PWM Guide

Zen Buzzer, Discrete LED, and Zen Tri-Colour LEDs

by Brian Fraser

Last update: Feb 10, 2024

Guide has been tested on

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

This document guides the user through:

1. Loading PWM support.
2. Driving the Zen cape's buzzer via PWM from a Linux terminal.
3. Driving a discrete LED via PWM.
4. Driving the Zen cape's tri-colour LED via three PWM channels from the Linux terminal.

Table of Contents

Formatting:

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

Revision History:

- Nov 1, 2021: Initial version.
- Mar 9, 2023: Updated for Bullseye (kernel 5.10).
- Feb 11, 2024: Added section on discrete LED for PWM.

1. PWM Basics

Pulse-width modulation (PWM) is a way of generating a digital wave form (think of a clock signal). You can specify two main components of the digital wave form:

1. Period: How much time is there between the start of one cycle and the next. This is the time between rising edges of the wave form.
2. Duty: This is the percentage of the cycle which the signal is high (or low, depending on its configuration).

Together, these two parameters allow you to generate waves such as those shown in Figure 1.

In some situations an analog voltage is needed. A PWM wave can be used to create such a voltage by applying extra hardware (capacitors) to smooth out, or average out, the wave form. For example, when the signal is between 0 and 3.3V, a 50% duty cycle would average out to 1.65V (half of 3.3V).

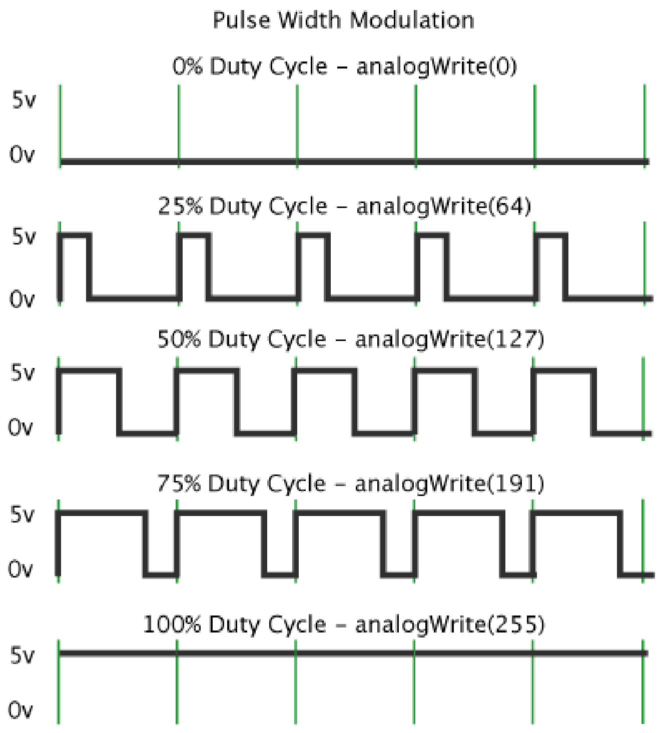


Figure 1: PWM wave forms for different duties, from https://www.arduino.cc/en/Tutorial/PWM

The BBG's PWM channels are listed below. Some channels are used by the Zen cape.

Zen Cape Use	**PWM Channel**	**BBB Pin**	**Linux Path**[1]	**Notes**
Buzzer	PWM-0A	P9-22	/dev/bone/pwm/**0/a/**	These two share a PWM timer: Period must be the same; duty cycle is independently.
unused	PWM-0B	P9-21	/dev/bone/pwm/**0/b**	
Blue LED	PWM-1A	P9-14	/dev/bone/pwm/**1/a**	These two share a PWM timer.
Red LED	PWM-1B	P9-16	/dev/bone/pwm/**1/b/**	
Green LED	PWM-2A	P8-19	/dev/bone/pwm/**2/a**	These two share a PWM timer.
unused	PWM-2B	P8-13	/dev/bone/pwm/**2/b**	

PWM channels which share hardware timers (such as 0A and 0B) cannot change their period independently; however, they can vary their duty cycle independently. See section 4. for more.

General info on the BBG's PWM can be found here:
https://elinux.org/Beagleboard:BeagleBone_cape_interface_spec#PWM

1 `/dev/bone/pwm/` folders links to `/sys/class/pwm/pwmchipX/`; where X is {3, 5, or 7} for PWM {0, 1, 2}

2. Linux PWM: Zen Cape Buzzer

NOTE ON SILENCE: The buzzer can be manually turned off by removing the jumper (black rectangle) just below the buzzer (left of the joystick) on the Zen cape.

1. Configure the necessary pins as PWM for the Zen Cape's Buzzer:
   ```
   (bbg)$ sudo config-pin p9_22 pwm
   ```
   - This configures the CPU's pin connected to the buzzer for use with PWM (each pin has multiple possible functions to select from).
   - You can list all the functions the pin can be configured for using:
     ```
     # config-pin -l P9_22
     ```
   - You can check the current state of the pin:[2]
     ```
     # config-pin -q P9_22
     ```
4. View the PWM files in the sysfs:
   ```
   (bbg$) cd /dev/bone/pwm/0/a/
   (bbg$) ls
   capture  duty_cycle  enable  period  polarity  power  uevent
   ```
5. Set the period of a cycle (via `period`, in ns), duration of each "on" pulse (`duty_cycle`, in ns), and if it is running (`run`, a 1 for on):
   ```
   (bbg$) echo 1000000 > period
   (bbg$) echo 500000  > duty_cycle
   (bbg$) echo 1       > enable
   ```
   - You can manually silence the buzzer by pulling the jumper beside it. When put the jumper back in, make sure you connect it to the correct pins! (not the big P8 header!)
   - Note on times:
     1 second
     = 1,000 miliseconds [ms]
     = 1,000,000 microseconds [us]
     = 1,000,000,000 nano-seconds [ns]
   - For the buzzer, an easy way to work is always make the `duty_cycle` half of the `period`. It's the period that controls the frequency of the sound played (the note).
6. Turn off with:
   ```
   (bbg$) echo 0 > enable
   ```
7. Make it play a lower pitched sound by giving it a larger period (ns):
   ```
   (bbg$) echo 2000000 > period
   (bbg$) echo 1000000 > duty_cycle
   ```
   - Ensure it's enabled (write a 1 to `enable`) for sound to be generated.
8. Make it play a higher pitched sound by giving it a smaller period. Since the period cannot be less than the duty cycle (period cannot be on longer than a single cycle), we must first set the duty to be less than we want the period to be:
   ```
   (bbg$) echo 0 > duty_cycle
   (bbg$) echo 100000 > period
   (bbg$) echo 50000 > duty_cycle
   ```

2 This is similar to the following command:
```
# cat /sys/devices/platform/ocp/ocp\:P9_22_pinmux/state
```

9. You can play specific notes on the buzzer. First find the frequency for the note you want (try online) and then compute the period by:
   Period = (1 / Frequency [cycles per s]) * 1,000,000,000 [ns per s]

   Set the duty to be half of the period.

   - For example, middle C is 261.6Hz. This gives a period of 3,822,256ns and duty 1,911,128ns:
     ```
     (bbg$) echo 0       > duty_cycle
     (bbg$) echo 3822256 > period
     (bbg$) echo 1911128 > duty_cycle
     ```

10. Troubleshooting:

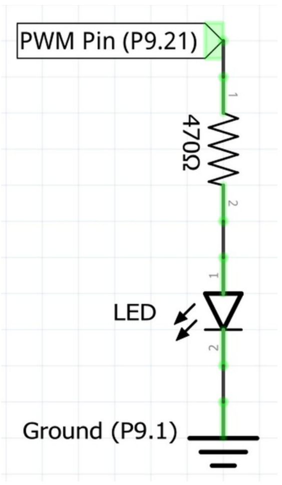
   - If `config-pin` fails with a message like the following, it likely means that the universal cape is not loaded, or the pin is mapped to another cape. Ensure you are not loading any unnecessary capes in `/boot/uEnv.txt` (should be fine with audio cape).
     ```
     $ sudo config-pin p9_22 pwm
     P9_22 pinmux file not found!
     bash: /sys/devices/platform/ocp/ocp*P9_22_pinmux/state: No such file or
     directory
     Cannot write pinmux file:
     /sys/devices/platform/ocp/ocp*P9_22_pinmux/state
     ```
   - No sound?
     - Ensure you have the jumper correctly installed connecting the two pins just below the buzzer (left of the joystick).
     - Ensure you have the PWM pins configured (with `config-pin`)
     - Ensure you have it running (`echo 1 > enable`)
     - Ensure you have set the duty to be half of the period.
   - While trying to change the period (or other operations), if you get:
     `"bash: echo: write error: Invalid argument"`
     - You may be trying to change `period` to a value less than the duty cycle. First change `duty_cycle` to 0, then retry your change.
     - You may be trying to change the period for a PWM channel when the other channel that shares that PWM timer has already set the period (both channels running on the same timer must have the same period, but may have different duty_cycle values). For example, if you have set the period for channel 0/a, and then later try to change the period for channel 0/b, you are likely to get this warning.

       Work arounds include:
       - Set both the A or B channels to the desired period, and then set the other channel to that exact same period. You may alter the duty_cycle independently.
       - If you need to change the period of the A and B channels (must be done to same value), you can try (for PWM channel 0/A or 0/B; change to chip 5 or 7 otherwise):
         ```
         (bbg)$ cd /sys/class/pwm/pwmchip3/
         (bbg)$ echo 0 > unexport                # Disable channel 0/A
         (bbg)$ echo 123456 > ./pwm1.period
         (bbg)$ echo 1 > export                  # Period now changed.
         ```

3. PWM an LED

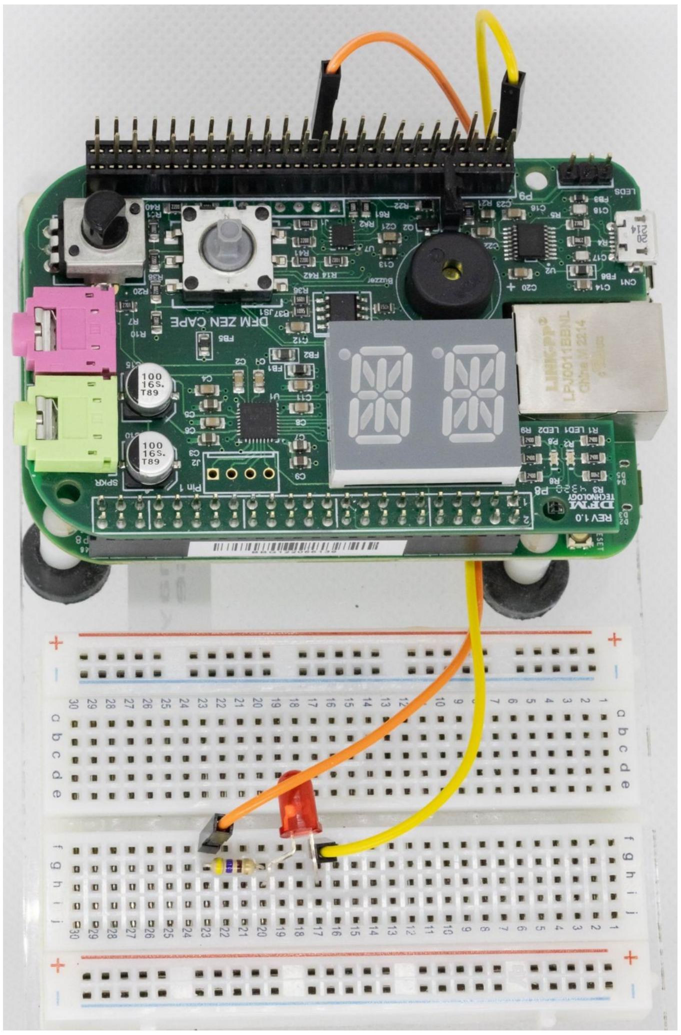
We can wire a discrete LED (single component) into a BBG PWM pin to turn on and off the LED at a desired frequency.

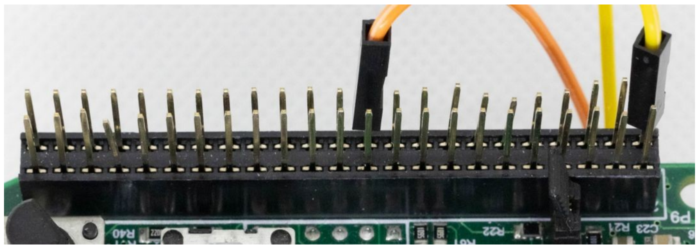
1. **Critical Tips**
   - Do all your wiring and wiring changes with the power to the BeagleBone turned ***off*** (safest!)
   - Try to have some space between the components you place on the breadboard to prevent them from unexpectedly shorting (touching the wire leads). For example, on the breadboard, wire your resistor into column A, and your LED into column E so that they don't touch.
   - 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.
   - Avoid static! Before working on the components, try grounding yourself by touching some unpainted metal on your computer, such as the USB port.
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 going to wire up is shown on the right. The LED is connected in series (one into the next) with the 470 ohm resistor. Note the LED has one long and one short lead (wire).

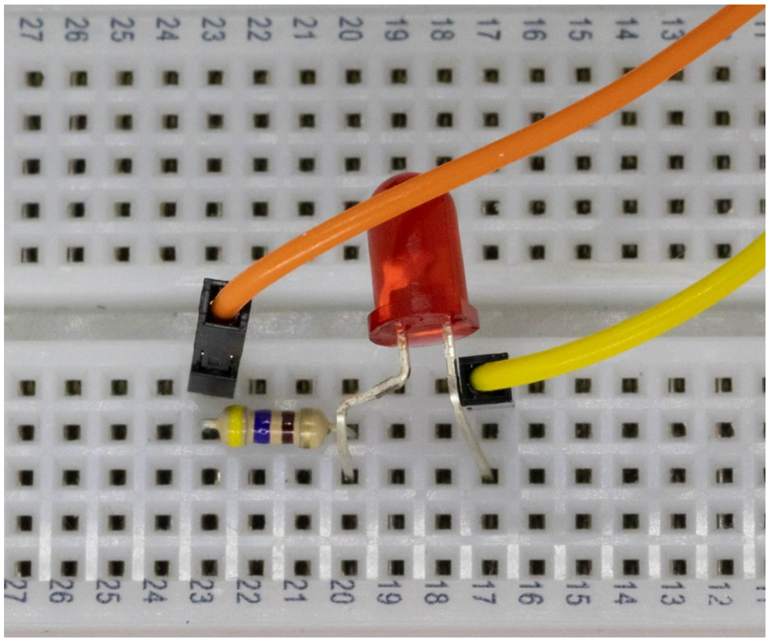


   - Connect the "top" of the 470 ohm resistor to:
     P9.21: PWM channel 0B
   - Connect the "middle" between the two components together. This will be one end of the resistor (either end), and **the longer** lead of the LED.
   - Connect the "bottom" of the LED (**the shorter** lead) to ground:
     P9.1: Ground (GND)
   - It does not matter whether the LED or the resistor is "on-top" (i.e., connected to the PWM pin).
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.

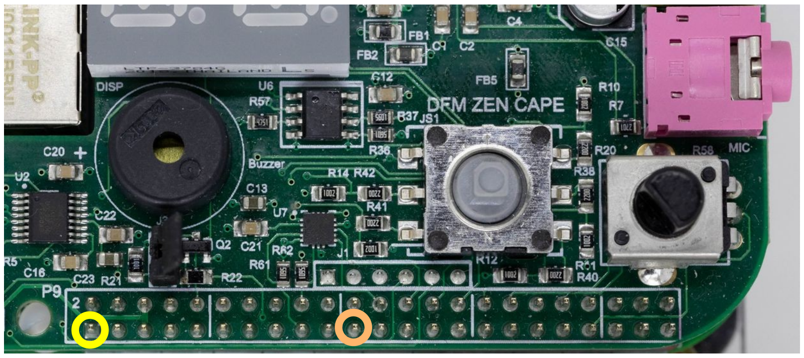
6. Here are some photos of the final wiring.
   - Photos taken from the "top" of the board, so P9 by the POT and joystick on the Zen cape.
   - Wires connect to breadboard in the order: P9.21 (PWM), P9.1 (Ground). Longer lead of LED has been bent.
   - Note the LED has been bent over onto its side so that it will flash down onto the breadboard, as needed if you are trying to have the LED light be sampled by a light sensor.







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 P9.1 (GND) and P9.21 (PWM) 48-pin P9 header.

4. Linux PWM: Zen Cape Tri-colour LED

1. Configure PWM for use:
   ```
   (bbg)$ config-pin P9_14 pwm   # red
   (bbg)$ config-pin P9_16 pwm   # green
   (bbg)$ config-pin P8_19 pwm   # blue
   ```
2. Set the period for the LED PWM channels.
   ```
   (bbg)$ echo 100000 > /dev/bone/pwm/1/b/period   # red
   (bbg)$ echo 100000 > /dev/bone/pwm/2/a/period   # green
   (bbg)$ echo 100000 > /dev/bone/pwm/1/a/period   # blue
   ```
   - The red and blue PWM channels share hardware so you cannot change the period of these channels independently. This is not an issue for us: we just need to set the duty cycle.
   - In fact, the software won't let you change the red/blue's period at all if you have both PWM channels' periods set. Specifically, you can set and change the period of one of the red or blue PWM channels until the other one is given a period. Then, you can only change the period of one to match the other.
   - So, in other words, set the period to something reasonable to start and then don't change it.
3. Enable the PWM
   ```
   (bbg$) echo 1 > /dev/bone/pwm/1/b/enable  # red
   (bbg$) echo 1 > /dev/bone/pwm/2/a/enable  # green
   (bbg$) echo 1 > /dev/bone/pwm/1/a/enable  # blue
   ```
4. LED brightness is controlled by how much time it is being turned on per cycle (the duty cycle).

Off	Set `duty_cycle` to 0
50%	Set `duty_cycle` to 50000
100%	Set `duty_cycle` to 100000

   - For example: red to 0%, green to 100%, blue to 0%:
     ```
     (bbg$) echo 0      > /dev/bone/pwm/1/b/duty_cycle
     (bbg$) echo 100000 > /dev/bone/pwm/2/a/duty_cycle
     (bbg$) echo 0      > /dev/bone/pwm/1/a/duty_cycle
     ```
5. Generate your own colours by specifying the RGB components. For example, purple is 50% red, 0% green, 50% blue.
   ```
   (bbg$) echo 50000  > /dev/bone/pwm/1/b/duty_cycle
   (bbg$) echo 0      > /dev/bone/pwm/2/a/duty_cycle
   (bbg$) echo 50000  > /dev/bone/pwm/1/a/duty_cycle
   ```
   - The colours of the LED don't mix together very well, so it can hard to create exactly the colour you have in mind. However, try putting a piece of paper over it to defuse the light and you may find it seems to mix better.

6. You may find the following script useful. Copy to a file `driveZenRGBLeds.sh`, change to executable and run with:

   ```
   # ./driveZenRGBLeds.sh 100000 50000 0
   #!/bin/sh
   # Drive the Zen RGB LED
   echo "Can pass R G B parameters (0 = off; 100,000 = on)"

   ## set -x
   ## Figure out values for writing
   RED=50000
   GREEN=50000
   BLUE=50000
   if [ $# -eq 3 ]; then
         RED=$1
         GREEN=$2
         BLUE=$3
   fi

   # Paths
   PATH_R=/dev/bone/pwm/1/b
   PATH_G=/dev/bone/pwm/2/a
   PATH_B=/dev/bone/pwm/1/a

   # Configure pins for PWM
   config-pin P9_14 pwm
   config-pin P9_16 pwm
   config-pin P8_19 pwm

   ## Setup period
   echo 100000 > $PATH_R/period
   echo 100000 > $PATH_G/period
   echo 100000 > $PATH_B/period

   ## Set the color (duty-cycle)
   echo "Red: $RED, Green: $GREEN, Blue: $BLUE"
   echo $RED   > $PATH_R/duty_cycle
   echo $GREEN > $PATH_G/duty_cycle
   echo $BLUE  > $PATH_B/duty_cycle

   ## Enable
   echo 1 > $PATH_R/enable
   echo 1 > $PATH_G/enable
   echo 1 > $PATH_B/enable
   ```

7. Troubleshooting:
   - "Permission denied" error trying to write to a file: You may need to use sudo tee, or have not yet exported the PWM for that pin.
   - "Invalid argument" when writing 1 to the `enable` file likely means you have not yet set the `period` or `duty_cycle` correctly.
   - Unable to change period of red/blue LED: This is by design, since both PWMs are linked and the period cannot be changed. It is best to not change the period of any of the LED PWM channels for this reason.