Contents

Justin's Beaglebone C I/O library (24-Volt DC Motor project) **1**

Summary . . . 1

Picture of setup . . . 2

Quick Start . . . 3

Configure Beaglebone, build, & run . . . 3

Turn motor in C . . . 4

Proportional controller in C . . . 4

Hardware setup . . . 5

Software . . . 7

C functions provided . . . 8

Details / Notes . . . 8

Background: sysfs entries . . . 8

Hardware setup notes . . . 9

Sign conventions . . . 9

Motor specifications . . . 9

Troubleshooting . . . 9

Justin's Beaglebone C I/O library (24-Volt DC Motor project)

This project lives on my GitHub at https://github.com/justinpearson/Beaglebone-Motor-Demo

Summary

Here is a very simple DC motor demo for the Beaglebone Black. It reads the motor shaft angle with a EQEP-based rotary encoder and drives the motor with PWM through a motor driver. The motor driver draws power through a disused desktop power supply's 12-Volt line.

The purpose of this demo is to show that **it's easy to use C to interface with the PWM, GPIO, and EQEP sysfs entries** that Beaglebone provides to access I/O from userspace. Only standard syscall functions like `open`, `close`, `read`, and `write` are needed to do I/O in C. Neato!

Some Python code is also provided for comparison. It uses the built-in Adafruit_BBIO library and Nathaniel Lewis's eqep.py module.

A lot of stuff is hard-coded for expediency. This makes it easy for newcomers to learn how to use C to interface with the Beaglebone's sysfs entries, without getting bogged down with C++ classes or device-tree overlays.

Picture of setup

Here's some pictures of the hardware.

Quick Start

Configure Beaglebone, build, & run

```
ssh debian@10.42.0.123
sudo su
date -s "13 Dec 2013 13:43"
export SLOTS=$(find /sys/devices -name slots)
echo am33xx_pwm > $SLOTS
echo bone_pwm_P8_34 > $SLOTS
echo bone_eqep1 > $SLOTS
echo 70 > /sys/class/gpio/export
echo 73 > /sys/class/gpio/export
./build
./test-pid
```

Turn motor in C

```c
#include "bb-simple-sysfs-c-lib.h"
void main() {
  setup();

  printf("Shaft angle BEFORE (deg): %lf\n", shaft_angle_deg());
  duty(50); // 50% duty cycle
  cw();     // clockwise
  unstby(); // disable 'stby' GPIO on motor driver
  run();    // set 'run' sysfs entry for PWM

  sleep(1); // let it run for a sec.

  duty(0);  // set 'duty' to 0
  stop();   // turn off 'run'
  stby();   // set 'stby' GPIO on motor driver

  printf("Shaft angle AFTER (deg): %lf\n", shaft_angle_deg());

  shutdown();

}
```

Proportional controller in C

```c
int main ( int argc, char *argv[] ) {

  setup();
  unstby();
  run();
  cw();

  double kp = -.015;
  double dt = 0.1;      // sec, time per iteration
  double max_time = 10; // sec, max time of sim
  int num_iters = max_time / dt;
  double freq = 1;      // Hz, controls how fast the reference angle changes

  int i=0;
  for( i=0; i<num_iters; i++ ) {
    double angle = shaft_angle_deg();
    double ref = 180 * sin(2.0 * M_PI * freq * dt * i); // deg
    double error = ref-angle;
    double v = kp * error;
    voltage(v);
    usleep(dt*1000000.0);
  }

  stop();
  stby();
  shutdown();
```

```
return 0;
}
```

Hardware setup

The hardware consists of:

- DC motor (Globe Motors 405A336)
- Motor driver (LMD18201T)
- Rotary encoder (US Digital)
- Beaglebone Black
- Dell desktop power supply
- 2 10-nF capacitors for motor driver
- Two 2N3906 transistors used for 5V-to-3.3V level-shifting the EQEP sensor

The wiring schematic is shown here:

24V motor
+ Beaglebone
Setup
-Justin Pearson
Dec 13, 2016
Dell desktop power supply
red 5V
black GND
black GND
yellow 12V
LMD18201T
Motor driver
orange bootstrap 1
red out1 2
white dir 3
white brake 4
green pwm 5
blue Vs 6
black pwr gnd 7
black sig gnd 8
n/c therm 9
black out2 10
orange bootst. 11
10uF
low = cw
high = ccw
low = motor turns
high = motor stops
10uF
to BB gnd: P9_01
5V-to-3.3V level shifter
2N3906 transistor
green
yellow
orange
3.3V BB P9-03
10k
1k
chA
gnd BB
same
green
yellow
orange
3.3V BB P9-03
10k
1k
chB
BB gnd
red
black
"Globe Motors"
24V motor
rotary encoder
molex connector
BB
PWM : P8_34
GPIO motor direction : P8_45
GPIO stby : P8_44
EQEP chA : P8_33
EQEP chB : P8_35
+3.3V : P9_03
+5V : P9_07
GND : P9_01
Rotary encoder (Molex connector)
1 gnd black
2 index white
3 chA orange
4 +5V red
5 chB yellow

In particular, note that:

- The motor driver has inputs for PWM, direction, and brake.
- Pin P8_34 is the PWM.
- Pin P8_45 (GPIO) ctrls motor direction.
- Pin P8_44 (GPIO) ctrls motor brake (standby).
- The rotary encoder puts out 5V, but the BB's GPIOs require 3.3V; the transistor circuits perform level-shifting from 5V to 3.3V.
- The rotary encoder's EQEP signal is read by the BBB's EQEP peripheral.

Software

The file `bb-simple-sysfs-c-lib.c/h` provides a very thin C interface to the Beaglebone Black's PWM, GPIO, and EQEP sysfs entries.

For expediency, I hard-coded the sysfs entries for the PWM, two GPIOs, and EQEP in `bb-simple-sysfs-c-lib.h`:

```
#define PWM_PATH           "/sys/devices/ocp.3/pwm_test_P8_34.18/"
#define GPIO_MOTORDIR_PATH "/sys/class/gpio/gpio70/"
#define GPIO_STBY_PATH     "/sys/class/gpio/gpio73/"
#define EQEP_PATH          "/sys/devices/ocp.3/48302000.epwmss/48302180.eqep/"
```

If these sysfs directories don't exist, execute the following lines to create them:

```
$ export SLOTS=$(find /sys/devices -name slots)
$ echo am33xx_pwm > $SLOTS
$ echo bone_pwm_P8_34 > $SLOTS
$ echo bone_eqep1 > $SLOTS
$ echo 70 > /sys/class/gpio/export
$ echo 73 > /sys/class/gpio/export
```

Notes:

- the `slots` file on my machine lives at `/sys/devices/bone_capemgr.9/slots`.
- The Exploring Beaglebone book's Fig 6-6 shows that P8_45 (that I connected to the motor driver's "direction" pin) is GPIO 70, and P8_44 (I connected to "brake" / standby) is GPIO 73.
- The EQEP directory may be named slightly different; find the precise one with

  ```
  $ find /sys/devices/ -iname "*qep*"
  ```

- The same goes for the PWM; use `find /sys/devices/ -name duty` to find it.
- Running Python's Adafruit library wipes out the sysfs entries, e.g, Adafruit_BBIO.PWM.cleanup(), so you will have to re-echo them to recreate them.

The motor driver draws power from a Dell desktop power supply's 12V line. I hard-coded the PWM period to 50kHz. The rotary encoder seems to have a resolution of 1500 lines per revolution:

```
#define MAX_VOLTAGE 11.7 // Volts, Dell desktop power supply
#define NS_PER_PWM_PERIOD 20000 // ns per PWM period
#define NS_PER_PWM_PERIOD_STR "20000"
#define EQEP_PER_REV 1500 // I counted by hand, rough estimate
```

C functions provided

- PWM
  - `stop()` / `run()`: write 0 / 1 to the "run" PWM sysfs entry
  - `rawduty(char* c, int len)`: write a string to the "duty" sysfs file: "20000" is 0% duty cycle, "0" is 100% duty cycle
  - `duty( double d )`: write 0 - 100% to the "duty" sysfs file
  - `voltage( double v)`: convert voltage v into a duty cycle & GPIO direction and change them appropriately
- GPIO
  - `stby()` / `unstby()`: set P8_44 to 1 / 0
  - `cw()` / `ccw()`: set P8_45 to 1 / 0
- EQEP
  - `int eqep_counts()`: read eqep "position" file as an int
  - `double shaft_angle_deg()`: gets eqep position and converts to degrees

Details / Notes

- Note: PWM period is set in something like `/sys/devices/ocp.3/pwm_test_P8_34.18/period` with units of "ns per PWM cycle".
- Note: in sysfs, the 'duty' file is given in ns, not %. Ex: if period is set to 20000 (ns), then duty takes value between 0 (for 100% duty cycle) to 20000 (for 0% duty cycle)
- Note: polarity is switched on pwm:
- to do 0% duty cycle, you must write same value to

  `/sys/devices/ocp.3/pwm_test_P8_34.18/duty`

  as you wrote to

  `/sys/devices/ocp.3/pwm_test_P8_34.18/period`.
- To get 100% duty, must write 0 to duty.
- Note: instead of using the GNU C lib functions 'fread' and 'fwrite', which may do some buffering, it is more direct to use the syscalls 'read' and 'write'.
- Note: considering using 'pread' and 'pwrite' instead of 'read' and 'write'. 'p' is for 'position', and in many examples I see folks seeking the read/write head to the beginning of the file for some reason? we need to rewind the reads (any maybe writes) to the beginning of the files each time? read/write seems to work for me, except for reading the eqep, where I use pread.

Background: sysfs entries

The BBB uses a sysfs filesystem to provide a userspace interface to the hardware. For example, set up a 50kHz PWM on pin P8_34 like this:

```
echo bone_pwm_P8_34 > /sys/devices/bone_capemgr.9/slots (or wherever your slots are)
```

That creates the directory

```
/sys/devices/ocp.3/pwm_test_P8_34.18/   (your .18 may be different)
```

with files like `duty`, `period`, and `run`. Now turn on the PWM:

```
echo 20000 > period  # 20000 ns per PWM cycle => 50kHz
echo 10000 > duty    # 50% duty cycle
echo 1 > run
```

Hardware setup notes

cw 1 rev: eqep changes by -1450

ccw 1 rev: eqep changes by 1500

- gearbox inside?

stby low: motor turns; hi: motor stops

pwm 10%: just barely turns. stutters. sometimes stops

dir pin low: motor turns cw; high: ccw

Sign conventions

For the `shaft_angle_deg()` and `voltage()` functions:

- ccw is positive angle
- positive motor voltage turns motor cw
- cw 1 rev => -1500 encoder ticks

Motor specifications

- motor coil resistance: 14 ohms
- motor coil inductance: 11.52 mH
- rotary encoder: 1500 lines / rev, roughly

Troubleshooting

The EQEP driver isn't included in the stock BBB kernel, so `echo bone_eqep1 > $SLOTS` will fail in `dmesg`; update kernel to latest with

```
cd /opt/scripts/tools/
git pull
sudo ./update_kernel.sh
sudo reboot
```

(Source: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian)

Now you should have

```
# find /lib/firmware -iname "*qep*"
/lib/firmware/bone_eqep0-00A0.dtbo
/lib/firmware/bone_eqep1-00A0.dtbo
/lib/firmware/bone_eqep2b-00A0.dtbo
/lib/firmware/bone_eqep2-00A0.dtbo
```

My `uname -a` shows:

```
Linux beaglebone 3.8.13-bone81 #1 SMP Fri Oct 14 16:04:10 UTC 2016 armv7l GNU/Linux
```

Make a shell variable `SLOTS` pointing to your `slots` file that organizes your DTOs:

```
$ export SLOTS=$(find /sys/devices -name slots)
```

On my BB, `$SLOTS` is `/sys/devices/bone_capemgr.9/slots`.

Load Device Tree Overlays:

```
$ echo am33xx_pwm > $SLOTS
$ echo bone_pwm_P8_34 > $SLOTS
$ echo bone_eqep1 > $SLOTS
```

Have them added automatically by adding to `/boot/uboot/uEnv.txt`:

```
optargs=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN \
   capemgr.enable_partno=BB-ADC,bone_pwm_P8_34,am33xx_pwm,bone_eqep1
```