SOEN 422

Tutorial 4

Installing Adafruit_BBIO Python Libraries

- Adafruit_BBIO is a high level Python library for manipulating GPIO pins on the Beaglebone Black, much like the Arduino libraries.
- First, you may need to make sure the system time is correct:

```
sudo ntpdate pool.ntp.org
```

- Next, we need to install python and its dependencies:

```
sudo apt-get update
sudo apt-get install build-essential
  python-dev python-setuptools python-pip python-smbus -y
```

- Finally, use `pip` to install the library:

```
sudo pip install Adafruit_BBIO
```

Importing Adafruit_BBIO

- There are three main packages, one general IO, one for PWM, and one for ADC:

```python
import Adafruit_BBIO.GPIO as GPIO
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.ADC as ADC
```

- To run your python script:

```
python yourPythonScript.py
```

Installing Bonescript Library

- Bonescript is a Node.js library which also allows you to use a high level API from a Javascript environment.
- First, you must install dependencies, Node.js and NPM, the Node Package Manager.

```
sudo apt-get install -y build-essential g++ curl libssl-dev
  apache2-utils git libxml2-dev
sudo apt-get install nodejs-legacy npm -y
```

- Now we can install Bonescript globally using NPM.

```
sudo npm install -g bonescript
```

Importing Bonescript

- To import Bonescript:

```
var b = require('bonescript');
```

- To run your Javascript program:

```
node yourJavascript.js
```

Linux Devices

- On Linux, all devices are exposed as virtual files through the file system.
- Beginning in Linux kernel 3.8, hardware devices and IO pins are defined through a data structure called the device tree which is loaded at boot time.
    - The device tree describes the hardware which is available and maps them onto a virtual file system.
- The device tree can be altered after booting by loading a device tree overlay which is a segment of this data structure with redefinitions to change mux modes and enable/disable devices.

Digital Output

- Most GPIO pins can be used for digital io when in mux mode 7 (see tables at [Derek Molloy's Website](#)).
- First the pin must be exported to indicate to the OS that you wish to use it. You must write the pin number used to the export file.

Digital Output

Bash

```
echo 67 > /sys/class/gpio/export
```

C

```
#include <stdio.h>

// Open the export file
FILE *exportFile;
exportFile = fopen("/sys/class/gpio/export", "w");

// Write the pin that we want to use to the file:
fprintf(exportFile, "67");

// Close the file
fclose(exportFile);
```

Digital Output

- Next, we must the data direction

Bash

```bash
echo out >> /sys/class/gpio/gpio67/direction
```

C

```c
// Open the Pin 67 Data Direction File
FILE *dataDirection67;
dataDirection67 = fopen("/sys/class/gpio/gpio67/direction", "w");

// Write the direction as output to the file:
fprintf(dataDirection67, "out");

// Close the file.
fclose(dataDirection67);
```

Digital Output

- Finally, we must write the value to output.

Bash

```
echo 1 >> /sys/class/gpio/gpio67/value
```

C

```
// Open the Pin 67 value file
FILE *value67;
value67 = fopen("/sys/class/gpio/gpio67/value", "w");

// Write the direction as HIGH (1), for LOW, use 0.
fprintf(value67, "1");

// Close the file.
fclose(value67);
```

Digital Output

- With the Python library:

```
import Adafruit_BBIO.GPIO as GPIO

GPIO.setup("P8_10", GPIO.OUT)
GPIO.output("P8_10", GPIO.HIGH)
GPIO.cleanup()
```

Digital Output

- With Bonescript:

```
var b = require('bonescript');

b.pinMode('P8_13', b.OUTPUT);

b.digitalWrite('P8_13', b.HIGH);
```

Digital Input

- The procedure is the same as for output, except for direction, we write "in" to the direction file, and we read the value file for the pin's value.

Bash

```bash
cat /sys/class/gpio/gpio67/value
```

C

```c
// Open the Pin 67 value file
FILE *value67;
value67 = fopen("/sys/class/gpio/gpio67/value", "r");

// Write the direction as HIGH (1), for LOW, use 0.
char ch = fgetc(value67);

// Close the file.
fclose(value67);
```

Digital Input

- With the Python library:

```python
import Adafruit_BBIO.GPIO as GPIO

GPIO.setup("P8_14", GPIO.IN)

if GPIO.input("P8_14"):
    print("HIGH")
else:
    print("LOW")
```

Digital Input

- With Bonescript:

```
var b = require('bonescript');

b.pinMode('P8_19', b.INPUT);

var callbackFunction = function (readvalue) {
  //Do something with readvalue
}

b.digitalRead('P8_19', callbackFunction);
```

Analog Output

- Analog Output is done in a similar fashion to Digital Output, except we must activate the PWM module and change the mux mode of a pin using a device tree overlay.

Bash

```
echo am33xx_pwm > /sys/devices/bone_capemgr.9/slots
echo bone_pwm_P9_14 > /sys/devices/bone_capemgr.9/slots
```

C

```
FILE *slots;
slots = fopen("/sys/devices/bone_capemgr.9/slots", "w");

// Activate the am33xx_pwm module
fprintf(slots, "am33xx_pwm");

// Activate the pwm mux mode on P9_14
fprintf(slots, "bonw_pwm_P9_14");

// Close the file.
fclose(slots);
```

Analog Output

- After activating the PWM module and setting the mux mode, a new directory should be available to manage the PWM of that pin.
- There are a few files here of note:
- `period`: Contains the period of the PWM signal in nanoseconds.
- `duty`: The duty cycle: ie. if period is 500000 and duty is 250000, then duty cycle is 50%.
- `run`: 1 if the pwm is on, 0 otherwise
- `polarity`: if 1: the signal starts at 3.3V for the duration of the duty, if 0: the signal starts at 0V and raises to 3.3V after the duration of the duty.

Analog Output

- An example of setting a 50% duty cycle:

Bash

```
echo 500000 > period
echo 250000 > duty
echo 1 > run
```

C

```
FILE *period;
FILE *duty;
FILE *run;

period = fopen("/sys/devices/ocp.3/pwm_test_P9_14.16/period", "w");
duty = fopen("/sys/devices/ocp.3/pwm_test_P9_14.16/duty", "w");
run = fopen("/sys/devices/ocp.3/pwm_test_P9_14.16/run", "w");

fprintf(period, "500000");
fprintf(duty, "250000");
fprintf(run, "1");

fclose(period);
fclose(duty);
fclose(run);
```

Analog Output

- With the Python library:

```
import Adafruit_BBIO.PWM as PWM

# 50% duty cycle.
PWM.start("P9_14", 50)
```

- Changing the duty cycle:

```
PWM.set_duty_cycle("P9_14", 25.5)
```

- Stopping PWM

```
PWM.stop("P9_14")
PWM.cleanup()
```

Analog Output

- With Bonescript:

```
var b = require('bonescript');

//analogWrite(pin, dutyCycle, frequency, callback)
// Frequency defaults to 2kHz, callback is optional.
b.analogWrite('P9_14', 0.7)
```

Analog Input

- The Beaglebone Black has 7 Analog Input pins: AIN0 - AIN7.
    - These are found on pins P9_33 and P9_35 - P9_40
    - **NOTE: The max input voltage is 1.8 V**
- First, we must activate the ADC device by loading a device tree overlay:

Bash

```bash
echo cape-bone-iio > /sys/devices/bone_capemgr.9/slots
```

C

```c
FILE *slots;
slots = fopen("/sys/devices/bone_capemgr.9/slots", "w");

// Activate the cape-bone-iio module
fprintf(slots, "cape-bone-iio");

// Close the file.
fclose(slots);
```

Analog Input

- Now, we can read the value from any of the ADC pins:

Bash

```
cat /sys/devices/ocp.3/helper.15/AIN1
```

C

```
FILE *AIN1;
char line[20];

AIN1 = fopen("/sys/devices/ocp.3/helper.15/AIN1", "r");

//Read the line from the file.
fgets(line, 20, AIN1);

// Close the file.
fclose(AIN1);
```

Analog Input

- With the Python library:

```
import Adafruit_BBIO.ADC as ADC

ADC.setup()

value = ADC.read("P9_40")

# Also Valid:
value = ADC.read("AIN1")
```

- **Note: There is a known bug where you may need to read values twice to get the latest value.**

Analog Input

- With Bonescript:

```
var b = require('bonescript');

b.analogRead('P9_36', printStatus);

function printStatus(readValue) {
  // Do something with readValue
}
```

References

- Derek Molloy
- Adafruit
- Bonescript

Copyrights