Lab 1 - GPIO

BeagleBone Green Getting Started and GPIO

This lab involves getting started with a BeagleBone Green (BBG) microSD card and programming the GPIO.

1. Getting Started

Instructions for getting started with the BBG are available at at at **https://beagleboard.org/getting-started** **(https://beagleboard.org/getting-started)** . This involves downloading the Debian Linux image and writing it to a microSD card (make sure you use the Debian image for the BBG). My version was:

```
Buster IoT (without graphical desktop) for BeagleBone and PocketBeagle via microSD card
AM3358 Debian 10.3 2020-04-06 4GB SD IoT image for PocketBeagle, BeagleBone, BeagleBone Blac
k, BeagleBone Black Wireless, BeagleBone Black Industrial, BeagleBone Blue, SeeedStudio Beagl
eBone Green, SeeedStudio BeagleBone Green Wireless, SanCloud BeagleBone Enhanced, Arrow Beagl
eBone Black Industrial and Mentorel BeagleBone uSomIQ - more info - sha256sum: 22448ba28d0d58
e25e875aac3b4e91eaef82e2d11c9d2c43d948ed60708f7434
```

Follow the instructions at **http://phwl.org/2021/connecting-beaglebone-to-internet/** **(http://phwl.org/2021/connecting-beaglebone-to-internet/)** to connect the BBG to the Internet.

2. GPIO

Since Linux v4.8, the standard way of using Linux GPIO has been via **libgpiod** **(https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/)** . Prior to the introduction of libgpiod, the sysfs interface was used, but sysfs is depreciated and was removed from the mainline Linux kernel in 2020. The best source of information on libgpiod **comes with the library** **(https://git.kernel.org/pub/scm/libs/libgpiod/libgpiod.git/about/)** and the (**libgpiod manual** **(http://phwl.org/assets/images/2021/02/libgpiod-ref.pdf)**).

The library comes with a number of command line tools to manipulate GPIOs:

- gpiodetect - list all gpiochips present on the system, their names, labels and number of GPIO lines

- gpioinfo - list all lines of specified gpiochips, their names, consumers, direction, active state and additional flags
- gpioget - read values of specified GPIO lines
- gpioset - set values of specified GPIO lines, potentially keep the lines exported and wait until timeout, user input or signal
- gpiofind - find the gpiochip name and line offset given the line name
- gpiomon - wait for events on GPIO lines, specify which events to watch, how many events to process before exiting or if the events should be reported to the console.

Here are some usage examples from the Linux documentation (some like `gpiomon` won't work on the BBG without an input source):

```
# Read the value of a single GPIO line.
$ gpioget gpiochip1 23
0

# Read two values at the same time. Set the active state of the lines
# to low.
$ gpioget --active-low gpiochip1 23 24
1 1

# Set values of two lines, then daemonize and wait for a signal (SIGINT or
# SIGTERM) before releasing them.
$ gpioset --mode=signal --background gpiochip1 23=1 24=0

# Set the value of a single line, then exit immediately. This is useful
# for floating pins.
$ gpioset gpiochip1 23=1

# Find a GPIO line by name.
$ gpiofind "USR-LED-2" gpiochip1 23

# Toggle a GPIO by name, then wait for the user to press ENTER.
$ gpioset --mode=wait `gpiofind "USR-LED-2"`=1

# Wait for three rising edge events on a single GPIO line, then exit.
$ gpiomon --num-events=3 --rising-edge gpiochip2 3
event:  RISING EDGE offset: 3 timestamp: [    1151.814356387]
event:  RISING EDGE offset: 3 timestamp: [    1151.815449803]
event:  RISING EDGE offset: 3 timestamp: [    1152.091556803]

# Wait for a single falling edge event. Specify a custom output format.
$ gpiomon --format="%e %o %s %n" --falling-edge gpiochip1 4 0 4 1156 615459801

# Pause execution until a single event of any type occurs. Don't print
```

```
    # anything. Find the line by name.
    $ gpiomon --num-events=1 --silent `gpiofind "USR-IN"`

    # Monitor multiple lines, exit after the first event.
    $ gpiomon --silent --num-events=1 gpiochip0 2 3 5
```

2. Laboratory Experiment

Part 1 - Setup (10%)

To use GPIO on the BBG, one must first login to the beagleboard:

```
phwl@PHWL-MBP lab1-gpio % ssh debian@beaglebone.local
Debian GNU/Linux 10

BeagleBoard.org Debian Buster IoT Image 2020-04-06

Support: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

debian@10.65.196.185's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Mar  3 04:58:05 2021 from 10.17.29.225
```

Then update the Debian package manager:

```
debian@beaglebone:~$ sudo apt update
[sudo] password for debian:
Hit:1 http://deb.debian.org/debian buster InRelease
Get:2 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:3 http://repos.rcn-ee.com/debian buster InRelease [3,078 B]
Get:4 http://deb.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:5 http://deb.debian.org/debian-security buster/updates/main armhf Packages [262 kB]
Get:6 http://repos.rcn-ee.com/debian buster/main armhf Packages [1,740 kB]
Fetched 2,122 kB in 50s (42.6 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
110 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Now install the libgpiod library and git:

```
debian@beaglebone:~$ sudo apt install libgpiod-dev git
Reading package lists... Done
Building dependency tree
Reading state information... Done
libgpiod-dev is already the newest version (1.4.1-2rcnee3~buster+20190906).
Suggested packages:
  gettext-base git-daemon-run | git-daemon-sysvinit git-doc git-el git-email
  git-gui gitk gitweb git-cvs git-mediawiki git-svn
The following packages will be upgraded:
  git
1 upgraded, 0 newly installed, 0 to remove and 109 not upgraded.
Need to get 4,542 kB of archives.
After this operation, 50.2 kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://deb.debian.org/debian buster/main armhf git armhf 1:2.20.1-2+deb10u3 [4,542 kB]
Fetched 4,542 kB in 3s (1,396 kB/s)
(Reading database ... 72582 files and directories currently installed.)
Preparing to unpack .../git_1%3a2.20.1-2+deb10u3_armhf.deb ...
Unpacking git (1:2.20.1-2+deb10u3) over (1:2.20.1-2+deb10u1) ...
Setting up git (1:2.20.1-2+deb10u3) ...
```

Clone the starting files for this lab

```
debian@beaglebone:~$ git clone https://github.com/phwl/elec3607-labquestions.git
Cloning into 'elec3607-labquestions'...
remote: Enumerating objects: 63, done.
remote: Counting objects: 100% (63/63), done.
remote: Compressing objects: 100% (39/39), done.
remote: Total 63 (delta 21), reused 52 (delta 13), pack-reused 0
Unpacking objects: 100% (63/63), done.
```

and install a Device Tree overlay to set the pinctl mode of the pins we will be using to Mode 7 (we will discuss Device Trees in more detail later in this course):

```
debian@beaglebone:~$ cd elec3607-labquestions/labs/lab1-gpio/
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ make install
sudo cp PHWL-GPIO-00A0.dtbo /lib/firmware
sudo cp uEnv.txt /boot
```

You will need to reboot the system for the Device Tree changes to take effect.

```
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ sudo shutdown -r now
```

You can check everything is working by detecting the GPIO chips and printing their labels and state.

```
debian@beaglebone:~$ gpiodetect
gpiochip0 [gpio] (32 lines)
gpiochip1 [gpio] (32 lines)
gpiochip2 [gpio] (32 lines)
gpiochip3 [gpio] (32 lines)
debian@beaglebone:~$ gpioinfo
gpiochip0 - 32 lines:
        line   0:  "MDIO_DATA"       unused   input  active-high
        line   1:   "MDIO_CLK"       unused   input  active-high
...
```

Now turn on the D5 LED for 10 seconds using

```
debian@beaglebone:~$ gpioset -m time -s 10 gpiochip1 24=1
```

and turn it off for 10 seconds

```
debian@beaglebone:~$ gpioset -m time -s 10 gpiochip1 24=0
```

Note that after `gpioset` exits, the data of the output is undefined. Place a listing of your program in you lab book.

Part 2 - S2 Button Input (30%)

Download, compile and execute the `blink` program which flashes the D5 LED as follows

```
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ ls
blink.c  Makefile  PHWL-GPIO-00A0.dtbo  PHWL-GPIO.dts  ssd.c
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ make
cc    -c -o ssd.o ssd.c
gcc -o ssd ssd.c -lgpiod
cc    -c -o blink.o blink.c
gcc -o blink blink.c -lgpiod
dtc -O dtb -o PHWL-GPIO-00A0.dtbo -b 0 -@ PHWL-GPIO.dts
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ ./blink
```

Here is a listing of `blink.c`

```c
/*
 * **    blink.c -    blink with 1s delay
 * */

#include <stdio.h>
#include <unistd.h>
#include <gpiod.h>

#define GPIOCHIP        1
#define GPIOLINE        24

int
main(int argc, char *argv[])
{
        struct gpiod_chip *output_chip;
        struct gpiod_line *output_line;
        int line_value;

        /* open chip and get line */
        output_chip = gpiod_chip_open_by_number(GPIOCHIP);
        output_line = gpiod_chip_get_line(output_chip, GPIOLINE);

23          /* config as output and set a description */
24          gpiod_line_request_output(output_line, "blink", GPIOD_LINE_ACTIVE_STATE_HIGH);
25  
26          for (;;)
27          {
28                  line_value = !line_value;
29                  gpiod_line_set_value(output_line, line_value);
30                  sleep(1);
31          }
32  
33          return 0;
34  }
```

Referring to the **libgpiod manual (http://phwl.org/assets/images/2021/02/libgpiod-ref.pdf)**, we obtain a descriptor for the chip specified by `GPIOCHIP` in line 20. Using that descriptor we obtain one for the line in line 21. In line 24, we configure the pin as an output, and give it the name “blink” (this is called a consumer in libgpiod). Line 28 toggles `line_value` between a true and false value, which is written to the line in line 29. We then sleep for 1 second before returning to the top of the infinite loop. You can press control-C to exit the program.

It is most convenient to edit the file blink.c directly from the command line. To do this, you can use the vi editor, and (refer to the **Vi summary (https://canvas.sydney.edu.au/courses/32687/pages/resources)** for further instructions):

```
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ vi blink.c
```

Modify the program so that, in addition to blinking the LED, it will print the status of the S2 button once a second as below.

```
debian@beaglebone:~/elec3607-labquestions/labs/lab1-gpio$ ./blink
S2=1
...
S2=0
```

Place a listing of your program in you lab book.

Part 3 - Seven Segment Display (60%)

Here is a listing of `ssd.c`, a program to be completed in this exercise.

```
/*
 * ssd.c -     count up and down on the SSD
 *
*/

#include <stdio.h>
#include <unistd.h>
#include <gpiod.h>

/* the SSD is entirely on this chip */
#define OUT_GPIOCHIP    0

/* the S2 button on the BBG */
#define IN_GPIOCHIP             2
#define IN_GPIOLINE             8

#define NELTS(x)        (sizeof(x) / sizeof(x[0]))      // calculate number of elements in x
#define SEGMENTS        NELTS(gpiossd)

static int      gpiossd[] = { 2, 13, 12, 4, 5, 15, 31 };

/* FILL THIS IN */
static int      ssd[][SEGMENTS] = {
        { 1, 1, 1, 1, 1, 1, 0 },        /* 0 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 1 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 2 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 3 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 4 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 5 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 6 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 7 */
        { 0, 0, 0, 0, 0, 0, 0 },        /* 8 */
        { 0, 0, 0, 0, 0, 0, 0 } };      /* 9 */

int
main(int argc, char *argv[])
{
    struct      gpiod_chip *output_chip;
    struct      gpiod_line *output_line[SEGMENTS];
        struct  gpiod_chip *input_chip;
        struct  gpiod_line *input_line;
    int         line_value;
        int             i, j;
        /* open input */
        input_chip = gpiod_chip_open_by_number(IN_GPIOCHIP);
        input_line = gpiod_chip_get_line(input_chip, IN_GPIOLINE);
        gpiod_line_request_input(input_line, "ssd");

    /* open /dev/gpiochip0 */
    output_chip = gpiod_chip_open_by_number(OUT_GPIOCHIP);
        for (i = 0; i < SEGMENTS; i++)
        {
                output_line[i] = gpiod_chip_get_line(output_chip, gpiossd[i
]);
                /* config as output and set a description */
                gpiod_line_request_output(output_line[i], "ssd", GPIOD_LINE_A
CTIVE_STATE_HIGH);
        }

        int d = 0;              /* the digit to display */
        for (;;)
        {
                /* display all segments */
                for (j = 0; j < SEGMENTS; j++)
                {
            /* FILL THIS IN */
                }
                sleep(1);
                /* update count */

        /* FILL THIS IN */
        }
    return 0;
}
```

Refer to the **Beaglebone P9 connector table** (http://exploringbeaglebone.com/wp-content/uploads/2019/01/533160-c06f009.png) for the mapping of pins to the gpiochip0 lines.

The data sheet for a seven segment display (SSD) is available **here** (https://au.element14.com/broadcom-limited/5082-7613/led-display-0-3-he-red/dp/1175576) .

- Calculate the resistor value so that the current to drive the LED (voltage is 3.3V-Vf where Vf=the diode forward voltage drop) doesn't exceed 4mA.
- After studying `ssd.c`, connect up the SSD to the BBG via a breadboard.
- Complete the "FILL THIS IN" parts of `ssd.c` so the value changes in value 0 to 9 every second then goes back to 0. When S2 is pressed it should count backwards.

Note that for the program to work, you will need to use the same GPIO lines as the program. Draw the circuit in your lab book and include a listing.