GPIO Initialization Framework & GPIO Operation Framework

GPIO Framework in Kernel space

Experiment 23: Prepare GPIO Initialization framework

Case 1:

Add printk's in init functions and at the start & end of probe function in drivers/gpio/gpio-omap.c

(Line no: 1764, 1246, 1361)

Run km-bbb-kernel-build.sh script and generate images.

Install kernel images in to target board using mmc (or) tftp (or) scp.

Reboot target board, run dmesg command and verfiy printk results in kernel log.

Expected Results:

Probe is invoked 4 times because in device tree source code compatible name “omap4-gpio” matches in gpio driver 4 times.

[0.255521] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.258392] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

[0.259600] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.261258] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

[0.262270] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.263659] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

[0.264641] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.265975] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

---

km@KM-BBB:/sys/class/gpio$ ls

export gpio45 gpio47 gpio87 gpio9 gpiochip32 gpiochip96

gpio44 gpio46 gpio86 gpio88 gpiochip0 gpiochip64 unexport

$ cat /sys/kernel/debug/gpio

Case 2:

Replace “gpio1, gpio2 and gpio3” compatible names with “test-omap” name in arch/arm/boot/dts/am33xx.dtsi file.

Run below command to build device tree source code.

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

Copy dtb image “km-bbb-am335x.dtb” to target board at “/boot/dtbs/uname -r/” location.

Reboot target board, run dmesg and verify printk results in kernel log.

Expected Results:

Probe is invoked only 1 time because in device tree source code compatible name “omap-gpio”

matches in gpio driver 1 time only. OMAP GPIO Device driver doesn’t recognize other 3 names as those

names were replaced by “test-omap” but driver can understand “gpio-omap” name.

[0.255399] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.258288] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

---

km@KM-BBB:/sys/class/gpio$ ls

export gpio9 gpiochip0 unexport

$ cat /sys/kernel/debug/gpio

After Testing, don't commit this modifications. revert the source code back to case 1.

$ git reset --hard Exp20

Experiment 23: GPIO Initialization Framework

Objective

Verify how the OMAP GPIO driver (gpio-omap.c) initializes GPIO controllers during boot, and how the **Device Tree (DTS) compatible name** drives the probe invocation.

Key Points

1. The GPIO framework is split into:
    - **Chip-level driver** (gpio-omap.c) – controls OMAP GPIO hardware.
    - **Generic GPIO library** (gpiolib.c) – abstracts operations for user-space via /sys/class/gpio or kernel APIs.
2. The probe function is called by the **platform device subsystem** when a matching device is found in the device tree.

Step-by-Step Flow

Case 1: Original DT compatible name

- You added printk() at the start & end of the probe function in drivers/gpio/gpio-omap.c:

```
omap_gpio_probe() {
  printk(KERN_INFO "drivers/gpio/gpio-omap.c:omap_gpio_probe:1246");
  ...
  printk(KERN_INFO "drivers/gpio/gpio-omap.c:omap_gpio_probe:1361");
}
```

- **Flow**:

1. Boot the kernel → device tree is parsed → omap4-gpio compatible nodes found 4 times.
2. Platform driver core calls omap_gpio_probe() **4 times**, once for each GPIO controller.
3. The kernel logs reflect the start and end of each probe call.

Dmesg Output Example:

```
[    0.255521] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246
[    0.258392] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361
... (repeated 4 times)
```

Result: /sys/class/gpio shows all GPIO chips (gpiochip0, gpiochip32, etc.).

Case 2: Modified DT compatible name

- Changed gpio1, gpio2, gpio3 names to "test-omap".
- Only **1 matching node** (omap-gpio) remains.

Flow:

1. Kernel parses DT → only 1 compatible match.
2. omap_gpio_probe() called **1 time**.
3. Only that GPIO chip is available in /sys/class/gpio.

Result in dmesg:

[0.255399] drivers/gpio/gpio-omap.c:omap_gpio_probe:1246

[0.258288] drivers/gpio/gpio-omap.c:omap_gpio_probe:1361

Takeaways from Experiment 23

- **Probe function** is **invoked per compatible device in the DTS**.
- Device Tree drives **driver initialization**.
- /sys/class/gpio is dynamically populated based on how many GPIO chips are initialized.

Experiment 24: Prepare GPIO Operation framework

Disable CONFIG_LEDS_GPIO configuration option to avoid continuous print messages.

Enable printk’s in the below mentioned functions:

drivers/gpio/gpiolib.c:

gpiod_request()

gpiod_direction_input()

gpiod_direction_output_raw_commit()

gpiod_direction_output()

gpiod_get_raw_value_commit()

gpiod_get_value()

gpiod_set_value()

gpiod_to_irq()

drivers/gpio/gpio-omap.c:

omap_set_gpio_direction()

omap_gpio_request()

omap_gpio_get_direction()

omap_gpio_input()

omap_gpio_get()

omap_gpio_output()

omap_gpio_set()

Run km-bbb-kernel-build.sh script and generate images.

Install kernel images in to target board using mmc (or) tftp (or) scp.

Reboot target board , run “$ cat /sys/class/gpio10/value” & “$ dmesg” commands in folder and verify printk results in kernel log.

root@KM-BBB:/sys/class/gpio/gpio10# cat direction

---

drivers/gpio/gpiolib.c:gpiod_get_direction:217 ->

drivers/gpio/gpiolib.c:gpiod_get_direction:232 -> status = chip->get_direction(chip, offset);

---

include/linux/gpio/driver.h:

---

struct chip

{

238 int (*request)(struct gpio_chip *chip, unsigned offset);

240 void (*free)(struct gpio_chip *chip, unsigned offset);

242 int (*get_direction)(struct gpio_chip *chip, unsigned offset);

244 int (*direction_input)(struct gpio_chip *chip, unsigned offset);

246 int (*direction_output)(struct gpio_chip *chip, unsigned offset, int value);

248 int (*get)(struct gpio_chip *chip, unsigned offset);

250 int (*get_multiple)(struct gpio_chip *chip, unsigned long *mask, unsigned long *bits);

253 void (*set)(struct gpio_chip *chip, unsigned offset, int value);

255 void (*set_multiple)(struct gpio_chip *chip, unsigned long *mask, unsigned long *bits);

258 int (*set_config)(struct gpio_chip *chip, unsigned offset, unsigned long config);

261 int (*to_irq)(struct gpio_chip *chip, unsigned offset);

263 void (*dbg_show)(struct seq_file *s, sruct gpio_chip *chip);

}

[160.402457] drivers/gpio/gpio-omap.c:omap_gpio_get_direction:990

---

root@KM-BBB:/sys/class/gpio/gpio10# echo out > direction

[2703.050477] drivers/gpio/gpiolib.c:gpiod_direction_output_raw:2640

[2703.056730] drivers/gpio/gpiolib.c:gpiod_direction_output_raw_commit:2598

[2703.063788] drivers/gpio/gpio-omap.c:omap_gpio_output:1027

[2703.069319] drivers/gpio/gpio-omap.c:omap_set_gpio_direction:10

root@KM-BBB:/sys/class/gpio/gpio10# echo 1 > value

[5252.691575] drivers/gpio/gpiolib.c:gpiod_set_value_nocheck:3198

After Testing, don't commit this modifications. revert the source code back to previous experiement.

$ git reset --hard Exp20

Experiment 24: GPIO Operation Framework

Objective

Understand **runtime GPIO operations**: input/output, direction setting, reading/writing values.

Key Points

1. Disable CONFIG_LEDS_GPIO to reduce print spam.
2. Add printk() to track **high-level calls** in:
    - drivers/gpiolib.c → generic interface
    - drivers/gpio/gpio-omap.c → hardware-specific operations

---

Step-by-Step GPIO Operation Flow

Kernel API → GPIO Chip Driver

Example: Setting GPIO direction to output

1. User runs:
2. echo out > /sys/class/gpio/gpio10/direction
3. **sysfs GPIO handler** calls gpiod_direction_output().
4. gpiolib.c → gpiod_direction_output_raw_commit() → chip->direction_output().
5. **Driver-specific implementation**:
6. omap_gpio_output(chip, offset, value)
7. omap_set_gpio_direction(chip, offset, direction)
8. printk logs show exact calls:
9. gpiolib: gpiod_direction_output_raw_commit
10. gpio-omap: omap_gpio_output
11. gpio-omap: omap_set_gpio_direction

Example: Writing GPIO value

1. User runs:

2. echo 1 > /sys/class/gpio/gpio10/value
3. gpiolib.c calls gpiod_set_value_nocheck().
4. Driver-specific omap_gpio_set() writes **hardware register**.
5. printk() logs:
6. drivers/gpio/gpiolib.c:gpiod_set_value_nocheck

GPIO Chip Structure (driver.h)

```
struct gpio_chip {
   int  (*request)(struct gpio_chip *chip, unsigned offset);
   void (*free)(struct gpio_chip *chip, unsigned offset);
   int  (*get_direction)(struct gpio_chip *chip, unsigned offset);
   int  (*direction_input)(struct gpio_chip *chip, unsigned offset);
   int  (*direction_output)(struct gpio_chip *chip, unsigned offset, int value);
   int  (*get)(struct gpio_chip *chip, unsigned offset);
   void (*set)(struct gpio_chip *chip, unsigned offset, int value);
   int  (*to_irq)(struct gpio_chip *chip, unsigned offset);
};
```

- **Flow**:
    1. User-space → gpiolib → chip operations
    2. Hardware registers updated via driver (gpio-omap.c)
    3. /sys/class/gpio reflects changes in direction/value

Key Observations

- **gpiolib** is **generic**, driver-independent.
- **gpio-omap** implements **chip-specific hardware access**.
- All GPIO operations are routed through **function pointers in struct gpio_chip**.
- Using printk() in both layers helps **trace exact kernel flow**.

Visual Flow (Summary)

User-space /sys/class/gpio

|

v

gpiolib.c API (gpiod_*)

|

v

gpio_chip->function (direction_input/output, get/set)

|

v

gpio-omap.c (omap_gpio_input/output/set/get)

|

v

Hardware GPIO registers

|

v

Physical Pin (LED, Switch, etc.)

✅ **Conclusion**

- Exp 23: **Initialization** – probe function called per compatible DTS node.
- Exp 24: **Operation** – direction/value set/read goes through gpiolib → chip driver → registers.
- printk in both generic and driver layers helps **visualize full GPIO flow** in kernel space.

1️⃣ **Test Procedure**

A. Experiment 23 – GPIO Initialization (Probe)

Objective: Verify that the OMAP GPIO driver initializes GPIO controllers correctly based on Device Tree compatible nodes.

Step-by-Step Test

1. **Preparation**

- Add printk() at the start and end of omap_gpio_probe() in drivers/gpio/gpio-omap.c.
- printk(KERN_INFO "omap_gpio_probe: start");
- printk(KERN_INFO "omap_gpio_probe: end");
- Make sure kernel build scripts are configured (km-bbb-kernel-build.sh).

2. **Build Kernel**
3. ./km-bbb-kernel-build.sh
4. **Deploy Kernel**
    - Copy kernel image to target BBB using one of:
        - MMC
        - TFTP
        - SCP
5. **Reboot BBB**
6. **Verify Probe Execution**
    - Check kernel log:
    - dmesg | grep omap_gpio_probe
    - Expected: **4 probe calls** for omap4-gpio (Case 1).
    - If compatible names are changed (test-omap), only **1 probe call** should appear.
7. **Check Sysfs GPIO Interface**
8. ls /sys/class/gpio
9. cat /sys/kernel/debug/gpio
    - Verify correct gpiochip entries appear based on probe count.
10. **Revert Code After Testing**
11. git reset --hard Exp20

B. Experiment 24 – GPIO Operations

Objective: Verify GPIO direction setting, reading, and writing functionality.

Step-by-Step Test

1. **Preparation**

- Disable CONFIG_LEDS_GPIO to avoid excessive printk logs.
- Add printk() in **gpiolib.c** and **gpio-omap.c** for key functions:
  - gpiod_request(), gpiod_direction_input(), gpiod_direction_output(), gpiod_get_value(), gpiod_set_value()
  - omap_gpio_get(), omap_gpio_output(), omap_set_gpio_direction(), etc.

2. **Build and Deploy Kernel**
3. ./km-bbb-kernel-build.sh
4. **Reboot BBB**
5. **Test GPIO Input**
   - Export GPIO if not already:
   - echo 10 > /sys/class/gpio/export
   - Check direction and read value:
   - cat /sys/class/gpio/gpio10/direction
   - cat /sys/class/gpio/gpio10/value
6. **Test GPIO Output**
   - Set direction to output:
   - echo out > /sys/class/gpio/gpio10/direction
   - Write value:
   - echo 1 > /sys/class/gpio/gpio10/value
   - echo 0 > /sys/class/gpio/gpio10/value
7. **Verify Kernel Logs**
8. dmesg | grep gpiod
9. dmesg | grep omap_gpio
   - You should see printk logs for each function call in the **correct sequence**:
     - gpiolib call → chip driver → hardware register
10. **Check Physical Response**
    - If a GPIO controls an LED, verify LED turns on/off as per value write.
11. **Cleanup**
12. echo 10 > /sys/class/gpio/unexport

13. git reset --hard Exp20

2️⃣ Debugging Techniques

A. Kernel Logging

- Use printk(KERN_INFO ...) to trace **function entry/exit** and key variable values.
- Use dmesg -w for **real-time log monitoring** during GPIO tests.
- Example:
- printk(KERN_INFO "omap_gpio_output: setting GPIO %d to %d\n", offset, value);

B. Sysfs Checks

- /sys/class/gpio/ interface provides:
  - direction – read/write
  - value – read/write
- /sys/kernel/debug/gpio shows:
  - **All GPIO chips**
  - **Pin ownership**
  - **Current direction/value**
- Useful to verify probe and runtime behavior.

C. Function Tracing

- Enable **ftrace** for GPIO functions:
- echo function > /sys/kernel/debug/tracing/current_tracer
- echo gpiod_ > /sys/kernel/debug/tracing/set_ftrace_filter
- cat /sys/kernel/debug/tracing/trace_pipe
- Shows **function call sequence** without modifying code.

D. Hardware Verification

- Connect **LEDs, switches, or multimeter** to GPIO pins.
- Verify **physical state matches software control**.

E. DT & Driver Mismatch Check

- Wrong Device Tree names → probe won't fire.
- Check with:
- dmesg | grep gpio
- Use of_node_get() or of_match_table in driver for debugging.

F. Common Issues & Fixes

Issue	Possible Cause	Fix
GPIO not exported	Missing echo <N> > export	Export GPIO in sysfs
Direction change fails	Driver pointer not set	Check chip->direction_output in gpio_chip
Probe not called	Device Tree mismatch	Verify compatible = "omap4-gpio"
Value not updating	Incorrect offset or GPIO number	Cross-check with /sys/kernel/debug/gpio

---

Summary

- Test procedure combines **probe verification** + **runtime GPIO operations**.
- Debugging combines **printk logging**, **sysfs inspection**, **ftrace**, and **hardware verification**.
- Always **revert changes after experiments** to maintain kernel integrity.

Step	Action / Command	Expected Output / Observation	Debug Check / Notes
Exp 23: GPIO Initialization – Probe			
1	Add printk() in omap_gpio_probe() at start & end		

Step	Action / Command	Expected Output / Observation	Debug Check / Notes
	(drivers/gpio/gpio-omap.c)		
2	Build kernel	./km-bbb-kernel-build.sh	Ensure no build errors
3	Copy kernel image to BBB via MMC/TFTP/SCP		
4	Reboot BBB		Boot successfully
5	Check dmesg	`dmesg	grep omap_gpio_probe`	
6	List GPIO chips	ls /sys/class/gpio	Chips like gpiochip0, gpiochip32 etc. visible
7	Check debug GPIO	cat /sys/kernel/debug/gpio	Confirm GPIO pins are mapped to correct chips
8	Modify DT compatible names (gpio1,2,3 -> test-omap)	Build DTB: make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs	Copy dtb to /boot/dtbs/<uname -r>/
9	Reboot BBB		Probe should be called **only 1 time**
10	Revert code	git reset --hard Exp20	Restore original DT & kernel source

Exp 24: GPIO Operation Framework			
1	Disable CONFIG_LEDS_GPIO in kernel config	Avoid continuous printk logs	
2	Add printk() in gpiolib.c functions: gpiod_request(), gpiod_direction_input(), gpiod_direction_output(), gpiod_get_value(), gpiod_set_value(), gpiod_to_irq()		Trace generic GPIO API calls
3	Add printk() in gpio-omap.c functions: omap_gpio_input(), omap_gpio_output(),

omap_set_gpio_direction(), etc. | | Trace driver-specific operations |
4	Build & deploy kernel	./km-bbb-kernel-build.sh	Boot without errors
5	Export GPIO	echo 10 > /sys/class/gpio/export	/sys/class/gpio/gpio10 folder created
6	Check direction	cat /sys/class/gpio/gpio10/direction	Should show in by default
7	Set GPIO direction to output	echo out > /sys/class/gpio/gpio10/direction	dmesg shows: gpiod_direction_output_raw_commit → omap_gpio_output → omap_set_gpio_direction
8	Write GPIO value HIGH	echo 1 > /sys/class/gpio/gpio10/value	dmesg shows: gpiod_set_value_nocheck
9	Write GPIO value LOW	echo 0 > /sys/class/gpio/gpio10/value	LED or pin should physically turn off
10	Read GPIO value	cat /sys/class/gpio/gpio10/value	Should return 0 or 1 matching last write
11	Read GPIO direction	cat /sys/class/gpio/gpio10/direction	Should return out
12	Unexport GPIO	echo 10 > /sys/class/gpio/unexport	/sys/class/gpio/gpio10 folder removed
13	Check /sys/kernel/debug/gpio	cat /sys/kernel/debug/gpio	Verify correct chip ownership & pin mapping
14	Optional: ftrace function trace	bash echo function > /sys/kernel/debug/tracing/current_tracer echo gpiod_ > /sys/kernel/debug/tracing/set_ftrace_filter cat /sys/kernel/debug/tracing/trace_pipe	Shows **sequence of GPIO API calls** in real time

---

Debugging Tips

- **Probe issues** → Check Device Tree compatible names.
- **Direction/value not working** → Verify chip->direction_output or chip->set function pointers.
- **No sysfs entries** → Ensure GPIO controller probe ran successfully.
- **Unexpected output** → Use printk in both gpiolib and driver layers to trace flow.
- **Physical verification** → Connect LEDs, switches, or multimeter to test actual pin state.