**I2C CONTOLLER PROGRAMMING**

Legacy platform device method for probe the device

step-1

- write a platform device structure for the platform device.

- write resources like iomem(base address of platform device)

and irqs information.

- If necessary provide platform data to platform device.

step-2

- write a platform driver structure for binding the platform

device with platform driver. (Note: Names should match in

both platform device and plaform driver structure).

step-3

- write a module init routine to start the driver.

-In init routine

- Register the platform device for attach the platform device to

platform bus. (Infact, the regestrationof platform device

process should be taken from board specific file, but that

file not available in linux-4.04, so that the code has

written in driver code it-self)

- Register the platform driver which is common for all same kind of

platform devices.

-Exit routine

- Exit routine is not necessary when writing the driver as static

driver(partof kernel).

- Exit code here is for understanding, but not mandatory.

- Unregister the common platform driver for platform devices.

- Unregister the platform devices from the platform bus.

- After registration of both platform device and platform driver,

when the names of both structures(platform\_device and platform\_driver)

matched, the driver probe get called.

step-4:

- Start the probe function. Which initializes the adapter(i2c controller),

algorithm, work queue, spinlock, and requesting irq ...

- The Important things to start I2C comm. are CLOCK, and PINMUX, Don't forget

to ENABLE.

- It also creates a new bus for invidual i2c controller, and give the number as

specified in platform device info.

- its look like as below

Platform\_bus(controled by kernel) ==> I2C contoller(adapter driver) ==>

==> I2C\_BUS(controlled by adapter driver) ==>

==> i2c\_slaves device drivers(slaves which connect to device) ==>

==> slave applications(used by end-user)

- Remove function which removes the I2cbus from platform bus, and free the

irq line, disable spinlock, free the work queue.

step-5:

- Irq handler, which plays very important role here.

- The job of Irq here is to get the status, from status register and give the

appropriate info to the controller(like 7-bit addr, data rd, data write).

-This Irq is called by algorithm(xfer) routine.

step-6:

- write a algorithm structure, which specifies the i2c functionality, and gives

the i2c xfer(tx/rx) function information to slave device(client).

- Then slave device driver uses I2C Exported API's to read/write from/to the

controller.

- Xfer routine is the key of the starting i2c communication.

- It Enables the START and IRQ in the I2C controller.

- It useses wait queue timeout to wait(sleep) untill controller operations

completed through IRQ handler.

- The last step of IRQ handler is WAKEUP the wait queue with proper condition.

- Then after IRQ handler execution, Queued work wake\_up from sleep state, and

stop the i2c comm. and put the status to IDLE.

step-7:

- Implemnt I2C required IO operations.

- i.e in this block just define all i2c required register operations as functions.

NOTE:

- SUNXI\_I2C\_DEBUG :

- This macro i2c communication debug msgs.u