Лекция 3.3 Кросс сборка для модуля для Beagle bone

Разработал: Максимов А.Н.

Кросс компиляция

- Установка необходимого ПО
- Сборка ядра
- Сборка модуля

Кросс компиляция - установка

- sudo apt-get install gcc-arm-linux-gnueab
- 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

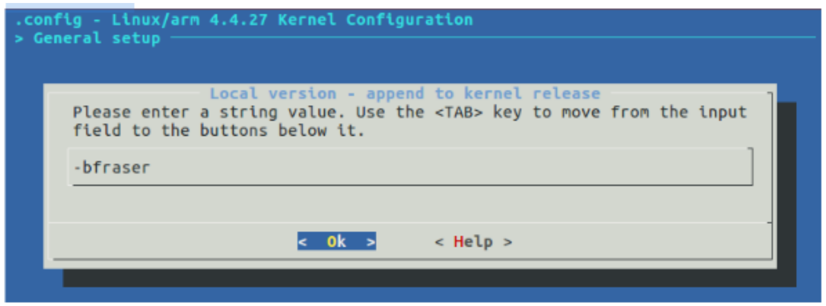
- make omap2plus_defconfig
- make LOADADDR=0x80008000 uImage dtbs -j4
- make modules -j4
- make

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
```

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



```
.config - Linux/arm 4.4.27 Kernel Configuration
> General setup
                Local version - append to kernel release
  Please enter a string value. Use the <TAB> key to move from the input
  field to the buttons below it.
  -bfraser
                    <  Ok  >         < Help >
```

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

Configure

Install compiler:

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

Compile

```
make ARCH=arm bb.org_defconfig

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

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

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

Копирование результатов

Copy “uImage” file from “arch/arm/boot directory to the BOOT partition.
Copy “am335x-boneblack.dtb” file from “arch/arm/boot/dts/” directory to the BOOT partition directory 2024_06.

/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

Модификация am325x-bonefoo.dts

```
/dts-v1/;
/plugin/;
/ {
        compatible = "ti,am335x-bone-black";
        fragment@0 {
                target-path = "/";
                __overlay__ {
                        my_device {
                                compatible = "brightlight,mydev";
                                status = "okay";
                                label = "Test";
                                my_value = <12>;
                        };
                };
        };
};
```

Модификация dts

Модифицировать arch/arm/dts/ti/omap/Makefile

```
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",
	}, { /* sentinel */ }
};
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);
```