

# GPIO Guide

by Brian Fraser

Last update: Feb 4, 2021

## This document guides the user through:

1. Reading and writing to GPIO on the BeagleBone via the command line terminal.
2. Using a C program to access GPIO.

## Table of Contents

1. GPIO Basics.....	2
2. Enabling a Pin for GPIO in Linux.....	2
3. Using a Pin.....	5
4. C Code.....	5
4.1 Writing.....	5
4.2 Reading.....	6
4.3 Edge Triggered.....	6
5. Useful References.....	6

## Formatting:

1. Host (desktop) commands starting with `(host)$` are Linux console commands:  
`(host)$ echo "Hello world"`
2. Target (board) commands start with `(bbg)$`:  
`(bbg)$ echo "On embedded board"`
3. Almost all commands are case sensitive.

## Revision History:

- Jan 15, 2021: Initial version for 2021
- Jan 18: Corrected `pFFile` typo in sample code.
- Jan 20: Removed mention of exporting pins for assignments.
- Jan 25: Corrected table name to P8/P9
- Feb 4: Added info on using GPIO via edge-triggered

## 1. GPIO Basics

General Purpose Input Output (GPIO) is using digital hardware pins. GPIO allows you to:

- 1) configure a pin for reading, and then read its state: either on (3.3V) or off (0V), or
- 2) configure a pin for output, and then drive the pin high (3.3V by writing a 1) to source current, or low (0V by writing a 0) to sink current.

A pin is either in input mode, or in output mode, but not both.

For input, since the pin is digital, it cannot tell you the voltage is 1.6V as it only returns 0 or 1. Also, if a pin is not connected to anything (called floating), it will still read 0 or 1 (perhaps seemingly randomly). Some circuits add hardware to pull-up floating inputs to 3.3V, thus always giving a 1 when floating instead of reading random garbage. Likewise, some circuits use hardware to pull-down the value to 0 when its floating. Pull-ups or pull-downs are “weak” enough (higher resistance) that when a real signal is connected to the pin, the pin reads that signal instead of the pull-up/down.

The BeagleBone Green hardware GPIO pins have a very limited ability to source (deliver) current (6mA<sup>1</sup>), and sink (accept) current (8mA). If you are trying to drive external hardware, be very careful!

## 2. Enabling a Pin for GPIO in Linux

All GPIO pins are controlled through Linux. First we must tell Linux that a pin is going to be used for GPIO (vs any of the other functions it can support).

1. Determine the Linux GPIO number for the pin of interest. On the BeagleBone, the P8 and P9 expansion headers allow easy access, and their Linux GPIO numbers are shown below.

# 65 possible digital I/Os

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BTN	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
GPIO_5	17	18	GPIO_4	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
GPIO_3	21	22	GPIO_2	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_113	GPIO_86	27	28	GPIO_88
GPIO_111	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
GPIO_110	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

- 1 The following pins are limited to sourcing 4mA: P9\_19, P9\_20, P9\_24, P9\_26, P9\_41, P9\_42; no additional restriction on sinking current. The 3.3V power pins can source up to a total of 250mA.

Source: <http://beagleboard.org/support/bone101>

- For the Zen cape, the GPIO signals correspond to the following Linux GPIO numbers:

Zen Cape Signal	Description	P8/P9 Pin	Linux GPIO Number
LED2BL	LED 2 blue	P9 #11	<b>30</b>
LED2RED	LED 2 red	P9 #12	<b>60</b>
LEDGRN	LED 2 green	P9 #13	<b>31</b>
JSUP	Joystick Up	P8 #14	<b>26</b>
JSRT	Joystick Right	P8 #15	<b>47</b>
JSDN	Joystick Down	P8 #16	<b>46</b>
JSLFT	Joystick Left	P8 #18	<b>65</b>
JSPB	Joystick Pushed -- A little tricky to push straight down.	P8 #17	<b>27</b>
LightStrip	Light-strip header	P8 #11	<b>45</b>
DISP_1	Alpha-numeric digit 1 drive	P8 #26	<b>61</b>
DISP_2	Alpha-numeric digit 2 drive	P8 #12	<b>44</b>

- Joystick notes:
    - “Left” is towards Ethernet jack on BeagleBone.
    - Press reasonably firmly to trigger the joystick; may hear a light click when pressed.**
    - Joystick pins may read 1 when not pressed, 0 when pressed.
- Change to the `sysfs` directory for GPIO. This file system gives access to many Linux devices:  
(bbg) \$ `cd /sys/class/gpio`
  - Tell Linux to handle the pin as GPIO by writing its “Linux GPIO number” to the `export` file.  
For example, if using LED2BL (Linux GPIO #30):  
(bbg) \$ `echo 30 > export`
    - Some pins may already be exported. If already exported this command gives the error:  
`write error: Device or resource busy`  
If it fails for this reason, then there is no problem because it is already exported.
    - If the BeagleBone has the “universal cape” loaded, it will export most of the available pins as GPIO. However, when you load another cape (such as the audio cape), then the universal cape is disabled and you must export the pins you need. Configuring capes is done via `/boot/uEnv.txt`, but discussion of this is beyond this guide.
    - After exporting a pin, it may take up to about 300ms before the kernel has the pin ready for use.
  - View the `/sys/class/gpio/` director; note new directory for `gpio30/`
    - Enter that directory:  
(bbg) \$ `cd gpio30`
    - View files:  
(bbg) \$ `ls`  
`active_low direction edge power/ subsystem@ uevent value`
  - Use the pin (next section).

6. When done using the pin, you can *optionally* disable GPIO for the pin by writing the Linux GPIO number to the `unexport` file. For example, to un-export pin 30:

```
(bbg) $ cd/sys/class/gpio  
(bbg) $ echo 30 > unexport
```

- If you un-export and then re-export a pin you may need to re-set its direction configuration. It's direction may seem to persist; however, you may need to reset it to correct errors.
- It is OK to leave a pin exported. This is reasonable in an embedded application because it will generally be doing one thing, and so there is little need to reset it to a default state.
- You cannot un-export pins which were exported by default, such as by the “universal” cape.

## 7. Troubleshooting

- If you try to export a pin and receive the error:  
`write error: Device or resource busy`  
It may mean that the pin has already been exported, or loaded at startup by a virtual cape. See if the pin is already mapped as GPIO (find the `gpio###/` folder). If present, just use it.
  - If it is present at boot on *your* system, realize that it may not be present on *all* BeagleBone's at boot. Especially the TA's while he/she is marking your assignments. Therefore, an application which uses these pins should export the pins it needs. If the export fails, it can be ignored and the program continue as the pin may already be there.
- If you try to unexport a pin and receive the error:  
`write error: Invalid argument`  
It may mean that the pin was not exported by the mechanism shown in this guide, but rather by a virtual cape, such as the universal cape. In this case, you are unable to unexport it because you never exported it via this mechanism.
  - If you do not need the pin for some use other than GPIO, then just leave it as GPIO.
  - If you do need to the pin for some use other than GPIO (for example, needing to setup the pin as a timer or serial port), then you may need to unload the virtual cape and/or change the boot configuration options for the board. This discussion is outside the scope of this guide, but is discussed in the Audio guide.

## 8. Free GPIO Pins

If you are looking to wire up your own electronics to the BeagleBone, here are some otherwise unused pins which should not conflict with the operation of the BeagleBone, or the Zen cape:

- P8-7 = Linux #66
- P8-8 = Linux #67
- P8-9 = Linux #69
- P8-10 = Linux #68
- P9-15 = Linux #48
- P9-23 = Linux #49

You can find these numbers via Derek Molloy's Beaglebone Black P8 (or P9) Header PDFs mirrored here:

<http://exploringbeaglebone.com/wp-content/uploads/resources/BBBP8Header.pdf>

<http://exploringbeaglebone.com/wp-content/uploads/resources/BBBP9Header.pdf>

- Find the pin you want on the left (“Head\_pin”), and then lookup Linux's GPIO pin number in “GPIO NO.”.
- Note one student had troubles getting his board to boot if he connected things to the following pins: P8\_31, P8\_41, P8\_43, P8\_44.

### 3. Using a Pin

1. Read a GPIO pin as input: using `gpio26` which is the joystick up on Zen cape:
  - Change to its folder:  
(bbg) \$ `cd /sys/class/gpio/gpio26`
  - Make the pin an input:  
(bbg) \$ `echo in > direction`
  - Read its value:  
(bbg) \$ `cat value`
  - Each time you read the `value` file it will return the digital value being read in for that pin.
  - You can write a 1 (or 0) to the `active_low` file to invert the values you are reading (i.e., read 0 when “high” instead of the usual 1).
2. Write to a GPIO pin as output: using `gpio30` which is the blue LED on the Zen cape:
  - Change to its folder:  
(bbg) \$ `cd /sys/class/gpio/gpio30`
  - Make the pin an output:  
(bbg) \$ `echo out > direction`
  - Write a value  
(bbg) \$ `echo 1 > value`  
or  
(bbg) \$ `echo 0 > value`
  - The value you write to the `value` file will be held until you change its value or disable GPIO. Note that hardware logic can be active low, which means you *may* need to write a 0 to turn some hardware on.

## 4. C Code

### 4.1 Writing

For each of the commands shown in the previous sections which uses `echo` to write data to a file, you can use C to do the same thing. Here is an example of echoing “30” to `/sys/class/gpio/export`:

```
// Use fopen() to open the file for write access.
FILE *pFile = fopen("/sys/class/gpio/export", "w");
if (pFile == NULL) {
    printf("ERROR: Unable to open export file.\n");
    exit(1);
}

// Write to data to the file using fprintf():
fprintf(pFile, "%d", 30);

// Close the file using fclose():
fclose(pFile);

// Call nanosleep() to sleep for ~300ms before use.
```

When exporting multiple pins, you’ll need to close the file between each export command (one pin at a time).

## 4.2 Reading

For each of the commands shown in the previous sections which uses `cat` to read data from a file, you can use C to do the same thing. Here is an example of some code reading a pin.

```
void readFromFileToScreen(char *fileName)
{
    FILE *pFile = fopen(fileName, "r");
    if (pFile == NULL) {
        printf("ERROR: Unable to open file (%s) for read\n", fileName);
        exit(-1);
    }

    // Read string (line)
    const int MAX_LENGTH = 1024;
    char buff[MAX_LENGTH];
    fgets(buff, MAX_LENGTH, pFile);

    // Close
    fclose(pFile);

    printf("Read: '%s'\n", buff);
}
```

## 4.3 Edge Triggered

It is possible to use the `epoll` syscall to have a program block until a GPIO pin changes its value. This is called being edge-triggered. See the sample code online for `edgeTrigger.c` example.

For more info, run:

```
(host)$ man epoll
```

## 5. Useful References

1. Walk-through of LEDs via terminal, plus discussion of GPIO.  
<http://robotic-controls.com/book/export/html/69>
2. BeagleBone GPIO  
<http://www.armhf.com/using-beaglebone-black-gpios/>
3. Kernel reference documents for GPIO:  
<https://www.kernel.org/doc/Documentation/gpio/gpio.txt>  
<https://www.kernel.org/doc/Documentation/gpio/sysfs.txt>