LED Guide

by Brian Fraser
Last update: Sept 22, 2022

Guide has been tested on

BeagleBone (Target):	**Debian 11.8**
PC OS (host):	**Debian 11.8**

This document guides the user through

1. Controlling the LEDs on the BeagleBone via the command line terminal.
2. Controlling the LEDs via C code

Table of Contents

Formatting

1. Commands for the host Linux's console are show as:
   `(host)$ echo "Hello PC world!"`
2. Commands for the target (BeagleBone) Linux's console are shown as:
   `(bbg)$ echo "Hello embedded world!"`
3. Almost all commands are case sensitive.

Revision History

- Jan 15: Updated for semester start
- Jan 19: Changed prompts to `(bbg)$` and `(host)$`
- Sept 2022: Updated commands for timer since files are now accessible to user

1. LEDs on BeagleBone

The five LEDs on the BeagleBone are shown below. Only LED3 through LED0 are controllable via software.

Power LED
(on when board is
powered on)

LED3
LED2
LED1
LED0

Figure 1: BeagleBone's 5 LEDs, above and below the Ethernet connector. BeagleBone Black shown in image; BeagleBone Green has the USB-A host connector where the 5V power connector is on the Black.

Default trigger for each LED:

- LED0: `heartbeat` = flashes about twice a second.
- LED1: `mmc0` = indicates activity on the uSD card (which is mapped to mmc0).
- LED2: `cpu0` = indicates processor activity.
- LED3: `mmc1` = indicates activity on the internal eMMC.
- Power: Always on when board has power. Not software controllable.

2. LEDs Controlled via Command Line

This guide requires a terminal connection to the BeagleBone via either serial or SSH. We will control the LEDs via the `sysfs` virtual file system which is exposed by the Linux kernel in the `/sys/` directory.

2.1 Turn On/Off LED

1. List all files in the `/sys/class/leds/` directory
   ```
   (bbg)$ cd /sys/class/leds
   (bbg)$ ls
   ```
   - This shows the four software-controlled LEDs on the board.
2. Change to directory for LED0
   ```
   (bbg)$ cd beaglebone\:green\:usr0
   ```
   - Note the ':' in path have to be escaped in the commands, but not in the output.
     Hint: Use tab-complete! Type `beag` and then press tab, then type `0`.
   - The path mentions green LEDs, but they are in fact blue.
3. Files of note in `/sys/class/leds/beaglebone:green:usr0` directory:
   - `trigger`: Specifies what, if anything, will cause the LED to turn on/off.
   - `brightness`: Direct control of LED on/off.
   - If a file is accessible only by the root user, you may need to use the following to write to it:
     `(bbg)$ echo 0 | sudo tee brightness`
     - This is the equivalent of "`echo 0 > brightness`", except running the tee program as superuser to pipe the output of echo into the file.
     - Note that "`sudo echo 0 > brightness`" won't work: it runs echo as super user, not writing to the file as super user.
4. Check the current value for trigger (output shown below command):
   ```
   (bbg)$ cat trigger
   none nand-disk mmc0 mmc1 timer oneshot [heartbeat] backlight gpio cpu0
   default-on transient
   ```
   - Note that `[heartbeat]` is in square brackets, indicating it's currently selected.
   - The first section of this document lists the default triggers for each LED.
5. Change trigger to "`none`" for direct control
   ```
   (bbg)$ echo none > trigger
   ```
6. Change `brightness` to 1 to turn on
   ```
   (bbg)$ echo 1 > brightness
   ```
   - Look at LED0 on the board; look at board to ensure it is now on.
7. Change `brightness` to 0 to turn off
   ```
   (bbg)$ echo 0 > brightness
   ```
   - Check LED0 is now off.
8. Return LED0 to flashing a heartbeat
   ```
   (bbg)$ echo heartbeat > trigger
   ```

2.2 Blinking Options

1. Change to the LED0 directory
   ```
   (bbg)$ cd /sys/class/leds/beaglebone\:green\:usr0
   ```
2. View files:
   ```
   (bbg)$ ls
   brightness  device  max_brightness  power  subsystem  trigger  uevent
   ```
3. Change the `trigger` to `timer`:
   ```
   (bbg)$ echo timer > trigger
   ```
4. View files:
   ```
   (bbg)$ ls
   brightness  delay_on  max_brightness  subsystem  uevent
   delay_off   device    power           trigger
   ```
   - Note the new files `delay_on`, `delay_off`
5. Set the timing to be on for 100ms and off for 900ms
   ```
   (bbg)$ echo 100 > delay_on
   (bbg)$ echo 900 > delay_off
   ```
   - This should make LED0 have a quick flash once a second.
6. Reverse the delays and see a long flash once a second.

3. LEDs Controlled via C

3.1 Control the trigger

1. Use `fopen()` to open the `trigger` file (as accessed in previous steps) for write access.
   - Example `fopen()` call:
     ```
     #define DA_TRIGGER_FILE_NAME_HERE "..."  // at top of file
     ...
     // inside your code, such as in like main()
     FILE *pLedTriggerFile = fopen(DA_TRIGGER_FILE_NAME_HERE, "w");
     ```
   - Check that the `fopen()` call succeed!
     ```
     if (pLedTriggerFile == NULL) {
           printf("ERROR OPENING %s.", DA_TRIGGER_FILE_NAME_HERE);
           exit(1);
     }
     ```
2. Write to the file the required trigger using `fprintf()`:
   ```
   int charWritten = fprintf(pLedTriggerFile, "none");
   if (charWritten <= 0) {
         printf("ERROR WRITING DATA");
         exit(1);
   }
   ```
3. Close the file using `fclose()`:
   ```
   fclose(pLedTriggerFile);
   ```

3.2 Setting the brightness

1. Open the `brightness` file (as used in previous sections) using `fopen()`.
2. Write the desired LED state (“1” or “0”) to the file using `fprintf()`.
3. Close the file using `fclose()`.
   - Note that you may have to close/open the `brightness` file if you want to make a number of changes to the brightness of the LED over time. Each time you want to change its brightness it works reliably to close and re-open the file.

3.3 Timing

The `nanosleep()` function suspends the current process for a certain amount of time, specified in seconds and nanoseconds (billionths of a second).

Example call to `nanosleep()`:

```
// Sleep 1.5 seconds
long seconds = 1;
long nanoseconds = 500000000;
struct timespec reqDelay = {seconds, nanoseconds};
nanosleep(&reqDelay, (struct timespec *) NULL);
```

Full example program, named `timing.c` using `nanosleep`:

```
// Timing test
#include <stdio.h>
#include <time.h>

int main()
{
      printf("Timing test\n");

      for (int i = 0; i < 5; i++) {
            long seconds = 1;
            long nanoseconds = 500000000;
            struct timespec reqDelay = {seconds, nanoseconds};
            nanosleep(&reqDelay, (struct timespec *) NULL);
            printf("Delayed print %d.\n", i);
      }
      return 0;
}
```

Compile `timing.c` with command (switch to cross compiler as needed):

```
gcc -Wall -g -std=c99 -D _POSIX_C_SOURCE=200809L -Werror timing.c -o timing
```

Note the POSIX option (`-D _POSIX_C_SOURCE=....`) which defines the `_POSIX_C_SOURCE` constant. This causes `nanosleep()`'s prototype to be included in the `time.h` header file; without it compiling generates a warning because `nanosleep()`'s prototype is compiled out (via `#define`'s). Since it is dangerous to allow your code to compile with warnings, the `-Werror` option is also included.

Consult the man page for `nanosleep()` (command `man nanosleep`) for more information.

4. Useful References

1. Walk-through of using terminal for LEDs and C++ code example.
   http://derekmolloy.ie/beaglebone-controlling-the-on-board-leds-using-c/
2. Walk-through of LEDs via termianl, plus discussion of GPIO.
   http://robotic-controls.com/book/export/html/69