GPIO Device Driver framework in u-boot source code

🔹 **Experiment 12: Prepare GPIO Device Driver framework (source code flow) [gpio set 10]**

GPIO Framework – gpio set 10

Write printf (“%s:%s:%d\n”,_FILE,func,LINE_) function in the below files, build the source code and test “gpio set 10” in u-boot command prompt.

Source files:

cmd/gpio.c (CONFIG_CMD_GPIO=y)

command line interface -> device access commands -> gpio

drivers/gpio/gpio-uclass.c (CONFIG_DM_GPIO=y)

Device Drivers -> GPIO Support -> Enable Driver Model for GPIO drivers

drivers/gpio/omap_gpio.c (CONFIG_OMAP_GPIO=y)

Device Drivers -> GPIO Support -> TI OMAP GPIO drivers

1. Code modifications

You’ll be adding debug prints in three key files. Use this format inside the relevant functions:

printf("%s:%s:%d\n", __FILE__, __func__, __LINE__);

a) cmd/gpio.c (command handler)

This file parses gpio set 10 from the U-Boot shell.

- Look for functions like:
- static int do_gpio(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
- Add printf() at the entry point and before calling gpio_request(), gpio_direction_output(), etc.

b) drivers/gpio/gpio-uclass.c (GPIO uclass)

This handles driver-model (DM) translation from the command to the GPIO driver.

- Look for functions like:
- int gpio_request(unsigned gpio, const char *label)
- int gpio_direction_output(struct udevice *dev, unsigned offset, int value)

- Add printf() inside these to trace.

c) drivers/gpio/omap_gpio.c (TI OMAP-specific GPIO driver)

This is the hardware driver for AM335x (BeagleBone).

- Add debug prints in:
- static int omap_gpio_direction_output(struct udevice *dev, unsigned offset, int value)
- static int omap_gpio_set_value(struct udevice *dev, unsigned offset, int value)
- These confirm when actual register writes happen.

---

2. Build

Rebuild U-Boot with:

make CROSS_COMPILE=arm-linux-gnueabihf- am335x_evm_defconfig

make -j$(nproc)

---

3. Testing on board

Boot into U-Boot prompt and run:

=> gpio set 10

Expected flow of printf()s:

cmd/gpio.c:do_gpio:123

drivers/gpio/gpio-uclass.c:gpio_request:87

drivers/gpio/gpio-uclass.c:gpio_direction_output:156

drivers/gpio/omap_gpio.c:omap_gpio_direction_output:203

drivers/gpio/omap_gpio.c:omap_gpio_set_value:230

---

4. Git Commit

git add cmd/gpio.c drivers/gpio/gpio-uclass.c drivers/gpio/omap_gpio.c

git commit -m 'Prepare GPIO framework - "gpio set 10"' -s -t Exp12

- **Experiment 13: GPIO Framework – gpio input 11**

GPIO framework [gpio input 11]

Write printf (“%s:%s:%d\n”,_FILE,func,LINE_) function in the below files, build the source code and test “gpio input 11” in u-boot command prompt.

Source files:

cmd/gpio.c

drivers/gpio/gpio-uclass.c

drivers/gpio/omap_gpio.c

Now test the input path.

1. Insert debug prints in same files

a) cmd/gpio.c

- In the branch handling gpio input <num>.

b) drivers/gpio/gpio-uclass.c

- Functions like:
- int gpio_direction_input(struct udevice *dev, unsigned offset)
- int gpio_get_value(struct udevice *dev, unsigned offset)

c) drivers/gpio/omap_gpio.c

- Functions like:
- static int omap_gpio_direction_input(struct udevice *dev, unsigned offset)
- static int omap_gpio_get_value(struct udevice *dev, unsigned offset)

---

2. Build again

make -j$(nproc)

---

3. Testing

At U-Boot prompt:

=> gpio input 11

Expected debug print flow:

cmd/gpio.c:do_gpio:145

drivers/gpio/gpio-uclass.c:gpio_direction_input:176

drivers/gpio/omap_gpio.c:omap_gpio_direction_input:212

drivers/gpio/gpio-uclass.c:gpio_get_value:198

drivers/gpio/omap_gpio.c:omap_gpio_get_value:245

---

4. Git Commit

git add cmd/gpio.c drivers/gpio/gpio-uclass.c drivers/gpio/omap_gpio.c

git commit -m 'Prepare GPIO framework - "gpio input 11"' -s -t Exp13

- **GPIO Flow in U-Boot**

U-Boot CLI Command

↓

cmd/gpio.c ← Command handler (parses "gpio set 10" / "gpio input 11")

↓

drivers/gpio/gpio-uclass.c ← Generic GPIO framework (Driver Model layer)

↓

drivers/gpio/omap_gpio.c ← OMAP-specific GPIO driver (AM335x hardware)

↓

SoC GPIO Registers ← Actual hardware pins toggled / read

- **For gpio set 10**

1. **cmd/gpio.c**
    - do_gpio() parses "gpio set 10"
    - Calls → gpio_request() and gpio_direction_output()
2. **gpio-uclass.c**
    - gpio_request() checks GPIO availability

- gpio_direction_output() routes to driver ops
- Calls → omap_gpio_direction_output()

3. **omap_gpio.c**
   - Configures pin direction = OUT
   - Calls → omap_gpio_set_value() to write register
4. **Hardware**
   - Pin 10 driven HIGH

- **For gpio input 11**

1. **cmd/gpio.c**
   - do_gpio() parses "gpio input 11"
   - Calls → gpio_request() and gpio_direction_input()
2. **gpio-uclass.c**
   - gpio_direction_input() routes to driver ops
   - Calls → omap_gpio_direction_input()
   - Then calls → omap_gpio_get_value()
3. **omap_gpio.c**
   - Configures pin direction = IN
   - Reads register → returns HIGH/LOW
4. **Hardware**
   - Pin 11 state read

U-boot customization

Experiment 14: REAL ASSIGNMENT

Add blink option to gpio u-boot command and make it blink every 100 msec delay up to user press Ctrl+C.

Syntax: gpio <input|set|clear|toggle|blink> <pin>

Example: gpio blink <pin>

Source files:

cmd/gpio.c

After build & testing,

Tag name: Exp13

Commit Message: u-boot customization: Add blink option to gpio u-boot command

Perfect 👌 Now you're stepping into **U-Boot command customization**. Let's carefully go through **Experiment 14** and **Experiment 15** so you can implement, build, test, and commit with clarity.

🔹 **Experiment 14: Add blink option inside gpio command**

1. Source File: cmd/gpio.c

Locate the do_gpio() function in cmd/gpio.c. Currently it handles:

- gpio input
- gpio set
- gpio clear
- gpio toggle

You'll extend it to handle gpio blink.

---

2. Code Modification (cmd/gpio.c)

Add a new case for "blink":

#include <linux/delay.h> // For mdelay()

```
static int do_gpio(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    int ret = 0, gpio, value;

    if (argc < 3)
        return CMD_RET_USAGE;

    gpio = simple_strtoul(argv[2], NULL, 10);

    if (!strcmp(argv[1], "set")) {
        ret = gpio_direction_output(gpio, 1);
    } else if (!strcmp(argv[1], "clear")) {
        ret = gpio_direction_output(gpio, 0);
    } else if (!strcmp(argv[1], "toggle")) {
        value = !gpio_get_value(gpio);
        ret = gpio_direction_output(gpio, value);
    } else if (!strcmp(argv[1], "input")) {
        value = gpio_get_value(gpio);
        printf("GPIO %d value = %d\n", gpio, value);
    } else if (!strcmp(argv[1], "blink")) {
        printf("Blinking GPIO %d. Press Ctrl+C to stop.\n", gpio);
        while (!ctrlc()) {                  // Exit when Ctrl+C is pressed
            gpio_set_value(gpio, 1);
            mdelay(100);
            gpio_set_value(gpio, 0);
            mdelay(100);
        }
        printf("\nStopped blinking GPIO %d\n", gpio);
    } else {
```

```
        return CMD_RET_USAGE;
    }

    return ret;
}
```

---

3. Build

make CROSS_COMPILE=arm-linux-gnueabihf- am335x_evm_defconfig

make -j$(nproc)

---

4. Testing

At U-Boot prompt:

=> gpio blink 10

Blinking GPIO 10. Press Ctrl+C to stop.

- Pin 10 will toggle every **100 ms**.
- Stop with **Ctrl+C**.

---

5. Git Commit

git add cmd/gpio.c

git commit -m 'u-boot customization: Add blink option to gpio u-boot command' -s -t Exp13

U-Boot CLI: "gpio blink <pin>"

```
        |
        ▼
```

cmd/gpio.c : do_gpio()

```
  - parse "blink"
  - while(!ctrlc()):
        gpio_set_value(pin, 1)
        mdelay(100)
```

gpio_set_value(pin, 0)

mdelay(100)

drivers/gpio/gpio-uclass.c - routes to driver ops

drivers/gpio/omap_gpio.c

- omap_gpio_set_value()
- actual register write

SoC GPIO Registers

- Pin toggles ON/OFF every 100ms

Experiment 15: REAL ASSIGNMENT

Implement blink command in u-boot command prompt. Blink command to control GPIO pin every 100msec delay up to user press Ctrl+C.

Syntax: blink <pin>

Example: blink 10

Source files:

cmd/blink.c

cmd/Makefile

After build & testing,

Tag name: Exp15

Commit Message: u-boot customization – Add blink command in u-boot command prompt

🔹 Experiment 15: Add a new standalone blink command

Now, instead of extending gpio, we create a **separate command** (blink) with its own source file.

---

1. Source Files

- cmd/blink.c → New file
- cmd/Makefile → Add entry for blink

---

2. Create cmd/blink.c

```
#include <common.h>
#include <command.h>
#include <asm/gpio.h>
#include <linux/delay.h>

static int do_blink(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
   int gpio;

   if (argc < 2)
      return CMD_RET_USAGE;
   gpio = simple_strtoul(argv[1], NULL, 10);
   printf("Blinking GPIO %d. Press Ctrl+C to stop.\n", gpio);
   gpio_request(gpio, "blink");
   gpio_direction_output(gpio, 0);
   while (!ctrlc()) {
      gpio_set_value(gpio, 1);
      mdelay(100);
      gpio_set_value(gpio, 0);
      mdelay(100);
```

```
    }

    printf("\nStopped blinking GPIO %d\n", gpio);
    gpio_free(gpio);
    return 0;
}

U_BOOT_CMD(
    blink, 2, 0, do_blink,
    "Blink GPIO pin every 100ms until Ctrl+C",
    "<pin>\n"
    "    - Blink given GPIO pin every 100ms until Ctrl+C is pressed"
);
```

---

3. Modify cmd/Makefile

Add this line near other commands:

```
obj-$(CONFIG_CMD_BLINK) += blink.o
```

---

4. Enable in U-Boot Config

In your board config (e.g., include/configs/am335x_evm.h) or via Kconfig, define:

```
#define CONFIG_CMD_BLINK
```

Cmd/kcinfig

```
config CMD_BLINK
  bool "blink command"
 depends on DM_GPIO
          help
 Enable the 'blink' command in U-Boot, which toggles a GPIO
 every 100 ms until Ctrl+C is pressed.
```

---

5. Build

make -j$(nproc)

---

6. Testing

At U-Boot prompt:

=> blink 10

Blinking GPIO 10. Press Ctrl+C to stop.

---

7. Git Commit

git add cmd/blink.c cmd/Makefile

git commit -m 'u-boot customization – Add blink command in u-boot command prompt' -s -t Exp14

After these, you'll have **two ways to blink**:

- gpio blink <pin> (extended command – Exp14)
- blink <pin> (new standalone command – Exp15)

U-Boot CLI: "blink <pin>"

cmd/blink.c : do_blink()

- gpio_request(pin, "blink")
- gpio_direction_output(pin, 0)
- while(!ctrlc()):

    gpio_set_value(pin, 1)

    mdelay(100)

    gpio_set_value(pin, 0)

    mdelay(100)

- gpio_free(pin)

▼

drivers/gpio/gpio-uclass.c

|

▼

drivers/gpio/omap_gpio.c

|

▼

SoC GPIO Registers

- Pin toggles ON/OFF every 100ms

Communicate with Hardware @ Linux command prompt (Experiment 16 & 17)

Experiment 16: Control GPIO pins from sys file system entry @ user space

1. Enter gpio class in sysfs file system,

$ cd /sys/class/gpio

2. Enter root login,

/sys/class/gpio/$ sudo su

3. Export pin 10,

/sys/class/gpio/# echo 10 > export

4. Enter gpio10 folder,

/sys/class/gpio/# cd gpio10

5. Change pin direction to output,

/sys/class/gpio/gpio10/# echo out > direction

6. LED ON,

/sys/class/gpio/gpio10/# echo 1 > value

7. LED OFF,

/sys/class/gpio/gpio10/# echo 0 > value

8. Unexport pin 10,

/sys/class/gpio/gpio10/# cd ..

/sys/class/gpio/# echo 10 > unexport

Experiment 16: Control GPIO pins via sysfs (User Space)

Sysfs provides a file-based interface to GPIOs.

On BBB, GPIOs are memory-mapped but exposed in `/sys/class/gpio`.

Steps Recap + Explanation:

1. **Enter GPIO class directory**

   cd /sys/class/gpio

   → This is where kernel exposes GPIO control.

2. **Switch to root**

   sudo su

3. **Export a GPIO (e.g., pin 10)**

   echo 10 > export

   → Creates `/sys/class/gpio/gpio10` directory.

4. **Enter gpio10 folder**

   cd gpio10

5. **Set direction (out/in)**

   echo out > direction

6. **Turn LED ON**

   echo 1 > value

7. **Turn LED OFF**

   echo 0 > value

8. **Unexport GPIO**

   cd ..

   echo 10 > unexport

   → Cleans up.

✅ **Test:** Connect an LED (through resistor) to GPIO10 → you'll see it toggle.

Experiment 17: Monitoring Mux Configuration and GPIO pins using debug file system.

1. Enter root login,

$ sudo su

2. To read 128 GPIO pins status,

 # cat /sys/kernel/debug/gpio

3. To read Mux configuration of each pin,

 # cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins

User Space (Shell commands: echo, cat)

↓

Sysfs Virtual Files (/sys/class/gpio/*)

↓

Kernel (GPIO sysfs driver layer)

↓

GPIO Subsystem (GPIO chip driver)

↓

Hardware (GPIO Pin → LED ON/OFF)

- **Experiment 17: Debugging GPIO & Pinmux via debugfs**

Debugfs exposes runtime kernel info, including pinmux and GPIO state.

1. **Become root**

   sudo su

2. **Check all GPIO pin states (0–127 on AM335x)**

   cat /sys/kernel/debug/gpio

   → Shows which pins are exported, direction, and values.

3. **Check Pinmux register values**

   cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins

   → Dumps pinmux configuration (which pin is GPIO, UART, I2C, etc.).

User Space (cat /sys/kernel/debug/*)

↓

DebugFS Interface

↓

Kernel Debug Subsystem

↓

Pinmux & GPIO Drivers

↓

Hardware State Registers (Read Only)

Experiment 18: Write hello world module program and compile with ARM cross toolchain and transfer the binary file to target board using SD Card (or) scp.

- Experiment 18: Hello World Kernel Module (Driver Space)**

This is your **first kernel module** → moves from *user space GPIO control* to *writing custom kernel code*.

(A) Write Module Source Code (`hello.c`)

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
static int __init hello_init(void)
{
   pr_info("Hello, world! Kernel module loaded.\n");
   return 0;
}
static void __exit hello_exit(void)
{
   pr_info("Goodbye, world! Kernel module unloaded.\n");
}
```

```
module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("KM-BBB");
MODULE_DESCRIPTION("Simple Hello World Module");
```

(B) Write Makefile

```
obj-m += hello.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

(C) Cross-Compile for ARM

On your host PC:

```
export ARCH=arm
export CROSS_COMPILE=arm-linux-gnueabihf-
make
```

This produces `hello.ko` (kernel object file).

(D) Transfer Module to BBB

Option 1: Copy to SD card and mount on BBB.

Option 2: Use `scp`:

```
scp hello.ko debian@<bbb-ip>:/home/debian/
```

(E) Load / Unload Module

On BBB (as root):

```
insmod hello.ko
dmesg | tail
```

→ Should show `Hello, world! Kernel module loaded.`

Remove:

rmmod hello

dmesg | tail

→ Should show `Goodbye, world! Kernel module unloaded.`

User Space (*insmod / rmmod* commands)

↓

Kernel Module Loader (*insmod → syscalls*)

↓

Kernel Space (*hello_init / hello_exit*)

↓

pr_info → Kernel Log Buffer

↓

User Space (*dmesg → prints logs*)

DTS & Kernel Driver Level

Mux Configuration of GPIO [LED, Buzzer & Switch] in device tree source code (Experiment 19 & 20)

Experiment 19: LED,Buzzer & Switch Mux configuration in u-boot

Case 1: Enable LED, BUZZER, and Switch MUX configuration in u-boot source code.

We have enabled Mux, already in “Experiments 9 to 11” inside u-boot repository.

Expected Results: LED, BUZZER and Switch devices should work properly in Linux command prompt.

Case 2: Disable LED, BUZZER, and Switch MUX configuration in u-boot source code. (switch to master

branch)

Once you switch to master branch inside u-boot repository, MUX is disabled.

Expected Results: LED, BUZZER and Switch devices don’t work in Linux command prompt.

🔹 **Experiment 19: LED, Buzzer & Switch Mux in U-Boot**

Case 1: MUX enabled in U-Boot

- You already did this in **Experiments 9–11** by editing:
- board/ti/am335x/mux.c

Example entry:

{OFFSET(lcd_data14), (MODE(7))}, /* LCD_DATA14 as GPIO0_10 (RED LED) */

✅ **Expected Result**:
When you boot into Linux, the pins are already muxed as GPIOs (by U-Boot initcode).
So at Linux prompt, echo 1 > /sys/class/gpio/gpio10/value will toggle LED, etc.

Case 2: MUX disabled (master branch U-Boot)

- If you checkout the master branch (without mux configs in U-Boot), then U-Boot will not set the pins as GPIO.
- Linux will boot with pins left in **default mode (LCD_DATA)**.

❌ **Expected Result**:
GPIO10/9/11 won't respond at Linux sysfs because pinmux isn't configured for GPIO.

Experiment 20: Mux Configuration of GPIO in device tree source code.

Enable LED, Buzzer and Switch MUX configuration in device tree source code.

Source files:

arch/arm/boot/dts/am335x-bone-common.dtsi

Tag name: Exp21

Commit Message: "Mux Config in DTS: LCD_DATA_13.GPIO0_9, LCD_DATA14.GPIO0_10 & LCD_DATA_15.GPIO0_11"

Expected Results: LED, BUZZER and Switch devices should work properly in Linux command prompt.

41 AM33XX_IOPAD(0x8d4, MUX_MODE7) /* lcd_data13.gpio0_9 (BUZZER) */

42 AM33XX_IOPAD(0x8d8, MUX_MODE7) /* lcd_data14.gpio0_10(RED LED) */

43 AM33XX_IOPAD(0x8dc, MUX_MODE7 | PIN_INPUT_PULLUP) /* lcd_data15.gpio0_11 (ENTER SWITCH) */

🔹 **Experiment 20: Mux in Linux Device Tree**

Instead of doing muxing in U-Boot, you do it in **Device Tree (DTS)**.

File:

arch/arm/boot/dts/am335x-bone-common.dtsi

Add mux settings:

AM33XX_IOPAD(0x8d4, MUX_MODE7) /* lcd_data13.gpio0_9 (BUZZER) */

AM33XX_IOPAD(0x8d8, MUX_MODE7) /* lcd_data14.gpio0_10 (RED LED) */

AM33XX_IOPAD(0x8dc, MUX_MODE7 | PIN_INPUT_PULLUP) /* lcd_data15.gpio0_11 (SWITCH) */

👉 After editing, rebuild DTB:

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

👉 Copy updated .dtb to SD card (/boot/dtbs/ on BBB).

✅ **Expected Result**:
Pins are muxed correctly by **kernel pinctrl driver**, not U-Boot.
LED, Buzzer, Switch work properly at Linux prompt.

GPIO Test Modules (Experiment 21)

Experiment 21: GPIO Device driver Test cases.

To test GPIO device controller operations such as input, output and interrupt we can use module programming technique.

These programs can be referred to as GPIO Test modules.

Source files:

gpio-output.c

gpio-input.c

gpio-interrupt.c

Process:

Cross compile the modules and transfer images (.ko files) to target board. Then load each module.

Test GPIO input/output/interrupt to read/write to the GPIO pins using cat/echo commands.

Unload the module.

- **Experiment 21: GPIO Test Modules**

Here you create **kernel modules** to test GPIO functionality (output, input, interrupt).

(A) gpio-output.c

```
#include <linux/gpio.h>
#include <linux/init.h>
#include <linux/module.h>

#define GPIO_LED 10   // GPIO0_10
static int __init gpio_out_init(void)
{
   gpio_request(GPIO_LED, "LED");
   gpio_direction_output(GPIO_LED, 0);
   pr_info("GPIO Output Test: Blinking LED on GPIO%d\n", GPIO_LED);
   gpio_set_value(GPIO_LED, 1);
   msleep(1000);
   gpio_set_value(GPIO_LED, 0);
   return 0;
}
static void __exit gpio_out_exit(void)
{
   gpio_set_value(GPIO_LED, 0);
```

```
    gpio_free(GPIO_LED);
    pr_info("GPIO Output Test: Module unloaded\n");
}

module_init(gpio_out_init);
module_exit(gpio_out_exit);
MODULE_LICENSE("GPL");
```

(B) gpio-input.c

```
#include <linux/gpio.h>
#include <linux/init.h>
#include <linux/module.h>
#define GPIO_SWITCH 11  // GPIO0_11
static int __init gpio_in_init(void)
{
    int value;
    gpio_request(GPIO_SWITCH, "SWITCH");
    gpio_direction_input(GPIO_SWITCH);

    value = gpio_get_value(GPIO_SWITCH);
    pr_info("GPIO Input Test: Switch value = %d\n", value);
    return 0;
}
static void __exit gpio_in_exit(void)
{
    gpio_free(GPIO_SWITCH);
    pr_info("GPIO Input Test: Module unloaded\n");
}
```

```
module_init(gpio_in_init);
module_exit(gpio_in_exit);
MODULE_LICENSE("GPL");
```

(C) gpio-interrupt.c

```
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/init.h>
#include <linux/module.h>

#define GPIO_SWITCH 11

static int irq_number;

static irqreturn_t switch_irq_handler(int irq, void *dev_id)
{
    pr_info("GPIO Interrupt Test: Switch pressed!\n");
    return IRQ_HANDLED;
}

static int __init gpio_irq_init(void)
{
    gpio_request(GPIO_SWITCH, "SWITCH");
    gpio_direction_input(GPIO_SWITCH);
    irq_number = gpio_to_irq(GPIO_SWITCH);

    request_irq(irq_number, switch_irq_handler, IRQF_TRIGGER_FALLING,
                "switch_irq", NULL);
```

```
    pr_info("GPIO Interrupt Test: Module loaded, waiting for interrupt\n");
    return 0;
}

static void __exit gpio_irq_exit(void)
{
    free_irq(irq_number, NULL);
    gpio_free(GPIO_SWITCH);
    pr_info("GPIO Interrupt Test: Module unloaded\n");
}

module_init(gpio_irq_init);
module_exit(gpio_irq_exit);
MODULE_LICENSE("GPL");
```

Build and Test

1. **Makefile**
2. obj-m += gpio-output.o
3. obj-m += gpio-input.o
4. obj-m += gpio-interrupt.o
5.
6. all:
7. 	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
8.
9. clean:
10. 	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
11. **Cross-compile**:
12. export ARCH=arm
13. export CROSS_COMPILE=arm-linux-gnueabihf-

14. make
15. **Transfer to BBB** (scp *.ko debian@<ip>:/home/debian/)
16. **Test on board**:
17. sudo insmod gpio-output.ko
18. sudo insmod gpio-input.ko
19. sudo insmod gpio-interrupt.ko
20. dmesg | tail

✅ **Learning Outcome**

- Exp19: Understand mux at U-Boot vs Linux
- Exp20: Learn DTS pinctrl configuration
- Exp21: First real GPIO device driver testing with **output, input, interrupt handling**