ELEC 424 - Project 2: gpiod

100 points

Overview

IMPORTANT NOTE: Back up your code constantly somewhere other than the Raspberry Pi. Improper device tree modifications can cause the Pi not to boot up anymore. If that happens, you will have to reflash your microSD card. We probably can prevent this by plugging in your microSD card into a laptop and modifying the config.txt file from there, but anything is possible. You know how hardware is.

Your goal is to toggle an external LED on and off from kernel space by making a driver that uses gpiod and interrupts.

You must use gpio pin 5 for the LED and 6 for the button. Note that these numbers don't correspond to the physical pin numbers, as we have discussed many times in class. The circuit is the same as assignment 2, but you must use the pins I've specified. Also note that usually gpio 6 for the button will usually have a pull up scheme where the value read will be 1 when the button is not pressed, meaning the button should usually connect gpio 6 to ground when pressed.

Rubric

1. **(50 points) In person demonstration of functionality. In person demonstrations can be done during instructor or TA office hours and possibly before or after class. We will also offer check off during 3-5pm on Friday Nov 3rd as well, WHICH WILL BE THE LAST TIME AVAILABLE TO DEMO YOUR PROJECT 2.**
    a. (10 pts) Module insertion message (from a simple printk() statement) appears in kernel log
    b. (10 pts) LED state changes via module
    c. (10 pts) Button state can be detected via module
    d. (10 pts) Interrupt service routine called and prints kernel log message when button is pressed, and debouncing works
    e. (10 pts) Button press toggles LED on/off using interrupt service routine, and toggling works again after module removed and inserted again
        i. Toggling should not work during the period when the module has been removed but not reinserted
2. **(50 points) Submission of relevant commented code files and report to Canvas**

a. (10 points) Code attempts to achieve objectives/requirements stated in instructions
b. (10 points) Code reasonably includes comments (at least every other line includes a comment)
c. (5 points) The following file(s) must be submitted in source form (.tbl, .c, etc.) - not a PDF
    i. gpiod_driver.c
    ii. Your <insert name>.dts file
    iii. config.txt
d. (25 points) PDF report that includes:
    i. (1 point) Title of assignment/project
    ii. (1 point) Your name
    iii. (5 points) 1 paragraph (at least 4 sentences) describing the goal of the project and the steps you took to complete it (include a statement on each key function)
    iv. (5 points) A 2nd paragraph (at least 4 sentences) describing what you found challenging/any bugs you had to fix, what you learned, and what you think would be another interesting application for this gpiod approach.
    v. (5 points) Include a screenshot showing a significant portion or all of your driver code.
    vi. (4 points) Include a screenshot of terminal output showing the messages printed by your code.
    vii. (4 points) All screenshots in the report must include a figure label with a short description.

Guidelines

- **Three files must be worked with to pull this off:**
    - Device tree overlay file
    - Boot configuration file config.txt which is in /boot/
        - You will have already played with this file in-class
        - We just need config.txt to include the compiled device tree overlay file
    - Module/driver file, which I will call gpiod_driver.c
- **For the device tree overlay**
    - You need to make a device tree overlay file that enables gpiod to access gpio pins 5 and 6 according to the “Device Tree” section of the documentation here (hosted by *The Linux Kernel Archives*).
        - You can start with the file contents here: https://stackoverflow.com/a/59950806 (posted by user Simon on *Stack Overflow*)

        - The modifications you should make:
            1. target should be replaced with target-path and should be set equal to the root node of the device tree, i.e., “/” (including the quotation marks)
                - See an example of this target-path here
                - This will ultimately make it so we have a fake device under the root node of the device tree that exposes gpio pins 5 and 6 for gpiod to use in kernel space
            2. hsncarr should be renamed to a name of your own choosing for the fake device that we are making
            3. compatible will have to be set equal to the same name that you later use in your driver code for compatible (you get to choose, so choose a new name)
            4. carr-gpios is following the syntax specified in the “Device Tree” section of the documentation here (hosted by *The Linux Kernel Archives*)
                - You want to replace carr with your own function name, and duplicate this line with yet another function name
                - Use function names for these two lines to correspond to your LED and your button
                - 17 in each line should be replaced with the appropriate gpio index
        - When you are done with editing the overlay file, compile it (replacing name_of_file with the name of your file):

          ```
          dtc -@ -I dts -O dtb -o name_of_file.dtbo name_of_file.dts
          ```
            - If you see errors, you will need to visually debug
        - The previous command compiles your overlay file, producing a compile file with the extension .dtbo
        - We need to copy this compiled file to where the boot configuration can see it:

          ```
          cp name_of_file.dtbo /boot/overlays/
          ```
        - If you end up needing to change your overlay file again, you will have to run the complication and copy commands again
- **For the boot configuration file config.txt**
    - The kernel is given the device tree at boot
    - We need to make sure modification to the device tree is included by modifying the configuration file to include our modified overlay
    - At the end of the config.txt file in /boot you need to have (where name_of_file refers to the name of your compiled device tree overlay):
        - `dtoverlay=name_of_file`
        - NOTE: Do not include the file extension
    - Once you are done with this, you have to reboot for the overlay to take effect and be implemented in the device tree

        - You will have to reboot anytime you make changes to the overlay file (after again compiling and again copying the compiled file)
- **For the module/driver file gpiod_driver.c**
    - The previous two modifications should make the GPIO pin accessible to your driver
    - Your driver doesn't exist yet, however! So now you need to write it, and the goal of your driver is to:
        - Have a button trigger an interrupt that toggles an LED, all in kernel space (i.e., all in your driver)
        - You MUST use GPIOD, not the old deprecated gpio method
    - Grab the template file and Makefile from Canvas (Files/Project 2/)
    - The template file has /*INSERT*/ or other notes wherever code where will definitely have to be added
    - Feel free to draw liberally (including copy and paste) interrupt code from here
        - That file was using the old gpio method (don't use those parts), we want to use their interrupt code with gpiod
        - Be sure to cite that reference if you use the code
        - The author uses IRQF_TRIGGER_RISING, you should change RISING to FALLING (because of the pull up behavior of our gpio pin for the button)
    - You will have to use gpiod [NOT gpio functions - those are deprecated] and irq (interrupt) functions. Below are the prototypes from /include/linux/gpio/consumer.h [real definitions appear to be here], https://elixir.bootlin.com/linux/latest/source/include/linux/interrupt.h, and elsewhere in the Linux source code to give you an idea of the inputs and outputs.
        - `struct gpio_desc *__must_check` **`devm_gpiod_get`**`(struct device *dev,`
          `const char *con_id,`
          `enum gpiod_flags flags);`
        - `void` **`gpiod_set_value`**`(struct gpio_desc *desc, int value);`
        - `int` **`gpiod_get_value`**`(const struct gpio_desc *desc);`
        - `int` **`gpiod_set_debounce`**`(struct gpio_desc *desc, unsigned int debounce);`
        - `int` **`gpiod_to_irq`**`(const struct gpio_desc *desc);`
        - `static inline int __must_check` **`request_irq`**`(unsigned int irq, irq_handler_t handler, unsigned long flags, const char *name, void *dev)`
        - `const void *`**`free_irq`**`(unsigned int irq, void *dev_id)`
    - The first function gets you a struct pointer of type gpio_desc that refers to the gpio pin for your LED, and the second function takes that pointer and can alter the gpio pin output value (0 or 1). Follow the wonder of slide 35 in Lecture 16 to figure out what to use for *dev on the first function because the devm_gpiod_get_index() function in the slide takes the same first input argument. You cannot just use "dev" - look at what is being passed along from the right screenshot to the left in the slide. For the LED output pin, *con_id will actually just be the function string that you used earlier for the device tree (i.e., relating to

<function>-gpios), and button will be similar (but a different name). Flags will be GPIOD_OUT_LOW (start with pin with output value 0) for the LED and GPIOD_IN for the button.
        - Note: we don't need MODULE_DEVICE_TABLE() for what we're doing despite it being shown on slide 33 of lecture 16
    - Your probe function must use `printk` (with a line feed "\n") to print some message to the kernel log (which can be viewed with `tail -f /var/log/kern.log`) when the module is inserted
    - You must implement switch debouncing using the `gpiod_set_debounce` function (you will do this in the probe function, not the interrupt service routine)
    - Be sure to free your irq in the remove function, otherwise your system may freeze when you reinsert the module
    - Your interrupt service routine must print a message to the log (printk) indicating that the button has been pressed
    - Here is the workflow you will use
        - Write the driver code
        - Compile it by using the new Makefile from canvas (Files/Project 2/) and running "sudo make" after changing
        - Make sure the module is removed (if previously inserted) by running: sudo modprobe -r gpiod_driver
            1. This is what we use for these types of modules, rather than insmod/rmmod
            2. Notice we don't put .ko for this function
        - Copy your newly compiled gpiod_driver.ko to a special spot for these kinds of modules [step 1 (making the folder) must be done the first time]:
            1. `sudo mkdir /lib/modules/$(uname -r)/misc/`
            2. `sudo cp gpiod_driver.ko /lib/modules/$(uname -r)/misc/`
        - Update the dependencies related to modules (again, something special we have to do for this kind of module)
            1. sudo depmod
        - Insert the module:
            1. sudo modprobe gpiod_driver
                - Again, no .ko extension for this command
        - Now try your button and see if it toggles the LED
        - You can also watch the kernel log in parallel (another terminal) to see if you are getting print statements from your driver related to what you are trying:
            1. `tail -f /var/log/kern.log`
        - If it doesn't work, you'll loop back through these bullet points after attempting to fix the code