1. Host and target setup 1h26m

1. About the instructor
2. Source code and course materials youtube
3. Host and target setup 1

4. Tool-chain download 2

https://releases.linaro.org/components/toolchain/binaries/7.5-2019.12/

5. Important Note
6. Installing gparted application
7. Tool-chain installation and PATH settings 3
8. Note for the students
9. Target preparation : Serial debug setup
10. Important documents
11. Understanding booting sequence of beaglebone black hardware 4

12. Preparing SD card for SD boot
13. Copying boot images to SD card 5
14. Booting BBB via SD card
15. Making SD boot default on BBB by erasing eMMC MBR

16. Updating Linux kernel image 7

pre-built image of the kernel : the kernel version is 4.4.62.

/home/scott/workspace/src/linux_bbb_4.14 3/26 4.14

```
  174  git init
  175  git clone https://github.com/beagleboard/linux.git linux_bbb_4.14
  178  cd linux_bbb_4.14/
  186  git checkout 4.14.108-ti-r130

  443  cd workspace/
  445  cd src
  447  cd linux_bbb_4.14/
```

kernel_image_update_5.10.pdf 5.10-rt

17. Linux kernel compilation

/home/scott/workspace/src/linux_bbb_4.14

18. Modules compilation

```
$ uname -r
  5.3.0-40.generic

$ make -C /lib/modules
  4.14.108/     5.3.0-28-generic/       5.3.0-40-generic/                       -> lecture


  scott@host:~/workspace/ldd/custom_drivers/001hello_world$ ll /lib/modules/
  4.14.108/     5.4.0-150-generic/      5.4.0-84-generic/                       -> 18.04 LTS
```

Ubuntu release	Arch	Kernel Version	
Ubuntu 20.04 LTS	64-bit x86	5.4 (GA)	
Ubuntu 18.04 LTS	64-bit x86	5.4 (HWE)	Hardware Enablement : the most recent versions of the Linux kernel.
Ubuntu 18.04 LTS	64-bit x86	4.15 (GA)	General Availabity : the most stable kernel with an original LTS
Ubuntu 16.04 LTS	64-bit x86	4.15 (HWE)	

```
$ make -C /lib/modules/5.3.0-40-generic/build/ M=$PWD modules
$ ls
  main.c main.ko main.mod mdin.mod.c main.mod.o main.o Makefile modules.order Module.symvers
$ sudo insmod main.ko
$ dmesg
  [164.xxxx] main: loading out-of-tree module tains kernel.
  [164.xxxx] main: module verfification failed: signature and /or required key missing - tainting kernel
  [164.xxxx] Hello world
$ sudo rmmod main.ko
$ dmesg
  [288.xxxx] Good bye world
```

30. Testing of an LKM on target

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -C /home/scott/workspace/src/linux_4.14/ M=$PWD modules     => 5.3.40 in
lecture
$ file main.ko
  main.ko: ELF 32-bit LSB relocatable, ARM, EABIS version 1 (SYSV), ....
$ modinfo main.ko
  filename:     /path/to/main.ko
  board:            Beaglebone black REV A5
  description:  a simple hello world kernel module
  author:           ME
  license:      GPL
  depends:
  name:         main
  vermagic:     4.14.108 SMP preemp mod_unload modversions ARMv7 p2v8
$
$ arm-linux-gnueabihf-objdump -h main.ko
  main.ko:      file format elf32-littlearm
```

19. Modules install
20. Update new boot images and modules in SD card
21. Enabling internet over USB 8

2. Linux Kernel module 1h38m

22. Introduction to Linux kernel module 43~46

23. User space Vs kernel space 89~90

24. LKM writing syntax 47~58

include/linux has all the kernel header files. e.g. linux/modules.h
stdio.h is a user header file.

25. __init and __exit macros 59~62 9

26. LKM entry point registration and other macros 63~66 10

linux_bbb_4.14

27. Hello World LKM 67~70 11 001

28. Building a Linux kernel module 71~75 12

LKM
- static
- dynamic
  - intree
  - out of tree : kbuild to build modules w/ "a prebuilt kernel source w/ config & headers"
    https://www.kernel.org/doc/Documentation/kbuild/modules.txt

Two ways to obtain a prebuilt kernel version:
- Download kernel from your distributor and build it by yourself
- Install the Linux-headers- of the target Linux kernel

```
make -C $KDIR -M $PWD [modules/modules_install/clean/help]

obj-<X> := <module_name>.o          -> <module_name>.ko

  X = n: do not compile
  X = y: compile and link it with kernel
  X = m: compile as dynamically loadable kernel module
```

29. Compilation and testing of an LKM

```
  section:
  idx name  size        VMA     LMA     file off algn
  0 .note.
  1 .text
  2 .init.text
  3 .exit.text
  4 .ARM.extab.init.text
  5 .ARM.exidx.init.text
  6 .ARM.extab.exit.text
  7 .ARM.exidx.exit.text
  8 .modinfo
  9 .rodata.str1.4
  10 __version
  11 .data
  12 .gnu.linkonce.this_module
  13 .plt
  14 .init.plt
  15 .bss
  16 .comment
  17 .note.GNU-stack
  18 .ARM.attributes

# sudo insmod main.ko
  [ 200.35xxx] main: loading out-of-tree module taints kernel
  [ 200.35xxx] Hello world
# dmesg | tail
  [ 200.35xxx] main: loading out-of-tree module taints kernel
  [ 200.35xxx] Hello world
# sudo rmmod main.ko
  [ 288.35xxx] Good bye world


/usr # dmesg    <- my BBB

[   10.627696] g_ether gadget: Ethernet Gadget, version: Memorial Day 2008
[   10.639452] g_ether gadget: g_ether ready
[   10.923157] g_ether gadget: high-speed config #2: RNDIS
[  121.502878] random: crng init done
[  215.820416] main: loading out-of-tree module taints kernel.
[  215.826708] Hello world
[  226.387175] Good bye world
/usr #
```

31. Makefile 13

KDIR = /lib/modules/$(shelluname -r)/build/ -> 5.3.0-40-generic

https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt)

You have to add the Linux kernel module inside the Linux kernel source tree and let the Linux build system builds that.
If you want to list your kernel module selection in kernel menuconfig, then create and use a Kconfig file

. Kconfig file

```
scott@vbox:~/.../src/linux_bbb_4.14/drivers/char/my_c_dev $ ls
 Kconfig    main.c                                          <= newly added

    menu “my custom modules”
        config CUSTOM_HELLOWORLD
            tristate “hello world module support”
            default m
    endmenu


scott@vbox:~/.../src/linux_bbb_4.14/drivers/char $ vi Kconfig              <= upper level Kconfig

    ...
  source "drivers/char/my_c_dev/Kconfig"

  endmenu



scott@vbox:~/.../src/linux_bbb_4.14/drivers/char/my_c_dev $ vi Makefile

    #obj-<config_item> += <module>.o                                       <= newly added

    obj-$(CONFIG_CUSTOM_HELLOWORLD) += main.o                              // $(CONFIG_CUSTOM_HELLOWORLD) if we don't know
    it’s y or n or m.



scott@vbox:~/.../src/linux_bbb_4.14/drivers/char $ vi Makefile             <= upper level Makefile

    ...
    obj-y                          += my_c_dev/

    // To select a kernel module under that menu, we have config item. But, select our folder we don’t have any config item.
```



```
write(fd, 0xab);    // echo 0xAB > /dev/rtc
```

Udev : populates /dev w/ device files

devices files are under VFS and identified by major/minor numbers

Let’s create a device number.
Your driver has to ask the kernel to dynamically allocate the device number or numbers.
Basically, what you should be doing here is, you should be using kernel APIs and kernel utilities(shown in Figure 6)
in order to a request various services from the kernel.

Now, to create a device number, you just have to use a kernel API alloc_chrdev_region().
So, you have to use alloc_chrdev_region(). This creates a device number. And for the registration, you can use these APIs
cdev_init() and cdev_add().

And after that, the driver should create a device files. For that, you can use these kernel APIs class_create() and
device_create().
The creation things we are going to do in module initialization function.

Whenever you load a module, so these creation a services must be executed and your driver must be ready to accept a system calls
from the user space program.

That’s why it makes sense to do this creation process in the module initialization function.

When you remove the module, it’s better you delete all those a resources what you requested from the kernel.
Otherwise, it will simply consume a resources of the kernel. That’s why, let’s say, if you use unregister_chrdev_region(),
it will delete the device number which is allocated for your module, that it can be reused for some other module.

After that, you can use cdev_del() to delete the registration, that will free some memory. class_destroy and device_destroy will
delete your device files.
That’s why, all these deletion things we are going to do in module clean-up function. Because these deletion things should be
executed whenever you remove the module.

```
scott@vbox:~/.../src/linux_bbb_4.14 $ make ARCH=arm menuconfig

    Device Drivers --->
      Character devices --->
        fastbit custom modules --->
    <m>    helloworld module support    : dynamically linked. '*' means statically linked.


<help>

There is no help available for this option.
Symbol: CUSTOM_HELLOWORLD [=m]
Type  : tristate
Prompt: helloworld module support
    Location:
    -> Device Drivers
        -> Character devices
        -> fastbit custom modules
    Defined at drivers/char/my_c_dev/Kconfig:2



scott@vbox:~/.../src/linux_bbb_4.14 $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules -j4

  scripts/kconfig/conf  --silentoldconfig Kconfig
    ...
    CC [M]  drivers/char/my_c_dev/main.o
    Building modules, stage 2.
    ..
    LD [M]  drivers/char/my_c_dev/main.ko



scott@vbox:~/.../src/linux_bbb_4.14/drivers/char/my_c_dev $ modinfo main.ko
    filename:       /home/scott/workspace/src/linux_bbb_4.14/drivers/char/my_c_dev/main.ko
    board:          Beaglebone black REV A5
    description:    A simple hello world kernel module
    author:         Kiran Nayak
    license:        GPL
    depends:
    intree:         Y                                               <= !!!!
    name:           main
    vermagic:       4.14.108+ SMP preempt mod_unload modversions ARMv7 p2v8
```



```
scott@host:~/workspace/ldd/custom_drivers/custom_drivers/001hello_world$ nm main.ko
0000000000000000 T cleanup_module
                 U __fentry__
0000000000000000 t helloworld_cleanup
0000000000000000 t helloworld_init
0000000000000000 T init_module
0000000000000000 r _note_6
                 U printk
0000000000000000 D __this_module                        <= !!!
000000000000004d r __UNIQUE_ID_author37
0000000000000000 r __UNIQUE_ID_board39
...
00000000000000ae r __UNIQUE_ID_vermagic36
0000000000000000 r ____versions


>> main.mod.c

#include <linux/build-salt.h>
#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

BUILD_SALT;

MODULE_INFO(vermagic, VERMAGIC_STRING);
MODULE_INFO(name, KBUILD_MODNAME);

__visible struct module __this_module           <= !!!
__section(.gnu.linkonce.this_module) = {
        .name = KBUILD_MODNAME,


struct file_operations {
    struct module *owner;                       <- __this_module : to prevent a module from being unloaded while the
    structure in use
    loff_t (*llseek) (struct file *, loff_t, int);

}
```

cdev_init()   -> just initialization

39. Character device registration contd.   121

cdev_add()   -> real registration

40. Character driver file operation methods   18

Character Driver System Calls

What is a file object?
  Whenever a file is opened, a file object is created in the kernel space.
  The file object stores information about the interaction between an open file and a user process.

def_chr_fops

```
open                                                                                    user
-----------------------------------------------------------------------------------------------
  -> do_sys_open                    --> return 'fd' to user space (-1 means failure)   kernel
       -> do_filp_open              --> 'file' object allocatioin
            -> do_dentry_open       --> using default (dummy) fops, e.g. chrdev_open for open
                 -> chrdev_open     --> replacing default fops
                      -> pcd_open
```

do_filp_open : a file object is created

ref) understanding linux kernel 3ed, p524

When device file gets created
  1) create device file using udev
  2) inode object gets created in memory and inode's i_rdev field is initialized with device number(dev_t)
  3) inode object's i_fop field is set to dummy default file operations (def_chr_fops)

When user process executes open system call
  1) user invokes open system call on the device file
  2) file object gets created   (do_filp_open)
  3) inode's i_fop gets copied to file object's f_op (dummy default file operations of char device file)
  4) open function of dummy default file operations gets called (chrdev_open)
  5) inode object's i_cdev field is initialized with cdev which you added during cdev_add (lookup happens using inode-> i_rdev field)

```
            return ret;
        }
```

41. Character driver file operation methods contd.

Open Method   Close system call
Release Method:
Write Method:
Llseek Method:

42. Implementing file operation methods

Accessing File Operations Structure

include/linux/fs.h

```
   struct file_operations {
       struct module *owner;
       loff_t (*llseek) (struct file *, loff_t, int);
       ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
       ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
       ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
       ...
       int (*clone_file_range)(struct file *, loff_t, struct file *, loff_t, u64);
       ssize_t (*dedupe_file_range)(struct file *, u64, u64, struct file *, u64);
   } __randomize_layout;
```

Implementing File Operation Functions
Method Name Replacement
Creating Method Variables
Adding Return Values

43. File operations structure initialization

```
CarBMW = {a, b, c, d};                      // C89
CarBMW = {.A = a, .B = b, .C = c, .D = d};  // C99
```

* 44. Creating device files   -> dynamic device file creation   127~133

6) inode->cdev->fops (this is a real file operations of your driver) gets copied to file->f_op
7) file->f_op->open method gets called (read open method of your driver)

```
static int chrdev_open(struct inode *inode, struct file *filp)
{
    const struct file_operations *fops;
    struct cdev *p;
    struct cdev *new = NULL;
    int ret = 0;
    spin_lock(&cdev_lock);
    p = inode->i_cdev;

    if (!p) {
        struct kobject *kobj;
        int idx;
        spin_unlock(&cdev_lock);
        kobj = kobj_lookup(cdev_map, inode->i_rdev, &idx);
        if (!kobj) return -ENXIO;
        new = container_of(kobj, struct cdev, kobj);
        spin_lock(&cdev_lock);
        p = inode->i_cdev;
        if (!p) {
            inode->i_cdev = p = new;
            list_add(&inode->i_devices, &p->list);
            new = NULL;
        } else if (!cdev_get(p))
            ret = -ENXIO;
    } else if (!cdev_get(p))
        ret = -ENXIO;

    spin_unlock(&cdev_lock);
    cdev_put(new);

    if (ret) return ret;
    ret = -ENXIO;
    fops = fops_get(p->ops);

    if (!fops) goto out_cdev_put;
    replace_fops(filp, fops);

    if (filp->f_op->open) {
        ret = filp->f_op->open(inode, filp);
        if (ret) goto out_cdev_put;
    }
    return 0;

out_cdev_put:
    cdev_put(p);
```

class_create and
device_create

45. Character driver cleanup function implementation

```
scott@host:~/workspace/.../002pseudo_char_driver$ sudo insmod pcd.ko

  [46707.310503] pcd_driver_init :Device number <major>:<minor> : 240:0
  [46707.310795] pcd_driver_init :Module init was successful

scott@host:~/workspace/.../002pseudo_char_driver$
scott@host:~/workspace/.../002pseudo_char_driver$ cd /sys/class/pc
  pcd_class/ pci_bus/   pci_epc/

scott@host:~/workspace/.../002pseudo_char_driver$ ls /sys/class/pcd_class
   pcd

scott@host:~/workspace/.../002pseudo_char_driver$ ls /sys/class/pcd_class/pcd
   dev  power  subsystem  uevent

scott@host:/sys/class/pcd_class/pcd$ cat dev
  240:0

scott@host:/sys/class/pcd_class/pcd$ cat uevent
  MAJOR=240
  MINOR=0
  DEVNAME=pcd

scott@host:/sys/class/pcd_class/pcd$ ll /dev/pcd

  crw------- 1 root root 240, 0 Apr 24 19:36 /dev/pcd

scott@host:/sys/class$ ls
                                                    ^^^^^^^^^
  ata_device  devcoredump     firmware     iommu     pcd_class     pps           rtc           thermal       wakeup
  ata_link    devfreq         gpio         leds      pci_bus       printer       scsi_device   tpm           watchdog
  ata_port    devfreq-event   graphics     mdio_bus  pci_epc       ptp           scsi_disk     tpmrm
  backlight   dma             hidraw       mem       phy           pwm           scsi_generic  tty
  bdi         dmi             hwmon        misc      powercap      rapidio_port  scsi_host     vc
  block       drm             i2c-adapter  mmc_host  power_supply  regulator     sound         vfio
  bsg         drm_dp_aux_dev  i2c-dev      nd        ppdev         remoteproc    spi_master    virtio-ports
  dax         extcon          input        net       ppp           rfkill        spi_slave     vtconsole
```

```
scott@host:/sys/class$ sudo rmmod pcd.ko
scott@host:/sys/class$ dmesg

  [46707.310503] pcd_driver_init :Device number <major>:<minor> : 240:0
  [46707.310795] pcd_driver_init :Module init was successful
  [47219.531427] pcd_driver_cleanup :module unloaded
```

4. Character driver file operation implementation 1h1m 134~

46. Understanding read method
47. Understanding error codes
48. Read method implementation

49. Understanding write method
50. Write method implementation

51. lseek method
52. lseek method implementation

53. Testing pseudo char driver

```
$ sudo -s
# insmod pcd.ko
# dmesg

  [54292.379629] pcd_driver_init :Device number <major>:<minor> : 240:0
  [54292.379692] pcd_driver_init :Module init was successful

# echo "Hello, welcome to the course" > /dev/pcd

  [54292.379629] pcd_driver_init :Device number <major>:<minor> : 240:0
  [54292.379692] pcd_driver_init :Module init was successful

  [54432.442779] pcd_open :open was successful                          -> echo issued an OPEN system cal.

  [54432.442787] pcd_write :write requested for 29 bytes                -> echo issued a WRITE system cal.
  [54432.442788] pcd_write :current file position = 0
  [54432.442788] pcd_write :Number of bytes successfully write = 29
  [54432.442789] pcd_write :Updated file position = 29

  [54432.442791] pcd_release :release was successful                    -> echo issued an RELEASE system cal.

# cat /dev/pcd
  Hello, welcome to the course



      [55338.920592] pcd_write :Updated file position = 512

      [55338.920593] pcd_write :write requested for 3860 bytes
      [55338.920593] pcd_write :current file position = 512
      [55338.920593] pcd_write :--No space left on the device           <-   count == 0

      [55338.920812] pcd_release :release was successful
```

54. Error handling

5. Char driver with multiple device nodes 1h49m 187~

55. pcd driver with multiple devices

56. Pcd driver with multiple devices code implementation part-1
57. Pcd driver with multiple devices code implementation part-2
58. Pcd driver with multiple devices code implementation part-3
59. Pcd driver with multiple devices code implementation part-4
60. Pcd driver with multiple devices code implementation part-5
61. Pcd driver with multiple devices code implementation part-6
62. Pcd driver with multiple devices code implementation part-7

63. Pcd driver with multiple devices testing

```
$ sudo insmod pcd_n.ko

  [ 3708.848439] pcd_driver_init : Device number <major>:<minor> = 240:0
  [ 3708.848484] pcd_driver_init : Device number <major>:<minor> = 240:1
  [ 3708.848499] pcd_driver_init : Device number <major>:<minor> = 240:2
  [ 3708.848514] pcd_driver_init : Device number <major>:<minor> = 240:3
  [ 3708.848535] pcd_driver_init : Module init was successful


$ echo hello > /dev/pcdev-1

  bash: /dev/pcdev-1: Permission denied

  [ 3708.848535] pcd_driver_init : Module init was successful
  [ 3708.8485XX] pcd_open : minor access = 0
  [ 3708.8485XX] pcd_open : open was unsuccessful
```

```
  [54679.429313] pcd_open :open was successful

  [54679.429319] pcd_read :read requested for 131072 bytes
  [54679.429319] pcd_read :current file position = 0
  [54679.429321] pcd_read :Number of bytes successfully read = 512
  [54679.429321] pcd_read :Updated file position = 512

  [54679.429324] pcd_read :read requested for 131072 bytes
  [54679.429325] pcd_read :current file position = 512
  [54679.429325] pcd_read :Number of bytes successfully read = 0            -> end of file
  [54679.429326] pcd_read :Updated file position = 512

  [54679.429331] pcd_release :release was successful


# cp test.txt /dev/pcd


  root@host:~/workspace/ldd/custom_drivers/custom_drivers/002pseudo_char_driver# cp pcd.c /dev/pcd
    cp: error writing '/dev/pcd': No space left on device

  root@host:~/workspace/ldd/custom_drivers/custom_drivers/002pseudo_char_driver# dmesg | tail - 20

    [55108.260918] pcd_open :open was successful

    [55108.260928] pcd_write :write requested for 4289 bytes
    [55108.260929] pcd_write :current file position = 0
    [55108.260929] pcd_write :Number of bytes successfully write = 512
    [55108.260930] pcd_write :Updated file position = 512

    [55108.260931] pcd_write :write requested for 3777 bytes              4289 - 512 = 3777
    [55108.260931] pcd_write :current file position = 512
    [55108.260932] pcd_write :Number of bytes successfully write = 0      -> end of memory
    [55108.260932] pcd_write :Updated file position = 512

    [55108.261188] pcd_release :release was successful


    root@host:~/workspace/ldd/custom_drivers/custom_drivers/002pseudo_char_driver# cp pcd.c /dev/pcd
    cp: error writing '/dev/pcd': Cannot allocate memory

    root@host:~/workspace/ldd/custom_drivers/custom_drivers/002pseudo_char_driver# dmesg | tail -10

    [55328.641690] pcd_driver_init :Module init was successful

    [55338.920580] pcd_open :open was successful

    [55338.920590] pcd_write :write requested for 4372 bytes
    [55338.920590] pcd_write :current file position = 0
    [55338.920591] pcd_write :Number of bytes successfully write = 512



$ strace dd if=pcd_n.c of=/dev/pcdev-1

  execve("/bin/dd", ["dd", "if=pcd_n.c", "of=/dev/pcdev-1"], 0x7ffd37eb4840 /* 53 vars */) = 0
  brk(NULL)                               = 0x55e06a28a000
  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
  fstat(3, {st_mode=S_IFREG|0644, st_size=79287, ...}) = 0
  mmap(NULL, 79287, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f18eb28a000
  close(3)                                = 0
  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0"..., 832) = 832
  fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f18eb288000
  mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f18eac84000
  mprotect(0x7f18eae6b000, 2097152, PROT_NONE) = 0
  mmap(0x7f18eb06b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f18eb06b000
  mmap(0x7f18eb071000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f18eb071000
  close(3)                                = 0
  arch_prctl(ARCH_SET_FS, 0x7f18eb289540) = 0
  mprotect(0x7f18eb06b000, 16384, PROT_READ) = 0
  mprotect(0x55e0692dd000, 4096, PROT_READ) = 0
  mprotect(0x7f18eb29e000, 4096, PROT_READ) = 0
  munmap(0x7f18eb28a000, 79287)           = 0
  rt_sigaction(SIGINT, NULL, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
  rt_sigaction(SIGUSR1, {sa_handler=0x55e0690d00e0, sa_mask=[INT USR1], sa_flags=SA_RESTORER, sa_restorer=0x7f18eacc2f10}, NULL,
  8) = 0
  rt_sigaction(SIGINT, {sa_handler=0x55e0690d00d0, sa_mask=[INT USR1], sa_flags=SA_RESTORER|SA_NODEFER|SA_RESETHAND,
  sa_restorer=0x7f18eacc2f10}, NULL, 8) = 0
  brk(NULL)                               = 0x55e06a28a000
  brk(0x55e06a2ab000)                     = 0x55e06a2ab000
  openat(AT_FDCWD, "/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
  fstat(3, {st_mode=S_IFREG|0644, st_size=3004224, ...}) = 0
  mmap(NULL, 3004224, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f18ea9a6000
  close(3)                                = 0
  openat(AT_FDCWD, "pcd_n.c", O_RDONLY)   = 3
  dup2(3, 0)                              = 0
  close(3)                                = 0
  lseek(0, 0, SEEK_CUR)                   = 0
  openat(AT_FDCWD, "/dev/pcdev-1", O_WRONLY|O_CREAT|O_TRUNC, 0666) = -1 EACCES (Permission denied)
  openat(AT_FDCWD, "/usr/share/locale/locale.alias", O_RDONLY|O_CLOEXEC) = 3
  fstat(3, {st_mode=S_IFREG|0644, st_size=2995, ...}) = 0
  read(3, "# Locale name alias data base.\n#"..., 4096) = 2995
  read(3, "", 4096)                       = 0
  close(3)                                = 0
  openat(AT_FDCWD, "/usr/share/locale/en_US.UTF-8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
  openat(AT_FDCWD, "/usr/share/locale/en_US.utf8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
```

```
openat(AT_FDCWD, "/usr/share/locale/en_US/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en.UTF-8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en.utf8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US.UTF-8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or
directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US.utf8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or
directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en.UTF-8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or
directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en.utf8/LC_MESSAGES/coreutils.mo", O_RDONLY) = -1 ENOENT (No such file or
directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en/LC_MESSAGES/coreutils.mo", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=578, ...}) = 0
mmap(NULL, 578, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f18eb29d000
close(3)                                = 0
write(2, "dd: ", 4dd: )                     = 4
write(2, "failed to open '/dev/pcdev-1'", 29failed to open '/dev/pcdev-1') = 29
openat(AT_FDCWD, "/usr/share/locale/en_US.UTF-8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en_US.utf8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en_US/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en.UTF-8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en.utf8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale/en/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US.UTF-8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US.utf8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en_US/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en.UTF-8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en.utf8/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/usr/share/locale-langpack/en/LC_MESSAGES/libc.mo", O_RDONLY) = -1 ENOENT (No such file or directory)
write(2, ": Permission denied", 19: Permission denied)     = 19
write(2, "\n", 1
)                       = 1
close(2)                                = 0
exit_group(1)                           = ?
+++ exited with 1 +++
```

```
scott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$ sudo dd if=pcd_n.c of=/dev/pcdev-2

  dd: writing to '/dev/pcdev-2': Cannot allocate memory
  2+0 records in
  1+0 records out
  512 bytes copied, 0.000306726 s, 1.7 MB/s

scott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$ dmesg | tail
  [ 4143.773984] pcd_open : minor access = 1
  [ 4143.773985] pcd_open : open was successful

  [ 4143.773999] pcd_write : Write requested for 512 bytes



  [ 4628.650781] pcd_read : Read requested for 131072 bytes
  [ 4628.650782] pcd_read : Current file position = 0
  [ 4628.650783] pcd_read : Number of bytes successfully read = 1024
  [ 4628.650784] pcd_read : Updated file position = 1024

  [ 4628.650794] pcd_read : Read requested for 131072 bytes
  [ 4628.650794] pcd_read : Current file position = 1024
  [ 4628.650795] pcd_read : Number of bytes successfully read = 0
  [ 4628.650795] pcd_read : Updated file position = 1024

  [ 4628.650801] pcd_release : release was successful
```



```
#define platform_driver_register(drv)   __platform_driver_register(drv, THIS_MODULE)
extern int __platform_driver_register(struct platform_driver *, struct module *);
```

```
  [ 4143.774000] pcd_write : Current file position = 0
  [ 4143.774001] pcd_write : Number of bytes successfully written = 512
  [ 4143.774002] pcd_write : Updated file position = 512

  [ 4143.774004] pcd_write : Write requested for 512 bytes
  [ 4143.774005] pcd_write : Current file position = 512
  [ 4143.774006] pcd_write : No space left on the device

  [ 4143.774289] pcd_release : release was successful



$ sudo dd if=pcd_n.c of=/dev/pcdev-2 count=1

  1+0 records in
  1+0 records out
  512 bytes copied, 0.00010661 s, 4.8 MB/s

$ dmesg | tail -15

  [ 4336.492774] pcd_open : minor access = 1
  [ 4336.492775] pcd_open : open was successful

  [ 4336.492784] pcd_write : Write requested for 512 bytes
  [ 4336.492785] pcd_write : Current file position = 0
  [ 4336.492785] pcd_write : Number of bytes successfully written = 512
  [ 4336.492786] pcd_write : Updated file position = 512

  [ 4336.492788] pcd_release : release was successful

scott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$ sudo dd if=pcd_n.c of=/dev/pcdev-3 count=1 bs=100

  1+0 records in
  1+0 records out
  100 bytes copied, 0.000108374 s, 923 kB/s


scott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$ sudo cat /dev/pcdev-3

  #include<linux/module.h>
  #include<linux/fs.h>
  #include<linux/cdev.h>
  #include<linux/device.h>
  #inclscott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$

scott@host:~/workspace/ldd/ex/003_pseudo_char_driver_multiple$ dmesg | tail -15

  [ 4628.650774] pcd_open : minor access = 2
  [ 4628.650775] pcd_open : open was successful
```

```
Platform driver structure     : struct platform_driver

Registering a platform device : int platform_device_register(struct platform_device *pdev);

Platform device structure     : struct platform_device
   probe(),              Called when matched platform device is found
   remove(),
   shutdown(),           Called at shut-down time to quiesce the device
   suspend(),            Called to put the device to sleep mode. Usually to a low power state
   resume(), ...         Called to bring a device from sleep mode
```



```
  - "matching" mechanism of the bus core

  The Linux platform core implementation maintains platform device and driver lists.
  Whenever you add a new platform device or driver, this list gets updated and matching mechanism triggers.

  Every bus type has its match function, where the device and driver list will be scanned.
```



```
  • Whenever a new device or a new driver is added, the matching function of the platform
    bus runs, and if it finds a matching platform device for a platform driver, the probe function of the matched driver will
    get called. Inside the probe function, the driver configures the detected device.

  • Details of the matched platform device will be passed to the probe function of the
    matched driver so that driver can extract the platform data and configure it.
```



```
  • Probe function must be implemented by the platform driver and should be registered during platform_driver_register().
  • When the bus matching function detects the matching device and driver, probe function of the driver gets called with
  detected platform device as an input argument
  • Note that probe() should in general, verify that the specified device hardware actually exists.
    Sometimes platform setup code can't be sure.
    The probing can use device resources, including clocks, and device platform_data.
  • The the probe function is responsible for
  • Device detection and initialization
  • Allocation of memories for various data structures,
  • Mapping i/o memory
  • Registering interrupt handlers
  • Registering device to kernel framework, user level access point creations, etc
  • The probe may return 0(Success) or error code. If probe function returns a non-zero value, meaning probing of a device has
  failed.
```

Remove function of the platform driver 242
- Remove function gets called when a platform device is removed from the kernel to unbind a device from the driver or when the kernel no longer uses the platform device
- Remove function is responsible for
- Unregistering the device from the kernel framework
- Free memory if allocated on behalf of a device
- Shutdown/De-initialize the devic

https://www.kernel.org/doc/Documentation/driver-model/platform.txt

70. Platform driver code exercise 245

essential functions

```
int pcd_platform_driver_remove(struct platform_device *pdev)
int pcd_platform_driver_probe(struct platform_device *pdev)

static int __init pcd_platform_driver_init(void)
{
        platform_driver_register();
        return 0;
}

static void __exit pcd_platform_driver_exit(void)
{
    platform_driver_unregister();
}
```

Code exercise: 248

Implementation of pseudo character driver as platform driver
- Repeat the exercise pseudo character driver with multiple devices as a platform driver.
- The driver should support multiple pseudo character devices(pcdevs) as platform devices
- Create device files to represent platform devices
- The driver must give open, release, read, write, lseek methods to deal with the devices

kernel module 1 : platform driver 250
kernel module 2 : platform device setup

Platform device setup 252

1. Create 2 platform devices and initialize them with required information
   - Name of a platform device
   - Platform data
   - Id of the device
   - Release function for the device
2. Register platform devices with the Linux kernel

```
kgdboc                  pcspkr            serial8250

$ dmesg | tail -30
  [18617.022770] pcdev_platform_init : Device setup module loaded

$ sudo rmmod pcd_device_setup.ko
$ dmesg | tail -30
  [18861.674563] pcdev_release : Device released
  [18861.674825] pcdev_release : Device released
  [18861.674869] pcdev_release : Device released
  [18861.674880] pcdev_release : Device released
  [18861.674881] pcdev_platform_exit : Device setup module unloaded

$ ls /sys/devices/platform/
  eisa.0                pcspkr                    serial8250
  'Fixed MDIO bus.0'    platform-framebuffer.0    uevent
  i8042                 power
  intel_rapl_msr.0      reg-dummy
  kgdboc                rtc_cmos
```

73. Platform driver code implementation part-1 : driver(module 1)

```
struct device_driver        in linux/device.h
platform_driver_register    in platform_device.h
```

74. Platform driver code implementation part-2

```
# insmod pcd_device_setup.ko
  [18861.674881] pcdev_platform_init : Device setup module loaded

# insmod pcd_platform_driver.ko
  [18861.674881] pcdev_platform_init : Device setup module loaded
  [18861.674881] pcd_platform_driver_probe : A device is detected
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded

# rmmod pcd_platform_driver.ko
  [18861.674881] pcdev_platform_init : Device setup module loaded
  [18861.674881] pcd_platform_driver_probe : A device is detected
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded
  [18861.674881] pcd_platform_driver_remove : A device is removed
  [18861.674881] pcd_platform_driver_cleanup : pcd platform driver unloaded

# rmmod pcd_device_setup
  [18861.674881] pcdev_platform_init : Device setup module loaded
  [18861.674881] pcd_platform_driver_probe : A device is detected
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded
  [18861.674881] pcd_platform_driver_remove : A device is removed
```

71. Platform device setup code implementation

```
struct pcdev_private_data       /*Device private data structure */
{
    struct pcdev_platform_data pdata;
    char *buffer;
    dev_t dev_num;
    struct cdev cdev;
};

struct pcdrv_private_data       /*Driver private data structure */
{
    int total_devices;
    dev_t device_num_base;
    struct class *class_pcd;
    struct device *device_pcd;
};

struct platform_device                                      347
{
    const char *name;           <= used to match
    int id;                     <= for multiple instances
    struct device dev;                                      311
}
```

72. Platform device setup code implementation contd. : setup device(module 2)

Platform device release function 251
- Callback to free the device after all references have gone away.
- This should be set by the allocator of the device

```
platform_add_devices(platform_pcdevs,ARRAY_SIZE(platform_pcdevs) );
platform_device_unregister(&platform_pcdev_1);

$ sudo insmod pcd_device_setup.ko

$ ls /sys/devices/platform/
  eisa.0                pcdev-A1x.0       platform-framebuffer.0      uevent
  'Fixed MDIO bus.0'    pcdev-B1x.1       power
  i8042                 pcdev-C1x.2       reg-dummy
  intel_rapl_msr.0      pcdev-D1x.3       rtc_cmos

  [18861.674881] pcd_platform_driver_cleanup : pcd platform driver unloaded

  [18861.674881] pcdev_release : Device released
  [18861.674881] pcdev_release : Device released
  [18861.674881] pcdev_platform_exit : Device setup module unloaded
```

or

```
# insmod pcd_platform_driver.ko
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded

# insmod pcd_device_setup.ko
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded
  [18861.674881] pcd_platform_driver_probe : A device is detected
  [18861.674881] pcdev_platform_init : Device setup module loaded

# rmmod pcd_device_setup
  [18861.674881] pcd_platform_driver_init : pcd platform driver loaded
  [18861.674881] pcd_platform_driver_probe : A device is detected
  [18861.674881] pcdev_platform_init : Device setup module loaded
  [18861.674881] pcd_platform_driver_remove : A device is removed
  [18861.674881] pcdev_release : Device released
  [18861.674881] pcdev_release : Device released
  [18861.674881] pcdev_platform_exit : Device setup module unloaded
```

75. Platform driver code implementation part-3

76. Platform driver code implementation part-4

```
    pdata = pdev->dev.platform_data;
or  pdata = (struct pcdev_platform_data*)dev_get_platdata(&pdev->dev);
```

Kernel memory allocation APIs 325
- kmalloc ()
- kfree ()

```
void* kmalloc( size_t size, gfp_t flags);//include/linux/slab.h 326
```

Used to allocate memory in kernel space by drivers and kernel functions
Memory obtained are physically(RAM) contiguous

gfp_t : get free pages

kzalloc - allocate memory. The memory is set to zero.

77. Platform driver code implementation part-5

78. Platform driver code implementation part-6

~ 335

to free allocated memory by kalloc()

```
struct platform_device {
    ...
    struct device   dev;
    ...
}

struct device {
    ...
    void *platform_data;        <=
    void *driver_data;          <=
    ...
}

// save the device private data pointer in platform device structure
pdev->dev.driver_data = dev_data;               or
dev_set_drvdata(*pdev->dev, dev_data);

==>

struct pcdev_private_data *dev_data = dev_get_drvdata(&pdev->dev);
kfree(dev_data->buffer);
kfree(dev_data);
```

79. Testing platform driver

```
# insmod pcd_device_setup.ko
# insmod pcd_platform_driver.ko
# lsmod                                 -> list all the modules
Module                  Size    Used by
pcd_platform_driver     16384   0       <= !!
pcd_device_setup        16384   0
...

# rmmod pcd_device_setup
#
# rmmod pcd_platform_driver
# insmod pcd_platform_driver.ko
# insmod pcd_device_setup.ko
#
# rmmod pcd_platform_driver
```

* 84. Platform device driver matching using platform device ids 254 ~ 256

device_id

```
static int platform_match(struct device *dev, struct device_driver *drv)    // where matching happens
{
    struct platform_device *pdev = to_platform_device(dev);
    struct platform_driver *pdrv = to_platform_driver(drv);

    of_driver_match_device(dev, drv)                // open firmware driver, ie device tree matching

    acpi_driver_match_device(dev, drv)              //

    return platform_match_id(pdrv->id_table, pdev); // id_table

    return (strcmp(pdev->name,drv->name) == 0);
}

static const struct platform_device_id *platform_match_id(const struct platform_device_id *id, struct platform_device *pdev)
{
    while (id->name[0]) {
        if (strcmp(pdev->name, id->name) == 0) {
            pdev->id_entry = id;                        <= updated to platform_device->id_entry
            return id;
        }
        id++;
    }
    return NULL;
}
```

if init and exit function consists of platform_driver_register and platform_driver_unregister only,

```
  =>  #define module_platform_driver(__platform_driver) \
            module_driver(__platform_driver, platform_driver_register, \
            platform_driver_unregister)
```

* 85. Fixing error handling in probe function

```
really_probe()
  -> drv->probe()
```

80. Linux device resource managed functions

336~340

```
devm_kzalloc()      <=>     kalloc & kfree      <= based on the existence of struct device
devm_gpio_get()     <=>     gpio_get()/put()
devm_request_irq()  <=>     request_irq()/free_irq()
```

www.kernel.org/doc/Documentation/driver-model/devres.txt

MFD, MUX, PCI, PHY, MEM, PWM, MDIO, IOMAP, INPUT, IRQ, IO region, IIC, CLOCK, DRM, GPIO, ...

* 81. Using device resource managed kernel functions

kxxxx() -> devm_kxxxx()

* 82. Testing with more platform devices

```
int platform_add_devices(struct platform_device **devs, int num) {
    for (int i = 0; i < num; i++) {
        platform_device_register(devs[i]);
    }
}
```

kernel crash w/ same ids

```
struct platform_device platform_pcdev_1 = {
    .name = "pcdev-A1x",
    .id = 0,
};


struct platform_device platform_pcdev_2 = {
    .name = "pcdev-A1x",
    .id = 0,                                    =>  1 !!!
};


rmmod -> "Killed"   => need to reboot !!!
```

83. Fixing kernel crash

```
# ls -l /dev/pcdev- [tab][tab]
```

7. Device tree 1h26m 258

86. Introduction to device tree

What is device tree? 260 ~ 262

- The "Open Firmware Device Tree", or simply Device Tree (DT),
  is a data exchange format used for exchanging hardware description data with the software or OS.
- More specifically, it is a description of hardware that is readable by an operating system
  so that the operating system doesn't need to hard code details of the machine.
- In short, it is a new and recommended way to describe non-discoverable devices(platform devices)
  to the Linux kernel, which was previously hardcoded into kernel source files.

Source : Documentation/devicetree/usage-model.txt

Device tree

- An operating system uses the Device Tree to discover the topology of the hardware at runtime,
  and thereby support a majority of available hardware without hardcoded information
  (assuming drivers were available for all devices)
- The most important thing to understand is that the DT is simply a data structure
  that describes the hardware. There is nothing magical about it,
  and it does not magically make all hardware configuration problems go away
- DT provides a language for decoupling the hardware configuration from the device driver
  and board support files of the Linux kernel (or any other operating system for that matter).
- Using it allows device drivers to become data-driven. To make setup decisions
  based on data passed into the kernel instead of on permachine hardcoded selections.
- Ideally, a data-driven platform setup should result in less code duplication
  and make it easier to support a wide range of hardware with a single kernel image.

Why DT is used ?

Linux uses DT for,
- Platform identification : identify the board or machine on which the kernel runs
- Device population:

- The kernel parses the device tree data and generates the required software data structure, which will be used by the kernel code.

Ideally, the device tree is independent of any os; when you change the OS, you can still use the same device tree file to describe the hardware to the new OS. That is, the device tree makes “adding of device information “ independent of OS

More reading
- https://elinux.org/Device_Tree_What_It_Is
- https://www.kernel.org/doc/Documentation/devicetree/usagemodel.txt

87. Writing device tree 265

Device tree specification
- You can get the full specification here
- https://www.devicetree.org/

Writing device tree

- The device tree supports a hierarchical way of writing hardware description at the soc level, common board level, and board-specific level. Most of the time, writing a new device tree is not difficult, and you can reuse most of the common hardware information from the device tree file of the reference board.

- For example, when you design a new board, which is slightly different from another reference board, then you can reuse the device tree file of the reference board and only add that information which is new in your custom board

Describing hardware hierarchy
- It comes at various level because the board has many device blocks
- SOC
- SOC has an on-chip processor and on-chip peripherals
- The board also has various peripherals onboard, like sensors, LEDs, buttons, joysticks, external memories, touchscreen, etc

Modular appoach

mod1 : SOC specific device tree file Board specific device tree file
(This DT file is used as an include file and can be used with another board which is based on same SOC)

mod2 : Board specific device tree file

linux/arch/arm/boot/dts/am335x-evm.dts => #include "am33xx.dtsi" (soc level device file)

```
&sgx {                <= reference
   status = "okay";
};
```

am33xx.dtsi

```
/ {
     compatible = "ti, am33xx";                <= overridden

     sgx: sgx@56000000 {
        compatible = "   ";
        ti, hwmods = "gfx";

     };
  };
};
```

```
#include "am335x-bone-common.dtsi"
#include "am335x-boneblack-common.dtsi"
```

Never edit top level files => it will be overridden later.

89. Device tree writing syntax 20

Device tree writing syntax 279
- Node name
- Node Label
- Standard and non-standard property names
- Different data type representation (u32, byte, byte stream, string, stream of strings, Boolean, etc)
- SoC node and children

Node name
- Refer device tree specification Release v0.3 from devicetree.org

https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.4

The unit-address component of the name is specific to the bus type on which the node sits.
It consists of one or more ASCII characters from the set of characters in Table 2.1.
The unit-address must match the first address specified in the reg property of the node.
If the node has no reg property, the @unit-address must be omitted and the node-name alone differentiates the node from other nodes at the same level in the tree.
The binding for a particular bus may specify additional, more specific requirements for the format of reg and the unit-address.

am335x-boneblack.dts 272, 278

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

88. Device tree structure 19

Overview of device tree structure
- ✓ Device tree is a collection of device nodes
- ✓ A ‘device node’ or simply called ‘a node’ represents a device. Nodes are organized in some systematic way inside the device tree file.
- ✓ They also have parent and child relationship, and every device tree must have one root node
- ✓ A node explains itself, that is, reveals its data and resources using its “properties.”

Root node

- The device tree has a single root node of which all other device nodes are descendants. The full path to the root node is /.
- All device trees shall have a root node, and the following nodes shall be present at the root of all device trees:
  - One /CPUs node
  - At least one /memory node

Chapter 3 :DEVICE NODE REQUIREMENTS Devicetree Specification Release v0.3a

How to write a device tree ?

- Remember that you most probably be writing device tree addons or overlays for your board-related changes but not for entire soc.
- The soc specific device tree will be given by the vendor in the form of device tree inclusion file (.dtsi) and you just need to include that in your board-level device tree
- Follow modulatory approach while writing device tree

am335x-boneblack.dts 272, 278

```
/ {
     model = "TI AM335x BeagleBone Black";          <= added
     compatible = "ti, am335x-bone-black", ... ;    <= override that of included files


  };                    <= root level node (children of root)
};                      <= root
```

ex. 281

```
i2c0: i2c@44e0b000 {
    compatible = "ti, omap4-i2c";
    #address-cells = <1>;
    #size-cells = <0>;
    ti, hwmods = "i2c1";
    reg = <0x44e0b000 0x1000>;      <= AM335x TRM ch2 memory map
    interrupts = <70>;
    status = "disabled";
};

i2c1: i2c@44e2a000 {
    compatible = "ti, omap4-i2c";
    #address-cells = <1>;
    #size-cells = <0>;
    ti, hwmods = "i2c2";
    reg = <0x4802a000 0x1000>;
    interrupts = <71>;
    status = "disabled";
};

i2c2: i2c@48060000 {

    ...
```

90. Device tree parent and child node 282 ~ 283

```
&i2c0 {
    status = "okay";

    tps: tps@24 {
        reg = <0x24>;   // i2c address
    };

    baseboard_eeprom: baseboard_eeprom@50 {
        reg = <0x50>

        baseboard_data: baseboard_data@0 {
            reg = <0 0x100>;
        };
    };
};
```

91. Device tree properties 287

label,

```
gpios,                      : standard property
default-state

linux,default-trigger       : custom property
```

Different types of properties
- Standard property
- Custom property (non-standard)
- Standard properties are those which is explained by the specification and the device-driver binding documentation
- Custom properties are specific to a particular vendor or organization which is not documented by the specification.
- That is why, when you use custom property, always begin with your organization name.

92. 'compatible' property 293

Root compatible property of BBB

```
/ {
    model = "TI AM335x BeagleBone Black";
    compatible = "ti,am335x-bone-black", "ti,am335x-bone", "ti,am33xx";
};
                Sorted string list from most compatible to least.

; Root compatible property is used for machine identification
```

Uses of compatible property
1. Machine identification and initialization
2. Match and load the appropriate driver for the device

```
    &i2c0 {
        status = "okay";                            !!!!

        tps: tps@24 {
            reg = <0x24>;   // i2c address
        };

        baseboard_eeprom: baseboard_eeprom@50 {
            compatible = "atmel,24c256";    <= match found in linux/drivers/misc/eeprom/at24.c
            reg = <0x50>

            #address-cells = <1>;
            #size-cells = <1>;
            baseboard_data: baseboard_data@0 {
                reg = <0 0x100>;
            };
        };
    };
```

linux/Documentation/devicetree/bindings/i2c/i2c-omap.txt

```
    Required properties:
    - compatible : Must be
        "ti,omap2420-i2c" for OMAP2420 SoCs
        "ti,omap2430-i2c" for OMAP2430 SoCs
        "ti,omap3-i2c" for OMAP3 SoCs
        "ti,omap4-i2c" for OMAP4+ SoCs
        "ti,om654-i2c", "ti,omap4-i2c" for AM654 SoCs
        "ti,j721e-i2c", "ti,omap4-i2c" for J721E SoCs

    Recommended properties:

    Optional properties:

    ex...
```

- Compatible strings and properties are first defined by the client program (OS , drivers) then shared with DT writer

Device tree bindings- points to remember

- Case 1 : When the driver is already available in the Linux kernel for the device ‘x,’
  but you just need to add device ‘x’ entry in the device tree
  then you must consult ‘x’ drivers binding document
  which guides you through creating device tree node for device ‘x.’

- Case 2 : When the driver is not available for the device ‘x,’
  then you should write you own driver, you should decide what properties to use
  (could be a combination of standard and non-standard property),
  you should then provide the device tree binding document
  describing what are all the properties and compatible strings a device tree write must include.

ex. lm75

linux/Documentation/devicetree/bindings/hwmon/lm75.txt

```
   sensor@48 {
      compatible = "st,stlm75";
      reg = <0x48>;
   }
```

linux/drivers/hwmon/lm75.c

```
   static const struct of_device_id lm75_of_match[] = {
      .compatible = "ti, tmpXXX";
      .data = (void *)tmpXXX
   };
```

```
    i2c0: i2c@44e0b000 {
        compatible = "ti, omap4-i2c";               <= matched !
        #address-cells = <1>;
        #size-cells = <0>;
        ti, hwmods = "i2c1";
        reg = <0x44e0b000 0x1000>;
        interrupts = <70>;
        status = "disabled";
    };
```

in linux/drivers/i2c/buses/i2c-omap.c -> drivers

```
    static struct platform_driver omap_i2c_driver = {
        .proble = omap_i2c_probe,
        .remove = omap_i2c_remove,
        .driver = {
            .name = "omap_i2c",
            .pm = &omap_i2c_pm_ops,
            .of_match_table = of_match_ptr(omap_i2c_of_match),
        },
    }

    static const struct of_device_id omap_i2c_of_match[] = {
        {
            .compatible = "ti, omap4-i2c",          <= matched !
            .data = &omap4_pdata,
        },
        {
            .compatible = "ti, omap3-i2c",
            .data = &omap3_pdata,
        },
        ...
    }
```

93. Device tree binding 298

Device tree bindings
- How do you know which property name and value pair should be used to describe a node in the device tree?
- Device tree binding document. The driver writer must document these details
- The properties that are necessary to describe a device in the device tree depends on the requirements of the Linux driver for that device
- When all the required properties are provided, the driver of charge can detect the device from the device tree and configure it.

ex. mpu 6050

linux/Documentation/devicetree/bindings/iio/imu/inv_mpu6050.txt

```
   mpu6050@68 {
      compatible = "invensense,mpu6050";
      reg = <0x68>;
      ...
   }
```

linux/drivers/hwmon/lm75.c

```
   static const struct of_device_id lm75_of_match[] = {
      .compatible = "ti, tmpXXX";
      .data = (void *)tmpXXX
   };
```

https://kernel.org/doc/Documentation/devicetree/bindings/i2c/i2c-omap.txt

Linux conventions to write device tree

hex constants are lower case
- use "0x" instead of "0X"
- use a..f instead of A..F, eg 0xf instead of 0xF

node names
- should begin with a character in the range 'a' to 'z', 'A' to 'Z'
- unit-address does not have a leading "0x" (the number is assumed to be hexadecimal)
- unit-address does not have leading zeros
- use dash "-" instead of underscore "_"

label names
- should begin with a character in the range 'a' to 'z', 'A' to 'Z'
- should be lowercase
- use underscore "_" instead of dash "-"

property names
- should be lower case
- should begin with a character in the range 'a' to 'z'
- use dash "-" instead of underscore "_"

https://elinux.org/Device_Tree_Linux#Linux_vs_ePAPR_Version_1.1

94. pcd device tree version

```
# git branch -a
```

```
 *4.14
  remote/origin/5.4
  remote/origin/HEAD -> origin/4.14

# git stash && git checkout 5.4

# git branch
  4.14
  4.9
 *5.4

# git stash apply

# git clone https://github.com/beagleboard/linux.git linux_bbb_5.4
```

95. Switching to Linux kernel version 5.4

```
# git checkout 5.4
```

96. Updating Linux kernel image of 5.4

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bb.org_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- uImage dtbs LOADADDR=0x80008000 -j4
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4 modules
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules_install


$ cd /home/kiran/workspace/ldd/src/linux_bbb_5.4
$ cd arch/arm/boot
$ cp uImage /media/kiran/BOOT/
$ cd dts
$ cp am335x-boneblack.dtb /media/kiran/BOOT/
$ cd /ib/modules/
$ sudo cp -a 5.4.47/ /media/kiran/ROOTFS/lib/modules/
$ sync

$ uname -r
5.4.47

$ ifconfig  -> reboot if usb0 & usb1 is not seen.
```

8. device tree nodes and platform driver 1h16m

97. Device tree nodes for pcd driver

```
static int platform_match(struct device *dev, struct device_driver *drv)
{

    of_driver_match_device(dev, drv)               // open firmware driver, ie device tree matching

    // Refer to 84 for details.

}

static inline int of of_driver_match_device(struct device *dev, const struct device_driver *drv)
{
    return of_match_device(drv->of_match_table, dev) != NULL;
}                                ^^^^^^^^^^^^^^


struct of_device_id {
    char name[32];
    char type[32];
    char compatible[128];
    const void *data;
};

const struct of_device_id *of_match_device(const struct of_device_id *matches, const struct device *dev)
{
    if ((!matches) || (!dev->of_node))  return NULL;
    return of_match_node(matchds, dev->of_node);           // try to mach
}


struct device_node *of_node;
struct device_node {
    const char *name;

    struct property *properties;
    struct device_node *parent;
    struct device_node *child;
    struct device_node *sibling;
    ...
};


>> pcd_platform_driver_dt.c

    #include<linux/of.h>

    struct of_device_id org_pcdev_dt_match[] =
    {
        {.compatible = "pcdev-A1x",.data = (void*)PCDEVA1X},
        {.compatible = "pcdev-B1x",.data = (void*)PCDEVB1X},
        {.compatible = "pcdev-C1x",.data = (void*)PCDEVC1X},
```

```
workspace/ldd/src/linux_bbb_5.4/arch/arm/boot/dts# vi am335x-boneblack.dts
#include "am335x-boneblack-lddcrs.dtsi"



workspace/ldd/src/linux_bbb_5.4/arch/arm/boot/dts# vi am335x-boneblack-lddcrs.dtsi
/ {
    pcdev-1 {
        compatible = "pcdev-E1x", "pcdev-A1x";
        org,size = <512>
        org,device-serial-num = “PCDEV1ABC123”
        org,perm = <0x11>
    };
    pcdev-2 {
        compatible = "pcdev-B1x";
        org,size = <1024>
        org,device-serial-num = “PCDEV2ABC123”
        org,perm = <0x11>
    };
    pcdev-3 {
        compatible = "pcdev-C1x";
        org,size = <256>
        org,device-serial-num = “PCDEV3ABC123”
        org,perm = <0x11>
    };
    pcdev-1 {
        compatible = "pcdev-D1x";
        org,size = <1024>
        org,device-serial-num = “PCDEV4ABC123”
        org,perm = <0x11>
    };
};

-> device tree compiler to get dtb
-> org is vendor or manufacture. this field is mandatory for non standard properties
-> integer property should be defined in <>
-> refer to Table 2.3: Property values in devicetree-specification.pdf
```

98. Pcd platform driver DT coding part-1

```
struct Platform_device_id in pcd_platform_driver_dt.c can be used to bind device and driver
only when device regiter function is called manually.
=> another list of device is needed inside device tree.
=> To process device nodes of the device file needs a separate structure.


in linux/drivers/platform.c


        {.compatible = "pcdev-D1x",.data = (void*)PCDEVD1X},
        { } /*Null termination*/

    };

    struct platform_driver pcd_platform_driver =
    {
        .probe = pcd_platform_driver_probe,         <= !!
        .remove = pcd_platform_driver_remove,
        .id_table = pcdevs_ids,
        .driver = {
            .name = "pseudo-char-device",
            .of_match_table = of_match_ptr(org_pcdev_dt_match)
        }
    };

    // extract properties and make a decision
    int pcd_platform_driver_probe(struct platform_device *pdev)
    {
        int ret;

        pr_info(dev,"A device is detected\n");
    }


    int pcd_platform_driver_remove(struct platform_device *pdev)
    {
        dev_info(&pdev->dev,"A device is removed\n");
        return 0;
    }
```

99. Testing device tree changes on board

```
>> am335x-boneblack-lddcrs.dtsi on VBox

    / {
        pcdev1: pcdev-1 {
            compatible = "pcdev-E1x","pcdev-A1x";
            org,size = <512>;
            org,device-serial-num = "PCDEV1ABC123";
            org,perm = <0x11>;
        };
    };

    # make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am335x-boneblack.dtb     <= source is .dts
    # scp arch/arm/boot/dts/am335x-boneblack.dtb debian@192.168.7.2:/home/debian/drivers

    or in Makefile,
```

```
    copy-dtb:
        scp     ~/workspace/ldd/source/linux_bbb_5.4/arch/arm/boot/dts/am335x-boneblack.dtb
        debian@192.168.7.2:/home/debian/drivers

    copy-drv:
        scp *.ko  debian@192.168.7.2:/home/debian/drivers


>> on beaglebone

    # lsblk
    mmcblk0
      +- mmcblk0p1      part        <= .dtb will be copied to here. needs to mount first
      +- mmcblk0p2      part /

    # mount /dev/mmcblk0p1 /mnt/
    # cd /mnt/
    # ls
      am335x-boneblack_4.14.dtb     MLO     uImage      uEnv.txt
    # cp /home/debian/drivers/am335x-boneblack.dtb .
    # sync
    # reboot

    # ls /sys/devices/platform/
      ...       pcdev-1     pcdev-2     pcdev-3     pcdev-4     ...
    # ls /sys/devices/platform/pcdev-1
      driver_override       modalias    of_node     power       subsystem       uevent
    # ls /sys/devices/platform/pcdev-1/of_node
      compatible    name    org,device-serial-num   org,perm    org,size
    # cat /sys/devices/platform/pcdev-1/of_node/compatible
      pcdev-E1xpcdev-A1x


    $ sudo insmod pcd_platform_driver_dt.ko

    $ dmesg
    [ 168.xx]  pcd_platform_driver_dt: loading out-of-tree module taint kernel.
    [ 168.xx]  pcd_platform_driver_probe : A device is detected
    [ 168.xx]  pcd_platform_driver_probe : A device is detected
    [ 168.xx]  pcd_platform_driver_probe : A device is detected
    [ 168.xx]  pcd_platform_driver_probe : A device is detected
    [ 168.xx]  pcd_platform_driver_init : pcd platform driver loaded

    $ sudo rmmod pcd_platform_driver_dt
    [ 168.xx]  pcd_platform_driver_remove : A device is remove
    [ 168.xx]  pcd_platform_driver_remove : A device is remove
    [ 168.xx]  pcd_platform_driver_remove : A device is remove
    [ 168.xx]  pcd_platform_driver_remove : A device is remove
    [ 168.xx]  pcd_platform_driver_cleanup : pcd platform driver unloaded



            driver_data =  pdev->id_entry->driver_data;
        }

        /*2. Dynamically allocate memory for the device private data  */
        dev_data = devm_kzalloc(&pdev->dev, sizeof(*dev_data),GFP_KERNEL);

        /*save the device private data pointer in platform device structure */
        dev_set_drvdata(&pdev->dev,dev_data);

        dev_data->pdata.size = pdata->size;
        dev_data->pdata.perm = pdata->perm;
        dev_data->pdata.serial_number = pdata->serial_number;

        /*3. Dynamically allocate memory for the device buffer using size
        information from the platform data */
        dev_data->buffer = devm_kzalloc(&pdev->dev,dev_data->pdata.size,GFP_KERNEL);

        /*4. Get the device number */
        dev_data->dev_num = pcdrv_data.device_num_base + pcdrv_data.total_devices;

        /*5. Do cdev init and cdev add */
        cdev_init(&dev_data->cdev,&pcd_fops);

        dev_data->cdev.owner = THIS_MODULE;
        ret = cdev_add(&dev_data->cdev,dev_data->dev_num,1);

        /*6. Create device file for the detected platform device */
        // 2nd argument : parent, e.g. dev_data->dev_num
        pcdrv_data.device_pcd = device_create(pcdrv_data.class_pcd,dev, dev_data->dev_num, NULL,\
                                "pcdev-%d", pcdrv_data.total_devices);


        if(IS_ERR(pcdrv_data.device_pcd)){
            dev_err(dev,"Device create failed\n");
            ret = PTR_ERR(pcdrv_data.device_pcd);
            cdev_del(&dev_data->cdev);
            return ret;
        }

        pcdrv_data.total_devices++;

        dev_info(dev,"Probe was successful\n");

        return 0;
    }


    int pcd_platform_driver_remove(struct platform_device *pdev)
    {
```

100. Pcd platform driver DT coding part-2 304~

```
    - two of device instantiations.
        the device setup file manually calling the function platform device registered as in pcd_device_setup.c
        the device tree file

    struct platform_device {
        struct device               dev;
        {
            struct device_node      *of_node;   // represents an associated device tree node
            struct fwnode_handle    *fwnode;    // fw device node
        };
    };


    linux/include/of.h

        of_property_read_string(const struct device_node *np,
                                const char *propname,
                                const char **out_string);

        of_property_read_u32(                           );



    pr_info(...) vs dev_info(dev, ...)  -> can't use in _init & _exit where dev is not available)


            // extract properties and make a decision
            int pcd_platform_driver_probe(struct platform_device *pdev)
            {
                int ret;

                struct pcdev_private_data *dev_data;
                struct pcdev_platform_data *pdata;
                struct device *dev = &pdev->dev;
                int driver_data;

                /* used to store matched entry of 'of_device_id' list of this driver */
                const struct of_device_id *match;

                /*match will always be NULL if LINUX doesnt support device tree i.e CONFIG_OF is off */
                match = of_match_device(of_match_ptr(org_pcdev_dt_match),dev);

                if(match){
                    pdata = pcdev_get_platdata_from_dt(dev);
                    driver_data = (long)match->data;
                }else{
                    pdata = (struct pcdev_platform_data*)dev_get_platdata(dev);



            #if 1
                struct pcdev_private_data  *dev_data = dev_get_drvdata(&pdev->dev);

                /*1. Remove a device that was created with device_create() */
                device_destroy(pcdrv_data.class_pcd,dev_data->dev_num);

                /*2. Remove a cdev entry from the system*/
                cdev_del(&dev_data->cdev);

                pcdrv_data.total_devices--;

            #endif
                dev_info(&pdev->dev,"A device is removed\n");
                return 0;
            }
```

101. Pcd platform driver DT coding part-3

```
    - As explained in part 100, there are two of device instantiations,
      device setup file & the device tree file
      To test device setup file, driver ko in 004 example is used.

    >> Vbox
     $ make                                 // 005_pcd_platform_driver_dt
     $ cd ../004_pcd_platform_driver
     $ make                                 // copy driver module to BBB

    >> BBB

     # insmod pcd_platform_driver_dt.ko                                          <- device tree
     [ XXXX    ] pseudo-char-device pcdev-1: A device is detected
     [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEV1ABC123
     [ XXXX    ] pcd_platform_driver_proble : Device size = 512
     [ XXXX    ] pcd_platform_driver_proble : Device permission = 17
     [ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
     [ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
     [ XXXX    ] pseudo-char-device pcdev-1: Probe was successful

     [ XXXX    ] pseudo-char-device pcdev-1: A device is detected
     [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEV2ABC456
       ...

     # ls -l /dev/pcdev-    [tab]
     pcdev-0         pcdev-1     pcdev-2     pcdev-3

     # insmod pcd_device_setup.ko                                                <- device setup code
     [ XXXX    ] pseudo-char-device pcdev-A1x.0: A device is detected               printed by dev_info (device name O)
     [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVABC111    printed by pr_info  (device name X)
     [ XXXX    ] pcd_platform_driver_proble : Device size = 512
```

```
[ XXXX    ] pcd_platform_driver_proble : Device permission = 17
[ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
[ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
[ XXXX    ] pseudo-char-device pcdev-A1x.0: Probe was successful

[ XXXX    ] pseudo-char-device pcdev-B1x.1: A device is detected
[ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVXYZ222
   ...

# ls -l /dev/pcdev-    [tab]
pcdev-0        pcdev-1     pcdev-2     pcdev-3     pcdev-4     pcdev-5     pcdev-6     pcdev-7

# ls /sys/devices/platform/
 eisa.0                  pcdev-1           pcdev-B1x.0             reg-dummy
 'Fixed MDIO bus.0'      pcdev-2           pcdev-C1x.0             rtc_cmos
 i8042                   pcdev-3           pcdev-D1x.0             serial8250
 intel_rapl_msr.0        pcdev-4           platform-framebuffer.0  uevent
 kgdboc                  pcdev-A1x.0       power

# /home/debian/drivers# rmmod pcd_platform_driver_dt.ko
[ YYYY    ] pseudo-char-device pcdev-D1x.3: A device is removed
[ YYYY    ] pseudo-char-device pcdev-C1x.2: A device is removed
[ YYYY    ] pseudo-char-device pcdev-B1x.1: A device is removed
[ YYYY    ] pseudo-char-device pcdev-A1x.0: A device is removed
[ YYYY    ] pseudo-char-device pcdev-4: A device is removed
[ YYYY    ] pseudo-char-device pcdev-3: A device is removed
[ YYYY    ] pseudo-char-device pcdev-2: A device is removed
[ YYYY    ] pseudo-char-device pcdev-1: A device is removed
[ YYYY    ] pcd_platform_driver_cleanup : pcd platform driver unloaded

# /home/debian/drivers# insmod pcd_platform_driver_dt.ko
[ XXXX    ] pseudo-char-device pcdev-1: A device is detected
[ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEV1ABC123
[ XXXX    ] pcd_platform_driver_proble : Device size = 512
[ XXXX    ] pcd_platform_driver_proble : Device permission = 17
[ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
[ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
[ XXXX    ] pseudo-char-device pcdev-1: Probe was successful

[ XXXX    ] pseudo-char-device pcdev-1: A device is detected
[ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEV2ABC456
   ...

[ XXXX    ] pseudo-char-device pcdev-A1x.0: A device is detected
[ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVABC111
[ XXXX    ] pcd_platform_driver_proble : Device size = 512
[ XXXX    ] pcd_platform_driver_proble : Device permission = 17
[ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
[ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
[ XXXX    ] pseudo-char-device pcdev-A1x.0: Probe was successful
```

```
            .of_match_table = of_match_ptr(org_pcdev_dt_match)  <= !!
        }
    };


    struct of_device_id org_pcdev_dt_match[] =
    {
        {.compatible = "pcdev-A1x",.data = (void*)PCDEVA1X},
        {.compatible = "pcdev-B1x",.data = (void*)PCDEVB1X},
        ...
    };

    int pcd_platform_driver_probe(struct platform_device *pdev)
    {
        struct device *dev = &pdev->dev;
        ...
        /* used to store matched entry of 'of_device_id' list of this driver */
        const struct of_device_id *match;

        /*match will always be NULL if LINUX doesnt support device tree i.e CONFIG_OF is off */
        match = of_match_device(of_match_ptr(org_pcdev_dt_match), dev);     <= drivers/of/device.c

        if(match){  // device tree
            ...
        }else{      // device setup
            ...
        }
```

9 device tree overlays 54m 317

103. Introduction to device tree overlays

DT overlays are device tree patches(dtbo) which are used to patch
or modify the existing main device tree blob(dtb)

am335x-boneblack.dtb
(This is board specific. This explains the hardware topology of the board)

The main dtb doesn’t include the hardware details to configure the cape
(device nodes, properties, pin configs)

Two ways you can include the device nodes for the cape device in the main dtb
1.Edit the main dtb itself (not recommended)
2.Overlay(a Patch which overlays the main dtb) (recommended)

Uses of overlays

```
[ XXXX    ] pseudo-char-device pcdev-B1x.1: A device is detected
[ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVXYZ222
   ...
```

102. Pcd platform driver DT coding part-4 314

CONFIG_OF configuration item
- In Linux, CONFIG_OF configuration item decides Device Tree and Open Firmware support
- If CONFIG_OF is not enabled, Linux doesn’t support hardware enumeration via device tree.
- All device tree processing functions which begin with of_* will be excluded from the kernel build
- For latest kernel this configuration item is enabled by default

$ vi linux/.config

```
CONFIG_OF=y                # device tree supported     make menu_config to change
```

Include/linux/of.h

```
// if CONFIG_OF enabled
#define of_match_ptr(_ptr) (_ptr)

// if CONFIG_OF disabled
#define of_match_ptr(_ptr) NULL
```

drivers/of/base.c

```
  #ifdef CONFIG_OF
  static inline struct devce_node *of_get_...(..)
  {
    ...
  }

  #else
  static inline struct devce_node *of_get_...(..)
  {
    return NULL;
  }
  #endif
```

```
struct platform_driver pcd_platform_driver =
{
    .probe = pcd_platform_driver_probe,
    .remove = pcd_platform_driver_remove,
    .id_table = pcdevs_ids,
    .driver = {
        .name = "pseudo-char-device",
```

1)To support and manage hardware configuration
(properties, nodes, pin configurations) of various capes of the board
2)To alter the properties of already existing device nodes of the main dtb
3)Overlays approach maintains modularity and makes capes management easier

ex.
I2c-touch-lcd.dts(this is overlay source file)
I2c-touch-lcd.dtbo(.dtbo indicates that this is a overlay to use with this LCD

Overlay DTS Format
- Refer : https://www.kernel.org/doc/Documentation/devicetree/overlaynotes.txt

Overlay DTS Format

The DTS of an overlay should have the following format:

```
{
    /* ignored properties by the overlay */

    fragment@0 {    /* first child node */

        /* phandle target of the overlay */
        target=<phandle> [, <phandle>, ...];
    or
        /* target path of the overlay */
        target-path="/path" [ , "/path", ...];

        __overlay__ {
            property-a; /* add property-a to the target */              !!!!
            node-a {    /* add to an existing, or create a node-a */    !!!!
                ...                                                     !!!!
            };
        };
    }

    fragment@1 {    /* second child node */
        ...
    };
    /* more fragments follow */
}
```

104. Device tree overlays exercise 323

DT Overlay exercise
Write a device tree overlay to disable/modify pcdev device nodes from the main dts

```
STEPS
  1. Create a device tree overlay file and add fragments to modify the target device nodes
  2. Compile the device tree overlay file to generate .dtbo file (Device tree overlay binary)
  3. Make u-boot to load the .dtbo file during board start-up

Overlay compilation
• Make sure that device tree compiler(dtc) is installed on your system
• Run the below command to generate .dtbo from .dts file
    dtc -O dtb -o <output-file-name> -I <input-file-name>

$ vi am335x-boneblack-lddcrs.dtsi

    / {
        pcdev1: pcdev-1 {
            compatible = "pcdev-E1x","pcdev-A1x";
            org,size = <512>;
            org,device-serial-num = "PCDEV1ABC123";
            org,perm = <0x11>;
        };

        ...
    };

- devicetree spec 2.3.4 status

    device is enabled by default if status property is missing


/dts-v1/;
/plugin/;

/{
    fragment@0 {
        target = <&pcdev1>;
        __overlay__ {
            status = "disabled";
        };
    };

    fragment@2 {
        target = <&pcdev3>;
        __overlay__ {
            org,size = <1048>;
            org,device-serial-num = "PCDEV4XXXXXX";
        };
    };
};


$ sudo apt install device-tree-compiler



        tag v2019.04 by Tom Rini    3c991644
        https://source.denx.de/u-boot/u-boot/-/tree/3c99166441bf3ea325af2da83cfe65430b49c066


105. Updating u-boot

    $ make ARCH=arm am335x_evm_defconfig
    #
    # configuration written to .config
    #
    $ make ARCH=arm menuconfig

    (2) delay in seconds before automatically booting

    $ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4


106. Updating u-boot contd.

    MKIMAGE MLO                             <=
    MKIMAGE MLO.byteswap
    CFGCHK  u-boot.cfg

    $ ls u-boot*
    u-boot      u-boot.cfg          u-boot.dtb      u-boot-dtb.img  u-boot.lds  u-boot-nodtb.bin  u-boot.sym
    u-boot.bin  u-boot.cfg.configs  u-boot-dtb.bin  u-boot.img      u-boot.map  u-boot.srec
                                                    ^^^^^^^^^^
    $ scp MLO u-boot.img debian@192.168.7.2:/home/debian/drivers

        >> in BBB

        $ sudo -s
        # mount /dev/mmcblk0p1 /mnt/
        # ls /mnt/
        am3335x-boneblack.dtb   MLO     u-boot.img      uEnv.txt    uImage      vmlinux     ...

        # cp /home/debian/drivers/MLO .
        # cp /home/debian/drivers/u-boot.img .
        # sync
        # reboot


107. Applying overlays using u-boot commands

    u-boot/doc$ vi README.fdt-overlays

        Manually Loading and Applying Overlays
        --------------------------------------

        1. Figure out where to place both the base device tree blob and the

$ dtc -@ -I dtb -o PCDEV0.dtbo PCDEV0.dts

    -@ : this creates some internal symbol table.

$ ls
PCDEV0.dtbo     PCDEV0.dts

$ scp PCDEV0.dtbo debian@192.168.x.x:/home/debian/drivers   // xfer to BBB

    >> in BBB

    ~/drivers$ sudo cp PCDEV0.dtbo  /lib/firmware/

    $ ls /lib/firmware/PCDEV*
    PCDEV0.dtbo     PCDEV-1.dtbo    PCDEV.dtbo

~/workpace/ldd/source/linux_bbb_5$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am335x-boneblack.dtb
~/workpace/ldd/custom_drivers/005_pcd_platform_driver_dt$ sudo make copy-dtb

    >> in BBB

    ~/drivers# ls

    am335x-boneblack.dtb    lcd_driver.c        MLO
    armmem                  lcd_driver.h        pcd_device_setup.ko
    bcopy.sh                lcd.h               PCDEV0.dtbo
    copy.sh                 lcd.o               PCDEV-1.dtbo
    ...

    ~/drivers# vi copy.sh
    umount /tmp
    mount /dev/mmcblk0p1 /tmp
    cp am335x-boneblack.dtb /tmp
    sync
    umount /tmp
    echo 4 > /proc/sys/kernel/printk

    ~/drivers# ./copy.sh            <= copy device tree binary to the root partition



    scott@host:~/workspace/src/u-boot_2017_05_rc2/u-boot$ git log
    commit f6c1df44b815a08585e7fd3805a1db51a5955d09 (HEAD, tag: v2017.05-rc2)
    Author: Tom Rini <trini@konsulko.com>
    Date:   Mon Apr 17 18:16:49 2017 -0400

    =>

    https://source.denx.de/u-boot/u-boot



    overlay. Make sure you have enough space to grow the base tree without
    overlapping anything.
      => setenv fdtaddr 0x87f00000
      => setenv fdtovaddr 0x87fc0000

    2. Load the base blob and overlay blobs
      => load ${devtype} ${bootpart} ${fdtaddr} ${bootdir}/base.dtb
      => load ${devtype} ${bootpart} ${fdtovaddr} ${bootdir}/overlay.dtb

    3. Set it as the working fdt tree.
      => fdt addr $fdtaddr

    4. Grow it enough so it can 'fit' all the applied overlays
      => fdt resize 8192

    5. You are now ready to apply the overlay.
      => fdt apply $fdtovaddr

    6. Boot system like you would do with a traditional dtb.
      For bootm:    => bootm ${kerneladdr} - ${fdtaddr}
      For bootz:    => bootz ${kerneladdr} - ${fdtaddr}


in u-boot shell:

  => setenv fdtaddr 0x87f00000
  => setenv fdtovaddr 0x87fc0000
  => load mmc 0:1 ${fdtaddr} am335x-boneblack.dtb
  94529 bytes read in 10 ms (9 MiB/s)
  => load mmc 0:2 ${fdtovaddr} /lib/firmware/PCDEV0.dtbo
  526 bytes read in 11 ms (45.9 KiB/s)

    . mmc 0:1 boot partition
    . mmc 0:2 rfs partition

  => fdt addr $fdtaddr      <- final dt can be found here
  => fdt resize 8192
  => fdt apply $fdtovaddr
  => setenv bootargs console=ttyO0, 115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext4 rootwait earlyprintk mem=512M
  => load mmc 0:1 ${loadaddr} uImage
  => bootm ${loadaddr} - ${fdtaddr}

  $ ls /sys/devices/platform
  pcdev-3   pcdev-4     ....            -> pcdev-1 & 2 are disabled by default via dt overlay

  $ sudo insmod pcd_platform_driver_dt.ko
  [ XXXX.    ] pseudo-char-device pcdev-3 : A device is detected
  [ XXXX.    ] pcd_platform_driver_probe : Device serial number = PCDEV4XXXXXX  <= changed via overlay
```

108. Applying overlays and testing using u-boot uEnv.txt file

uEnv-dtbo.txt

```
console=ttyS0,115200n8

overlays=PCDEV0.dtbo PCDEV1.dtbo
dtb=am335x-boneblack.dtb
dtbopath=/lib/firmware

ovenvsetup=setenv fdtaddr 0x87f00000;setenv fdtovaddr 0x87fc0000;
fdtload=load mmc 0:1 ${fdtaddr} ${dtb};
fdtcmd=fdt addr $fdtaddr;fdt resize 8192;
fdtovload=for i in ${overlays};
          do echo Applying overlay...;
          load mmc 0:2 ${fdtovaddr} ${dtbopath}/${i}; fdt apply $fdtovaddr;
          done

bootsettings=setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext4 rootwait earlyprintk mem=512M
mmcboot= run ovenvsetup ; run fdtload; run fdtcmd; run fdtovload;echo Booting from microSD ...; setenv autoload no ; load mmc
0:1 ${loadaddr} uImage ; run bootsettings ; bootm ${loadaddr} - ${fdtaddr}
uenvcmd=run mmcboot
```

```
$ scp PCDEV1.dtbo debian@192.168.7.2:/home/debian/drivers
$ scp uEnv-dtbo.txt debian@192.168.7.2:/home/debian/drivers
```

```
    >> in BBB

    ~drivers$ sudo  mount /dev/mmcblk0p1 /mnt/
    ~drivers$ sudo cp uEnv-dtbo.txt /mnt/uEnv.txt
    ~drivers$ sync
    ~drivers$ reboot

    ...
    Apllying overlay
    526 bytes read in 11 ms (45.9 KiB/s)
    Apllying overlay
    229 bytes read in 23 ms (8.8 KiB/s)
    Booting from SD
    debian@beaglebone:~$ ls /sys/devices/platform/
    ...     pcdev-4     ...         1 ~ 3 are disabled by overlays

    debian@beaglebone:~$ sudo insmod pcd_platform_driver_dt.ko
    [ XXXX    ] pseudo-char-device pcdev-1: A device is detected
    [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEV1ABC123
    [ XXXX    ] pcd_platform_driver_proble : Device size = 512
    [ XXXX    ] pcd_platform_driver_proble : Device permission = 17
```

```
    struct device_private   *p;

    const char      *init_name; /* initial name of the device */
    const struct device_type *type;

    const struct bus_type   *bus;   /* type of bus device is on */
    struct device_driver *driver;   /* which driver has allocated this
                       device */
    void        *platform_data; /* Platform specific data, device
                       core doesn't touch it */
    void        *driver_data;   /* Driver data, set and get with
                       dev_set_drvdata/dev_get_drvdata */
#ifdef CONFIG_PROVE_LOCKING
    struct mutex    lockdep_mutex;
#endif
    struct mutex        mutex;  /* mutex to synchronize calls to
                       * its driver.
                       */
};
```

Include/linux/device.has

• At the lowest level, every device in a Linux system is represented by an instance of struct device

• The device structure contains the information that the device model core needs to model the system

• Most subsystems, however, track additional information about the devices they host. As a result, it is rare for devices to be represented by bare device structures; instead, that structure, like kobject structures, is usually embedded within a higher level representation of the device

```
struct platform_device {
    const char  *name;
    int     id;
    bool        id_auto;
    struct device   dev;
    u64     platform_dma_mask;
    struct device_dma_parameters dma_parms;
    u32     num_resources;
    struct resource *resource;

    const struct platform_device_id *id_entry;

    /*
    * Driver name to force a match. Do not set directly, because core
    * frees it.  Use driver_set_override() to set or clear it.
```

```
    [ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
    [ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
    [ XXXX    ] pseudo-char-device pcdev-1: Probe was successful
    [ XXXX    ] pcd_platform_driver_init : pcd platform driver loaded
```

10 Linux device driver model 1h32m 341

109. Linux device model

• Linux device model is nothing but a collection of various data structures, and helper functions that provide a unifying and hierarchical view of all the busses, devices, drivers present on the system. You can access the whole Linux device and driver model through a pseudo filesystem called sysfs. Which is mounted at /sysfs.
• Different components of the Linux device model is represented as files and directories through sysfs
• Sysfs exposes underlying bus, device, and driver details and their relationships in the Linux device model.

• Device ----- struct device
• Device driver ----- struct device_driver
• Bus ----- struct bus_type
• Kobject ----- struct kobject
• Ksets ----- struct kset
• Kobject type ----- struct kobj_type

• Definition of a device
  • Under Linux device driver model, anything which can be represented by an instance of the data structure struct device is a device

• Definition of a driver
  • Anything which can be represented by an instance of the data structure struct device_driver is a driver

Example : Consider the case of a platform device

CPU -- platform bus -- platform device(ADC)

It’s a device and it’s device type is “ platform device” because it is hanging on platform bus type

```
struct device {
    struct kobject kobj;
    struct device       *parent;
```

```
    */
    const char *driver_override;

    /* MFD cell pointer */
    struct mfd_cell *mfd_cell;

    /* arch specific additions */
    struct pdev_archdata    archdata;
};
```

Struct platform_device
  +-- subsystem specific information
  +-- struct device
        +-- device specific information
              • parent
              • associated device tree node
              • bus_type
              • device_driver
              • driver_data
              • platform_data

CPU -- I2C bus -- I2C client device(RTC)

It’s a device and it’s device type is “i2c client” because it is hanging on i2c bus type.

```
    struct i2c_client {
        unsigned short flags;               /* div., see below                */
        #define I2C_CLIENT_PEC      0x04    /* Use Packet Error Checking      */
        #define I2C_CLIENT_TEN      0x10    /* we have a ten bit chip address */
                                            /* Must equal I2C_M_TEN below     */
        #define I2C_CLIENT_SLAVE    0x20    /* we are the slave               */
        #define I2C_CLIENT_HOST_NOTIFY 0x40 /* We want to use I2C host notify */
        #define I2C_CLIENT_WAKE     0x80    /* for board_info; true iff can wake*/
        #define I2C_CLIENT_SCCB     0x9000  /* Use Omnivision SCCB protocol   */
                                            /* Must match I2C_M_STOP|IGNORE_NAK */
        unsigned short addr;                /* chip address - NOTE: 7bit      */
                                            /* addresses are stored in the    */
                                            /* _LOWER_ 7 bits                 */
        char name[I2C_NAME_SIZE];
        struct i2c_adapter *adapter;        /* the adapter we sit on          */
        struct device dev;                  /* the device structure           */  !
        int init_irq;                       /* irq set at initialization      */
        int irq;                            /* irq issued by device           */
        struct list_head detected;
      #if IS_ENABLED(CONFIG_I2C_SLAVE)
        i2c_slave_cb_t slave_cb;            /* callback for slave mode        */
      #endif
```

```
    void *devres_group_id;              /* ID of probe devres group         */
};
```

```
                        CPU
                         |
<----------------- platform bus ------------------>
                         |
               I2C host controller (on chip)           ; struct platform_device
                         |
<-------------------- I2C bus -------------------->
          |
 I2C client device(RTC)  ; struct i2c_client
    (off chip device)
```

-> It’s a device and it’s device type is “i2c client”
because it is hanging on i2c bus type.

2 drivers:
. struct platform_driver
+-- platform specific information
+-- struct device_driver
device specific information

. struct i2c_driver
+-- I2C specific information
+-- struct device_driver
device specific information

```
struct platform_driver {
    int (*probe)(struct platform_device *);
    int (*remove)(struct platform_device *);
    void (*remove_new)(struct platform_device *);
    void (*shutdown)(struct platform_device *);
    int (*suspend)(struct platform_device *, pm_message_t state);
    int (*resume)(struct platform_device *);
    struct device_driver driver;                                    !
    const struct platform_device_id *id_table;
    bool prevent_deferred_probe;
    bool driver_managed_dma;
};
```

```
struct i2c_driver {
    unsigned int class;
    int (*probe)(struct i2c_client *client);
    void (*remove)(struct i2c_client *client);
    void (*shutdown)(struct i2c_client *client);
    void (*alert)(struct i2c_client *client, enum i2c_alert_protocol protocol,
            unsigned int data);
```

```
    .probe  = ds1347_probe,
};

module_i2c_driver(ds1347_driver);
```

110. kobjects 355

Kobject

- Kobject stands for kernel object which is represented by struct kobject
- Kobjects are a fundamental building block of Linux device and driver hierarchy
- Kobjects are used to represent the ‘containers’ in the sysfs virtual filesystem
- Kobjects are also used for reference counting of the ‘containers’.
- It has got its name, type, and parent pointer to weave the Linux device and driver hierarchy
- Using kobjects you can add attributes to the container,
  which can be viewed/altered by the user space.
- The sysfs filesystem gets populated because of kobjects, sysfs is a user
  space representation of the kobjects hierarchy inside the kernel

```
struct kobject {
    const char              *name;              struct kobj_type {
    struct list_head        entry;                  void (*release)(struct kobject *kobj);
    struct kobject          *parent;                const struct sysfs_ops *sysfs_ops;
    struct kset             *kset;                  const struct attribute_group **default_groups;
    const struct kobj_type  *ktype;     ------> ...
    /* sysfs directory entry */                 };
    struct kernfs_node      *sd;
    struct kref             kref;       ------> struct kref {
    unsigned int state_initialized:1;               refcount_t refcount;
    unsigned int state_in_sysfs:1;              };
    unsigned int state_add_uevent_sent:1;
    unsigned int state_remove_uevent_sent:1;
    unsigned int uevent_suppress:1;
#ifdef CONFIG_DEBUG_KOBJECT_RELEASE
    struct delayed_work     release;
#endif
};
```

What are containers ?

- Kobjects are rarely or never used as stand-alone kernel objects.
  Most of the time, they are embedded in some other structure
  that we call container structure, which describes the device diver model’s components.
- Some example of container structure could be
  struct bus,
  struct device,
  struct device_driver

```
    int (*command)(struct i2c_client *client, unsigned int cmd, void *arg);
    struct device_driver driver;                                    !
    const struct i2c_device_id *id_table;
    int (*detect)(struct i2c_client *client, struct i2c_board_info *info);
    const unsigned short *address_list;
    struct list_head clients;
    u32 flags;
};
```

linux/drivers/i2c/buses/i2c-omap.c

```
static struct platform_driver omap_i2c_driver = {
  .probe  = omap_i2c_probe,
  .remove = omap_i2c_remove,
  .driver = {
      .name = "omap_i2c",
      .pm = &omap_i2c_pm_ops,
      .of_match_table = of_match_ptr(omap_i2c_of_match),
  },
};

static int __init omap_i2c_init_driver()
{
  return platform_driver_register(&omap_i2c_driver);
}
```

linux/drivers/rtc/rtc-ds1374.c

```
static struct i2c_driver ds1374_driver = {
  .driver = {
      .name = "rtc-ds1374",
      .of_match_table = of_match_ptr(ds1374_of_match),
      .pm = &ds1374_pm,
  },
  .probe  = ds1374_probe,
  .remove = ds1374_remove,
  .id_table = ds1374_id,
};

module_i2c_driver(ds1374_driver);
```

linux/drivers/rtc/rtc-ds1347.c

```
static struct spi_driver ds1347_driver = {
  .driver = {
      .name = "rtc-ds1347",
  },
```

- The container structure's embedded kobject enables the container
  to become part of an object hierarchy.

```
Struct platform_device          <-- device type
  +-- struct device             <-- container
        +-- kobj                <-- embedded kobject
            • kref              <-- reference counter
```

111. kobject type and kset 361

- Type of a kobject is determined based on,
  type of the container in which the kobject is embedded

struct kobj_type (ktype)

Now, the type of a kobject is controlled using the structure called struct kobj_type

This structure is used to define the default behavior(e.g. attributes)
for a group of kobjects of same container type

Struct kobj_type
- struct kobj_type object or simply ktype object is an object
  which defines the behavior for the container object.
- Behaviors are manifested in terms of attributes and file operation methods
  that handle those attributes.
- The ktype also controls what happens to the kobject when it is created and destroyed.
- Every structure that embeds a kobject needs a corresponding ktype.

Instead of each kobject defining its own behavior, the behavior is stored in a ktype,
and kobjects of the same “type” point at the same ktype structure,
thus sharing the same behavior.

If you want to assign a type to your kobject, then you have to create
an object of type ‘struct kobj_type’ and initialize the ‘ktype’ field of the kobject

each kobject doesn't have its own release method.
release method provided by kobj_type object.

```
struct kobj_type {
    void (*release)(struct kobject *kobj);
    const struct sysfs_ops *sysfs_ops;
    struct attribute **default_attrs;                   // obsolete
    const struct attribute_group **default_groups;
...
};
```

```
void my_object_release(struct kobject *kobj)
{
    struct my_object *mine = container_of(kobj, struct my_object, kobj);
    /* Perform any additional cleanup on this object, then... */
    kfree(mine);
}
```

- sysfs_ops
  it points to sys operation structure which lists the methods
  to operate on the default attributes created for the kobject of this ktype

- default_attrs pointer
  a list of default attributes that will be automatically
  created for any kobject that is registered with this ktype

Kset 369

- Kset as its name indicates it's a set of kobjects of same type
  and belongs to a specific subsystem.

- Basically kset is a higher-level structure which 'collects'
  all lower level kobjects which belong to same type
  -> top level container class for kobjects

```
struct kset {
    struct list_head list;      <-- The list field is the head of the doubly linked
    spinlock_t list_lock;           circular list of kobjects included in the kset
    struct kobject kobj;
    const struct kset_uevent_ops *uevent_ops;
} __randomize_layout;
```

```
                          The directory associated with a kobject always
          [kobject]      / appears in the directory of the parent kobject
           +- parent   <-+
kset object     |
    +-- [kobj]  |         [kobject]
          +-- parent  ----   +- parent
```

Linux device driver model hierarchy 371

```
scott@host:~$ ls /sys/devices/platform      => 'devices' is subsystem. 'platform' is kset.
 eisa.0              kgdboc                   reg-dummy
 'Fixed MDIO bus.0'  pcspkr                   rtc_cmos
 i8042               platform-framebuffer.0   serial8250
 intel_rapl_msr.0    power                    uevent

 => 'eisa.0, kgdboc, reg-dummy, and so on' are kobjects of the same type.
```

```
  drwxr-xr-x 3 root root    0 May 21 13:04 i2c-dev
  -r--r--r-- 1 root root 4096 May 21 13:04 name
  --w------- 1 root root 4096 May 21 13:04 new_device
  drwxr-xr-x 2 root root    0 May 21 13:04 power
  lrwxrwxrwx 1 root root    0 May 21 13:04 subsystem -> ../../../../bus/i2c
  -rw-r--r-- 1 root root 4096 May 21 13:04 uevent
```

```
scott@host:/sys/bus$ tree i2c           => subsystem
  i2c                                       => bus_type (subsystem)
  ├── devices                               => kset
  │   └── i2c-0 -> ../../../devices/pci0000:00/0000:00:07.0/i2c-0
  │                   ; kobject (client devices)
  ├── drivers                           => kset
  │   ├── 88PM860x                          => kobject
  │   │   ├── bind                              . attribute
  │   │   ├── uevent                            . attribute
  │   │   └── unbind                            . attribute
  │   ├── aat2870
  │   │   └── uevent
```

```
scott@host:/sys/bus$ tree i2c
  i2c
  ├── devices                   <- devices attached to an i2c bus type
  ├── drivers                   <- drivers attached to an i2c bus type
```

112. sysfs and kobject attributes 378

Sysfs is a virtual in-memory file system which provides
1. Representation of the kobject hierarchy of the kernel.
2. The complete topology of the devices and drivers of the system
   in terms of directories and attributes.
3. Attributes to help user space application
   to interact with devices and drivers
4. Standard methods to access devices using 'classes'

Using sysfs
- sysfs is always compiled in if CONFIG_SYSFS is defined
- You can access it by doing mount -t sysfs sysfs /sys
- Documentation :
  https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

```
/sys/class/pcd_class$ ls
 pcdev-0    pcdev-1     pcdev-2     pcdev-3
```

```
scott@host:/sys/bus$ ls -l platform/
 total 0
 drwxr-xr-x  2 root root    0 May 21 12:56 devices
 drwxr-xr-x 47 root root    0 May 21 12:56 drivers
 -rw-r--r--  1 root root 4096 May 21 12:56 drivers_autoprobe
 --w-------  1 root root 4096 May 21 12:56 drivers_probe
 --w-------  1 root root 4096 May 21 12:56 uevent
```

```
Struct platform_device          <-- device type
   +-- struct device            <-- container
         +-- kobj               <-- embedded kobject
            • kref              <-- reference counter
```

```
scott@host:~$ ls -l /sys/devices/platform/i8042
 total 0
 lrwxrwxrwx 1 root root    0 May 21 09:53 driver -> ../../../bus/platform/drivers/i8042
 -rw-r--r-- 1 root root 4096 May 21 09:53 driver_override
 -r--r--r-- 1 root root 4096 May 21 09:53 modalias
 drwxr-xr-x 2 root root    0 May 21 09:53 power
 drwxr-xr-x 6 root root    0 May 21 09:53 serio0
 drwxr-xr-x 5 root root    0 May 21 09:53 serio1
 lrwxrwxrwx 1 root root    0 May 21 09:53 subsystem -> ../../../bus/platform
 -rw-r--r-- 1 root root 4096 May 21 09:53 uevent

    => all above are default attributes of a kobject which depend on kobj_type.
```

```
scott@host:~$ ls /sys/bus
 ac97         eisa          mipi-dsi     platform  usb
 acpi         event_source  mmc          pnp       virtio
 cec          gpio          nd           rapidio   vme
 clockevents  hid           node         scsi      workqueue
 clocksource  i2c           nvmem        sdio      xen
 container    isa           parport      serial    xen-backend
 cpu          machinecheck  pci          serio
 dax          mdio_bus      pci-epf      snd_seq
 edac         memory        pci_express  spi
```

```
scott@host:~$ ls /sys/bus/i2c
 devices  drivers  drivers_autoprobe  drivers_probe  uevent

scott@host:/sys/bus$ ls -l  i2c/devices/i2c-0/
  total 0
  --w------- 1 root root 4096 May 21 13:04 delete_device
  lrwxrwxrwx 1 root root    0 May 21 13:04 device -> ../../0000:00:07.0
```

```
/sys/class/pcd_class$ ls -l pcdev-0
  -r--r--r-- 1 root root 4096 May 21 13:04 dev
  lrwxrwxrwx 1 root root    0 May 21 13:04 device    -> ../../pcdev-A1x.0   ; parent device
  drwxr-xr-x 2 root root    0 May 21 13:04 power
  lrwxrwxrwx 1 root root    0 May 21 13:04 subsystem -> ../../../../class/pcd_class
  -rw-r--r-- 1 root root 4096 May 21 13:04 uevent

/sys/class/pcd_class$ cat pcdev-0/dev
  237:0
```

```
/sys/class/pcd_class/pcd$ cat uevent

  MAJOR=240
  MINOR=0
  DEVNAME=pcdev-0
```

. refer to #45

```
scott@host:/sys/class$ ll ata_device/dev1.0/power/
  total 0
  drwxr-xr-x 2 root root    0 May 21 19:28 ./
  drwxr-xr-x 3 root root    0 May 21 19:28 ../
  -rw-r--r-- 1 root root 4096 May 21 19:28 async
  -rw-r--r-- 1 root root 4096 May 21 19:28 autosuspend_delay_ms
  -rw-r--r-- 1 root root 4096 May 21 19:28 control
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_active_kids
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_active_time
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_enabled
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_status
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_suspended_time
  -r--r--r-- 1 root root 4096 May 21 19:28 runtime_usage

scott@host:/sys/class/ata_device/dev1.0/power/ cat runtime_active_kids
  0
```

Exercise 384

```
struct attribute
{
    const char *name; => This name will show in the sysfs kobject directory
                         as attribute's name

    umode_t mode;     => This controls the read/write permission
                         for the attribute file from user space programs
```

```
        #ifdef CONFIG_DEBUG_LOCK_ALLOC
            boolignore_lockdep:1;
            struct lock_class_key*key;
            struct lock_class_keyskey;
        #endif
    };
```

113. Creating sysfs attributes 385

Mode of an attribute
- S_IRUGO – world-read-only
- S_IRUSR – owner read-only
- S_IRUGO | S_IWUSR : world-read and only owner write

APIs for managing sysfs files (Attributes)

```
int sysfs_create_file(struct kobject *kobj, const struct attribute *attr);
void sysfs_remove_file(struct kobject *kobj, const struct attribute *attr);
int sysfs_create_groups(struct kobject *kobj, const struct attribute_group **groups);
int sysfs_create_group(struct kobject *kobj, const struct attribute_group *grp);
int sysfs_chmod_file(struct kobject *kobj, const struct attribute *attr, umode_t mode);
```

include/linux/sysfs.h

file I/O operations on sysfs attributes

- Once you create a sysfs files (attributes), you should provide read and write methods for them so that user can read value of the attribute or write a new value to the attribute .
- If you see struct attribute there is no place to hook read/write methods for an attribute. Basically it just represents name of a attribute and the mode

```
/* interface for exporting device attributes */
struct device_attribute
{
   struct attribute attr;
   ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
   ssize_t (*store)(struct device *dev, struct device_attribute *attr,
                                        const char *buf, size_t count);
};
```

Use this structure if your goal is to create attributes for a device and to provide show/store methods

```
int sysfs_create_file(struct kobject *kobj, const struct attribute *attr);
```

-> kobj from struct device, attr from DEVICE_ATTR macro.

116. pcd sysfs kernel module testing

```
$ insmod pcd_device_setup.ko
$ insmod pcd_sysfs.ko
   [ XXXX    ] pseudo-char-device pcdev-A1x.0: A device is detected
   [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVABC111
   [ XXXX    ] pcd_platform_driver_proble : Device size = 512
   [ XXXX    ] pcd_platform_driver_proble : Device permission = 17
   [ XXXX    ] pcd_platform_driver_proble : Config item 1 = 60
   [ XXXX    ] pcd_platform_driver_proble : Config item 2 = 21
   [ XXXX    ] pseudo-char-device pcdev-A1x.0: Probe was successful

   [ XXXX    ] pseudo-char-device pcdev-B1x.1: A device is detected
   [ XXXX    ] pcd_platform_driver_proble : Device serial number = PCDEVXYZ222
    ...
$ ls /sys/class/pcd_class
    pcdev-0     pcdev-1     pcdev-2     pcdev-3
$ ls /sys/class/pcd_class/pcdev-0
    dev     device      max_size    power   serial_num  subsystem       uevent
```

117. show and store methods of the sysfs attributes 393

Show and store methods
- Show method is used to export attribute value to the user space
- Store method is used to receive a new value from the user space for an attribute

Show method
Prototype:

```
ssize_t (*show)(struct device *dev, struct device_attribute *attr , char *buf);
```

This function is invoked when user space program tries to read the value of the attribute.

Here you must copy the value of the attribute in to the ‘buf’ pointer.

Note that ‘buf’ is not user level pointer .
Its provided by the kernel buffer so its size is limited to PAGE_SIZE .
For ARM architecture it is 4KB (4096 Bytes) long
So , while copying data to ‘buf’ pointer the size shouldn’t exceed 4096 bytes.
To copy data in to ‘buf’ you can either use sprintf() or scnprintf()

Show method

- include/linux/device.h

Creating device_attribute variables

- Instead of manually creating variables of struct device_attribute and initializing them, use the DEVICE_ATTR_XX macros given in include/linux/device.h

```
#define DEVICE_ATTR(_name, _mode, _show, _store) \
    struct device_attribute dev_attr_##_name = __ATTR(_name, _mode, _show, _store)
#define DEVICE_ATTR_PREALLOC(_name, _mode, _show, _store) \
    struct device_attribute dev_attr_##_name = \
        __ATTR_PREALLOC(_name, _mode, _show, _store)
#define DEVICE_ATTR_RW(_name) \
    struct device_attribute dev_attr_##_name = __ATTR_RW(_name)

#define DEVICE_ATTR_RO(_name) \
    struct device_attribute dev_attr_##_name = __ATTR_RO(_name)
#define DEVICE_ATTR_WO(_name) \
    struct device_attribute dev_attr_##_name = __ATTR_WO(_name)
```

Usage

```
static struct device_attribute dev_attr_bar = {
    .attr = {
        .name = “bar",
        .mode = S_IWUSR | S_IRUGO,
    },
    .show = show_bar,
    .store = store_bar,
};
```

is equivalent to doing:

```
static DEVICE_ATTR(bar, S_IWUSR | S_IRUGO, show_bar, store_bar);
```

114. pcd_sysfs_attributes coding part 1
115. pcd_sysfs_attributes coding part 2

linux/include/linux/device.h

```
ssize_t (*show)(struct device *dev, struct device_attribute *attr, char *buf);
ssize_t (*store)(struct device *dev, struct device_attribute *attr,
                                        const char *buf, size_t count);
```

create sysfs attribute in the sysfs directory
=> after creating device file, device_creat();

linux/fs/sysfs
linux/include/linux/sysfs.h

Difference between drivers read method and drivers show method

1)Read method is used by the user space to read large amounts of data from the driver .
2)Show method is used for reading a single value data or an array of similar values or data whose length is less than PAGE_SIZE
3)Use show method to read any configuration data of your driver or device.

=> use READ if it's over 4KB.

Return value of show method :
show() methods should return the number of bytes copied into the buffer or an error code

Store method

```
ssize_t (*store)(struct device *dev, struct device_attribute *attr, \
                                        const char *buf, size_t count);
```

This is invoked when user wants to modify the value of the sysfs file .
In this method , ‘buf’ points to the user data.
'count' parameter is the amount of user data being passed in.

The maximum amount of data which the ‘buf’ pointer can carry is limited PAGE_SIZE
The data carried by the ‘buf’ is NULL terminated .

store() should return the number of bytes used from the buffer.
If the entire buffer has been used, just return the count argument.

118. pcd_sysfs_attributes coding part 3

```
struct pcdev_private_data *dev_data = dev_get_drvdata(dev->parent);

static inline void *dev_get_drvdata(const struct device *dev)
{
    return dev->driver_data;
}

static inline void dev_set_drvdata(struct device *dev, void *data)
{
    dev->driver_data = data;
}
```

=> how to get dev ?

```
struct device *device_create(const struct class *class, struct device *parent, \
                dev_t devt, void *drvdata, const char *fmt, ...)       ^^^^^^

int pcd_platform_driver_probe(struct platform_device *pdev)
{
```

```
        ...
        struct device *dev = &pdev->dev;

        ...
        dev_set_drvdata(&pdev->dev, dev_data);  // &pdev->dev == dev

        pcdrv_data.device_pcd = device_create(pcdrv_data.class_pcd, dev, dev_data->dev_num,NULL,\
                                "pcdev-%d",pcdrv_data.total_devices);
        ...
    }

// TODO still don't get why it's parent. need to revisit


$ insmod pcd_device_setup.ko
$ insmod pcd_sysfs.ko
$ ls /sys/class/pcd_class
   pcdev-0     pcdev-1     pcdev-2     pcdev-3
$ ls /sys/class/pcd_class/pcdev-0
   dev     device      max_size    power   serial_num  subsystem       uevent
$ cat /sys/class/pcd_class/pcdev-3/cat serial_num
  PCDEVXYZ4444

   int kstrtol(const char *s, unsigned int base, long *res)

$ cat /sys/class/pcd_class/pcdev-3/cat max_size
  512

$ echo 100 > max_size
$ cat /sys/class/pcd_class/pcdev-3/cat max_size
  100
$ echo "hello world" > max_size
  bash: echo: write error: cannot allocate memroy
$ dmesg
  no space left on the devie
```

119. Attribute grouping 398

Attribute grouping

- Instead of calling sysfs_create_file to create a sysfs file for every attribute. You can finish it off in one call using attribute grouping
  int sysfs_create_group(struct kobject *kobj,const struct attribute_group *grp);

```
int sysfs_create_group(struct kobject *kobj, const struct attribute_group *grp);
```

122. Pad configuration register 420, 445

- Data sheet table 4.2

register details from TRM
pin details from datasheet

TRM 9.2.2.1
    default mode is 7

123. Linux GPIO subsystem 423

```
              consumers (any client drivers like foo, keypad)

                          |

gpio driver  ->  gpiolib (provides APIs)  <- gpiolib-sysfs driver
(led, keys)
                          |

                gpio controller driver
                  (struct gpio_chip)
                          |

                  gpio controller
```

```
gpio driver         : /sys/class/leds           drivers/led/leds-gpio.c
gpio driver         : /sys/class/input          drivers/input/keyboard/gpio_keys

gpiolib-sysfs driver: /sys/class/gpio/gpioN     drivers/gpio/gpiolib-sysfs.c

gpiolib             :                           drivers/gpio/gpiolib.c

gpio controllr driver:                          drivers/gpio/gpio-omap.c for AM335x
```

124. Consumer accessing GPIO pins

Consumer accessing GPIO pins

- Producer
GPIO controller
 GPIO_0[31:0]
- Consumer
Keypad driver
- Consumer dt node

```
include/linux/sysfs.h

struct attribute_group {
   const char          *name;
   umode_t (    *is_visible)(struct kobject *, struct     attribute *, int);
   umode_t (*is_bin_visible)(struct kobject *, struct bin_attribute *, int);
   struct attribute     **attrs;
   struct bin_attribute **bin_attrs;
};

struct attribute *pcd_attrs[] =
{
   &dev_attr_max_size.attr,
   &dev_attr_serial_num.attr,
   NULL
};

struct attribute_group pcd_attr_group =
{
  .attrs = pcd_attrs
};

int pcd_sysfs_create_files(struct device *pcd_dev)
{
  return sysfs_create_group(&pcd_dev->kobj, &pcd_attr_group);
}

$ insmod pcd_sysfs.ko
$ ls /sys/class/pcd_class
  dev device max_size power serial_num subsystem uevent
$ cat /sys/class/pcd_class/max_size
  512
```

11 Linux GPIO subsystem 32m 410

120. Introduction

121. GPIOs of BBB

Accessing GPIOs on BBB header

- TRM chap 25

- SRM 7.1 Expansion Connectors
  Table 12. Expansion Header P8 Pinout

- Data sheet 4.1.2 & table 4.2

Keypad dt node (7 pins)

- Platform specific data
Dt node for the keypad device

Representing GPIOs in consume dt node
  . GPIO number
  . GPIO controller number
  . Active high or active low status

- Keypad driver
The consumer driver accesses
the dt node to extract the information
about the gpois used to connect
the keypad and configures them

GPIO device tree property

GPIOs being consumed are represented by <function>-gpios property

- <function> being the purpose of this GPIO for the consumer
- Omitting <function > is OK but not recommended
- <function>-gpio is OK but not recommended
- So best practice is always use in the form : <function>-gpios

Documentation/gpio/board.txt

GPIOs mappings are defined in the consumer device's node, in a property named <function>-gpios, where <function> is the function the driver will request through gpiod_get(). For example:

```
   foo_device {                                          <= consumer node
           compatible = "acme,foo";                      <= consumer driver
           ...
           led-gpios = <&gpio 15 GPIO_ACTIVE_HIGH>, /* red */     <= consumer flags
                       <&gpio 16 GPIO_ACTIVE_HIGH>, /* green */
                       <&gpio 17 GPIO_ACTIVE_HIGH>; /* blue */

           power-gpios = <&gpio 1 GPIO_ACTIVE_LOW>;
   };
```

Properties named <function>-gpio are also considered valid and old bindings use it but are only supported for compatibility reasons and should not be used for newer bindings since it has been deprecated.

This property will make GPIOs 15, 16 and 17 available to the driver under the "led" function, and GPIO 1 as the "power" GPIO:

foo_device : A GPIO consumer dt node

led-gpios, power-gpios : Function name. Driver binding document tells you

```
                        what name to use. ‘Conn-id’

    <&gpio 15 GPIO_ACTIVE_HIGH>
        +- phandle to the GPIO controller to which this pin belongs to
        +- local offset to the GPIO line
        +- gpio flags


    GPIO consumer flags

        flags is defined to specify the following properties:

            * GPIO_ACTIVE_HIGH - GPIO line is active high
            * GPIO_ACTIVE_LOW - GPIO line is active low
            * GPIO_OPEN_DRAIN - GPIO line is set up as open drain
            * GPIO_OPEN_SOURCE - GPIO line is set up as open source
            * GPIO_PERSISTENT - GPIO line is persistent during suspend/resume and maintains its value
            * GPIO_TRANSITORY - GPIO line is transitory and may loose its electrical state during suspend/resum

        include/dt-bindings/gpio/gpio.h



12 GPIO sysfs driver implementation                2h

    125. Exercise GPIO Sysfs driver implementation                      439

        Exercise

          Write a GPIO sysfs driver.

            The goal of this exercise is to handle GPIOs of the hardware through Sysfs interface

          The driver should support the below functionality

          1) The driver should create a class "bone_gpios" under /sys/class (class_create)

          2) For every detected GPIO in the device tree, the driver should create a device
             under /sys/class/bone_gpios (device_create)

          3) the driver should also create 3 sysfs files(attributes) for every gpio device

            attribute 1) direction:
                        used to configure the gpio direction
                        possible values: ‘in’ and ‘out’
                        mode : (read /write)

            attribute 2) value:
                        used to enquire the state of the gpio or to write a new value to the gpio


        {
            &gpio_attr_group,
            NULL
        };

        int gpio_sysfs_probe(struct platform_device *pdev);
        int gpio_sysfs_remove(struct platform_device *pdev);

        struct of_device_id  gpio_device_match[] =
        {
            {.compatible = "org,bone-gpio-sysfs"},
            { }     // NULL terminated
        };

        struct platform_driver gpiosysfs_platform_driver =
        {
            .probe = gpio_sysfs_probe,
            .remove = gpio_sysfs_remove,
            .driver = {
                .name = "bone-gpio-syfs",
                .of_match_table = of_match_ptr(gpio_device_match)   // dt base
            }
        };

        int __init gpio_sysfs_init(void)
        {
            gpio_drv_data.class_gpio = class_create(THIS_MODULE,"bone_gpios");
            platform_driver_register(&gpiosysfs_platform_driver);
            return 0;
        }

        void __exit gpio_sysfs_exit(void)
        {
            platform_driver_unregister(&gpiosysfs_platform_driver);
            class_destroy(gpio_drv_data.class_gpio);
        }

        module_init(gpio_sysfs_init);
        module_exit(gpio_sysfs_exit);


    127. GPIO Sysfs driver implementation part-2

        . Table 12 of SRM shows that GPIO 66(pin 7) is gpio2.2.

           pin  proc    name        mode0           mode2       mode7
            7   R7      TIMER4      gpmc_advn_ale   timer4      gpio2[2]

        . Table 8 in 6.6 of SRM shows that LED USR1 is GPIO_2_22(PROC pin U15).
                possible values : 0 and 1 (read/write)

            attribute 3) label:
                        used to enquire label of the gpio (read-only )

          4) implement show and store methods for the attributes


        Device-Driver binding information

            The device tree for this driver must be created like below

            gpio_devs
            {
                compatible = "org,bone-gpio-sysfs";

                gpio_1 {    // child node 1
                    label = "gpio1.21";                         /* optional */
                    bone-gpios = <&gpio1 21 GPIO_ACTIVE_HIGH>;  /* mandatory */
                };

                gpio_2 {    // child node 2
                    label = "gpio1.22";
                    bone-gpios = <&gpio1 22 GPIO_ACTIVE_HIGH>;
                };
            }



    126. GPIO Sysfs driver implementation part-1

        static DEVICE_ATTR_RW(direction);
        static DEVICE_ATTR_RW(value);
        static DEVICE_ATTR_RO(label);

        static struct attribute *gpio_attrs[] =
        {
            &dev_attr_direction.attr,
            &dev_attr_value.attr,
            &dev_attr_label.attr,
            NULL
        };

        static struct attribute_group gpio_attr_group =
        {
            .attrs = gpio_attrs
        };

        static const struct attribute_group *gpio_attr_groups[] =


            LED   GPIO SIGNAL PROC PIN
            --------------------------
            USR0    GPIO1_21    V15
            USR1    GPIO1_22    U15
            USR2    GPIO1_23    T15
            USR3    GPIO1_24    V16



        $ vi arch/arm/boot/dts/am335x-boneblack-lddcrs.dtsi

            / {
                pcdev1: pcdev-1 {
                    compatible = "pcdev-E1x","pcdev-A1x";
                    org,size = <512>;
                    org,device-serial-num = "PCDEV1ABC123";
                    org,perm = <0x11>;
                };

                ...

                bone_gpio_devs {
                    compatible = "org,bone-gpio-sysfs";
                    gpio1 {
                        label = "gpio2.2";
                        bone-gpios = <&gpio2 2 GPIO_ACTIVE_HIGH>;
                    };

                    ...

                    gpio4 {
                        label = "usrled0:gpio1.21";
                        bone-gpios = <&gpio1 21 GPIO_ACTIVE_HIGH>;
                    };

                    gpio5 {
                        label = "usrled1:gpio1.22";
                        bone-gpios = <&gpio1 22 GPIO_ACTIVE_HIGH>;
                    };

                    gpio6 {
                        label = "usled2:gpio1.23";
                        bone-gpios = <&gpio1 23 GPIO_ACTIVE_HIGH>;
                    };


                };//bone_gpio_devs
            }; //root node
```

```
$ vi am33xx-l4.dtsi (line# 168)         // arch/arm/dts/am33xx-l4.dtsi (not in 4.14 but in 5.4)

    gpio0: gpio@0 {                 --> node for gpio0
        ...
    }

    gpio1: gpio@0 {                 --> node for gpio1
        ...
    }

    gpio2: gpio@0 {                 --> node for gpio2
        ...
    }
```

128. GPIO Sysfs driver implementation part-3

```
>> include/linux/of.h

#define for_each_child_of_node(parent, child)   \
    child = of_get_next_child(parent, child)

#define for_each_available_child_of_node(parent, child) \
    child = of_get_next_available_child(parent, child)

int gpio_sysfs_probe(struct platform_device *pdev)
{
    struct device *dev = &pdev->dev;
    struct device_node *parent = pdev->dev.of_node;
    struct device_node *child = NULL;

    struct gpiodev_private_data *dev_data;

    gpio_drv_data.total_devices = of_get_child_count(parent);
    gpio_drv_data.dev = devm_kzalloc(dev, sizeof(struct device *) * gpio_drv_data.total_devices , GFP_KERNEL);

    int i = 0;
    for_each_available_child_of_node(parent, child) {
        dev_data = devm_kzalloc(dev, sizeof(*dev_data), GFP_KERNEL);

        const char *name;
        if(of_property_read_string(child, "label", &name))
            snprintf(dev_data->label, sizeof(dev_data->label),"unkngpio%d",i);
        else
            strcpy(dev_data->label,name);
```

GPIO manipulations

- What you do with GPIO lines?
  - Configure its direction (input or output)
  - Change its output state to 0 or 1
  - Read input state
  - Configure output type (push-pull/open-drain)
  - Enable/Disable pull-up/pull-down resistors

Configuring direction and flags

Function	note
int gpiod_direction_input(struct gpio_desc *desc);	set the GPIO direction to input
int gpiod_direction_output(struct gpio_desc *desc, int value);	set the GPIO direction to output by taking initial output value of the GPIO
int gpiod_direction_output_raw (struct gpio_desc *desc, int value)	set the GPIO direction to output without regard for the ACTIVE_LOW status
int gpiod_configure_flags (struct gpio_desc *desc, const char *con_id, unsigned long lflags, enum gpiod_flags dflags)	helper function to configure a given GPIO

```
>> kernel.org/doc/Documentation/gpio/gpio.txt
```

130. GPIO Sysfs driver implementation part-5

```
    #define DEVICE_ATTR_RO(_name) \
        struct device_attribute dev_attr_##_name = __ATTR_RO(_name)
    #define DEVICE_ATTR_WO(_name) \
        struct device_attribute dev_attr_##_name = __ATTR_WO(_name)


    -> equivalent to :
       static DEVICE_ATTR_RO(...);


ssize_t direction_show(struct device *dev, struct device_attribute *attr,char *buf)
{
    return 0;
}

ssize_t direction_store(struct device *dev, struct device_attribute *attr,const char *buf, size_t count)
{
    return 0;
}

static DEVICE_ATTR_RW(direction);
static DEVICE_ATTR_RW(value);
static DEVICE_ATTR_RO(label);
```

131. GPIO Sysfs driver implementation part-6

```
gpio_drv_data.dev[i] = device_create_with_groups(gpio_drv_data.class_gpio, dev, 0, dev_data, gpio_attr_groups,\
                       dev_data->label);
```

132. GPIO Sysfs driver implementation part-7

```
ssize_t direction_show(struct device *dev, struct device_attribute *attr,char *buf)
{
    struct gpiodev_private_data *dev_data = dev_get_drvdata(dev);

    int dir;
    char *direction;

    dir = gpiod_get_direction(dev_data->desc);
    if(dir < 0) return dir;
    /* if dir = 0 , then show "out". if dir =1 , then show "in" */
    direction = (dir == 0) ? "out":"in";

    return sprintf(buf,"%s\n",direction);
}

ssize_t direction_store(struct device *dev, struct device_attribute *attr,const char *buf, size_t count)
{
    int ret;
    struct gpiodev_private_data *dev_data = dev_get_drvdata(dev);

    if(sysfs_streq(buf,"in") )
        ret = gpiod_direction_input(dev_data->desc);
    else if (sysfs_streq(buf,"out"))
        ret = gpiod_direction_output(dev_data->desc,0);
    else
        ret = -EINVAL;

    return ret ? : count;
}

bool sysfs_streq(const char *s1, const char *s2);
```

129. GPIO Sysfs driver implementation part-4

GPIO kernel functions to manage GPIOs

- The GPIO subsystem exposes many kernel function which the consumer driver can use to manage the GPIOs.
- Consumer driver should include linux/gpio/consumer.h header file where gpio manipulation function prototypes are available

Acquire and dispose a GPIO

```
>> drivers/gpio/gpiolib.c

Acquire
  struct gpio_desc* gpiod_get(struct device *dev, const char *con_id, \
                    enum gpiod_flags flags)

Dispose
  void gpiod_put(struct gpio_desc *desc)
```

. Device-managed variants of GPIO functions are also available.
  Check consumer.txt   -> devm_gpiod_get() in consumer.h/gpiolib-devres.c

. Since parent is referenced from child, alternative api is used.

```
struct gpio_desc *devm_fwnode_get_gpiod_from_child(struct device *dev,
                    const char *con_id,
                    struct fwnode_handle *child,    <= !!!!!
                    enum gpiod_flags flags,
                    const char *label);
```

```
    dev_data->desc = devm_fwnode_get_gpiod_from_child(dev, "bone", &child->fwnode,\
                        GPIOD_ASIS,dev_data->label);

    ret = gpiod_direction_output(dev_data->desc, 0);
    gpio_drv_data.dev[i] = device_create_with_groups(gpio_drv_data.class_gpio, dev, 0, dev_data, gpio_attr_groups,\
                           dev_data->label);
    i++;
}

    return 0;
}
```

```
ssize_t value_show(struct device *dev, struct device_attribute *attr,char *buf)
{
    struct gpiodev_private_data *dev_data = dev_get_drvdata(dev);
    int value;
    value = gpiod_get_value(dev_data->desc);
    return sprintf(buf,"%d\n",value);
}

ssize_t value_store(struct device *dev, struct device_attribute *attr,const char *buf, size_t count)
{

    struct gpiodev_private_data *dev_data = dev_get_drvdata(dev);
    int ret;
    long value;

    ret = kstrtol(buf,0,&value);
    if(ret)
        return ret;

    gpiod_set_value(dev_data->desc,value);

    return count;
}

ssize_t label_show(struct device *dev, struct device_attribute *attr,char *buf)
{
    struct gpiodev_private_data *dev_data = dev_get_drvdata(dev);
    return sprintf(buf, "%s\n", dev_data->label);
}
```

133. GPIO Sysfs driver implementation part-8

```
/*Driver private data structure */
struct gpiodrv_private_data
{
    int total_devices;
    struct class *class_gpio;
    struct device **dev;            <= to hold information till removing
};

int gpio_sysfs_remove(struct platform_device *pdev)
{
    int i;

    for(i = 0 ; i < gpio_drv_data.total_devices ; i++){
        device_unregister(gpio_drv_data.dev[i]);
    }
```

```
        led4 {
            label = "beaglebone:green:usr2";
            gpios = <&gpio1 23 GPIO_ACTIVE_HIGH>;
            linux, default-trigger = "cpu0";
            default-statue = "off";
        }; */

        led5 {
            label = "beaglebone:green:usr3";
            gpios = <&gpio1 24 GPIO_ACTIVE_HIGH>;
            linux, default-trigger = "mmc1";
            default-statue = "off";
        };
    };
```

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am3335x-boneblack.dtb
$ sudo make copy-dtb    // am335x-boneblack.dtb
$ make copy-drv         // gpio-sysfs.ko, gpiosysfs.ko
```

```
    >> in BBB

    ~/drivers# ls -l /sysdevice/platform/
      bone_gpio_devs/   pcdev-3/    ...
      leds/             pcdev-4/
      pcdev-1/          power/
      pcdev-2/          uevent

    ~/drivers# inmod gpio-sysfs.ko
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: Total devices found = 6
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.2
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.3
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.4
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.21
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.22
      [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.23
      [ XXXX.XXX ] gpio_sysfs_init : module load success
    ~/drivers# ls /sys/class/bone_gpios
      gpio2.2   gpio2.3   gpio2.4   usrled0:gpio1.21   usrled1:gpio1.22   usrled2:gpio1.23
    ~/drivers# ls /sys/class/bone_gpios/usrled1:gpio1.22
      device  direction  label  power   subsystem  uevent  value
    ~/drivers# cat /sys/class/bone_gpios/usrled1:gpio1.22/label
      usrled1:gpio1.22
    ~/drivers# cat /sys/class/bone_gpios/usrled1:gpio1.22/value
      0
    ~/drivers# cat /sys/class/bone_gpios/usrled1:gpio1.22/direction
      out
    ~/drivers# echo 1 > /sys/class/bone_gpios/usrled1:gpio1.22/value
    ~/drivers# echo 0 > /sys/class/bone_gpios/usrled1:gpio1.22/value
    ~/drivers# echo "in" > /sys/class/bone_gpios/usrled1:gpio1.22/direction
```

```
    return 0;
}
```

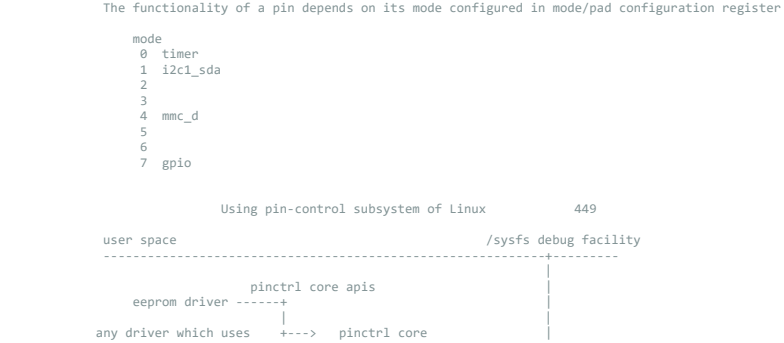
134. Gpio sysfs driver testing

```
$ vi am33xx-bone-common.dtsi

    &am33xx_pinmux {
        user_leds_s0: user_leds_s0 {
            pinctrl-single,pins = <
                AM33XX_PADCONF(AM335X_PIN_GPMC_A5, PIN_OUTPUT_PULLDOWN, MUX_MODE7)
                AM33XX_PADCONF(AM335X_PIN_GPMC_A6, PIN_OUTPUT_PULLUP, MUX_MODE7)
                AM33XX_PADCONF(AM335X_PIN_GPMC_A7, PIN_OUTPUT_PULLDOWN, MUX_MODE7)
                AM33XX_PADCONF(AM335X_PIN_GPMC_A8, PIN_OUTPUT_PULLUP, MUX_MODE7)
            >;
        };

        i2c0_leds: pinmux_i2c0_pins {
            pinctrl-single,pins = <
                AM33XX_PADCONF(AM335X_PIN_I2C0_SDA, PIN_OUTPUT_PULLUP, MUX_MODE0)
                AM33XX_PADCONF(AM335X_PIN_I2C0_SCL, PIN_OUTPUT_PULLUP, MUX_MODE0)
            >;
        };

    }

    ...

    leds {
        pinctrl-names = "default";
        pinctrl-0 = <&user_ldes_s0>;

        compatible = "gpio-leds";
    /*
        led2 {
            label = "beaglebone:green:usr0";
            gpios = <&gpio1 21 GPIO_ACTIVE_HIGH>;
            linux, default-trigger = "heartbeat";
            default-statue = "off";
        };

        led3 {
            label = "beaglebone:green:usr1";
            gpios = <&gpio1 22 GPIO_ACTIVE_HIGH>;
            linux, default-trigger = "mmc0";
            default-statue = "off";
        };
```

```
    struct device                           struct device_node

        // associated device tree node
    +- struct device_node *of_node          +-- struct fwnode_handle fwnode

       // firmware device node              *The function properly finds the corresponding
    +- struct fwnode_handle *fwnode;         GPIO using whatever is the * underlying firmware
                                             interface and then makes sure that the GPIO *
                                             descriptor is requested before it is returned
                                             to the caller

    @fwnode: firmware node containing GPIO refrernce
    @con_id: function within the GPIO consumer
```



```
    The functionality of a pin depends on its mode configured in mode/pad configuration register

        mode
         0  timer
         1  i2c1_sda
         2
         3
         4  mmc_d
         5
         6
         7  gpio
```

```
        pin ctrl dev    |           |

                          pin controller driver

kernel          pinctrl-single.c    |       vendor given pin
                generic device tree |       controller driver
        +---->  based pinctrl driver|
        |       (drivers/pinctrl/   |
        |        pinctrl-single.c)  |
        |                           |
--------+---------------------------+---------------------------------
        |                           |                             H/W
  pin configuration       pin cntrlr H/W     i2c2_sda
     dt code              (pad config reg)  <--------->  EEPROM
(pin controller node) <---    AM335X         <--------->     |
                                |            i2c2_scl       |
                                |                           |
                                +---------------------------+
                                            GPIO
```

• Pin control core
  • Pinctrl core implementation you can find in drivers/pinctrl/core.c
  • This provides pin control helper functions to any consumer drivers
  • Maintains pin exclusivity for a device
  • Provides debug interface to user space through sysfs

Pin control subsystem of linux
  • Pin control subsystem of Linux is used to configure pins of a uprocessor/ucontroller

  • What is pin configuration ?
    • Pin mode configuration for alternate functions
    • Slew rate adjustment
    • Driver strength
    • Pull up and pull down resistor enable/disable
    • Output type control (push pull, open drain)

  Device tree nodes
    1. Node which explains the pin controller
    2. Node which explains configuration details for individual pins
    3. Node which claims the pins (Client device node)

  Client device
    A Client device is nothing but any hardware module whose signals are
    affected by pin configuration. Again, each client device must be
    represented as a node in the device tree, just like any other hardware module.

```
        reg = <0x54>;
        #address-cells = <1>;
        #size-cells = <1>;
        cape0_data: cape_data@0 {
            reg = <0 0x100>;
        };
    };
```

pinctrl-single,pins property 458

  The pin configuration nodes for pinctrl-single are specified as pinctrl
  register offset and value pairs using pinctrl-single,pins. Only the bits
  specified in pinctrl-single,function-mask are updated. For example, setting
  a pin for a device could be done with:

```
        pinctrl-single,pins = <0xdc 0x118>;
```

  Where 0xdc is the offset from the pinctrl register base address for the
  device pinctrl register, and 0x118 contains the desired value of the
  pinctrl register. See the device example and static board pins example
  below for more information.

  Reference :
     Documentation/devicetree/bindings/pinctrl/pinctrl-single.txt

Pinctrl related properties to be used with client device nodes

  • For full discussion visit :
    Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt

  pinctrl-names : The list of names to assign states. List entry 0 defines
    the name for integer state ID 0, list entry 1 for state ID 1, and so on.

  pinctrl-<id> : List of phandles, each pointing at a pin configuration node
                 within a pin controller

Pin state id and pin state name
  You can define, different set of pin configuration values for different state of the device
  A device's state could be default, sleep, idle, active, etc
  Different pin states help to achieve dynamic configurability of pins

    Device' state Default => pins will be configured for i2c functionality
    Device' state sleep   => Pins will be configured as input to save power

How to write a pin-controller node ?
  • This is explained by your pin controller driver's binding document
  • If you are using generic pinctrl driver then refer pinctrl-single.txt
    under Documentation/devicetree/bindings/pinctrl/

am33xx-l4.dtsi

for controller node & child

```
    &am33xx_pinmux {                        // pin controller node

        i2c2_pins: pinmux i2c2 pins {       // child of pin controller

            pinctrl-single,pins = <         // property

                // init
                AM33XX_PADCONF(AM335X_PIN_I2C2_SDA, PIN_INPUT_PULLUP, MUX_MODE0)
                AM33XX_PADCONF(AM335X_PIN_I2C2_SCL, PIN_INPUT_PULLUP, MUX_MODE0)
            >;
        };
```

pinctrl-single, pins // property name : driver name, property name

property name = < init part >

for client node

```
    cape_eeprom0: capre_eeprom0@54 {
        pintctrl-names = "default";
        pintctrl-0 = <&i2c2_pins>;

        compatible = "atmel,24c256";


        i2c2_pins: pinmux i2c2 pins {
            pinctrl-single,pins = <
                AM33XX_PADCONF(AM335X_PIN_I2C2_SDA, PIN_INPUT_PULLUP, MUX_MODE0)
                AM33XX_PADCONF(AM335X_PIN_I2C2_SCL, PIN_INPUT_PULLUP, MUX_MODE0)
            >;
        };

        i2c2_pins_sleep: pinmux i2c2 pins_sleep {
            pinctrl-single,pins = <
                AM33XX_PADCONF(AM335X_PIN_I2C2_SDA, PIN_INPUT_PULLDOWN, MUX_MODE7)
                AM33XX_PADCONF(AM335X_PIN_I2C2_SCL, PIN_INPUT_PULLDOWN, MUX_MODE7)
            >;
        };
```

Here, a pin controller has 2 nodes. The first one configures the pins for i2c
functionality and the second one configures the same pins for gpio input
functionality .

The client device will use the first one during normal operation , and when
driver decides to put device to sleep, it will use the second one to
reconfigure the pins to stop leakage of current thus achieving low power

Client device node with 2 pin states

```
    cape_eeprom0: capre_eeprom0@54 {

        pintctrl-names = "default", "sleep";        # 0, 1
        pintctrl-0 = <&i2c2_pins>;                  # 0
        pintctrl-1 = <&i2c2_pins_sleep>;            # 1

        compatible = "atmel,24c256";
        reg = <0x54>;
        #address-cells = <1>;
        #size-cells = <1>;
        cape0_data: cape_data@0 {
            reg = <0 0x100>;
        };
    };
```



```
                                lcd_app
                                 /  \
user space              open    |    |   read/write
                                |    |
                sysfs gpio attributes (direction, value)
```

```
                                  |
-----------------------------------+--------------------------------
                                  |
kernel                        gpio-sysfs.ko  (show, store - attr val)
                                  |
-----------------------------------+--------------------------------
                                  |                              H/W
                              BBB gpios <-----> 16 x 2 LCD
```

Components required
- 16x2 Character LCD (1602A HD44780 LCD)
- 10K potentiometer
- Breadboard
- Connecting wires
- Beaglebone black

138. Significance of LCD application 476

P8.45	gpio2.6	4(RS)	register selection (char. vs. cmd)
p8.46	gpio2.7	5(RW)	read/write
p8.43	gpio2.8	6(EN)	enable
p8.44	gpio2.9	11(D4)	data line 4
p8.41	gpio2.10	12(D5)	data line 5
p8.42	gpio2.11	13(D6)	data line 6
p8.39	gpio2.12	14(D7)	data line 7
p9.1(GND)		15(BKLTA)	backlight anode(+)
p9.7(sys_5V supply)		16(BKLTA)	backlight cathode(-)
p9.1(GND)		1(VSS/GND)	Ground
p9.7(sys_5V supply		2(VCC)	+5V supply

139. Adding pin configuration node for gpios

Steps
1. Connect LCD to BBB
2. Add required gpio entries to the gpio-sysfs device tree node
3. Recompile the dts file and make BBB boots with modified dtb
4. Load the gpio-sysfs driver
5. Make sure that all required gpio devices are formed under /sys/class/bone_gpios
6. Download the lcd application files attached with this video. lcd_app.c, lcd.c, gpio.c
7. Cross compile the lcd application and test it on the target

$ vi am335x-boneblack-lddcrs.dtsi

refer to 448

$ vi am335x-l4.dtsi

```
    am33xx_pinmux: pinmux@800 {             // pin controller node
            compatible = "pinctrl-single";
            reg = <0x800 0x238>;
            #pinctrl-cells = <1>;
            pinctrl-single,register-width = <32>;
            pinctrl-single,function-mask = <0x7f>;
    };
```

Include/dt-bindings/pinctrl/am33xx.h

```
#define PULL_DISABLE            (1 << 3)
#define INPUT_EN                (1 << 5)
#define SLEWCTRL_SLOW           (1 << 6)
#define SLEWCTRL_FAST           0

#define PIN_OUTPUT              (PULL_DISABLE)
#define PIN_OUTPUT_PULLUP       (PULL_UP)
#define PIN_OUTPUT_PULLDOWN     0
#define PIN_INPUT               (INPUT_EN | PULL_DISABLE)
#define PIN_INPUT_PULLUP        (INPUT_EN | PULL_UP)
#define PIN_INPUT_PULLDOWN      (INPUT_EN)

#define AM335X_PIN_LCD_DATA0                    0x8a0
#define AM335X_PIN_LCD_DATA1                    0x8a4
#define AM335X_PIN_LCD_DATA2                    0x8a8
#define AM335X_PIN_LCD_DATA3                    0x8ac
#define AM335X_PIN_LCD_DATA4                    0x8b0
#define AM335X_PIN_LCD_DATA5                    0x8b4
#define AM335X_PIN_LCD_DATA6                    0x8b8
#define AM335X_PIN_LCD_DATA7                    0x8bc
#define AM335X_PIN_LCD_DATA8                    0x8c0
#define AM335X_PIN_LCD_DATA9                    0x8c4
#define AM335X_PIN_LCD_DATA10                   0x8c8
#define AM335X_PIN_LCD_DATA11                   0x8cc
#define AM335X_PIN_LCD_DATA12                   0x8d0
#define AM335X_PIN_LCD_DATA13                   0x8d4
#define AM335X_PIN_LCD_DATA14                   0x8d8
#define AM335X_PIN_LCD_DATA15                   0x8dc
#define AM335X_PIN_LCD_VSYNC                    0x8e0
#define AM335X_PIN_LCD_HSYNC                    0x8e4
#define AM335X_PIN_LCD_PCLK                     0x8e8
#define AM335X_PIN_LCD_AC_BIAS_EN               0x8ec
```

```
/ {
    ...

    bone_gpio_devs {
        compatible = "org,bone-gpio-sysfs";
        pinctrl-single,names = "default";
        pinctrl-0 = <&p8_gpio>;

        gpio1 {
            label = "gpio2.6";
            bone-gpios = <&gpio2 6 GPIO_ACTIVE_HIGH>;
        };

        ...

        gpio1 {
            label = "gpio2.12";
            bone-gpios = <&gpio2 12 GPIO_ACTIVE_HIGH>;
        };

    };//bone_gpio_devs
}; //root node

&tda19988 {
    status = "disabled";
}

&am33xx_pinmux {        // pin controller node

    p8_gpios: bone_p8_gpios
        //pinctrl-single,pins = < 0x00 0x00 >;  // init addr & val
        // AM33XX_PADCONF(AM335X_PIN_GPMC_A5, PIN_OUTPUT_PULLDOWN, MUX_MODE7)
        pinctrl-single,pins = <
            AM33XX_PADCONF(AM335X_PIN_GPMC_ALE, PIN_OUTPUT, MUX_MODE7)
            /*AM33XX_PADCONF(AM335X_PIN_LCD_DATA0, PIN_OUTPUT, MUX_MODE7)*/
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA1, PIN_OUTPUT, MUX_MODE7)
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA2, PIN_OUTPUT, MUX_MODE7)
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA3, PIN_OUTPUT, MUX_MODE7)
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA4, PIN_OUTPUT, MUX_MODE7)
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA5, PIN_OUTPUT, MUX_MODE7)
            AM33XX_PADCONF(AM335X_PIN_LCD_DATA6, PIN_OUTPUT, MUX_MODE7)
        >;
};
```

```
=> include/dt-bindings/pinctrl/omap.h

    #define OMAP_IOPAD_OFFSET(pa, offset)   (((pa) & 0xffff) - (offset))

    #define AM33XX_PADCONF(pa, dir, mux)    OMAP_IOPAD_OFFSET((pa), 0x0800) ((dir) | (mux))
```

Please note that, the pin configuration happens before calling the probe function
of the consumer driver, only if this property is set to 'default'.

If this property set to some other value, such as sleep, or idle, or something else,
then the pin control subsystem of Linux won't trigger any pin configuration
before calling probe function of the consumer driver.

$ vi am335x-boneblack-common.dtsi

```
&am33xx_pinmux {
        nxp_hdmi_bonelt_pins: nxp_hdmi_bonelt_pins {
                pinctrl-single,pins = <
                        AM33XX_PADCONF(AM335X_PIN_XDMA_EVENT_INTR0, PIN_OUTPUT_PULLUP, MUX_MODE7)
                        AM33XX_PADCONF(AM335X_PIN_LCD_DATA0, PIN_OUTPUT, MUX_MODE0)
                        AM33XX_PADCONF(AM335X_PIN_LCD_DATA1, PIN_OUTPUT, MUX_MODE0)
                        AM33XX_PADCONF(AM335X_PIN_LCD_DATA2, PIN_OUTPUT, MUX_MODE0)
                        AM33XX_PADCONF(AM335X_PIN_LCD_DATA3, PIN_OUTPUT, MUX_MODE0)
                        AM33XX_PADCONF(AM335X_PIN_LCD_DATA4, PIN_OUTPUT, MUX_MODE0)
                            ...

&i2c0 {
        tda19988: tda19988@70 {
                compatible = "nxp,tda998x";
                reg = <0x70>;
                nxp,calib-gpios = <&gpio1 25 0>;
                interrupts-extended = <&gpio1 25 IRQ_TYPE_LEVEL_LOW>;

                pinctrl-names = "default", "off";
                pinctrl-0 = <&nxp_hdmi_bonelt_pins>;
                pinctrl-1 = <&nxp_hdmi_bonelt_off_pins>;

        => nxp_hdmi_bonelt_pins referenced !!!
        => needs to disable in am335x-boneblack-lddcrs.dtsi : status = "disabled"
```

```
$ sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- am3335x-boneblack.dtb
$ sudo make copy-dtb    // am335x-boneblack.dtb
```

```
~/drivers# reboot

~/drivers# inmod gpio-sysfs.ko
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: Total devices found = 9
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.6
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.7
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.8
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.9
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.10
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.11
  [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.12
```

```
[ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.22
[ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.23
[ XXXX.XXX ] gpio_sysfs_init : module load success
~/drivers# ls /sys/class/bone_gpio
   gpio2.10 gpio2.12 gpio2.7 gpio2.9 usrled2:gpio1.23
   gpio2.11 gpio2.6  gpio2.8 usrled1:gpio1.22
~/drivers# ls /sys/kernel/debug/pinctrl
  44e10800.pinmux-pinctrl-single    pinctrl-devices     pinctrl-maps
  44e10800.rtc                      pinctrl-handles
~/drivers# ls /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single
 gpio-ranges pingroups pinmux-functions pinmux-pins pins
~/drivers# cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pingroups
 ...
 group:bone_p8_gpios
 pin 40 (PIN40)
 pin 41 (PIN41)
 pin 42 (PIN42)
 pin 43 (PIN43)
 pin 44 (PIN44)
 pin 45 (PIN45)
 pin 46 (PIN46)
~/drivers# cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins
 ...
 pin 40 (PIN40) 44e10894 0000002f   pinctrl-single      2f 0010_1111
 pin 41 (PIN41) 44e10898 0000002f   pinctrl-single
 pin 42 (PIN42) 44e1089c 0000002f   pinctrl-single
 pin 43 (PIN43) 44e108a0 0000002f   pinctrl-single
 pin 44 (PIN44) 44e108a4 0000002f   pinctrl-single
 pin 45 (PIN45) 44e108a8 0000002f   pinctrl-single
 pin 46 (PIN46) 44e108ac 0000002f   pinctrl-single
```

140. Exploring LCD code

```
                lcd_app.c       (LCD application)
                  |
                lcd.c           (LCD related functions )
                  |
                gpio.c          (File handling of sysfs attributes
                  |              (open, read, write, close))
          sysfs gpio attributes
           (direction , value )   (GPIO sysfs entries with attributes)
                  |
           Driver's store methods
              for attributes
```

141. Sending command to LCD 477

142. Creating LCD command code

```
~/drivers# ls /sys/class/lcd/
 LCD16X2

~/drivers# ls /sys/class/lcd/LCD16X2
 device  lcdcmd  lcdscroll  lcdtext  lcdxy  power  subsystem  uevent

 // lcd* are lcd attributes

~/drivers# echo 0x1 > /sys/class/lcd/LCD16X2/lcdcmd     // clear screen
~/drivers# echo 0x2 > /sys/class/lcd/LCD16X2/lcdcmd     // return home
~/drivers# echo -n "Good morning" > /sys/class/lcd/LCD16X2/lcdtext
 [ xxxx.xxx ] lcd LCD16X2: lcdtext: Good morning        // @1st row
~/drivers# echo 21 > /sys/class/lcd/LCD16X2/lcdxy       // move to (1,0)
~/drivers# echo -n "World" > /sys/class/lcd/LCD16X2/lcdtext
 [ xxxx.xxx ] lcd LCD16X2: lcdtext: World               // @2nd row
~/drivers# cat /sys/class/lcd/LCD16X2/lcdxy
 (2,1)
~/drivers# echo "on" > /sys/class/lcd/LCD16X2/lcdscroll // shift to left
~/drivers# echo "on" > /sys/class/lcd/LCD16X2/lcdscroll // shift to left
~/drivers# echo "on" > /sys/class/lcd/LCD16X2/lcdscroll // shift to left
~/drivers# echo 0x2 > /sys/class/lcd/LCD16X2/lcdcmd     // return home
```

```
usleep : user level lib
udelay : kernel lib
mdelay
```

```
. am335x-boneblack-lddcourse.dtsi

   bone_gpio_devs {
           ...
           status = "disabled";


    lcd16x2 {
           compatible = "org,lcd16x2";
           pictrl-names = "default";
           pinctrl-0 = <&p8_gpios>;
           status = "okay";
           rs-gpios = <&gpio2 2  GPIO_ACTIVE_HIGH>;
           rw-gpios = <&gpio2 7  GPIO_ACTIVE_HIGH>;
           en-gpios = <&gpio2 8  GPIO_ACTIVE_HIGH>;
           d4-gpios = <&gpio2 9  GPIO_ACTIVE_HIGH>;
           d5-gpios = <&gpio2 10  GPIO_ACTIVE_HIGH>;
           d6-gpios = <&gpio2 11  GPIO_ACTIVE_HIGH>;
           d7-gpios = <&gpio2 12  GPIO_ACTIVE_HIGH>;
    };
```

143. Testing LCD application over gpio sysfs

```
   $ vi am335x-boneblack-lddcrs.dtsi

           gpio1 {
               //label = "gpio2.6";
               //bone-gpios = <&gpio2 6 GPIO_ACTIVE_HIGH>;
               label = "gpio2.2";
               bone-gpios = <&gpio2 2 GPIO_ACTIVE_HIGH>;

               ...

           };


       &tda19988 {
           status = "disabled";
       }

       &am33xx_pinmux {        // pin controller node

           p8_gpios: bone_p8_gpios
               pinctrl-single,pins = <
                   AM33XX_PADCONF(AM335X_PIN_GPMC_ALE, PIN_OUTPUT, MUX_MODE7)
                   /*AM33XX_PADCONF(AM335X_PIN_LCD_DATA0, PIN_OUTPUT, MUX_MODE7)*/
                   AM33XX_PADCONF(AM335X_PIN_LCD_DATA1, PIN_OUTPUT, MUX_MODE7)
                   AM33XX_PADCONF(AM335X_PIN_LCD_DATA2, PIN_OUTPUT, MUX_MODE7)
                       ...
           >;
       };


       >> in BBB

       ~/drivers# inmod gpio-sysfs.ko
         [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: Total devices found = 9
         [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.2
         [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.7
         [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = gpio2.8
         ...
         [ XXXX.XXX ] bone-gpio-sysfs bone_gpio_devs: GPIO label = usrled0:gpio1.23
         [ XXXX.XXX ] gpio_sysfs_init : module load success

       ~/drivers# ./lcd_app.elf
```

144. Assignment : Implementing LCD platform driver

```
       >> in BBB
```

14 Linux synchronization services 1h35m

145. Avoiding race conditions
146. Linux locking services
147. Spinlock Vs Mutex
148. Linux spinlock functions
149. Mutex
150. Using locking functions in the code