ECE471: Embedded Systems – Fall 2024
Homework 4: Linux GPIO Interface

Due: Friday, 4 October 2024, 5:00pm EDT

1. **Use your Raspberry-Pi to access the GPIOs.**

   - You will need a breadboard, an LED, some resistors, and some wire for this homework. In class I handed out some female/male hookup wires for connecting the Pi to the breadboard. You should source the remaining parts yourself, although if for some reason you cannot let me know and I can help you find some.
   - Download the code from:
     `https://web.eece.maine.edu/~vweaver/classes/ece471/ece471_hw4_code.tar.gz`
     and copy it to the Raspberry-Pi.
   - Uncompress/unpack it: `tar -xzvf ece471_hw4_code.tar.gz`
   - Change into the ece471_hw4_code directory: `cd ece471_hw4_code`

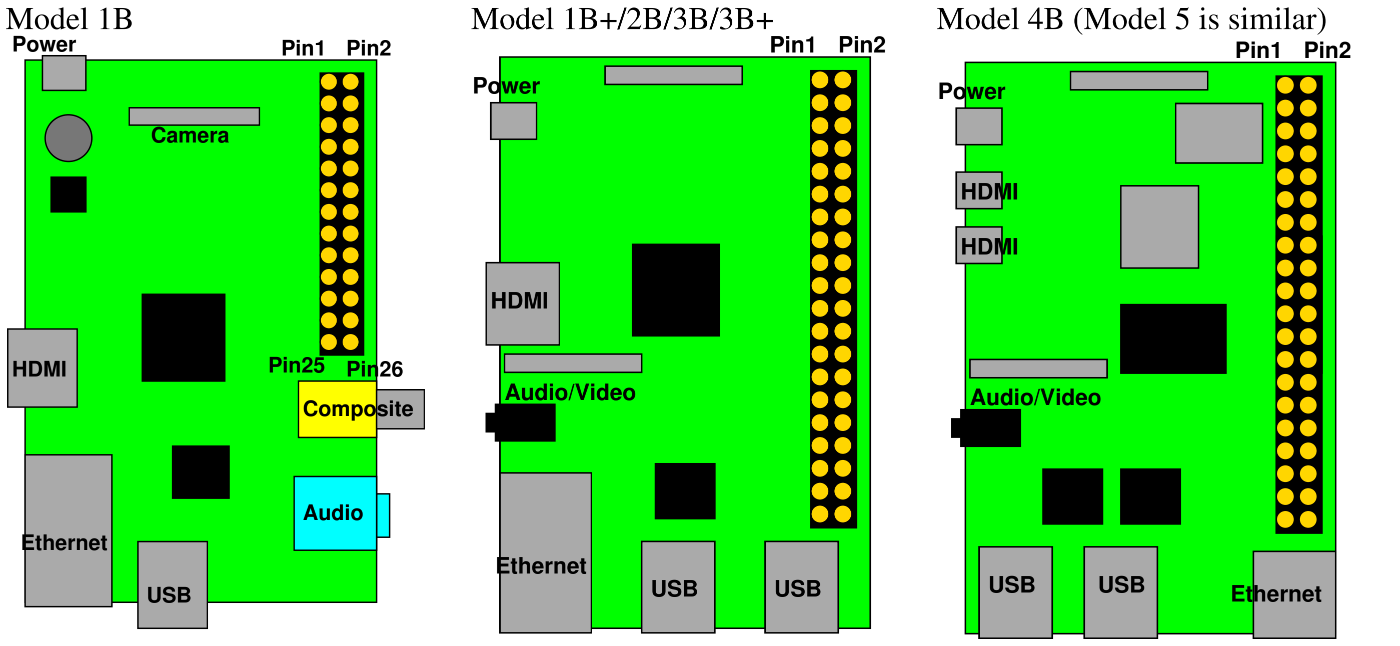

Figure 1: Raspberry Pi Layout

2. **Hook up an LED to a GPIO pin (3 points total)**

   (a) Hook up an LED to GPIO18. Figure 2 shows the appropriate circuit and Figure 1 and Table 1 might be helpful.

   (b) Modify `blink_led.c` so that it makes the LED blink. Blink at 1Hz with a 50% duty cycle: it should blink with a pattern of 0.5 seconds on, 0.5 seconds off, repeating forever (on Linux to exit out of a program in an infinite loop you can press control-C). Be sure you have the timing right!

Table 1: Raspberry Pi Header Pinout

3.3V	1	2	5V
GPIO2 (SDA)	3	4	5V
GPIO3 (SCL)	5	6	GND
GPIO4 (1-wire)	7	8	GPIO14 (UART_TXD)
GND	9	10	GPIO15 (UART_RXD)
GPIO17	11	12	GPIO18 (PCM_CLK)
GPIO27	13	14	GND
GPIO22	15	16	GPIO23
3.3V	17	18	GPIO24
GPIO10 (MOSI)	19	20	GND
GPIO9 (MISO)	21	22	GPIO25
GPIO11 (SCLK)	23	24	GPIO8 (CE0)
GND	25	26	GPIO7 (CE1)
ID_SD (EEPROM)	27	28	ID_SC (EEPROM)
GPIO5	29	30	GND
GPIO6	31	32	GPIO12
GPIO13	33	34	GND
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
GND	39	40	GPIO21

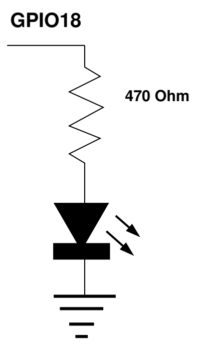

Figure 2: LED connected to GPIO18.

(c) Code requirements:

   i. Write your code in C.
   ii. Make sure the code compiles *without warnings*.
   iii. Comment your code!
   iv. Be sure to check for errors (especially at open time)! If you detect an error, print a message and then clean up and exit the program.

(d) Hints

   i. If you get a permissions error when trying to run things, the easiest (though not very secure) way to get it going is to use the `sudo` utility which temporarily runs a program with root permissions, i.e. `sudo ./blink_led`. You can permanently give permission for a user to access the gpio device with the command `sudo addgroup username gpio` where username is your username.
   ii. `usleep()` can be used to sleep a certain number of microseconds.

(e) The Linux GPIO interface was covered in lecture. A brief overview follows:

   i. Open the device

```
int fd,rv;
/* Open the gpio device */
fd=open("/dev/gpiochip0",O_RDWR);     // FIXME: proper error handling
```

   ii. Set up a request struct

```
struct gpiohandle_request req;
memset(&req,0,sizeof(struct gpiohandle_request));
req.flags = GPIOHANDLE_REQUEST_OUTPUT;
req.lines = 1;
req.lineoffsets[0] = 23;    // FIXME: replace with proper GPIO number
req.default_values[0] = 0;
strcpy(req.consumer_label, "ECE471");
rv = ioctl(fd, GPIO_GET_LINEHANDLE_IOCTL, &req);
```

   iii. Set gpio value

```
struct gpiohandle_data data;
data.values[0]=0;    // value to output (0 or 1)
rv=ioctl(req.fd,GPIOHANDLE_SET_LINE_VALUES_IOCTL,&data);
```

(f) Be sure to check for errors. If you encounter an error, notify the user (print a message) and then exit the program if there's no way to continue. You can use the `exit()` function to exit a Linux program.

3. **Read input from a switch (3 points total)**

(a) Connect GPIO17 to a switch as shown in Figure 3. If you do not have a switch, you can instead just use a piece of wire connected to 3.3V and touch the other end to the proper pin.

(b) Modify `read_switch.c` so that it loops forever waiting for keypress events: it should print to the screen when the key changes state (meaning when the key is pressed, it should print once that it was pressed, then after the key is released it should prince once that it was released). Please use the `switch_pressed()` and `switch_released()` routines provided in the debounce.c file for printing the press/release messages, these include timestamps that can help with debugging.

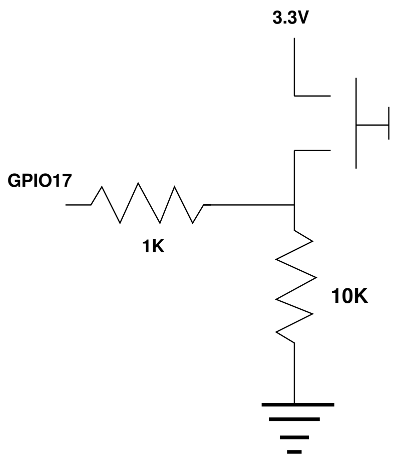

Figure 3: Switch connected to GPIO.

(c) Each time you press and release the switch it should only print *two* messages, one when the key is pressed and one when it is released.

(d) You can implement this by polling (constantly reading the input in a loop) you do not need to use the fancier interrupts/`poll()` interface.

(e) You will, however, need to de-bounce the switch in software
Note: to properly debounce you will need to do at least two reads and only report a switch change if they match. Doing one read and then sleeping is not a proper debounce.

(f) The GPIO input code is similar to that for doing GPIO output with the following differences:

   i. When setting up the request struct, use `GPIOHANDLE_REQUEST_INPUT` instead of `GPIOHANDLE_REQUEST_OUTPUT`

   ii. To read the value use

```
memset(&data, 0, sizeof(data));
rv = ioctl(req.fd, GPIOHANDLE_GET_LINE_VALUES_IOCTL, &data);
/* the read value is in data.values[0] */
```

(g) Note: you will need to do an `ioctl()` each time you read the value. The `data` variable doesn't map to the hardware registers, it's just a plain integer value that is only updated when you ask the kernel to write the GPIO value into it via `ioctl()`

4. **Something cool: (1 point total)**
Edit `gpio_extra.c` and do something cool. Put a short description of what you did in the README.

Here are some suggestions:

- Have the switch toggle the LED on and off.
- Have the LED blink a Morse code message.
- Hook up a second LED to GPIO23 and have them blink alternately.

5. **Questions to Answer: (2 points total)**
   Put the answer to these in the README.

   (a) Why is it good to use `usleep()` rather than a busy loop?

   (b) How does having an operating system make life easier when programming GPIOs?

   (c) How did you implement the debounce of the switch input?

6. **Linux Command Line Exploration (1 point total)**

   Try out the `dmesg` program. This shows all of the system boot messages. (Note, in some systems you might need to be administrator for this to work, in the case you'll need to do `sudo dmesg`). Try piping the output into `less` so you can easily scroll back and look at the messages:
   `dmesg | less`

   (a) You can use `grep` to search for text. Find out what your machine type is by running:
   `dmesg | grep Machine`
   Report your machine type in the README file.

   (b) You can use `uname` to find out more about your system. Run:
   `uname -a`
   and report your kernel version in the README file.

   (c) You can use `df` to find out how much disk is free. Run:
   `df -h`
   and report the free space on your rootfs filesystem.

   (d) What does the `-h` option to the `df` command do?

7. **Submitting your work.**

   - Run `make submit` which will create `hw4_submit.tar.gz`
     You can verify the contents with `tar -tzvf hw4_submit.tar.gz`
   - e-mail the `hw4_submit.tar.gz` file to me by the homework deadline. Be sure to send the proper file!