Servo & PWM Guide: SG90 9g Micro Servo

For Linux Kernel 4.9.78-ti-r94 BeagleBone Firmware Image

by Catherine Riopel, Noel Riopel, Daniel Gomes Maia Filho - with additions and adaptations from Dr. Brian Fraser's PWM Guide.

Created: April 12, 2022

This document guides the user through:

1. Loading PWM support.
2. Wiring and schematic of a SG90 9g Micro Servo.
3. Driving a SG90 9g Micro Servo via PWM from a Linux terminal.

Table of Contents

Formatting:

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

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.

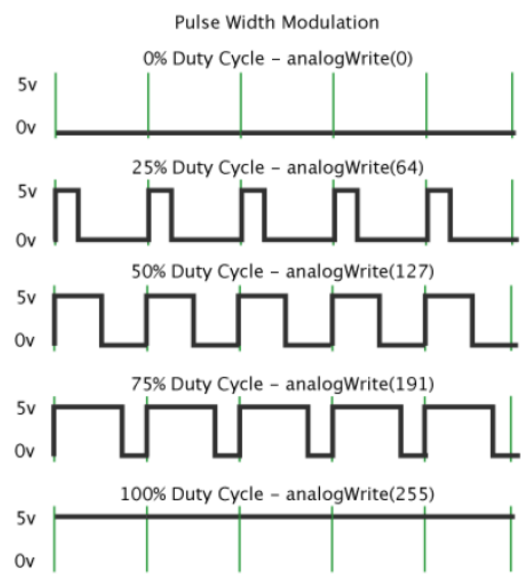


Figure 1: PWM wave forms for different duties, from https://www.arduino.cc/en/Tutorial/PWM

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).

Zen cape aside:
We will not be utilizing the Buzzer nor the LEDs for this guide, but for reference, with the universal cape enabled, the PWM channels used by the Zen cape are listed below. Note that not all PWM channels are used by the Zen cape: some are unused on the BBG, and others are used by the HDMI hardware.

Zen Cape Use	PWM Channel	BBG Pin	Linux Path	Notes
Buzzer	PWM-0A	P9-22	/sys/class/pwm/pwmchip0/pwm0/	
Red LED	PWM-1B	P9-16	/sys/class/pwm/pwmchip2/pwm1/	Shares PWM hardware timer with Blue LED (PWM-1A)
Green LED	PWM-2A	P8-19	/sys/class/pwm/pwmchip4/pwm0/	
Blue LED	PWM-1A	P9-14	/sys/class/pwm/pwmchip2/pwm0/	Shares PWM hardware timer with Red LED (PWM-1B)

Note that for PWM channels which share PWM hardware timers (red and blue), you cannot change the period of these channels independently. See Section 4 for more.

2024 Note from Dr. Brian

Changing uEnv.txt unnecessary; consult PWM guide.

2. Load PWM Device Tree – Edit uEnv.txt

Linux can learn about the PWM hardware in two ways, the universal cape, and the overlays in the device tree found in `/boot/uEnv.txt`. For this servo & PWM guide we will be utilizing device tree overlays. To load PWM overlays via uEnv.txt, do the following

1. Update the cape overlays:
   ```
   # sudo apt-get update
   # sudo apt-get install bb-cape-overlays
   ```
   - You must have an internet connection for this to succeed.
2. Ensure you have an up-to-date version (as of April 12, 2022):
   ```
   # apt-cache show bb-cape-overlays
   Package: bb-cape-overlays
   Version: 4.14.20210821.0-0~buster+20210821
   Architecture: armhf
   ....
   ```
3. Copy the current uEnv.txt file. This is necessary for recovering from anything going wrong!
   ```
   # sudo cp /boot/uEnv.txt /boot/uEnv-BeforePwm.txt
   ```
4. Edit the uEnv.txt file to enable the PWM overlays
   ```
   # sudo nano /boot/uEnv.txt
   ```
   - Edit the Additional custom capes section.
     Below shows the setup you'll have if you are already loading the audio cape and the I2C-1 cape. If you are not loading these, you may comment out those lines
     ```
     ###Additional custom capes
     uboot_overlay_addr4=/lib/firmware/BB-BONE-AUDI-02-00A0.dtbo
     uboot_overlay_addr5=/lib/firmware/BB-I2C1-00A0.dtbo
     uboot_overlay_addr6=/lib/firmware/BB-PWM0-00A0.dtbo
     uboot_overlay_addr7=/lib/firmware/BB-PWM1-00A0.dtbo
     uboot_overlay_addr3=/lib/firmware/BB-PWM2-00A0.dtbo
     ```
   - The highlighted overlays are PWM overlays, these are maintained and found in the following repository: https://github.com/beagleboard/bb.org-overlays/tree/master/src/arm.
   - To get a deeper understanding of which overlay controls which pins, take a look at the following dts files,
     P9-21(ehrpwm0B) & P9-22(ehrpwm0A):
     https://github.com/beagleboard/bb.org-overlays/blob/master/src/arm/BB-PWM0-00A0.dts
     P9-14(ehrpwm1A) & P9-16(ehrpwm1B):
     https://github.com/beagleboard/bb.org-overlays/blob/master/src/arm/BB-PWM1-00A0.dts
     P8-13(ehrpwm2B) & P8-19(ehrpwm2A):
     https://github.com/beagleboard/bb.org-overlays/blob/master/src/arm/BB-PWM2-00A0.dts
   - Note that you can use `uboot_overlay_addr0` through `uboot_overlay_addr7` for any of the capes being loaded. If you use 0-3, it replaces the cape support for any automatically detected capes configured to load at that slot. If you have no physical capes connected (which are automatically detected), then you can use any of the 8 slots you like. This is why the above snippet uses 3 through 7. The number dictates the order they are loaded, which should not matter here.
   - **You only need to load the capes you need, for this servo guide we will only be utilizing** `BB-PWM0-00A0.dtbo`.
5. Reboot the target
6. Troubleshooting
   - See the Audio guide's last section (on the course website) to recover from a corrupted `uEnv.txt`. You should be able to copy back the `/boot/uEnv-BeforePwm.txt` to restore your previous working state.

3. Servo Basics

A servo motor is a motor whose shaft position can be controlled precisely. The motor has an internal servomechanism that provides positional feedback so that the servo may know what position its shaft is in. We may control the position of the shaft with PWM.

In Figure 2 we see 3 digital wave forms for our SG90 9g Micro Servo.
To move our servo to the 0-, 90-, and 180-degree positions, we want to set the period to 20ms as shown.

But what does that mean? Let's brush up on some physics:
A **period** can be defined as the leading edge/start of one pulse to the leading edge of another pulse.
The **pulse width** can be defined as the measure of the elapsed time between the leading and trailing edge of one pulse.
A **duty cycle** is the percentage of the period that the signal is active/on.

Now we have some understanding of reading these digital wave forms, we can see that the duty cycle for one period varies between a pulse width of 1ms (0 degrees) to 2ms (180 degrees). So, by manipulating the timing for which the signal is on, we also affect the percentage of the period the signal is on, thus we affect the duty cycle.

A period of 20ms and duty cycle of 1ms will move the shaft to a 0-degree position.
We can also achieve a 90-degree position with a duty cycle of 1.5ms, and a 180-degree shaft position with a duty cycle of 2ms.

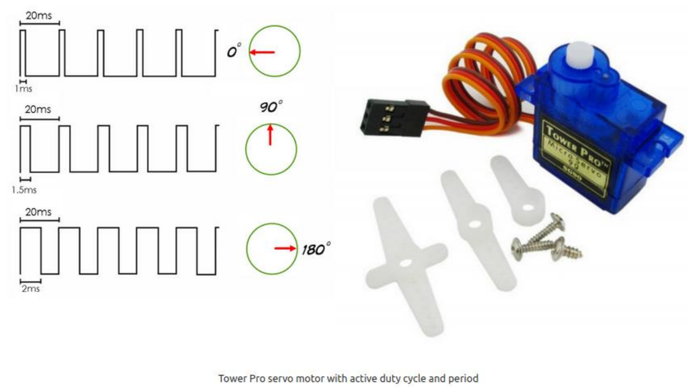


Figure 2: Image of the SG90 9g Micro Servo and its respective digital wave forms that describe period, pulse-width, and shaft degree. From, https://www.engineersgarage.com/servo-motor-sg90-9g-with-89c51-microcontroller/

4. Wiring the Servo

Understanding the schematic

Figure 3 shows a schematic of connecting a servo to the BeagleBone Green.

The servo motor has three wires, brown for ground, orange for the signal, red for power. The operating voltage needed for this servo is in between approximately 4.8V to 6V, since the BBG has a 5V pin we will use that. As such, we connect the red wire to a SYS_5V pin on the BBG either P9-7 or P9-8, the brown wire to a DGND pin (there are several on the P9 and P8 headers), and the orange wire to a EHPWM designated pin (8 available across P9 and P8). See Figure 4 for more details.

The schematic also shows two 470Ω resistors in series, which achieves a 940Ω resistance. Why this resistance?

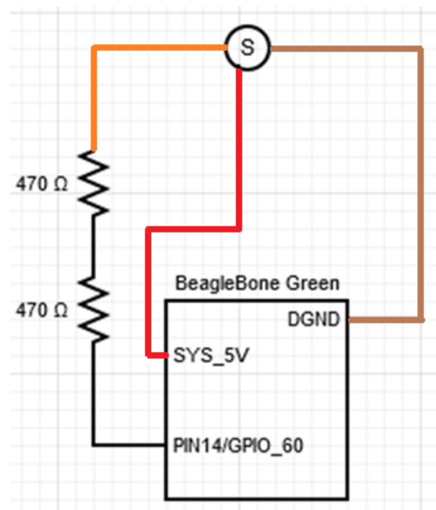


Figure 3: schematic that describes the wiring of the servo to the BBG include resistance and BBG pins needed.

Here is the physics of the “servo to resistor to resistor to GPIO pin” series:

Total resistance = 470Ω + 470Ω = 940Ω

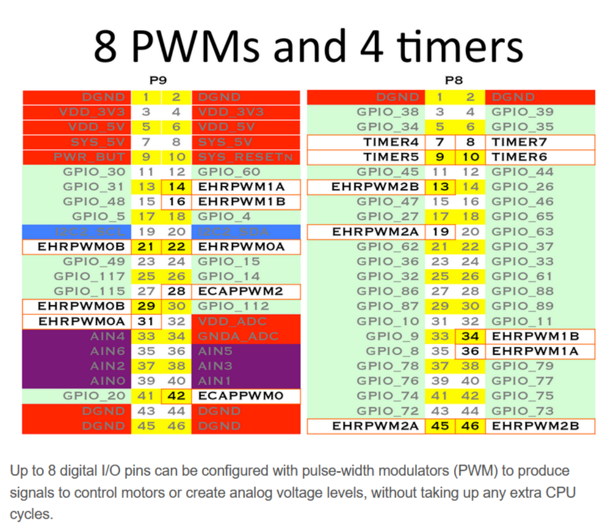
Voltage = 5V

Current = V/R = 5V/940Ω = 0.00532A (Same throughout)

V1 = I1 * R1 = 0.00532A * 470Ω = 2.5V

V2 = I2 * R2 = 0.00532A * 470Ω = 2.5V

Since the GPIO pin we use (the pin connected to the orange wire) can take 3.3V, we are under 3.3V.



8 PWMs and 4 timers

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	TIMER4	7	8	TIMER7
PWR_BUT	9	10	SYS_RESETN	TIMER5	9	10	TIMER6
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	EHRPWM1A	EHRPWM2B	13	14	GPIO_26
GPIO_48	15	16	EHRPWM1B	GPIO_47	15	16	GPIO_46
GPIO_5	17	18	GPIO_4	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	EHRPWM2A	19	20	GPIO_63
EHRPWM0B	21	22	EHRPWM0A	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	GPIO_15	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	GPIO_14	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	ECAPPWM2	GPIO_86	27	28	GPIO_88
EHRPWM0B	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
EHRPWM0A	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	EHRPWM1B
AIN6	35	36	AIN5	GPIO_8	35	36	EHRPWM1A
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	ECAPPWM0	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	EHRPWM2A	45	46	EHRPWM2B

Up to 8 digital I/O pins can be configured with pulse-width modulators (PWM) to produce signals to control motors or create analog voltage levels, without taking up any extra CPU cycles.

Figure 4, Highlighted PWM and timer headers for BBG, from https://beagleboard.org/Support/bone101/#headers.

5. Linux PWM: Servo

For this guide, as we will be using the `BB-PWM0-00A0.dtbo` overlay. This overlay corresponds to P9-21 (EHRPWM0B) and P9-22 (EHRPWM0A) pins, we will only be using P9-21.

1. To begin export the PWM functionality for the servo, we first need to find out what pwmchip is in use. The pwmchips change on every boot, so, if you have multiple PWM overlays enabled in `uEnv.txt`, you must first figure out which pwmchip corresponds to which pins. Fun! If you have only one PWM overlay enabled in `uEnv.txt`, then your job is easy and there should only be one pwmchip shown if you navigate to `/sys/class/pwm`. Follow your case below:
   - If you have multiple overlays:
     1. Although the pwmchips assigned will change with each boot, what stays constant it the address in which the EHRPWMs reside, so let's find those.
        If we look at Figure 5, the red lines indicate the start address in which we may find the PWMS.
        From the headers in Figure 4, we can gather that,
        P9-21 & P9-22 correspond to ePWM0 (overlay `BB-PWM0-00A0.dtbo`),
        P9-14 & P9-16 correspond to ePWM1 (overlay `BB-PWM1-00A0.dtbo`), and
        P8-13 & P8-19 correspond to ePWM2 (overlay `BB-PWM2-00A0.dtbo`).

        TEXAS INSTRUMENTS

        ARM Cortex-A8 Memory Map www.ti.com

        Table 2-3. L4_PER Peripheral Memory Map (continued)

Device Name	Start_address (hex)	End_address (hex)	Size	Description
PWM Subsystem 0	0x4830_0000	0x4830_00FF		PWMSS0 Configuration Registers
eCAP0	0x4830_0100	0x4830_017F	4KB	PWMSS eCAP0 Registers
eQEP0	0x4830_0180	0x4830_01FF		PWMSS eQEP0 Registers
ePWM0	0x4830_0200	0x4830_025F		PWMSS ePWM0 Registers
	0x4830_0260	0x4830_1FFF	4KB	Reserved
PWM Subsystem 1	0x4830_2000	0x4830_20FF		PWMSS1 Configuration Registers
eCAP1	0x4830_2100	0x4830_217F	4KB	PWMSS eCAP1 Registers
eQEP1	0x4830_2180	0x4830_21FF		PWMSS eQEP1 Registers
ePWM1	0x4830_2200	0x4830_225F		PWMSS ePWM1 Registers
	0x4830_2260	0x4830_3FFF	4KB	Reserved
PWM Subsystem 2	0x4830_4000	0x4830_40FF		PWMSS2 Configuration Registers
eCAP2	0x4830_4100	0x4830_417F	4KB	PWMSS eCAP2 Registers
eQEP2	0x4830_4180	0x4830_41FF		PWMSS eQEP2 Registers
ePWM2	0x4830_4200	0x4830_425F		PWMSS ePWM2 Registers
	0x4830_4260	0x4830_5FFF	4KB	Reserved
Reserved	0x4830_6000	0x4830_DFFF	32KB	Reserved
LCD Controller	0x4830_E000	0x4830_EFFF	4KB	LCD Registers
	0x4830_F000	0x4830_FFFF	4KB	Reserved
Reserved	0x4831_0000	0x4831_1FFF	8KB	Reserved
	0x4831_2000	0x4831_2FFF	4KB	Reserved
Reserved	0x4831_3000	0x4831_7FFF	20KB	Reserved
Reserved	0x4831_8000	0x4831_BFFF	16KB	Reserved
	0x4831_C000	0x4831_CFFF	4KB	Reserved
Reserved	0x4831_D000	0x4831_FFFF	12KB	Reserved
Reserved	0x4832_0000	0x4832_5FFF	16KB	Reserved
Reserved	0x4832_6000	0x48FF_FFFF	13MB-152KB	Reserved

        Figure 5, Peripheral Memory Map located on Pg. 182 of TI-am3359 TRM.
     2. Now that we know which addresses pertain to which ePWM's we can find which pwmchip each one uses. Based off of commands found in the beagleboard repo, https://github.com/beagleboard/bb.org-overlays/blob/master/examples/cape-unversal-pwm.txt, we can see the matching addresses from the memory map and `ls` to find the appropriate chip.

```
#P9.21/P9.22
#ls -lh /sys/devices/platform/ocp/48300000.epwmss/48300200.pwm/pwm/
#P9.14/P9.16
#ls -lh /sys/devices/platform/ocp/48302000.epwmss/48302200.pwm/pwm/
#P8.13/P8.19
#ls -lh /sys/devices/platform/ocp/48304000.epwmss/48304200.pwm/pwm/
```

If we do the first `ls` command for ePWM address 48300200, the output should look something like this:

```
total 0
drwxrwxr-x 5 root pwm 0 Jan  1  2000 pwmchip0
```

The highlighted pwmchip shows us which pwmchip we will use for P9-21 and P-22!

- If you have only one PWM overlay, your job is easy and there should only be one pwmchip shown in `/sys/class/pwm`.

2. Now that we have the pwmchip we need, we can export the pin we will use to control our servo, pin P9-21. We know that P9-21 corresponds to EHRPWM0B & P9-22 corresponds to EHRPWM0A via Figure 4. Both belong to the same pwmchip, so to only use one we must export that specific pin. To export an 'A' EHRPWM pin you navigate to the pwmchip and write a 0 to the export file, to export a 'B' EHRPWM pin you write a 1 to the export file.
   Let's say that on boot our P9-21 corresponds to pwmchip0, as such, P9-21 is a 'B' ePWM pin so we may write a 1 to the export file:
   ```
   (bbg)$ /sys/class/pwm/pwmchip0$ echo 1 > export
   ```
   If you list the files in the pwmchip directory, you should see:
   ```
   (bbg)$:/sys/class/pwm/pwmchip0$ ls
   device  export  npwm  power  pwm1  subsystem  uevent  unexport
   ```

3. Next, we navigate to `pwm1` and set the period to 20ms, the period takes in ns so 20ms = 20000000ns:
   ```
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ echo 20000000 > period
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ cat period
   20000000
   ```
   - Note in the event you want to use both pins know that these pins share hardware so you may not change the period of these channels independently
   - In fact, the software won't let you change P9-21/P9-22's period at all if you have both PWM channels' periods set. Specifically, you can set and change the period of one of the P9-21 or P9-22's PWM channels until the other one is given a period. Then, you can only change the period of one to match the other.
   - In other words, set the period to something reasonable to start and then don't change it.

4. Now we can set the duty cycle. Let's move the position of the shaft to a 90-degree position, meaning we need to give the duty cycle a value of 1.5ms or 1500000 ns:
   ```
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ echo 1500000 > duty_cycle
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ cat duty_cycle
   15000000
   ```

5. Finally, we enable the pin and watch the servo move:
   ```
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ echo 1 > enable
   (bbg)$ /sys/class/pwm/pwmchip0/pwm-0:1$ cat enable
   1
   ```

7. Troubleshooting:

- If you get the messages for an unknown folder when you try to export the PWM? You likely don't yet have the hardware support loaded yet. Follow the steps in section 2 - Load PWM Device Tree.
- "Permission denied" error trying to write to a file: You either incorrectly used sudo tee (if executing from a .sh script) or have not yet exported the PWM for that pin.
- "Device or resource busy" when trying to export: the PWM is likely already exported.
- "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 P9-21/P9-22: This is by design, since both PWMs are linked, and the period cannot be changed.