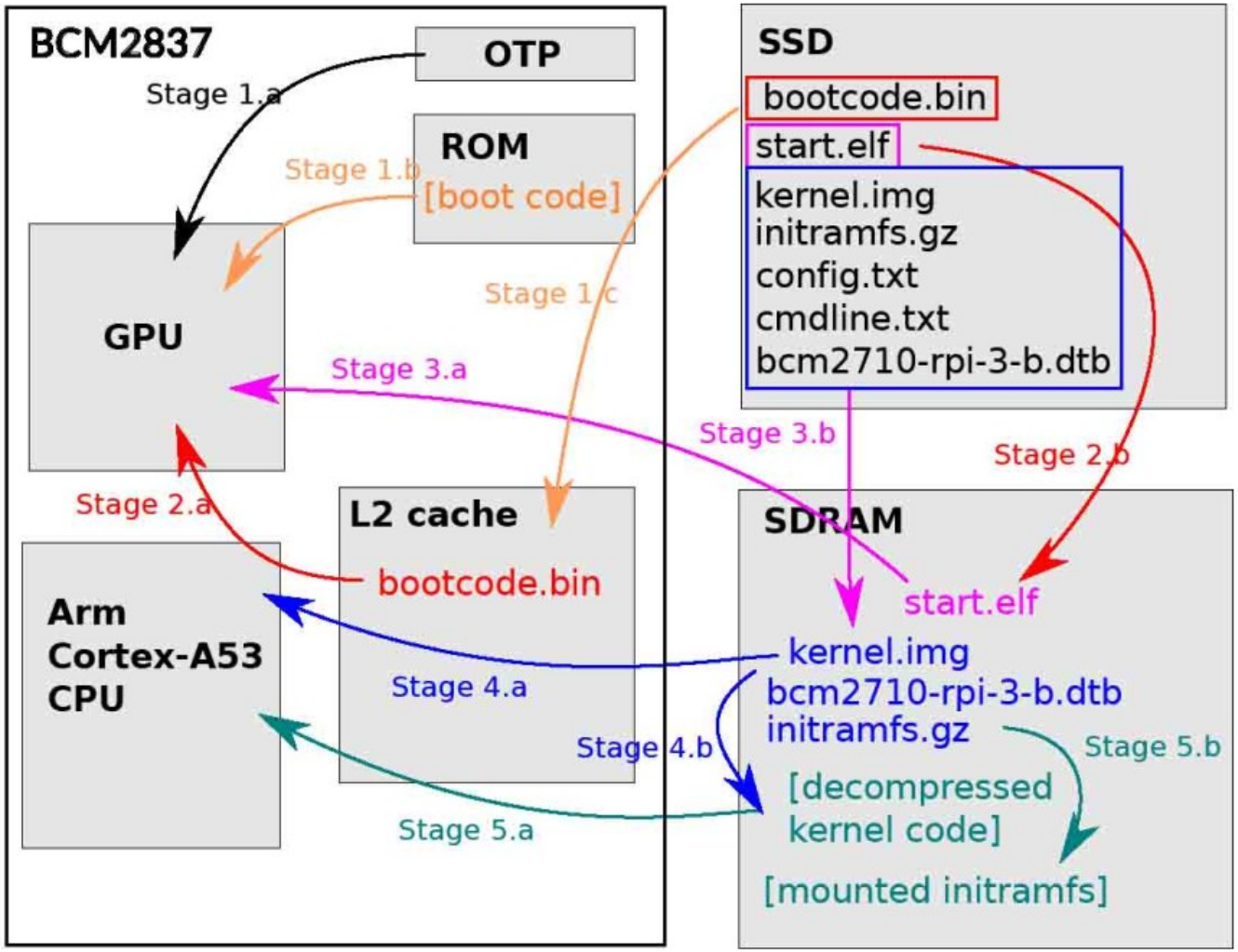
Лекция 4.1 Кросс сборка для модуля для Beagle bone

Разработал: Максимов А.Н.

Кросс компиляция

- Установка необходимого ПО
- Сборка ядра
- Сборка модуля

Процесс загрузки ARM



https://www.codeinsideout.com/blog/linux/bootup-sequence/#overview

Загрузчик на встраиваемых устройствах

- Процессор как правило имеет встроенный код для загрузки
- Этот код способен загрузить Stage 1 загрузчик в SRAM

из MMC, NAND, SPI flash, UART (transmitting data over the serial line), etc.

- Stage 1 загрузчик инициализирует DRAM и устройства и загружает Stage 2 загрузчик в память

Один из основных универсальных загрузчиков для встраиваемых систем это U-Boot

U-Boot

- Лицензия GPLv2, как и Linux
- В свободном доступе на http://www.denx.de/wiki/U-Boot
- Последняя версия исходного кода доступна в Git репозитории: http://git.denx.de/?p=u-boot.git;a=summary
- С 2008 года релизы выходят каждые 3 месяца, версии именуются YYYY.MM

Пример сборки u-boot на Raspberry PI3

Скачайте исходный код для uboot:
git clone git://git.denx.de/u-boot.git

Найдите конфигурационный файл для PI3:
ls configs | grep rpi
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- rpi_3_defconfig

Можно изменить настройки
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- menuconfig

Соберите u-boot
make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- -j 4

После сборки, скопируйте “u-boot” на “boot” раздел SD карты, и переименуйте его в “kernel8.img”.
Число должно быть больше “7” (предыдущие числа используются загрузчиками предыдущих стадий)

Создайте “boot script” для u-boot (rpi_bootscript.txt). Файл может включать:

```
mmc dev 0
fatload mmc 0:1 ${kernel_addr_r} Image
fatload mmc 0:1 ${fdt_addr_r} bcm2710-rpi-3-b.dtb
setenv bootargs "console=serial0,115200 console=tty1 root=/dev/sda1 rw rootfs type=ext4 rootwait devtmpfs.mount=0"
booti ${kernel_addr_r} - ${fdt_addr_r}
```

https://yuhei1-horibe.medium.com/building-and-booting-fully-customized-os-on-raspberry-pi-f743899c79d

U-Boot

Обычно устанавливается на флеш-память для запуска устройством. В зависимости от устройства, возможны различные варианты установки:

- CPU предоставляет монитор загрузки, с которым возможно взаимодействовать через UART или USB порт
- CPU загружает со съемного накопителя (MMC) перед загрузкой с несъемного (NAND)
- U-Boot уже предустановлен и его возможно использовать для установки новой версии
- JTAG интерфейс, для записи в флеш-память без запущенной какой-либо системы

Оболочка u-boot

U-Boot имеет оболочку, предоставляющую достаточно богатый набор команд для управления загрузкой системы. Для подробной справки, введите ***help*** в командной сроке U-Boot

```
U-Boot 2022.04 (May 29 2022 - 10:30:21)
OMAP36XX/37XX-GP ES1.2, CPU-OPP2, L3-165MHz, Max CPU Clock 1 Ghz
IGEPv2 + LPDDR/NAND
I2C:
ready
DRAM: 512 MiB
NAND: 512 MiB
MMC:
OMAP SD/MMC: 0
Die ID #255000029ff800000168580212029011
Net:
smc911x-0
U-Boot #
```

U-Boot в основном используется для загрузки и запуска ядра, но также он может использоваться для замены ядра и корневой ФС сохраненной во флеш-памяти.
Для передачи файлов между устройством и рабочей станции могут быть использованы:

- Сеть, U-Boot как правило имеет драйвер сетевого чипа
- USB, если U-Boot содержит драйвер USB контроллера вашей платформы
- SD или microSD карты, если U-Boot содержит драйвер SD контроллера вашей платформы
- Последовательный порт

Копирование результатов

Скопировать из “arch/arm/boot “**uImage**” на SD карте
Скопировать “*am335x-boneblack.dtb*” из “arch/arm/boot/dts/” в каталог 2024_06 на SD карте

/extlinux/extlinux.conf

```
label buildroot
    kernel /2024_06/uImage
    devicetree /2024_06/am335x-boneblack.dtb
    append console=ttyO0,115200 root=/dev/mmcblk0p2 rootwait
```

https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/DriverCreationGuide-FullKernelDownload.pdf

SD карта для загрузки beegle bone

sudo umount /dev/sdc1
sudo dd if=/dev/zero of=/dev/sdc bs=1M count=16
sudo cfdisk /dev/sdc

sudo mkfs.vfat -a -F 32 -n boot /dev/sdc1
sudo mkfs.ext4 -L rootfs -E nodiscard /dev/sdc2
sync

Создание разделов на SD

```
sudo fdisk /dev/sdc

Created a new DOS disklabel with disk identifier 0x3eb86cd2.
Command (m for help): p
Disk /dev/sdc: 7,5 GiB, 7990149120 bytes, 15605760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x3eb86cd2

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-15605759, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-15605759, default 15605759): +64M
Created a new partition 1 of type 'Linux' and of size 64 MiB.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2):
First sector (133120-15605759, default 133120):
Last sector, +sectors or +size{K,M,G,T,P} (133120-15605759, default 15605759):
Created a new partition 2 of type 'Linux' and of size 7,4 GiB.

Command (m for help): t
Partition number (1,2, default 2): 1
Hex code (type L to list all codes): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.

Command (m for help): a
Partition number (1,2, default 2): 1
The bootable flag on partition 1 is enabled now.
Command (m for help): w
```

SD карта для загрузки beegle bone

sudo fdisk /dev/sdc
[sudo] password for alex:

Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Command (m for help): p
Disk /dev/sdc: 7,3 GiB, 7817134080 bytes, 15267840 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x2ceb63e5

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdc1	*	2048	133119	131072	64M	**c**	**W95 FAT32 (LBA)**
/dev/sdc2		133120	15267839	15134720	7,2G	**83**	**Linux**

Command (m for help):

SD карта для загрузки beegle bone

sudo umount /dev/sdc1
sudo dd if=/dev/zero of=/dev/sdc bs=1M count=16
sudo cfdisk /dev/sdc

sudo mkfs.vfat -a -F 32 -n boot /dev/sdc1
sudo mkfs.ext4 -L rootfs -E nodiscard /dev/sdc2
sync

SD карта для загрузки beegle bone

cp MLO u-boot.img zImage am335x-boneblack.dtb /media/user/boot/
mkdir extlinux && touch extlinux/extlinux.conf
gedit ./extlinux/extlinux.conf

Развертывание файловой системы:
sudo tar -C /media/user/rootfs/ -xf rootfs.tar
sudo umount /dev/sdc1

SD карта для загрузки beegle bone

cp MLO u-boot.img zImage am335x-boneblack.dtb /media/**user**/boot/
cd /media/user/boot/
mkdir extlinux && touch extlinux/extlinux.conf
gedit .**/extlinux/extlinux.conf**

Развертывание файловой системы:
sudo tar -C /media/user/rootfs/ -xf rootfs.tar
sudo umount /dev/sdc1

Кросс компиляция - установка

- sudo apt-get install gcc-arm-linux-gnueabi
- sudo apt-get install flex yacc bison u-boot-tools

Кросс компиляция

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabi-
export CC=/usr/bin/arm-linux-gnueabi-gcc

make omap2plus_defconfig
make LOADADDR=0x80008000 uImage dtbs -j4
make modules -j4
make
```

Кросс компиляция

```
obj-m += hello.o
KDIR := /work/2025_02/linux-6.6.68
all:
    make -C $(KDIR) M=$(PWD) modules
clean:
    make -C $(KDIR) M=$(PWD) clean
```

Задача – соберем и загрузим модуль на BB

Установка версии

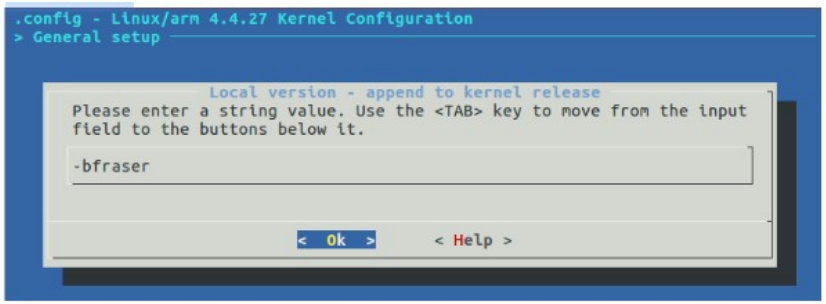
When the blue kernel configuration menu appears, change the Local Version to be your user ID:

- Select General Setup --> , and press Enter.
- Select Local version - append to kernel release and press Enter.
- Type in a dash and your SFU ID (your login) and press Enter, such as, and shown below in

the screen shot:

-bfraser

Screen shot



https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/DriverCreationGuide-FullKernelDownload.pdf

Конфигурирование

Установка компилятора:

apt-get install gcc-arm-linux-gnueabi
sudo apt-get install liblz4-tool
sudo apt-get install u-boot-tools

Компиляция:
make ARCH=arm bb.org_defconfig

export CC=/usr/bin/arm-linux-gnueabi

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage dtbs
LOADADDR=0x80008000 -j4

Копирование результатов – установка на microsdSD

Скопировать “uImage” из “arch/arm/boot на раздел BOOT.
Скопировать “am335x-boneblack.dtb” из “arch/arm/boot/dts/” на раздел BOOT.

/extlinux/extlinux.conf

```
label buildroot
        kernel /2024_06/uImage
        devicetree /2024_06/am335x-boneblack.dtb
        append console=ttyO0,115200 root=/dev/mmcblk0p2 rootwait
```

https://opencoursehub.cs.sfu.ca/bfraser/grav-cms/cmpt433/guides/files/DriverCreationGuide-FullKernelDownload.pdf

Device tree

"Open Firmware Device Tree", или просто Devicetree (DT), представляет собой структуру данных и язык для описания аппаратного обеспечения, которое не может быть определено в режиме Plug&play (например, информацию о pci устройствах можно определить) .

Это описание аппаратного обеспечения, которое доступно для чтения операционной системой (и загрузчиком), так что операционной системе не нужно жестко кодировать детали конфигурации компьютера.

https://www.devicetree.org/specifications/

Device tree

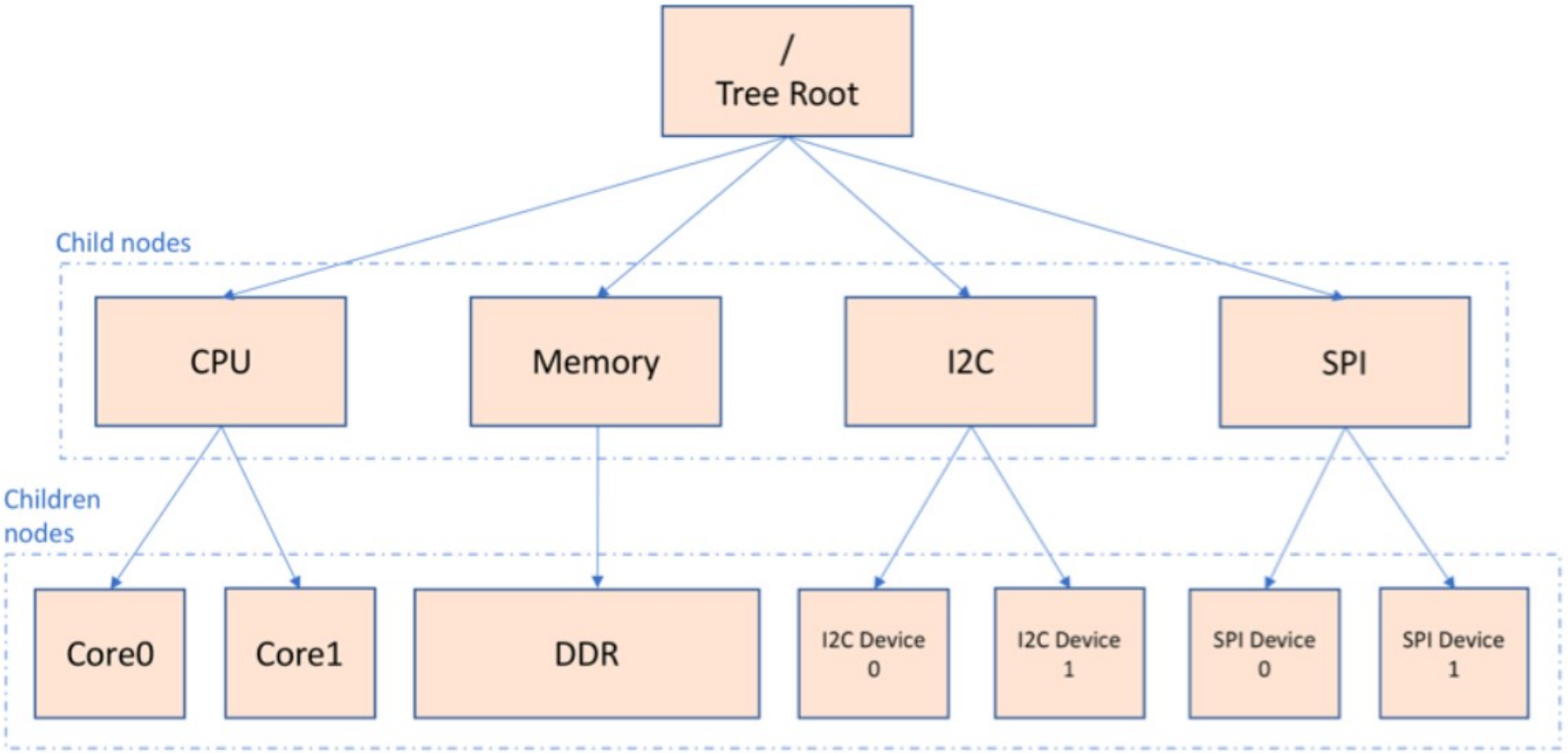
- Device tree – это коллекция узлов устройств (device node)
- Узлы device tree представляют устройства
- Между узлами могут быть отношения предок-потомок
- В device tree должен быть один корневой узел *"/"*
- Узел может содержать пары property/value, содержащие информацию об узле

Device tree

Типичное дерево устройств может содержать следующую информацию:

- Число CPU
- Размер и расположение RAMs
- Информация о шинах и бриджах
- Информация о подключениях устройств
- Контроллеры прерываний и информация о подключениях IRQ линий
- Специфичная информация о конфигурации устройств:
  - Ethernet MAC адрес
  - Входной input clock
  - и др.

Device tree


/
Tree Root
Child nodes
CPU
Memory
I2C
SPI
Children nodes
Core0
Core1
DDR
I2C Device 0
I2C Device 1
SPI Device 0
SPI Device 1

Device tree

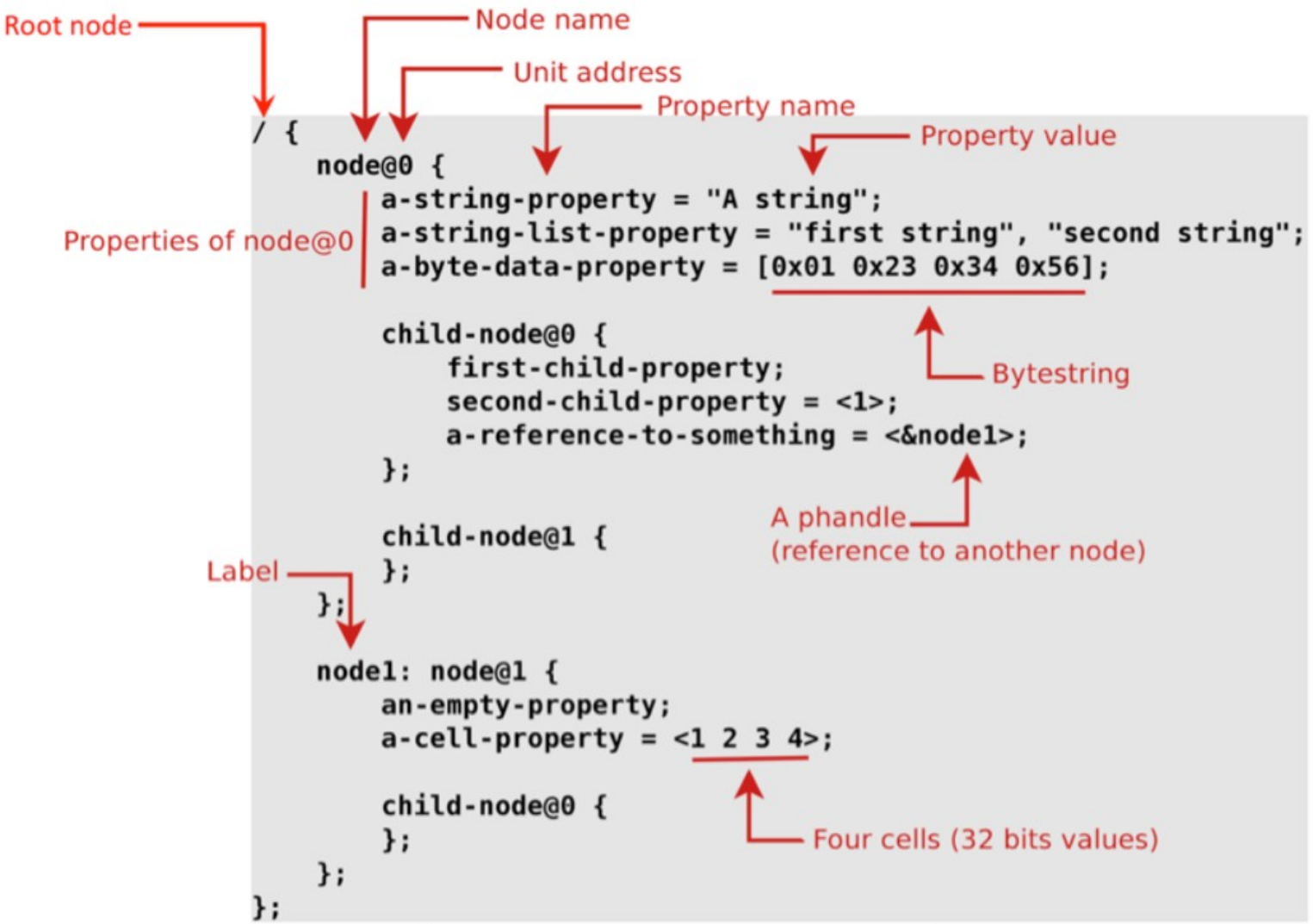


Figure 2 Simple Device Tree Template (© Thomas Petazzoni, Device Tree for Dummies)

Device tree

Дерево устройств обычно описывается в текстовом виде, известном как DTS Пример device tree:

```
/dts-v1/;
/include/ "common.dtsi";
/ {
    node1 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];
        cousin: child-node1 {
            first-child-property;
            second-child-property = <1>;
            a-string-property = "Hello, world";
        };
        child-node2 {
        };
    };
    node2 {
        an-empty-property;
        a-cell-property = <1 2 3 4>; /* каждая ячейка — uint32 */
        child-node1 {
            my-cousin = <&cousin>;
        };
    };
};
/node2 {
    another-property-for-node2;
};
```

https://bootlin.com/blog/enabling-new-hardware-on-raspberry-pi-with-device-tree-overlays/

Инструменты для работы с dts

Plugin для редактирования dts в VSCode
DeviceTree Language Support for Visual Studio Code

Для работы с описанием дерева устройств предназначен компилятор dtc

*.dtd - файлы Device Tree Binary - иерархическое описание структуры периферии в бинарном формате. dtb файлы используются uboot и ядром Linux для получения информации об аппаратное обеспечении SOC

*.dts - Device Tree Source - представление дерева устройств в человекочитаемом формате. Для генерации *.dtb файла из *.dts используется утилита dtc.

Для установки dtc в Ubuntu
sudo apt-get install device-tree-compiler

RISoft

Как получить device tree для машины

В исходниках Linux arch/$ARCH/boot/dts
В исходниках U-boot arch/$ARCH/dts

Linux отображает текущую конфигурация device tree через sysfs в подкаталоге /sys/firmware/devicetree/base/

Возможно посмотреть изменения в device tree после изменения конфигурации:

dtc --sort -I fs -O dts /sys/firmware/devicetree/base > device_tree.out

или

tar zcf device_tree.tar.gz /sys/firmware/devicetree/base/

tar xvf device-tree.tar.gz

dtc --sort -I fs -O dts sys/firmware/devicetree/base/ > device_tree.out

Получить dts из dtb:

$ dtc --sort -I dtb -O dts your_dtb_file.dtb > device_tree.out

https://community.silabs.com/s/article/kba-linux-device-tree-debugging?language=en_US

Как получить device tree для машины

В исходниках Linux arch/$ARCH/boot/dts
В исходниках U-boot arch/$ARCH/dts

Raspbery PI 4 - linux/arch/arm/boot/dts/bcm2711-rpi-4-b.dts
(базируется на файле для SoC
linux/arch/arm/boot/dts/bcm2711.dtsi)

https://community.silabs.com/s/article/kba-linux-device-tree-debugging?language=en_US

Оверлеи dtbo

Механизм оверлеев позволяет во время загрузки накладывать на *.dtb файлы патчи.

dtbo патч, включающий какую-то аппаратную возможность платы, называется оверлеем. Формат файла - такой же, как у .dtb - файл можно прочитать при помощи утилиты dtc.

Обычно файлы находятся в папке /boot/dtb/overlay
В файле /boot/config.txt можно указать, какие оверлеи загужать при старте

Сборка оверлея:
dtc -@ -I dts -O dtb -o my_overlay.dtbo my_overlay-overlay.dts

https://devdotnet.org/post/rabota-s-gpio-na-primere-banana-pi-bpi-m64-chast-2-device-tree-overlays/
https://blog.stabel.family/raspberry-pi-4-device-tree/

Динамическая загрузка оверлеев

Утилиту **dtoverlay** можно использовать для динамической загрузки оверлеев device trees.

dtoverlay -a – получилуть список доступных оверлеев
dtoverlay -l – получить список загруженных оверлеев
dtoverlay -h <overlay_name> - получить информацию об оверлее и их параметрах
dtoverlay <overlay_name> <param_name>=<param_value> для добавления оверлея с параметрами
dtoverlay -r <overlay_name> для удаления динамически загруженного оверлея

https://blog.stabel.family/raspberry-pi-4-device-tree/

Отладка device tree

Проверка дерева устройств во время выполнения

Для проверки, какой файл дерева устройств загружается при загрузке.
$ sudo vcdbg log msg

Получить дополнительную информация на следующей загрузке можно путем добавлени в /boot/config.txt

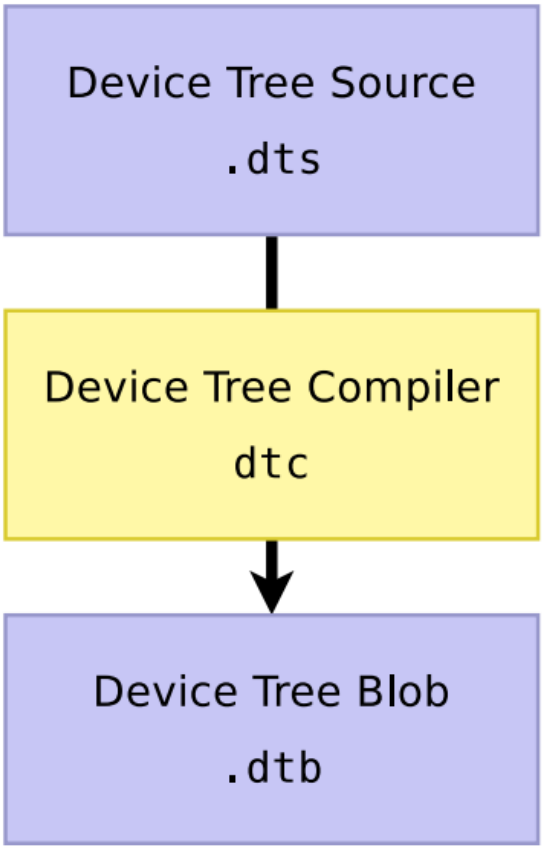
dtdebug=1

Device tree и платформенные драйвер

```
/ {
  #address-cells = <1>;
  #size-cells = <1>;
  model = "TI AM335x BeagleBone Black";
  compatible = "ti,am335x-bone-black", "ti,am335x-bone", "ti,am33xx";

  cpus { ... };
  memory@80000000 { ... };
  chosen { ... };
  ocp {
    intc: interrupt-controller@48200000 { ... };
    usb0: usb@47401300 { ... };
    l4_per: interconnect@44c00000 {
      i2c0: i2c@40012000 { ... };
    };
  };
};
```

```
$ dtc -I dts -O dtb -o foo.dtb foo.dts
$ ls -l foo.dt*
-rw-r--r-- 1 thomas thomas 169 ... foo.dtb
-rw-r--r-- 1 thomas thomas 102 ... foo.dts
```



Компиляция Device tree

Компилияция device tree (нужен dtc)

sudo apt-get install device-tree-compiler
dtc -I dts -O dtb -o 1st-overlay.dtb 1st-overlay.dts

https://bootlin.com/blog/enabling-new-hardware-on-raspberry-pi-with-device-tree-overlays/

Назначение шины platform

Шина platform предназначена для выполненеия следующих задач:

- Интеграции устройств на встраиваемых системах в модель устройств не оснащенными

шинами с поддеркой технологий Plug&Play, Hot plug и идентификацию устройств

- Шина platform является псевдо шиной и предоставляет интерфейс platform driver / platform device во встраиваемых систьемах

- Устройства представляемые в виде platform device обычно напрямую подключены к центральному процессору

Модификация am335x-bonefoo.dts

```
/dts-v1/;

#include "am33xx.dtsi"
#include "am335x-bone-common.dtsi"
#include "am335x-boneblack-common.dtsi"
#include "am335x-boneblack-hdmi.dtsi"

/ {
    model = "TI AM335x BeagleBone Black";
    compatible = "ti,am335x-bone-black", "ti,am335x-bone", "ti,am33xx";

    my_device {
        compatible = "brightlight,mydev";
        status = "okay";
        label = "Test";
        my_value = <12>;
    };
};
```

Модификация dts

Модифицировать arch/arm/dts/ti/omap/Makefile
(For 6.1.80 – arch/arm/boot/dts/Makefile)

Add line:

am335x-bonefoo.dtb \

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- **dtbs**

Пример

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mod_devicetable.h>
#include <linux/property.h>
#include <linux/platform_device.h>
#include <linux/of_device.h>

static int dt_probe(struct platform_device *pdev);
static int dt_remove(struct platform_device *pdev);

static struct of_device_id my_driver_ids[] = {
        {
                .compatible = "brightlight,mydev",
        }, {  }
};
MODULE_DEVICE_TABLE(of, my_driver_ids);

static struct platform_driver my_driver = {
        .probe = dt_probe,
        .remove = dt_remove,
        .driver = {
                .name = "my_device_driver",
                .of_match_table = my_driver_ids,
        },
};
```

Перед сборкой модуля для ядра должны быть собраны модули

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules

Пример overlay

```
static int dt_probe(struct platform_device *pdev) {
	struct device *dev = &pdev->dev;
	const char *label;
	int my_value, ret;
	printk("dt_probe - Now I am in the probe function!\n");
	if(!device_property_present(dev, "label")) {
		printk("dt_probe - Error! Device property 'label' not found!\n");
		return -1;
	}
	if(!device_property_present(dev, "my_value")) {
		printk("dt_probe - Error! Device property 'my_value' not found!\n");
		return -1;
	}
	ret = device_property_read_string(dev, "label", &label);
	if(ret) {
		printk("dt_probe - Error! Could not read 'label'\n");
		return -1;
	}
	printk("dt_probe - label: %s\n", label);
	ret = device_property_read_u32(dev, "my_value", &my_value);
	if(ret) {
		printk("dt_probe - Error! Could not read 'my_value'\n");
		return -1;
	}
	printk("dt_probe - my_value: %d\n", my_value);
	return 0;
}
```

Пример overlay

```
static int dt_remove(struct platform_device *pdev) {
        printk("dt_probe - Now I am in the remove function\n");
        return 0;
}
/**
 * @brief This function is called, when the module is loaded into the kernel
 */
static int __init my_init(void) {
        printk("dt_probe - Loading the driver...\n");
        if(platform_driver_register(&my_driver)) {
                printk("dt_probe - Error! Could not load driver\n");
                return -1;
        }
        return 0;
}

/**
 * @brief This function is called, when the module is removed from the kernel
 */
static void __exit my_exit(void) {
        printk("dt_probe - Unload driver");
        platform_driver_unregister(&my_driver);
}
module_init(my_init);
module_exit(my_exit);
```

Работа с оборудованием

```
#include "am335x-boneblack.dts"
/* Метка, am335xx_pinmux, ссылается на существующий pin muxing определенный в am33xx.dtsi */

&am33xx_pinmux {
    uart2_pins: uart2_pins { //This section adds muxing for UART2
/*  The AM33XX_IOPAD macro comes from omap.h. It provides the absolute physical address of the given
address, as opposed to an offset from the base of the register. The first argument is the address obtained
from the processor technical reference manual.    */
        pinctrl-single,pins = <
            AM33XX_IOPAD(0x954, PIN_OUTPUT_PULLDOWN | MUX_MODE1)
            AM33XX_IOPAD(0x950, PIN_INPUT_PULLUP | MUX_MODE1)
            >;
    };
};
/*  The label here references the device defined in the am33xx.dtsi include file. The fields populated here
will override those in the original defintion.*/

&uart2 {
    compatible = "serial"; //This is used when identifying a compatibel device driver
    status = "okay"; ///Setting the status to "okay" enables the device
    //This is the name that corresponds to the pin control mode on the line below it
    pinctrl-names = "default";
    //This last entry binds the UART2 device to the pin control node defined above
    pinctrl-0 = <&uart2_pins>;
};
```

Работа с оборудованием

```
#include <linux/fs.h>
#include <linux/of.h>
#include <linux/io.h>
#include <linux/pm_runtime.h>
#include <linux/platform_device.h>
#include <linux/init.h>
#include <linux/uaccess.h>

static struct of_device_id hw_match_table[] = {
        {
            .compatible = "serial",
        },
};
static struct platform_driver hw_plat_driver = {
    .driver = {
        .name = "serial",
        .owner = THIS_MODULE,
        .of_match_table = hw_match_table},
    .probe = hw_probe,
    .remove = hw_remove
};
module_platform_driver(hw_plat_driver);
```

Работа с оборудованием

```
static int hw_probe(struct platform_device *pdev)
{
    struct resource *res;
    res = platform_get_resource(pdev, IORESOURCE_MEM,
0);


…
    host->base = ioremap(res->start, resource_size(res));
или
   dev->regs = devm_ioremap_resource(&pdev->dev, res);

   dev->irq = platform_get_irq(pdev, 0);
   …
}
```

https://docs.kernel.org/driver-api/device-io.html

Работа с оборудованием

```
static int hw_remove(struct platform_device *pdev) {

    …
    return 0;
}
```