# Controllo dei motori LEGO con Linux Embedded

## Un progetto divertente

Angelo Dureghello
adureghello@baylibre.com

Linux Day, sabato 26 Ottobre 2024

# Angelo Dureghello

- Lavora per BayLibre SAS, Nizza, sviluppo SW open-source,
- attivo nello sviluppo HW e SW di sistemi embedded dal 2001,
- grande interesse per l'open-source,
- Linux kernel contributor, 61 patch (Author), 101 Log references, 1 driver MAINTAINERS flag
- U-boot "custodian", per l'architettura m68k/ColdFire,
- contributor occasionale in altri progetti open-source, Yocto, gdb, etc,
- speaker at ELCE Berlin and Fosdem,
- "family-man", sposato con due figlie 11 e 13 anni.

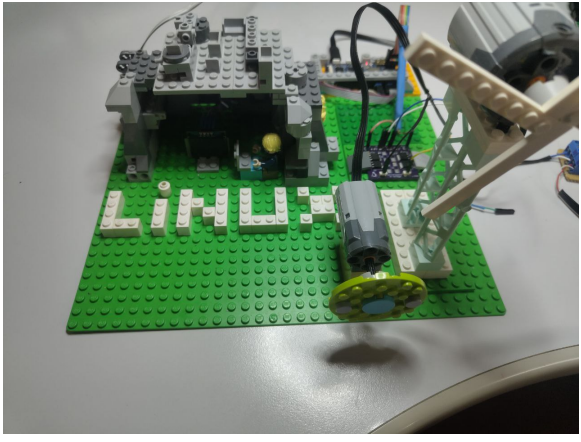
- MCU : microcontrollore
- SoC : System On Chip
- FDT : Flat Device Tree
- PWM : Pulse Width Modulation
- BJT : Bipolar Junction Transistor
- MOSFET : Metal-Oxide-Semiconductor Field-Effect

# Controllo dei motori LEGO con Linux Embedded

- PARTE 1 - Introduzuione sui motori LEGO
- PARTE 2 - PWM
- PARTE 3 - Elettronica di controllo
- PARTE 4 - Configurazione Linux embedded
- Appendice - Link utili

# Controllo dei motori LEGO con Linux Embedded

## PARTE 1 - Introduzuione sui motori LEGO




LINUX DAY ITALIA

# Introduzuione sui motori LEGO

# Controllo dei motori LEGO con Linux Embedded

## PARTE 1 - Introduzuione sui motori LEGO

| 9 V supply |  2838 |  2986 |  71427 |  43362 |  5292 |  47154 |  NXT |  E-Motor |  PF Medium |  PF XL |  9V Train |  RC Train |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rotation speed (rotations per minute) | 4100 rpm | 35 rpm | 360 rpm | 340 rpm | 1700 rpm / 1240 rpm | 460 rpm | 170 rpm | 780rpm | 405 rpm | 220 rpm | 2000rpm | 2000rpm |
| No-load current | 35 mA | 6 mA | 3.5 mA | 9 mA | 160 mA | 31 mA | 60 mA | 17.5mA | 65 mA | 80 mA | 90mA | 90mA |

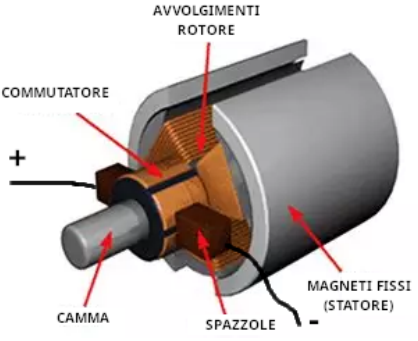
| 9 V supply |  PF Train |  PF Large |  EV3 Large |  EV3 Medium |  PUP Train |  PUP medium |  Boost Ext. |  Boost Int. |  Control+ L |  Control+ XL |  Spike M |  Spike L |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rotation speed (rotations per minute) | 1900rpm | 390 rpm | 175rpm | 260 rpm | 1760 rpm | 380 rpm | 255 rpm | 350 rpm | 315 rpm | 330 rpm | 228 rpm | 213 rpm |
| No-load current | 90mA | 120 mA | 60mA | 80 mA | 100 mA | 60 mA | 41 mA | 140 mA | 120 mA | 60 mA | 100 mA | 110 mA |

Reperibili preso LEGO o su ebay, venduti in set o singoli.

Motori 9V DC brushed utilizzati:

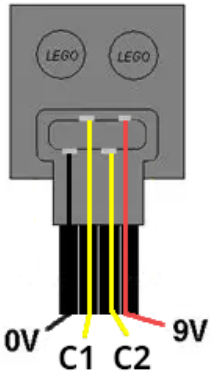
8883, M-Motor

8882, XL-Motor




Tramite le spazzole e il commutatore la corrente scorre negli avvolgimenti del rotore, creando un campo magnetico indotto nelle armature. Quando un lato del rotore e lo statore hanno campi magnetici dello stesso segno si respingono, e il motore inizia a girare. Quando le armature si allineano sui segni opposti, il commutatore inverte il senso di passaggio della corrente modificando il campo magnetico, e il motore continua a girare.

- Semplicemente, applicando 9VDC al motore, ruota in un senso,
- invertendo le polarita', ruota nell'altro senso,
- tuttavia, in questa demo vogliamo variare anche la velocita', vedremo come ....



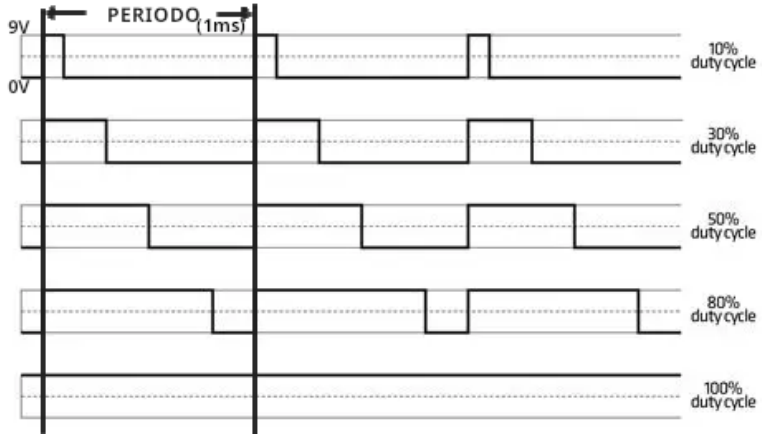

- Questo connettore e' progettato per il battery pack LEGO,
- l'alimentazione del motore e' nei due fili interni, C1 e C2,
- i contatti 0 e 9V vengono usati lato battery pack per altri scopi, non ci riguardano,
- **In questa demo i connettori sono stati rimossi, useremo solo C1 e C2.**

# PWM

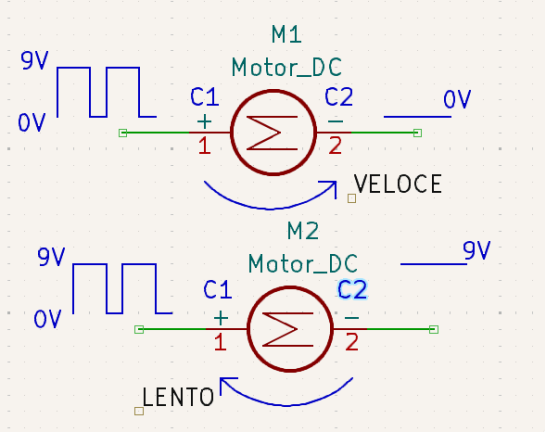
# Controllo dei motori LEGO con Linux Embedded

## PARTE 2 - PWM

- PWM significa "Pulse Width Modulation", modulazione a larghezza d'impulso,
- e' un segnale di forma squadrata,
- e' un segnale periodico, si ripete ciclicamente ad ogni intervallo di tempo fisso (detto periodo),
- e' un segnale con la parte positivia di larghezza variabile,
- e' un segnale dove il periodo (tempo in cui il segnale si ripete) non varia,
- con "duty cycle" (percentuale) si indica la larghezza della parte positiva del segnale,
- il periodo utilizzato per questi motori e' 1ms.


PERIODO (1ms)
9V
0V
10% duty cycle
30% duty cycle
50% duty cycle
80% duty cycle
100% duty cycle

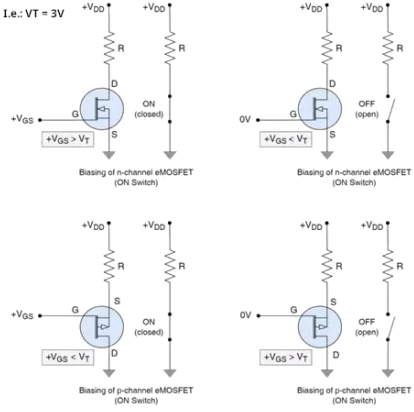
- Controlliamo il motore tra C1 e C2, con:
- un segnale PWM tra 0 e 9V sul terminale C1,
- un segnale fisso di 0 o 9V sul terminale C2,
- cambiando il livello da 0 a 9V e viceversa su C2 (segnale fisso), potremo invertire la direzione di rotazione.
- ad ogni cambio di direzione, il duty cycle assumera' un valore invertito.

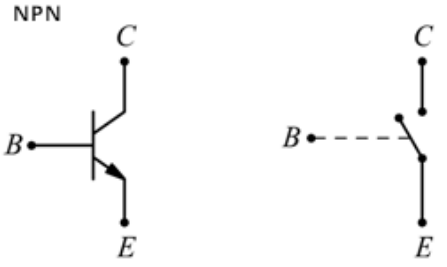

M1
Motor_DC
9V
0V
C1
C2
+
−
1
2
0V
VELOCE
M2
Motor_DC
9V
0V
C1
C2
+
−
1
2
9V
LENTO

- La corrente assorbita dal motore ha un picco di 300mA all'avvio, si assesta poi attorno ai 100mA,
- la velocita' del motore si potrebbe controllare anche variando la tensione, ma in genere non si fa, il motore ha maggior spunto e potenza sotto carico utilizzando PWM,
- le CPU hanno delle uscite di controllo (pin GPIO) a tensioni piuttosto basse (1,8, 3,3), derivate in genere dal dominio Vdd_IO,
- le CPU possono fornire solo correnti molto basse come "drive current", non possono pilotare direttamente dei motori
- serve pertanto un cicuito intermedio che sopporti e fornisca la potenza necessaria a pilotare il motore come **il ponte ad H**.

Ponte ad H

- Costituito da 4 transistor, generalmente BJT o MOSFET,
- in questa demo si compone principalmente di 4 MOSFET, 2 canale P, 2 canale N, utilizzati come interruttori,
- si trova in commercio per pochi euro, come clickboard (https://www.mikroe.com/click-boards), o similari,
- possibile autocostruirlo facilmente, come in questa demo.

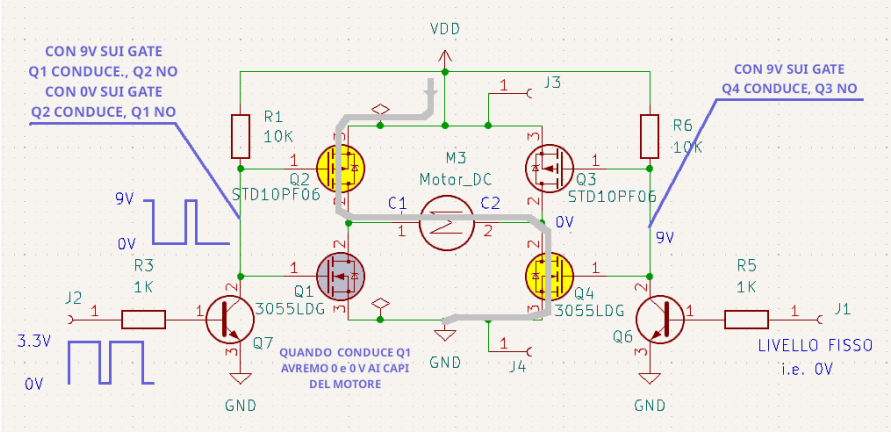

I.e.: VT = 3V
+VDD
+VDD
R
R
D
+VGS
G
ON
(closed)
+VGS > VT
S
Biasing of n-channel eMOSFET
(ON Switch)
+VDD
+VDD
R
R
D
0V
G
OFF
(open)
+VGS < VT
S
Biasing of n-channel eMOSFET
(ON Switch)
+VDD
+VDD
R
R
S
+VGS
G
ON
(closed)
+VGS < VT
D
Biasing of p-channel eMOSFET
(ON Switch)
+VDD
+VDD
R
R
S
0V
G
OFF
(open)
+VGS > VT
D
Biasing of p-channel eMOSFET
(ON Switch)



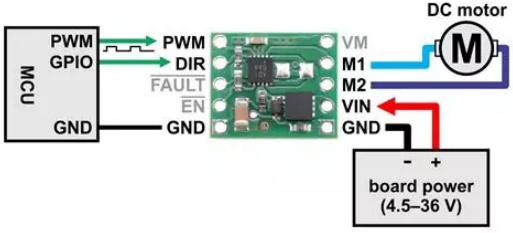

Approssimando:
Vbe > 0.6V => SATURAZIONE, Vce 0.1V, CE chiuso
Vbe = 0V => INTERDIZIONE, CE = APERTO

# Controllo dei motori LEGO con Linux Embedded

## PARTE 3 - Elettronica di controllo



- Esistono molte tipologie di H-bridge sul mercato, piu o meno complessi,
- molti utilizzano circuiti integrati di controllo evoluti,
- molti hanno piu ingressi ed uscite per piu motori,
- in ogni caso, dovremo sempre collegare
  - GND e 9V come alimentazione DC del ponte (2 fili)
  - C1 e C2 per comandare il motore (2 fili)
  - Ingresso (IN), PWM, livello fisso, e massa (3 fili)

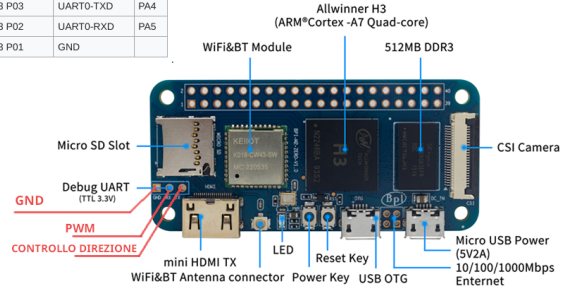


Cofigurazione Demo

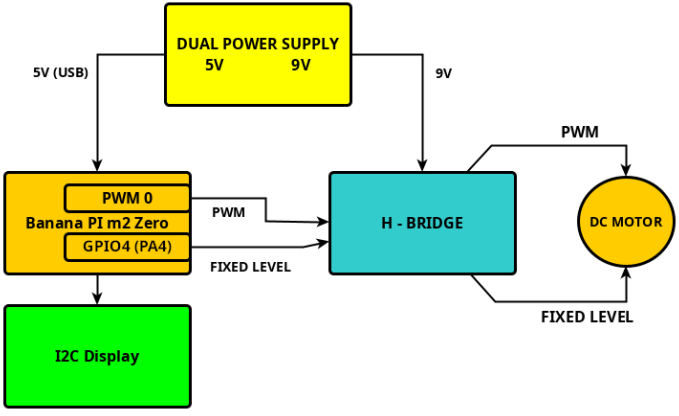
- Alimentatore duale, per comodita',,
- Banana PI M2 Zero, https://wiki.banana-pi.org/Banana_Pi_BPI-M2_ZERO,
- display I2C SSD1307,
- ponte ad H autocostruito,
- motore LEGO M o L.

## Cofigurazione Demo

| jumper CON3 of Banana pi BPI-M2 Zero | | |
|---|---|---|
| CON3 Pin Name | Default Function | GPIO |
| CON3 P03 | UART0-TXD | PA4 |
| CON3 P02 | UART0-RXD | PA5 |
| CON3 P01 | GND | |




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PA4 | I/O | DIS | Z | UART0_TX | - | - | - | PA_EINT4 |
| PA5 | I/O | DIS | Z | UART0_RX | PWM0 | - | - | PA_EINT5 |
| PA6 | I/O | DIS | Z | SIM_PWREN | - | | | PA_EINT6 |
| PA7 | I/O | DIS | Z | SIM_CLK | - | | | PA_EINT7 |

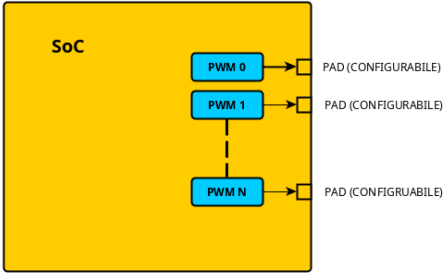

DUAL POWER SUPPLY
5V
9V
5V (USB)
9V
PWM
PWM 0
Banana PI m2 Zero
GPIO4 (PA4)
PWM
FIXED LEVEL
H - BRIDGE
DC MOTOR
FIXED LEVEL
I2C Display

# Configuraizone Linux embedded

Generare il PWM in Linux embedded

- Hardware (necessario modulo hardware e relativo driver)
  - MCU (microcontrollori, si (modello appropriato)
  - SoC (generalmente si),
  - CPU pure (x86, etc), no.
- In mancanza di PWM hardware, si può generarlo via software (bitbanging, tramite gpio)
  - Linux lo supporta, c'e' il driver /drivers/pwm/pwm-gpio.c
  - Documentazione: Documentation/devicetree/bindings/pwm/pwm-gpio.yaml.
  - non particolarmente affidabile, può variare con carico CPU, interrupts, etc.

**Un SoC puo' avere uno o piu moduli PWM hardware.**
**In genere, le uscite possono essere direzionate a diverse piazzole,**
**in Linux si utilizza il "devicetree" per questo (nodo "pinmux").**



Nei sistemi Linux Embedded, i dispositivi non sono rilevati dinamicamente, come nel PC, (dove generalmente vengono rilevati sui bus PCI, USB etc.). Si utilizzano dei file speciali per dichiarare i dispositivi esistenti sulla scheda, chiamati **devicetree**.

Affinche' il modulo PWM funzioni, in Linux sono necessari:

- 1 - abilitare il supporto PWM (framework),
- 2 - il **driver** che cotnrolla il modulo PWM,
- 3 - il **device** PWM, da abilitare tramite **devicetree**.

Nota: una scheda potrebbe avere un chip PWM separato, controllabile in SPI o I2C.
In questo caso, anche i relativi drivers (I2C o SPI) dovranno essere abilitati.

## 1 - Individuare e abilitare il driver del kernel

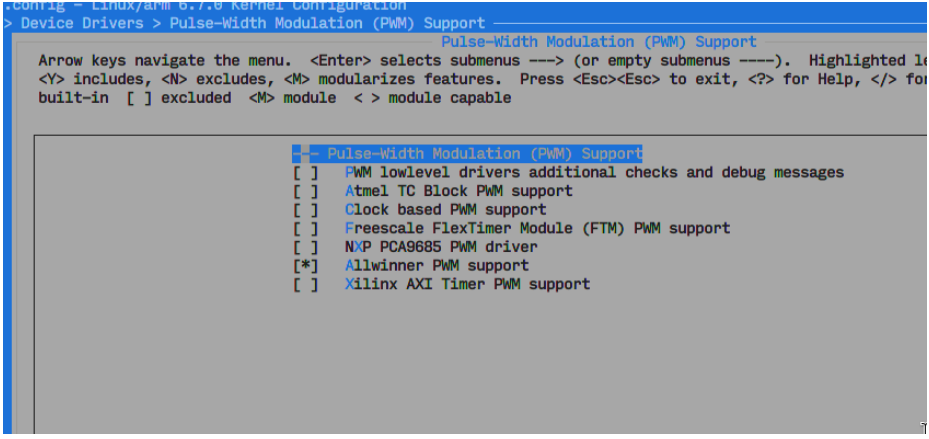
Già attivo nel sistema (generalmente si) ? **Bene, OK**

```
$ ls /sys/class/pwm/pwmchip0
device     npwm       pwm0       uevent
export     power      subsystem  unexport   ===> OK, pwmchip attivo
# (Se supporto config.gz abilitato nel kernel)
$ zcat /proc/config.gz | grep PWM
# CONFIG_SENSORS_PWM_FAN is not set
# CONFIG_REGULATOR_PWM is not set
# CONFIG_LEDS_PWM is not set
# CONFIG_COMMON_CLK_PWM is not set
CONFIG_PWM=y                          <======= framework abilitato
# CONFIG_PWM_ATMEL_TCB is not set
# CONFIG_PWM_AXI_PWMGEN is not set
CONFIG_PWM_CLK=y
# CONFIG_PWM_DWC is not set
# CONFIG_PWM_FSL_FTM is not set
# CONFIG_PWM_GPIO is not set
CONFIG_PWM_XILINX=y                   <=======  driver abilitato (built-in) !
```

1 - Individuare e abilitare il driver del kernel
Il driver e' mancante ? => Necessario ricompilare il kernel

- da **make menuconfig**, i driver PWM si trovano in "Device Drivers" -> "Pulse-Width Modulation (PWM) Support"
- necessario individuare il driver relativo al SoC in uso
- aiutarsi eventualmente con "nano drivers/pwm/Kconfig" (vedi in seguito, slide 32)

make menuconfig



in aiuto, si può visionare "drivers/pwm/Kconfig"

```
          will be called pwm-fsl-ftm.

config PWM_HIBVT
        tristate "HiSilicon BVT PWM support"
        depends on ARCH_HISI || COMPILE_TEST
        depends on HAS_IOMEM
        help
          Generic PWM framework driver for HiSilicon BVT SoCs.

          To compile this driver as a module, choose M here: the module
          will be called pwm-hibvt.

config PWM_IMG
        tristate "Imagination Technologies PWM driver"  <=== nome esotico
        depends on HAS_IOMEM
        depends on MFD_SYSCON
        depends on COMMON_CLK
        depends on MIPS || COMPILE_TEST    <=== utilizzato in architettura MIPS
        help
          Generic PWM framework driver for Imagination Technologies
          PWM block which supports 4 channels.

          To compile this driver as a module, choose M here: the module
          will be called pwm-img

config PWM_IMX1
        tristate "i.MX1 PWM support"
        depends on ARCH_MXC || COMPILE_TEST
        depends on HAS_IOMEM
        help
```

## 2 - abilitare il modulo PWM da devicetree

- i devicetree sono file sorgenti **.dts**
- compilati con il compilatore **dtc** che genera un **.dtb**
- il **.dtb** e' processato dal bootloader, passato al kernel al suo avvio,
- bisogna individuare il devicetree sorgente **.dts** della nostra scheda,
- che e' generalmente incluso nei sorgenti del kernel,
- ad esempio, per architetture ARM, si trova in **arch/arm/boot/dts/**
- il suo nome ricordera' il nome della scheda
- può includere blocchi comuni ad altre schede **.dtsi**

2 - abilitare il modulo PWM da devicetree, modifiche per banana pi m2 zero

- Nei devicetree, il carattere & si utilizza per riferirsi ad un nodo esistente e modificarlo,
- nel caso delal BananaPI M2 Zero, abbiamo modificato il nodo originale, che si trova in arch/arm/boot/dts/allwinner/sunxi-h3-h5.dtsi,
- il suo stato era "disabled", dovremo portato a "okay" (abilitato) e aggiungere alcune informazioni.

```
$ nano arch/arm/boot/dts/allwinner/sunxi-h3-h5.dtsi
...
    pwm: pwm@1c21400 {
        compatible = "allwinner,sun8i-h3-pwm";
        reg = <0x01c21400 0x8>;
        clocks = <&osc24M>;
        #pwm-cells = <3>;
        status = "disabled";
    };
```

2 - abilitare il modulo PWM da devicetree, modifiche per banana pi m2 zero

Modifichiamo dunque il devicetree (file dts) relativo alla BananPi M2 Zero aggiungendo:

```
$ nano arch/arm/boot/dts/allwinner/sun8i-h2-plus-bananapi-m2-zero.dts
...
&pwm {  /* abilito il modulo pwm, unico per questo SoC */
        pinctrl-names = "default";
        pinctrl-0 = <&pwm_pins>;    /* <== rimanda alla configurazione pin d'uscita */
        status = "okay";            /* <== fondamentale, abilita il device,*/
                                    /*     altrimenti il modulo non sara' inizializzato al boot */
};
...
&pio {
    ....

        pwm_pins: pwm-pins {
                pins = "PA5";       /* utilizzero' la piazzola d'uscita gpio PA5 */
                function = "pwm0";
        };
};
```

2 - abilitare il modulo PWM da devicetree, modifiche per banana pi m2 zero

E' possibile anche creare un nodo consumer e definire periodo (nanosecondi) e polarita', in questo modo:

```
1    bl: backlight {
2        /* three parameters, PWM number in the pwmchip, period, polarity */
3        pwms = <&pwm 0 5000000 PWM_POLARITY_INVERTED>;
4        pwm-names = "backlight";
5    };
```

Altrimenti, abilitando il PWM da userspace (esportando un pwm), il periodo sara' impostato di default a **1000000 ns (1ms)**

- una volta modificato il deviceetree, va compilato con:

  ```
  dtc -I dts -O dtb -o devicetree_file_name.dtb devicetree_file_name.dts
  ```

- e copiato nella partizione di boot della vostra sdcard, dove si trova il kernel (zImage, uImage), sovrascrivendo il vecchio dtb
- se non si vuole modificare il file dts sorgete, un altra soluzione e' utilizzare un dtso (dts overlay), compilato in un file "dtbo" che verra' applicato "sopra" al devicetree originale. https://www.kernel.org/doc/html/latest/devicetree/overlay-notes.html

2 - abilitare il modulo PWM da devicetree, modifiche per banana pi m2 zero

- dopo il reboot, verifichiamo il PWM sia funzionante
- esiste "/sys/class/pwm/pwmchip0/" ? Se non presente, qualcosa e' andato storto.
- possiamo anche visualizzare informazioni da debugfs

```
root # mount -t debugfs none /sys/kernel/debug
root # cat /sys/kernel/debug/pwm
platform/1c21400.pwm, 1 PWM device
 pwm-0   ((null)             ): period: 0 ns duty: 0 ns polarity: normal
```

**PWM abilitato**, utilizzo del PWM tramite interfaccia "sysfs"

- Se CONFIG_SYSFS e' abilitato nella configurazione del kernel (quasi sempre),
- interfaccia PWM esposta in **/sys/class/pwm/**,
- ogni mpodulo PWM rilevato sara' esportato come pwmchipN, dove N e' un indice assegnato al chip PWM chip.
- nella directory troviamo:
  - npwm : numero di canali supportati nel chip,
  - export : per esportare (ed utilizzare) un canale PWM (write-only),
  - unexport : rimuove un canale PWM da sysfs (write-only).

Abilitiamo (esportiamo) il canale per essere utilizzato:

```
$ echo 0 > /sys/class/pwm/pwmchip0/export
$ ls  sys/class/pwm/pwmchip0/
/root/lugts # ls -al /sys/class/pwm/pwmchip0/
drwxr-xr-x    4 0        0               0 Jan  1 23:38 .
drwxr-xr-x    3 0        0               0 Jan  1 23:38 ..
lrwxrwxrwx    1 0        0               0 Jan  1 23:38 device -> ../../../1c21400.pwm
--w-------    1 0        0            4096 Jan  1 23:38 export
-r--r--r--    1 0        0            4096 Jan  1 23:38 npwm
drwxr-xr-x    2 0        0               0 Jan  1 23:38 power
drwxr-xr-x    3 0        0               0 Jan  1 23:38 pwm0 <=== eccolo
lrwxrwxrwx    1 0        0               0 Jan  1 23:38 subsystem -> ../../../../../../class/pwm
-rw-r--r--    1 0        0            4096 Jan  1 23:38 uevent
--w-------    1 0        0            4096 Jan  1 23:38 unexport
```

Una volta esportato (abilitato) il canale, troveremo una nuova directory **pwm0**, e nella directory avremo:

```
/root/lugts # ls /sys/class/pwm/pwmchip0/pwm0/
capture     enable      polarity    uevent
duty_cycle  period      power
```

file che sono le proprieta' con cui configurare il PWM.
Nota: sysfs e' uno pseudo filesystem esistente solo in memoria, il suo accesso e' immediato.

Le seguenti proprieta' sysfs sono ora disponibili:

- **period** : il periodo del segnael PWM (read/write), in nanosecondi, e' la somma del tempo attivo e inattivo del segnale PWM,
- **duty_cycle** : il tempo attivo del segnale PWM, (read/write) in nanosecondi, deve essere inferiore o uguale al periodo,
- **polarity**' : cambia la polarita' del segnale PWM (read/write). Funziona solo se il chip supporta l'inversione di polarita'. Valore e' una stringa "normal" o "inversed".
- **enable** : abilita o disabilita il segnale PWM (read/write), (0 - disabled, 1 - enabled).

- Attualmente, ufficialmente, sysfs e' l'unico modo per controllare il PWM,
- files utilizzabili da qualsiasi linguaggio, tuttavia ...
- c'e' una "patchset" in fase di accettazione che crea un "character device" per ogni pwmchip: patchset link
- e una libpwm che lavora con entrambi i metodi libpwm link

Commentiamo lo script di demo "intro.sh"

```
init()
{
        if [ ! -e /sys/class/pwm/pwmchip0/pwm0 ]; then
                echo 0 > /sys/class/pwm/pwmchip0/export
        fi
        # period 1ms (in ns)
        echo 1000000 > /sys/class/pwm/pwmchip0/pwm0/period
        echo 0 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
        echo 1 > /sys/class/pwm/pwmchip0/pwm0/enable
        # direction gpio
        if [ ! -e /sys/class/gpio/gpio4 ]; then
                echo 4 > /sys/class/gpio/export
        fi
        echo out > /sys/class/gpio/gpio4/direction
        echo 1 > /sys/class/gpio/gpio4/value
        # stop motor
        echo 0 > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
}
```

Commentiamo lo script di demo "intro.sh"

```
pwm()
{
        if [ $# -gt 2 ]; then
                val=$((100 - $1))
        else
                val=$1
        fi
        echo $(($val * 10000)) > /sys/class/pwm/pwmchip0/pwm0/duty_cycle
        clear_screen
        msg "       PWM        "
        blank_line
        msg "       ${1} %     " ${2}
}
```

Commentiamo lo script di demo "intro.sh"

```
change_direction()
{
        clear_screen
        if [ $# -eq 0  ]; then
                msg " Changing  PWM  "
                msg "  direction ... " 2
        fi
        dir=`cat /sys/class/gpio/gpio4/value`
        if [ $dir == "1" ]; then
                echo 0 > /sys/class/gpio/gpio4/value
        else
                echo 1 > /sys/class/gpio/gpio4/value
        fi
}
```

Commentiamo lo script di demo "intro.sh"

```
show()
{
        # pwm duty_cycle time inverted_flag
        pwm 0 0.2
        pwm 10 0.5
        pwm 20 0.5
        pwm 50 0.5
        pwm 70 0.5
        pwm 100 2
        pwm 0 1
        change_direction
        pwm 0 0.5 i
        pwm 10 0.5 i
        pwm 50 0.5 i
        pwm 70 0.5 i
        pwm 100 2 i
        pwm 0 1 i
        change_direction
}
```

## libpwm

```
1  int main(int argc, char *const argv[])
2  {
3    struct pwm_chip *chip;
4    struct pwm *pwm;
5    struct pwm_waveform wf = {
6      .period_length_ns = 50000,
7      .duty_length_ns = 25000,
8      .duty_offset_ns = 0,
9    };
10   int ret;
11
12   chip = pwm_chip_open_by_number(chipno);
13   if (!chip) {
14     perror("Failed to open pwmchip0");
15     return EXIT_FAILURE;
16   }
17
18   ...
```

## libpwm

```
1
2   pwm = pwm_chip_get_pwm(chip, pwmno);
3   if (!pwm) {
4     perror("Failed to get pwm0");
5     return EXIT_FAILURE;
6   }
7
8   ret = pwm_set_waveform(pwm, &wf);
9   if (ret < 0) {
10    perror("Failed to configure PWM");
11    return EXIT_FAILURE;
12  }
13
14  ...
```

Dovremmo avere ora tutti gli ingredienti per divertirci.

# Domande ?

# Controllo dei motori LEGO con Linux Embedded

## Appendice: link utili

- Linux kernel, sorgenti ufficiali
- Linux kernel PWM, devicetree documentation
- Browse codice kernel online
- libpwm
- pwmtest.c (libpwm C example)
- Banana PI m2 Zero wiki
- Allwinner H2+ datasheet

Grazie per l'ascolto !!