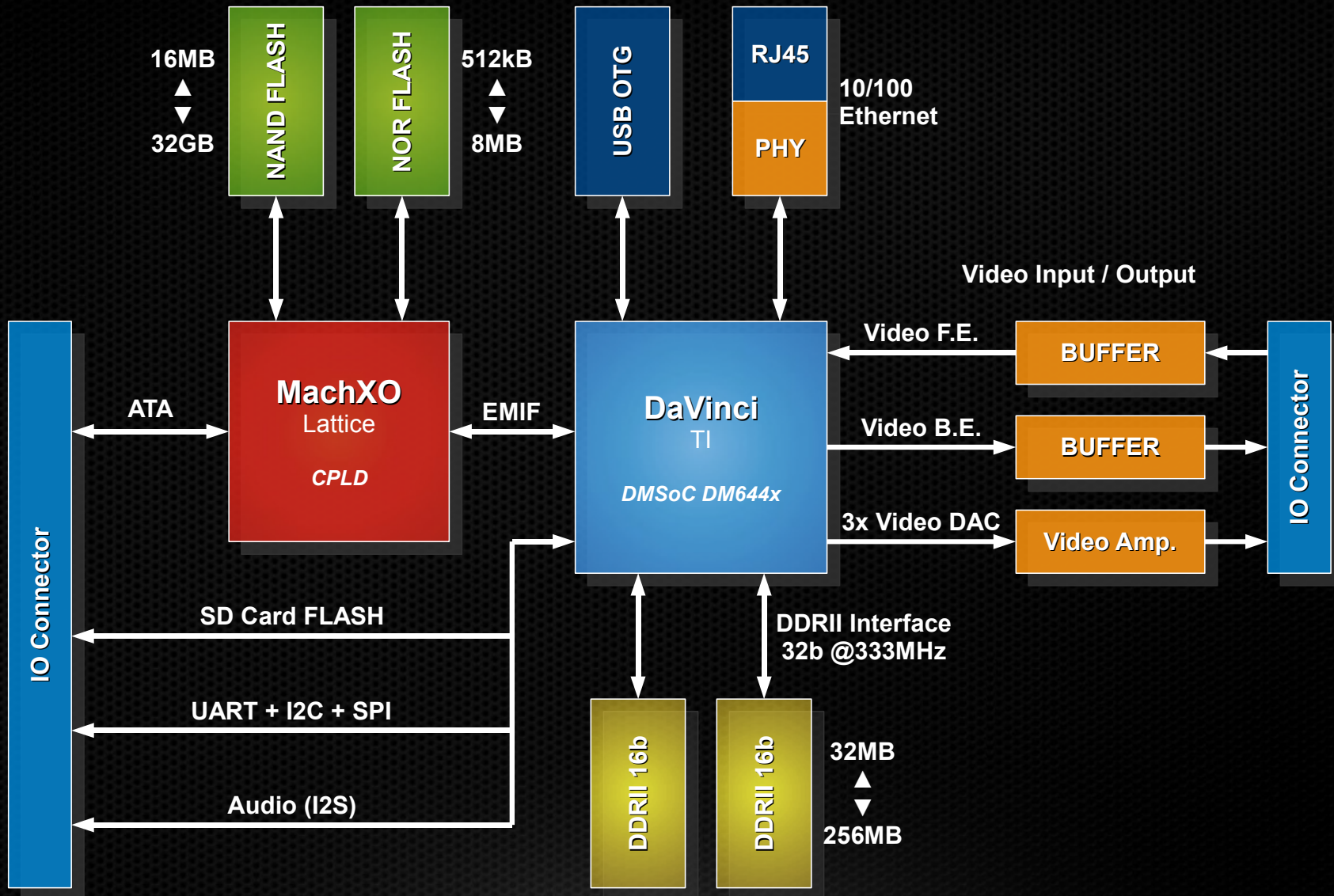


Semihalf HASE-I

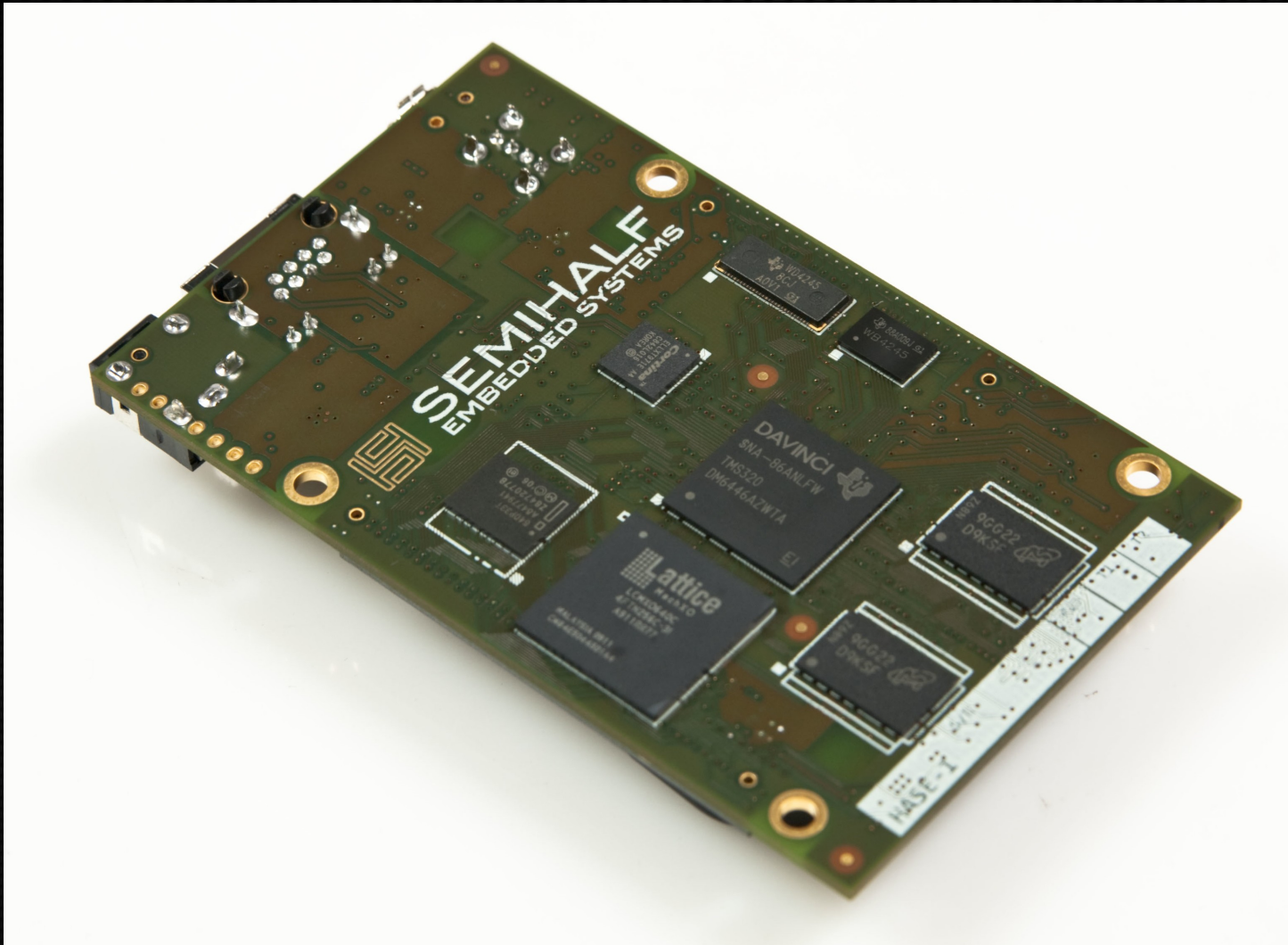
■ Charakterystyka

- Dwurdzeniowy procesor TI TMS320DM6441/6443/6446
 - ARM926EJ-S
 - DSP C64x+
- RAM 256 MB (max) DDR2
- NOR Flash 512KB – 32MB
- NAND Flash 16MB – 4GB NAND
- 1x port sieciowy: 10/100-Mbit Ethernet, PoE
- 1x port USB 2.0 OTG
- Programowalny układ CPLD
- Wymiary: 95mm x 58mm x 20mm

Semihalf HASE-I



Semihalf HASE-I



Wymagania początkowe

- Maszyna z FreeBSD do budowania skróśnego (host)
- Egzemplarz urządzenia na który portujemy (target)
- Specyfikacje, instrukcje, dokumentacje
- Debugger JTAG (Opcjonalnie)

Kompilacja skrótna kernela

- Ustalamy katalog z plikami wynikowymi
 - `export MAKEOBJDIRPREFIX=/obj`
- Najpierw musimy zbudować toolchain:
 - `$ make kernel-toolchain
TARGET_ARCH=arm`
- Następnie możemy przystąpić do kompilacji samego kernela:
 - `$ make buildkernel
TARGET_ARCH=arm KERNCONF=DAVINCI`

Struktura portu

sys/

- arm/
 - arm/
 - at91/
 - conf/
 - DAVINCI
 - ...
 - davinci/
 - files.davinci
 - std.davinci
 - davinci_machdep.c
 - ...

files.davinci

arm/arm/bus_space_generic.c	standard	
arm/arm/irq_dispatch.S	standard	
arm/arm/cpufunc_asm_arm9.S	standard	
arm/arm/cpufunc_asm_armv5.S	standard	
arm/davinci/davinci_machdep.c	standard	
arm/davinci/davinci_devices.c	standard	
arm/davinci/davinci_space.c	standard	
arm/davinci/davinci_ainct.c	standard	
arm/davinci/davinci_timer.c	standard	
arm/davinci/davinci_obio.c	standard	
arm/davinci/davinci_psc.c	standard	
arm/davinci/davinci_pll.c	standard	
arm/davinci/uart_cpu_davinci.c	optional	uart
arm/davinci/uart_bus_davinci.c	optional	uart
dev/uart/uart_dev_ns8250.c	optional	uart
arm/davinci/if_dve.c	optional	dve
arm/davinci/davinci_mmc.c	optional	dvmmc

std.davinci

```
files          "../davinci/files.davinci"  
cpu            CPU_ARM9  
makeoptions   CONF_CFLAGS="-march=armv5te"  
options       PHYSADDR=0x80000000  
options       STARTUP_PAGETABLE_ADDR=0x80000000  
makeoptions   KERNPHYSADDR=0x80100000  
options       KERNPHYSADDR=0x80100000  
makeoptions   KERNVIRTADDR=0xc0100000  
options       KERNVIRTADDR=0xc0100000
```

sys/arm/conf/DAVINCI

```
ident          DAVINCI
include        "../davinci/std.davinci"
makeoptions    MODULES_OVERRIDE=""
makeoptions    WERROR="-Werror"

options        ALT_BREAK_TO_DEBUGGER
options        DDB
options        DIAGNOSTIC
# [...]

device        loop
device        md
device        pty
device        random
device        uart
```

machdep

- Odczytuje metadane przekazane z bootloadera lub używa domyślnych
- Odpowiada za inicjalizację MMU przed włączeniem pmapa
- Rezerwuje miejsce w przestrzeni adresowej dla urządzeń takich jak UART czy kontroler przerwań
- Jest podobny dla wszystkich portów ARM-owych, można „pożyczyć” go sobie z innego portu, np. z `sys/arm/at91/at91_machdep.c`

machdep

```
/* Static device mappings. */
static const struct pmap_devmap davinci_devmap[] = {
    {
        /*
         * Add the CFG bus peripherals.
         */
        DAVINCI_CFGBUS_BASE,
        DAVINCI_CFGBUS_PHYS_BASE,
        DAVINCI_CFGBUS_SIZE,
        VM_PROT_READ|VM_PROT_WRITE,
        PTE_NOCACHE,
    },
    {
        0, 0, 0, 0, 0,
    }
};
```

machdep

- `initarm(void *arg, void *arg2)`
 - Tutaj zaczyna się uruchamianie kernela
- `cpu_reset()`
 - Odpowiada za reset urządzenia
 - Pisząc port musimy ją zaimplementować
- `davinci_ramsize()`
 - Zwraca ilość pamięci RAM zainstalowanej w urządzeniu.

Wczesna konsola

■ UART

- Gdzie w pamięci znajdują się jego rejestry?
- Jakiego jest typu?
 - Najczęściej zgodny z 16550, który zaś jest pochodną 8250

■ Dwa ważne pliki

- `uart_cpu_davinci.c`
- `uart_bus_davinci.c`

uart_cpu_davinci.c

```
int
uart_cpu_eqres(struct uart_bas *b1, struct uart_bas *b2)
{
    return (b1->bsh == b2->bsh);
}

int
uart_cpu_getdev(int devtype, struct uart_devinfo *di)
{
    di->ops = uart_getops(&uart_ns8250_class);
    di->bas.chan = 0;
    di->bas.bst = obio_tag;
    di->bas.bsh = DAVINCI_UART0_BASE;
    di->bas.regshft = 2;
    di->bas.rclk = DAVINCI_UART_CLK;
    di->baudrate = 115200;
    di->databits = 8;
    di->stopbits = 1;
    di->parity = UART_PARITY_NONE;

    uart_bus_space_io = NULL;
    uart_bus_space_mem = obio_tag;
    return (0);
}
```

uart_bus_davinci.c

```
static device_method_t uart_davinci_methods[] = {
    DEVMETHOD(device_probe,      uart_davinci_probe),
    DEVMETHOD(device_attach,     uart_bus_attach),
    DEVMETHOD(device_detach,    uart_bus_detach),
    { 0, 0 }
};

static driver_t uart_davinci_driver = {
    uart_driver_name,
    uart_davinci_methods,
    sizeof(struct uart_softc),
};

static int
uart_davinci_probe(device_t dev)
{
    struct uart_softc *sc;

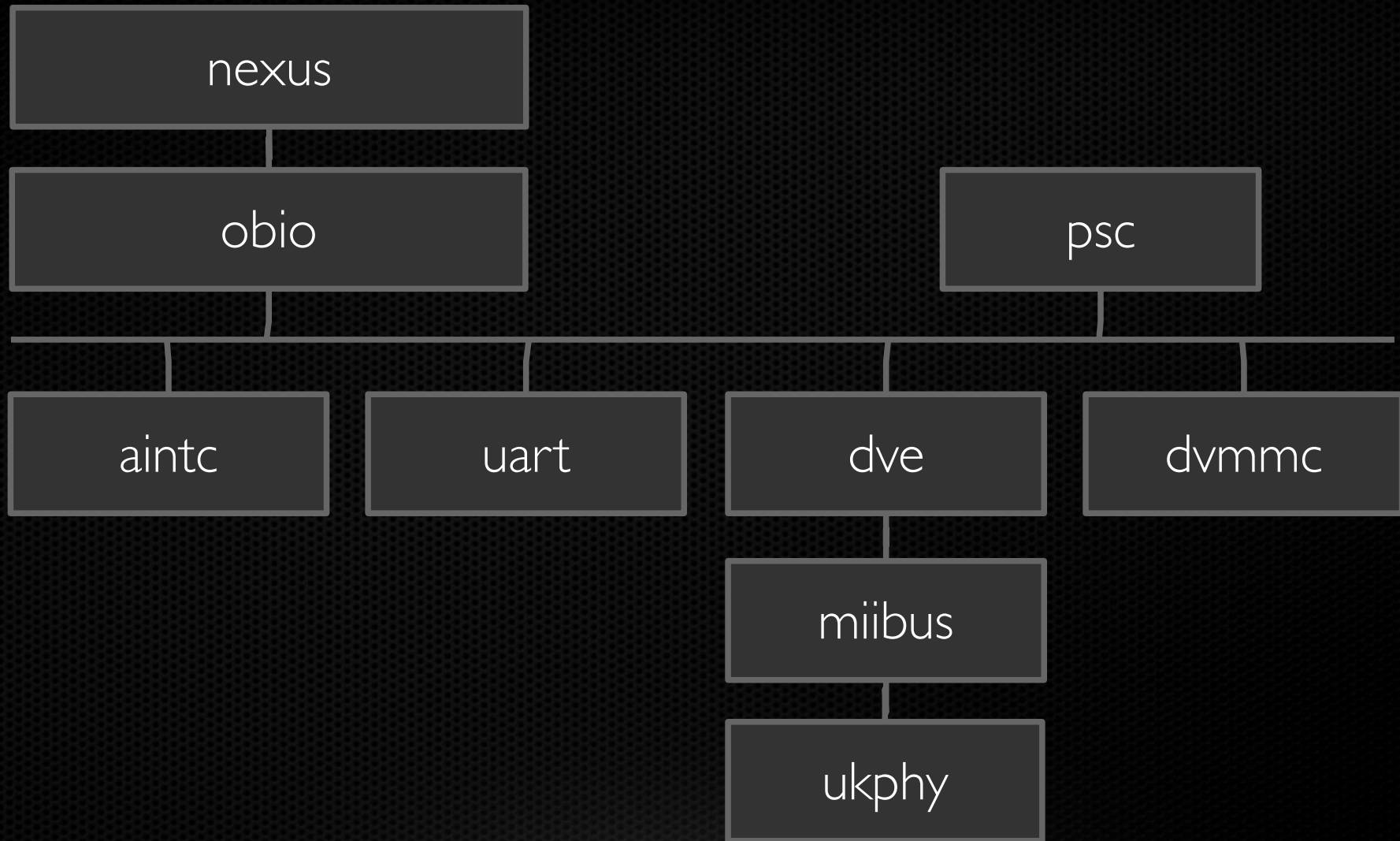
    sc = device_get_softc(dev);
    sc->sc_class = &uart_ns8250_class;

    return (uart_bus_probe(dev, 2, DAVINCI_UART_CLK, 0, 0));
}
```

OBIO

- **On-Board I/O**
 - Pseudo-szyna do której w systemie podpięte są wszystkie pozespoły naszego SoC-a
 - Zarządza przestrzenią adresową pamięci oraz zapewnia dostęp do niej za pomocą mechanizmu `bus_space(9)`
 - Przydziela zasoby (okna pamięci i przerwania) sterownikom

Hierarchia urządzeń w systemie



Alokacja zasobów

- **obio_alloc_resource(...)**
 - Przydzielanie zasobów urządzeniom leżącym na szynie OBIO
 - Musimy gdzieś przechowywać informacje o tym, jakie obszary pamięci i jakie przerwania wskazać danemu urządzeniu
 - Plik „hints”
 - Zapisana na stałe w kodzie tabelka z danymi o zasobach każdego z urządzeń

Alokacja zasobów

```
struct obio_device davinci_devices[] = {
    { "aintc", DAVINCI_AINTC_BASE, DAVINCI_AINTC_SIZE,
      { -1 },
      { -1 },
    },
    { "psc", DAVINCI_PSC_BASE, DAVINCI_PSC_SIZE,
      { DAVINCI_PSCINT, -1 },
      { -1 },
    },
    { "timer", DAVINCI_TIMERS_BASE, DAVINCI_TIMERS_SIZE,
      { DAVINCI_TINT0, DAVINCI_TINT1, -1 },
      { DAVINCI_PSC_TIMER0_IDX, -1 },
    },
    { "uart", DAVINCI_UART0_BASE, DAVINCI_UART_SIZE,
      { DAVINCI_UART0INT, -1 },
      { DAVINCI_PSC_UART0_IDX, -1 },
    },
    ...
}
```

bus_space(9)

- **Bus space tag**
 - Definiuje operacje na przestrzeni adresowej
- **Bus space handle**
 - Opisuje wycinek przestrzeni adresowej

bus_space(9)

```
struct bus_space root_tag = {  
    [...]  
    /* read (single) */  
    generic_bs_r_1,  
    generic_armv4_bs_r_2,  
    generic_bs_r_4,  
    NULL,  
    [...]  
    /* write (single) */  
    generic_bs_w_1,  
    generic_armv4_bs_w_2,  
    generic_bs_w_4,  
    NULL,  
    [...]
```

Kontroler przerwań

- Zależny od SoC-a, multipleksuje wiele przerwań w jeden sygnał IRQ lub FIQ w rdzeniu ARM
- W DaVinci jest to „Advanced Interrupt Controller” - AINTC

davinci_aintc.c

- Alokacja okna pamięci z rejestrami kontrolera przerw
- `arm_get_next_irq(int last)`
 - Zwraca numer następnego po *last* przerwania, które aktualnie wystąpiło
- `arm_mask_irq(int irq)`
 - Maskuje przerwanie *irq*
- `arm_unmask_irq(int irq)`
 - Przywraca przerwanie *irq*

Timer

- Urządzenie wyzwalające w zaprogramowanych odstępach czasu przerwanie
- `DELAY(int usec)`
 - Blokuje na ustaloną liczbę mikrosekund. Musi być dostępna bardzo wcześnie, nawet przed wykryciem timera.
- `hardclock()`
 - Wywoływany z przerwania zegarowego

MFS root

- Umożliwia zaszytie w pliku kernela obrazu filesystemu, np. UFS
- Przydatny, gdy nie mamy jeszcze ani dostępu do sieci ani do żadnego urządzenia pamięci masowej

MFS root

```
options      FFS
options      MD_ROOT
options      MD_ROOT_SIZE=8192 # kilobajty
makeoptions  MFS_IMAGE=/home/[...]/mdroot.bin
```

Port DaVinci

- **Aktualnie obsługiwane**
 - Multiuser mode
 - Rootfs zamontowany po NFS lub z karty SD
 - Kontroler przerwań, timery
 - Ethernet
 - Kontroler MMC/SD
 - NOR flash

Port DaVinci

- **Perspektywy na przyszłość**
 - Ujarzmienie potężnego silnika DMA – EDMA3
 - Jeden z projektów Google Summer of Code 2010
 - Komunikacja z rdzeniem DSP
 - Video back- i frontend
 - Wymaga kompletnego frameworku do obsługi video – odległa przyszłość
 - ATA, GPIO, NAND flash, I2C, itd
 - W najbliższej przyszłości

FDT

- Zastępuje wielokrotnie (i niepotrzebnie) powielany kod szyn OBIO
- Eliminuje konieczność zapisywania adresów urządzeń i przerwań w plikach źródłowych
 - Zamiast tego wszystkie informacje o urządzeniach w systemie i zależnościach między nimi są zapisane w jednym pliku w formacie .dts

Podziękowania

- The FreeBSD project
- Zespół programistów Semihalf

Odnośniki

- **Kod źródłowy portu**
 - Perforce: `//depot/user/jcee1/dm644x`
- **Specyfikacja TI DM6446**
 - <http://focus.ti.com/docs/prod/folders/print/tms320dm6446.html>

Dziękuję za uwagę!

Pytania, komentarze?

FreeBSD/arm na DaVinci DMSoC

Jakub Klama
jceel@semihalf.com

MeetBSD 2010, Kraków