LED Guide

by Brian Fraser
Last update: Jan 21, 2025

Guide has been tested on

BeagleBone (Target):	**Debian 12.8**
PC OS (host):	**Debian 12.8**

This document guides the user through

1. Controlling the LEDs on the BeagleY-AI 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:
   ```
   (byai)$ echo "Hello embedded world!"
   ```
3. Almost all commands are case sensitive.

Revision History

- Jan 21, 2025

1. LEDs on BeagleY-AI

The BeagleY-AI has a single dual-colour (red/green) LED.

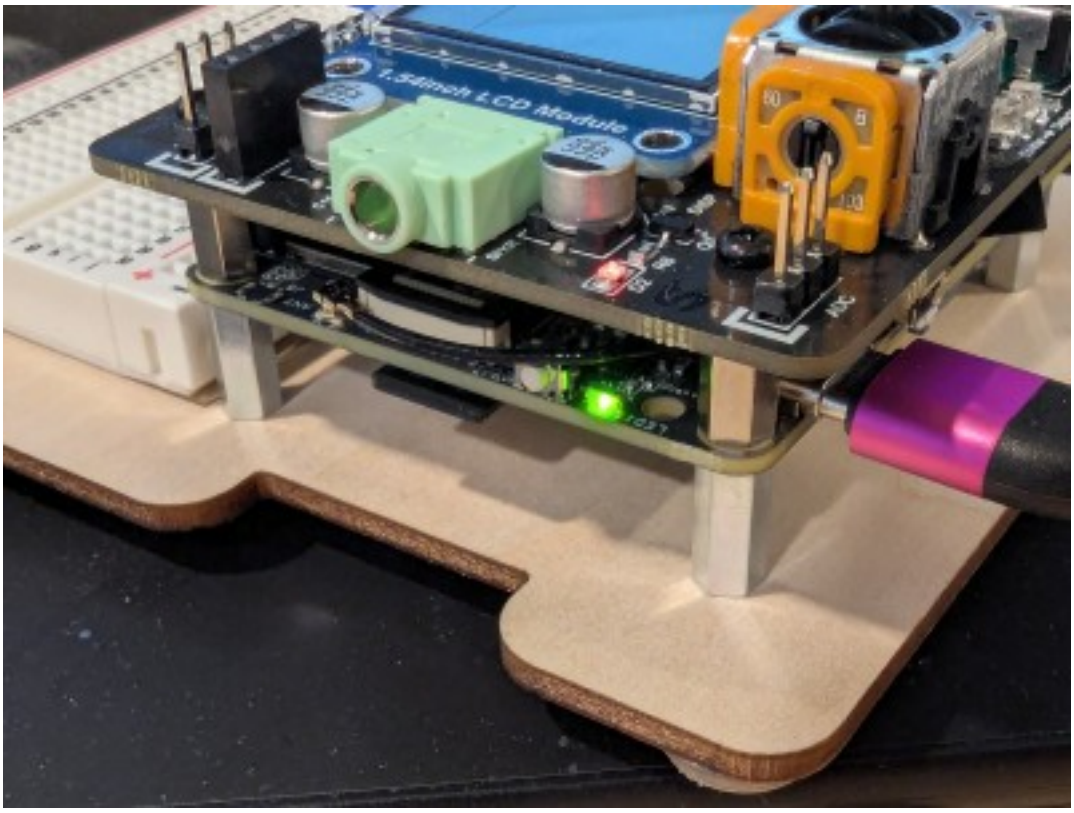

Figure 1: BeagleY-AI's dual-colour LED.

Each colour of the LED is given a directory in the Linux file system to control it independently of the other colour.

Default triggers for LEDs:

- ACT (Green): `heartbeat` = flashes about twice a second.
- PWR (Red): `none` = Off

With a dual-colour LED, one can control each colour separately. When both LEDs are on, the colours will seem to merge a little. With the BYAI's dual-colour LED, when both colours are on the red colour strongly dominates.

2. LEDs Controlled via Command Line

This guide requires a terminal connection to the target. 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
   ```
   (byai)$ cd /sys/class/leds
   (byai)$ ls
   ```
   - This shows quite a few LED devices; however only two of them are physical LEDs on the BeagleY-AI: `ACT` (green channel) and `PWR` (red channel).
2. Change to directory for the ACT LED (green):
   ```
   (byai)$ cd /sys/class/leds/ACT
   ```
3. Files of note in `/sys/class/leds/ACT` directory:
   - `trigger`: Specifies what, if anything, will cause the LED to turn on/off.
   - `brightness`: Direct control of LED on/off.
   - This file is accessible via a regular user so we can use:
     ```
     (byai)$ echo 1 > brightness
     ```

     If a file is accessible only by the root user, you may need to use the following to write to it:
     ```
     (byai)$ 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):
   ```
   (byai)$ cat trigger
   none usb-gadget usb-host rfkill-any rfkill-none kbd-scrolllock kbd-numlock
   kbd-capslock kbd-kanalock kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-
   altlock kbd-shiftllock kbd-shiftrlock kbd-ctrlllock kbd-ctrlrlock timer
   oneshot disk-activity disk-read disk-write ide-disk mtd nand-disk [heartbeat]
   backlight cpu cpu0 cpu1 cpu2 cpu3 activity default-on panic mmc2 mmc1
   8000f00.mdio:00:link 8000f00.mdio:00:1Gbps 8000f00.mdio:00:100Mbps
   8000f00.mdio:00:10Mbps bluetooth-power phy0rx phy0tx phy0assoc phy0radio
   rfkill0
   ```
   - Note that `[heartbeat]` is in square brackets, indicating it's currently selected.
5. Change trigger to "`none`" for direct control
   ```
   (byai)$ echo none > trigger
   ```
6. Change `brightness` to 1 to turn on
   ```
   (byai)$ echo 1 > brightness
   ```
   - Look at the LED on the board; look at board to ensure it is now on and green.
7. Change `brightness` to 0 to turn off
   ```
   (byai)$ echo 0 > brightness
   ```
   - Check the LED is now off.
8. Return green LED to flashing a heartbeat
   ```
   (byai)$ echo heartbeat > trigger
   ```

9. Note that when the red LED is lit up, it may seem to dominate over the green colour. Therefore, you may not want to have both the red and green turned on at once.

2.2 Blinking Options

1. Change to the green LED directory
   ```
   (byai)$ cd /sys/class/leds/ACT
   ```
2. View files:
   ```
   (byai)$ ls
   brightness  device  invert  max_brightness  power  subsystem  trigger  uevent
   ```
3. Change the `trigger` to `timer`:
   ```
   (byai)$ echo timer > trigger
   ```
4. View files:
   ```
   (byai)$ ls
   brightness  delay_off  delay_on  device  max_brightness  power
   subsystem  trigger  uevent
   ```
   - Note the new files `delay_on`, `delay_off`
5. Set the timing to be on for 100ms and off for 900ms
   ```
   (byai)$ echo 100 > delay_on
   (byai)$ echo 900 > delay_off
   ```
   - This should make the green LED 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. Make a module to provide a higher-level interface to control the LEDs.
2. 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 a function to initialize an LED
     FILE *pLedTriggerFile = fopen(DA_TRIGGER_FILE_NAME_HERE, "w");
     ```
   - Check that the `fopen()` call succeed!
     ```
     if (pLedTriggerFile == NULL) {
           perror("Error opening LED trigger file");
           exit(EXIT_FAILURE);
     }
     ```
3. Write to the file the required trigger using `fprintf()`:
   ```
   int charWritten = fprintf(pLedTriggerFile, "none");
   if (charWritten <= 0) {
         perror("Error writing data to LED file");
         exit(EXIT_FAILURE);
   }
   ```
4. Close the file using `fclose()`:
   ```
   fclose(pLedTriggerFile);
   ```

3.2 Turning LED On / Off

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()`.
   - Each time you want to set the brightness, you likely have to open and close the brightness file. If you leave the file open then it won’t register your individual requests to control the LED.

3.3 Timing

When changing LED trigger, it may take a little time before Linux has all necessary files available to us. Therefore after changing the trigger, try putting in a 100ms desay.

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");

	// Sleep for 1.5s
	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 feature flag (`-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 terminal, plus discussion of GPIO.
   http://robotic-controls.com/book/export/html/69