A2D Guide:
Zen POT, Photoresistor, Analog Joystick

by Brian Fraser
Last update: Feb 11, 2024

This document guides the user through

1. Reading the value of an A2D input on the BeagleBone via the command line terminal.
2. Wiring up and reading a photoresistor.
3. Wiring up and reading a 2-axis analog joystick.
4. Using a C program to access the A2D.

Guide has been tested on

BeagleBone (Target):	**Debian 11.4** (and beyond)
PC OS (host):	**Debian 11.5** (and beyond)

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

- Sept 17, 2019: Changed to support not using the cape manager.
- Jan 28: Changed how prompts are shown
- Oct 2022: Added notes on 2-axis joystick readings
- Feb 2024: Added section on using a photoresistor

1. A2D Basics

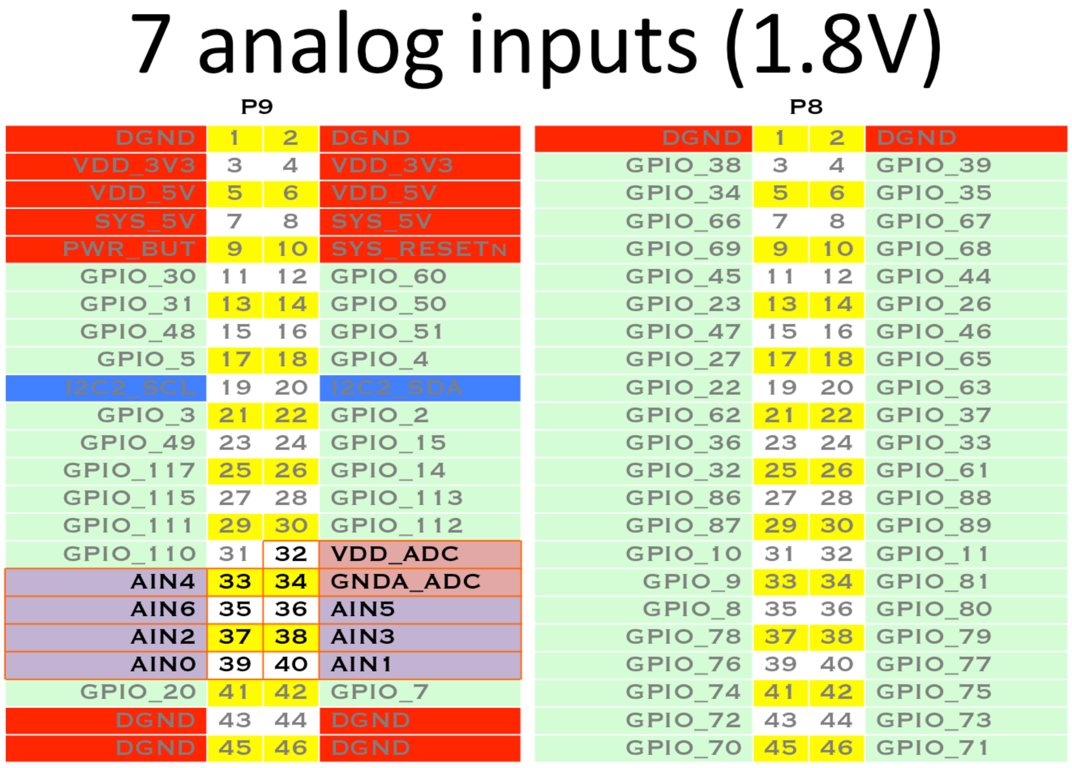
Internally in a computer, values are stored as digital: either on (1) or off (0). In reality, signals are analog, which means they are a voltage level and not just on or off. The potentiometer, for examples, is a knob the user can turn and generate a voltage between some min and max levels. The Analog to Digital converter (A2D or ADC) is used convert an analog signal (such as 1.2V) into a digital number in the computer (such as 2731).

The hardware has a limited range of voltages it can tolerate without being damaged. On the BeagleBone, this range is 0 to 1.8V. Be very careful not to exceed 1.8V on the input, even though there are 3.3V and 5V voltages also on the BeagleBone. On the Zen cape, the potentiometer (POT) has been wired to give a value between 0 and 1.8V.

2. Enabling the A2D in Linux

All A2D pins are controlled through Linux, so we must know which pin we want to access.

1. Determine which A2D input channel is being used. On the BeagleBone, the P9 expansion headers allow easy access to the 7 analog inputs.



7 analog inputs (1.8V)

P9

DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3
VDD_5V	5	6	VDD_5V
SYS_5V	7	8	SYS_5V
PWR_BUT	9	10	SYS_RESETN
GPIO_30	11	12	GPIO_60
GPIO_31	13	14	GPIO_50
GPIO_48	15	16	GPIO_51
GPIO_5	17	18	GPIO_4
I2C2_SCL	19	20	I2C2_SDA
GPIO_3	21	22	GPIO_2
GPIO_49	23	24	GPIO_15
GPIO_117	25	26	GPIO_14
GPIO_115	27	28	GPIO_113
GPIO_111	29	30	GPIO_112
GPIO_110	31	32	VDD_ADC
AIN4	33	34	GNDA_ADC
AIN6	35	36	AIN5
AIN2	37	38	AIN3
AIN0	39	40	AIN1
GPIO_20	41	42	GPIO_7
DGND	43	44	DGND
DGND	45	46	DGND

P8

DGND	1	2	DGND
GPIO_38	3	4	GPIO_39
GPIO_34	5	6	GPIO_35
GPIO_66	7	8	GPIO_67
GPIO_69	9	10	GPIO_68
GPIO_45	11	12	GPIO_44
GPIO_23	13	14	GPIO_26
GPIO_47	15	16	GPIO_46
GPIO_27	17	18	GPIO_65
GPIO_22	19	20	GPIO_63
GPIO_62	21	22	GPIO_37
GPIO_36	23	24	GPIO_33
GPIO_32	25	26	GPIO_61
GPIO_86	27	28	GPIO_88
GPIO_87	29	30	GPIO_89
GPIO_10	31	32	GPIO_11
GPIO_9	33	34	GPIO_81
GPIO_8	35	36	GPIO_80
GPIO_78	37	38	GPIO_79
GPIO_76	39	40	GPIO_77
GPIO_74	41	42	GPIO_75
GPIO_72	43	44	GPIO_73
GPIO_70	45	46	GPIO_71

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

- For the Zen cape, the potentiometer is wired to AIN0 (analog input 0)

2. Change to the `sysfs` directory for the A2D readings. The `sys` file system gives access to many Linux devices. The folder is `/sys/bus/iio/devices/iio:device0` however the ':' must be escaped for the Linux command line (but not in your C program!):[1]
   ```
   (bbg)$ cd /sys/bus/iio/devices/iio\:device0
   ```
3. To read voltage 0:
   ```
   (bbg)$ cat in_voltage0_raw
   ```
   - Change the 0 to the desired channel number.
   - The value will be between 0 and 4095 (4K – 1)

     You can compute the voltage with the formula:
     ```
     voltage = (value / max) * reference_voltage
     ```
     So, if you just read 3103 it relates to the real world voltage of:
     ```
     voltage = (3103 / 4095) * 1.8
             = 1.36V
     ```
   - *Tip: If doing this sort of math in C, make sure you use the correct data types to avoid a rather unhelpful integer division.*
4. Troubleshooting:
   - If you are changing the position of the potentiometer on the Zen cape but the value being read from `in_voltage0_raw` does not change, ensure the hardware is correctly connected. For example, if you are using the Zen cape, ensure it is fully seated on the BBG (virtually none of the P8/P9 pins should be visible between the BBG and Zen).
   - If the file `/sys/bus/iio/devices/iio:device0/in_voltage0_raw` does not exist, then double check the following in `/boot/uEnv.txt`:
     - Enabled the UBoot overlays:
       ```
       enable_uboot_overlays=1
       ```
     - Commented out (with a # in front) the A2D disable command:
       ```
       #disable_uboot_overlay_adc=1
       ```

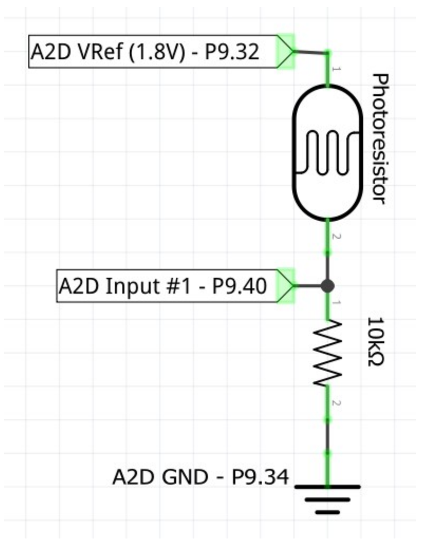
1 On older kernels, such as early 4.x, the A2D cape must be loaded using the cape manager, with a command such as:
```
 echo BB-ADC > /sys/devices/platform/bone_capemgr/slots
```
On the 4.9 kernel, this functionality is done through the UBoot bootloader, with the A2D cape loaded by default. See /boot/uEnv.txt for where to configure the device overlays in UBoot.

3. Using Light Sensor

A photoresistor (also called a photo cell, or phototransistor) is a discrete component (component we can wire into our breadboard). The resistance of the photoresistor changes depending on how much light it is exposed to. We can use an A2D voltage reading to infer the amount of light.

This photoresistor video guide shows how to wire up and use the component.

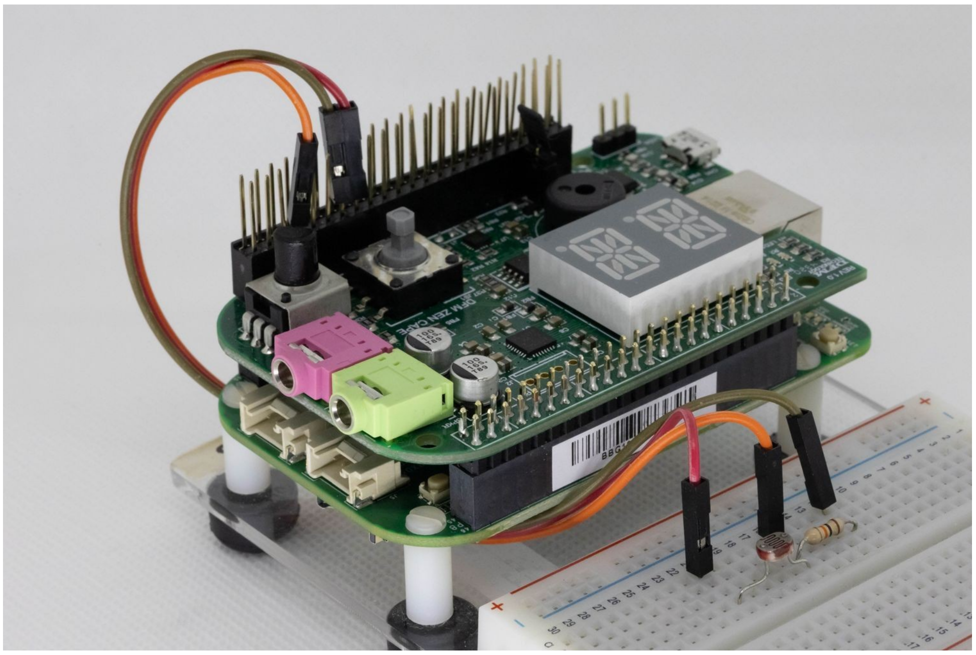
1. **Critical Tips**
   1. Do all your wiring and wiring changes with the power to the BeagleBone turned ***off***.
   2. Try to have some space between the components you place on the breadboard to prevent them from unexpectedly shorting (touching the bare wire leads).
   3. Note that it's OK for the plastic coated parts of the wires to touch; it's only the bare metal that can conduct electricity.
   4. Avoid static! Before working on the components, try grounding yourself by touching some unpainted metal on your computer, such as the USB port, or metal on a light switch.
2. Turn power off to BeagleBone.
3. If not done so already, place breadboard onto plastic mounting plate beside the BeagleBone.
   - Peel paper off back of breadboard and stick it to the mounting plate; discard extra metal plate if any.
4. The circuit we are creating is shown on the right in a schematic form. It does not matter which direction the resistors and photoresistors are placed: they can be turned around "backwards" without a problem.

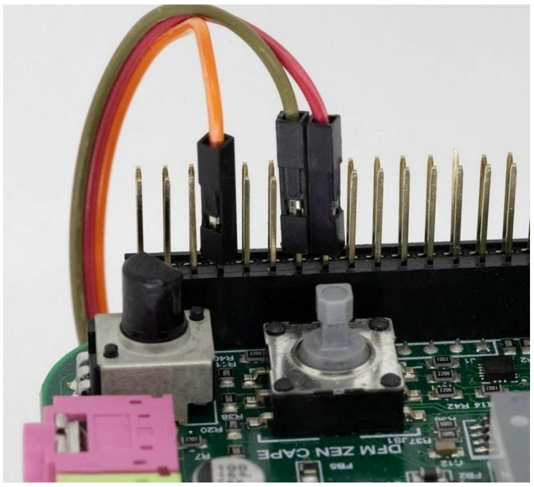


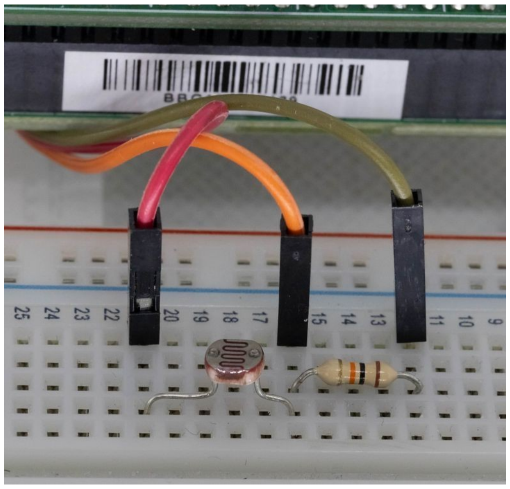
   - The photoresistor and 10k ohm resistor[2] are connected in series (one into the next). The diagram shows the photoresistor on top (it has the arrow through it), and the 10k ohm resistor on the bottom.
   - Connect the "top" of the photoresistor to:
     P9.32: 1.8V A2D reference voltage
   - Connect the "middle" between the components together, and connect this to:
     P9.40: A2D input pin #1.
   - Connect "bottom" of the 10k10 resistor to:
     P9.34: A2D ground (A2D is a precise device it has a separated ground from GPIO).
5. Use a 48-pin extension header (included in bag of loose parts) to connect wires to P9. Without it, the pins are too short to connect the wires onto directly.
   - Connect the 48-pin extension header (black with pin out the one side) by wiggling it down onto the P9 header.
   - Make sure the pins on the header to not get bend and touch (short) when transporting the board. This could damage the board.

2 It's OK to use a different strength resistor. Changing the value will adjust the range of values you read from the A2D as the light level changes. I recommend using something close to 10k ohms if possible.

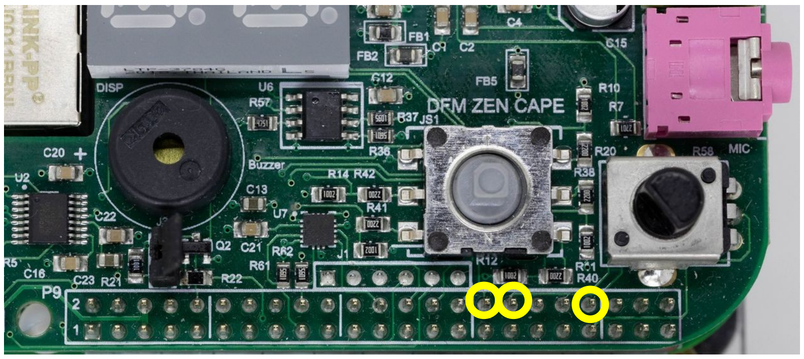
6. Here are some photos of the final wiring.
    - Photos taken from the “top” of the board, so P9 by the POT and joystick on the Zen cape.
    - Wires connect to breadboard in the order: P9.32, P9.40, P9.34.







7. To the right is a closeup of how to count the P9 header. Board in rotated so BBG is “down” and breadboard is “up.”
    - P9 pins are numbered from 1 on the left bottom.
    - Count from left, bottom row to top row. Bottom row is all odd numbers; top is even.



Location of P9.32, P9.34, and P9.40 on 48-pin P9 header.

8. Read the voltage using:
```
(bbg)$ cd /sys/bus/iio/devices/iio\:device0
(bbg)$ cat in_voltage0_raw
```
   - Try reading the voltage when exposing the sensor to direct light; note the value.
   - Try reading the voltage when blocking all light from the sensor; compare its value.
9. Troubleshooting:
   - If the value read from the A2D does not change significantly (by at least 500) between direct light and no light, try the following:
     - Ensure that the photoresistor and the 10k resistor are wired into the same column on the breadboard so that they make a connection.
     - Ensure that the A2D sensing wire is connecte to the same column as the connection between the photoresistor and the 10k resistor.
     - Try swapping each wire out with a different one (they are sometimes defective)
   - If the value read is always very low (<10) it's likely that there is a good connection to ground, but that something is wrong with connecting the photoresistor to 1.8V.
   - If the value read is always very high (>4000) it is likely that there is something wrong with the 10k resistor's connection to ground.

4. Using 2-Axis Analog Joystick

Some joysticks, such as this one by Adafruit (and its breakout board to fit in a breadboard) produce analog outputs. We can wire this up to the BBG and read the joystick.
Note: This hardware is not included in the hardware kit for this class.

1. Wire the joystick as follows:

P9.32 (`VDD_ADC`)	to +V to the joystick
P9.34 (`GNDA_ADC`)	to -V to the joystick
P9.27 (`AIN2`)	to X output on the joystick
P9.38 (`AIN3`)	to Y output on the joysticks

   (other `AINx` pins may be used)
2. Read X position:
```
(bbg)$ cat /sys/bus/iio/devices/iio:device0/in_voltage2_raw
```
   Read Y position:
```
(bbg)$ cat /sys/bus/iio/devices/iio:device0/in_voltage3_raw
```
   Note: Depending on the orientation of your joystick on your breadboard, you may need to swap the X and Y readings if it's turned 90 degrees.
3. Often a 50% value for each axis will correspond to the joystick being centred.
   You may want to scale the A2D values (0 through 4095) to be relative to -1.0 to 1.0. This conversion allows you to remap X/Y, if needed, or invert an axis if needed (multiply by -1).

   Hint: When doing conversions, print out the joystick readings and converted values to the screen to ensure code works as expected for your wiring and orientation.

5. C Code

Assuming the cape is already loaded, the following program will display the current A2D reading and the voltage it relates to until the user hits control-c:

```
// Demo application to read analog input voltage 0 on the BeagleBone
// Assumes ADC cape already loaded by uBoot:

#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>

#define A2D_FILE_VOLTAGE0  "/sys/bus/iio/devices/iio:device0/in_voltage0_raw"
#define A2D_VOLTAGE_REF_V  1.8
#define A2D_MAX_READING    4095

int getVoltage0Reading()
{
    // Open file
    FILE *f = fopen(A2D_FILE_VOLTAGE0, "r");
    if (!f) {
        printf("ERROR: Unable to open voltage input file. Cape loaded?\n");
        printf("       Check /boot/uEnv.txt for correct options.\n");
        exit(-1);
    }

    // Get reading
    int a2dReading = 0;
    int itemsRead = fscanf(f, "%d", &a2dReading);
    if (itemsRead <= 0) {
        printf("ERROR: Unable to read values from voltage input file.\n");
        exit(-1);
    }

    // Close file
    fclose(f);

    return a2dReading;
}

int main()
{
    while (true) {
        int reading = getVoltage0Reading();
        double voltage = ((double)reading / A2D_MAX_READING) * A2D_VOLTAGE_REF_V;
        printf("Value %5d ==> %5.2fV\n", reading, voltage);
    }
    return 0;
}
```

Compile with:

```
arm-linux-gnueabihf-gcc -std=c99 -D _POSIX_C_SOURCE=200809L potDriver.c -o potDriver
```