CMPT433 Project How-To Guide
Group: Harambe

Using PWM to Control External Devices

Pulse width modulation (PWM) is a great way to control external devices using the Beaglebone Green. It allows devices that expect varying voltage levels to be controlled using the digital outputs of the Beaglebone. By rapidly switching a digital signal on and off, we can approximate an analog voltage based on the percentage of time the signal is on vs the time it is off. Using PWM we can control the speed of a fan, the brightness of an LED, or the frequency of a speaker. However, most devices run at higher voltages than the 3.3V of the Beaglebone, and draw much greater current than the few milliamps each pin can provide. This guide will provide a brief overview of using MOSFETs to overcome this limitation, along with step-by-step instructions on how to enable and use the PWM outputs on the Beaglebone.
While guides exist on using PWM with the Beaglebone, this guide will focus on the mapping between physical PWM outputs and the linux sysfs file system.

Part 1: Using MOSFETs

Using an N-channel MOSFET will allow us to control high voltage and high current devices from the PWM outputs of our Beaglebone. The MOSFET acts as a switch, and instead of directly controlling our device (fan, LED, etc.), we control the MOSFET, which in turn controls our device. The MOSFET requires very little current to activate, and operates at a range of voltages low enough that we can control it directly from the Beaglebone.

Parts needed:

- 1 x IRLB8721PBF MOSFET (models with similar specifications will also work)
- 1 x 10KΩ resistor
- 1 x DC power supply (voltage based on device being controlled)

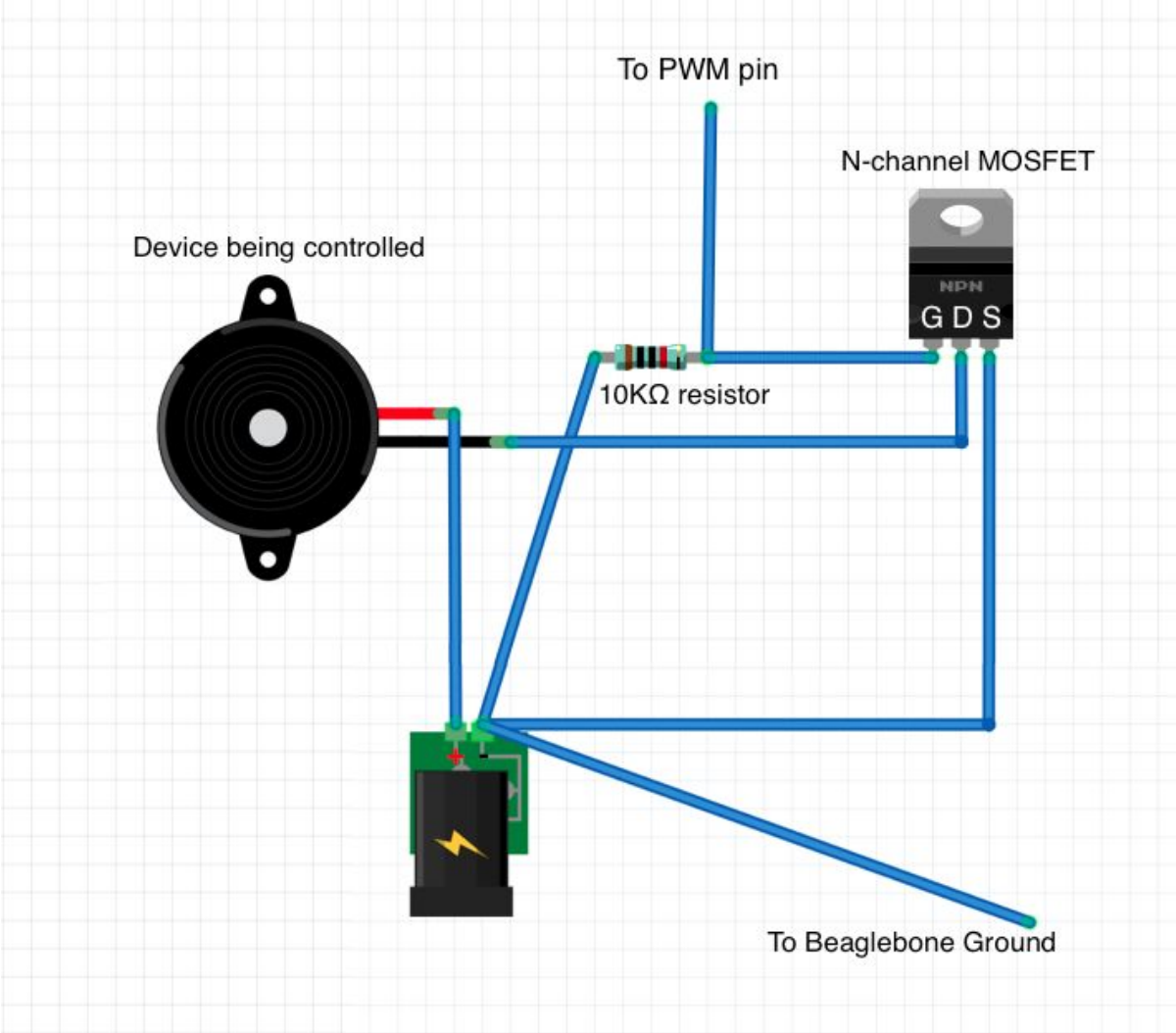
If you’re controlling an inductive load (things like motors or fans), it’s a good idea to put a schottky diode between the positive and negative terminals of the device, but that is beyond the scope of this guide.

The MOSFET will have three pins: Gate, Drain, and Source. Usually they will be in that order, from left to right, but check your specific model’s datasheet to confirm.

The functions of each MOSFET pin are as follows:

- The Gate pin will be connected to the PWM output of your Beaglebone, and switches the MOSFET on or off.
- The Drain pin connects to the negative terminal of the device you are controlling. This pin controls whether the device is connected to ground. When the MOSFET is on, the drain pin becomes connected to ground, completing the circuit and turning our device on.
- The Source pin connects directly to ground. When the MOSFET is turned on, this pin is connected to drain.

The 10KΩ resistor is placed between ground and the gate pin of the MOSFET. It is called a pull-down resistor, and ensures that the MOSFET is turned off unless we specifically output a high-logic level to the PWM pin.



Once you've connected everything according to the above circuit, you should have two wires connected to your Beaglebone: the Beaglebone PWM pin output connected to the Gate pin of the MOSFET, and the Beaglebone ground pin connected to the ground of your power supply.

Part 2: Controlling PWM using C

Using the linux sysfs file system we can easily control various hardware by writing and reading files. However, for PWM control, there is the problem that the sysfs directories corresponding to each physical pin are dynamically set, and so upon each reboot or loading of a cape, the pin represented by a sysfs folder may change.

To overcome this, at our program's runtime, we need to use a lookup between the gpio number and pin header number, and then another lookup from header number to chip address number (we could do this with one lookup, but this gives us the flexibility to address pins by either GPIO or header number). We finally look in a subdirectory of the OCP directory to find which pwmchip name our pin has been set to.

If this seems all rather confusing, don't worry too much. Using the class below, you can create a PWM object that will handle PWM sysfs for you.

https://csil-git1.cs.surrey.sfu.ca/CMPT433public/PWM/raw/master/PWM.cpp
https://csil-git1.cs.surrey.sfu.ca/CMPT433public/PWM/raw/master/PWM.h

Using the provided class, we can initialize a PWM object with:

```
PWM pwm50(51);
```

The constructor for the PWM class will load the "cape-universaln" cape (make sure you don't have any capes loaded that conflict with this cape. It will then set the duty cycle, period, polarity, and enable paths to the correct locations, as well as set the pin to pwm mode. Finally it will export the correct pwm number (0 or 1) for the pin.

Once we have created the PWM object, we can set its period and duty cycle as follows:

```
pwm50.setPeriod(10000000);
pwm50.setDutyCycle(5000000);
```

This sets the pin to a 50% duty cycle.

Finally, we enable the pin by calling enable()

```
pwm50.enable();
```

By setting different duty cycles, you can vary the speed of a fan, brightness of an LED, or pitch of a speaker!