



SigmaStar Camera PWM User Guide

Version 0.1



© 2020 SigmaStar Technology Corp. All rights reserved.

SigmaStar Technology makes no representations or warranties including, for example but not limited to, warranties of merchantability, fitness for a particular purpose, non-infringement of any intellectual property right or the accuracy or completeness of this document, and reserves the right to make changes without further notice to any products herein to improve reliability, function or design. No responsibility is assumed by SigmaStar Technology arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

SigmaStar is a trademark of SigmaStar Technology Corp. Other trademarks or names herein are only for identification purposes only and owned by their respective owners.

**REVISION HISTORY**

Revision No.	Description	Date
0.1	<ul style="list-style-type: none">Initial release	12/18/2019



TABLE OF CONTENTS

REVISION HISTORY	i
TABLE OF CONTENTS	ii
1. PWM parameters	1
2. The PWM KERNEL Settings	2
2.1. Setting DTS	2
2.2. An Example of the Hardware Design	2
3. The PWM driver framework	3
4. STANDARD LINUX PWM control IN KERNEL	4
4.1. Controlling PWM at Console.....	4
4.2. 在 User mode Console 下控制 Motor	5



1. PWM PARAMETERS

- Duty_cycle :
 - For example: echo 25 > duty_cycle → it will generate 25% duty cycle
- Period (= "Frequency")
 - For example: echo 2000 > period → it will generate 2 KHz waveform
- Polarity :
 - For example: echo inversed > polarity → it will inverse output waveform. The default is normal.
- Enable/disable :
 - For example: echo 1 > enable → it will enable output waveform.



2. THE PWM KERNEL SETTINGS

2.1. Setting DTS

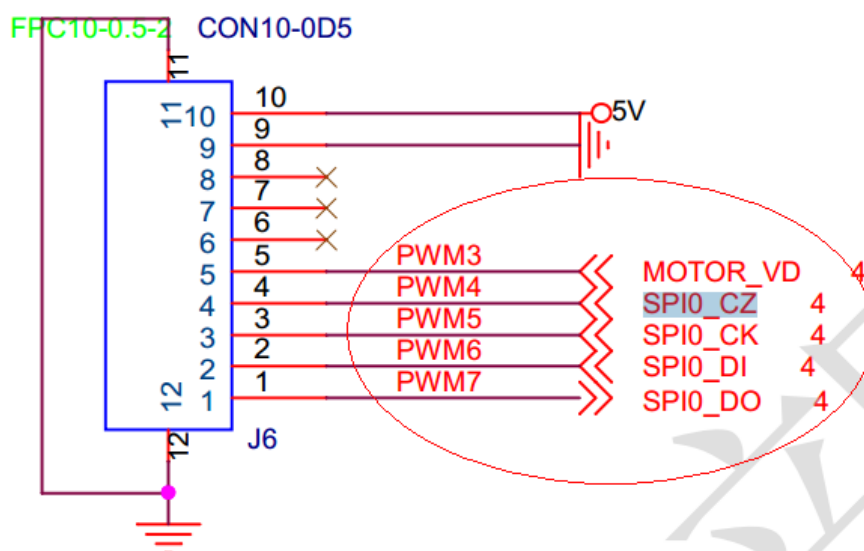
```
pwm {
    compatible = "ssstar,infinity-pwm";
    reg = <0x1F003400 0x600>;
    clocks = <&CLK_xtali_12m>;
    npwm = <11>;
    pad-ctrl = <PAD_PWM0 PAD_PWM1 PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN PAD_UNKNOWN>;
    status = "ok";
}
```

npwm=<11>; indicates there are 11 sets of pwm(s) for control;

pad-ctrl=<...>; to set pad config(s) due to GPIO sharing

- As an example above, it only set correct pads for PAD_PWM0 and PAD_PWM1. It only enable PWM0 and PWM1 among the 11 sets of pwm(s).

2.2. An Example of the Hardware Design



If you want to use PWM4, you need to enable it in the DTS pad-ctrl=<...> by the read pad name listed in drivers/ssstar/include/\gpio.h. From this case, it is PAD_PM_SPI_CZ.

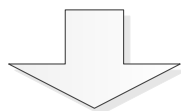
```
91 #define PAD_PM_SPI_CZ 70
92 #define PAD_PM_SPI_CK 71
```



3. THE PWM DRIVER FRAMEWORK

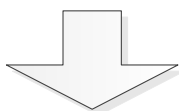
Pwm fs layer: sysfs.c

To generate kobject.
pwm: period/duty_cycle/enabe
The node was created.



Pwm core layer : core.c

To define file_operations.
warp pwm drv layer.



Pwm drv layer: mdrv_pwm/mhal_pwm

mdrv layer:
To define func: enable() /
disable()/ polarity()
mhal layer:
The implementation of mdrv layer.



4. STANDARD LINUX PWM CONTROL IN KERNEL

4.1. Controlling PWM at Console

1. Export PWM number (for example USB PAD_PWM0)

Command:

```
cd /sys/class/pwm/pwmchip0
```

```
echo 0 > export
```

2. Set period (frequency) / duty_cycle / polarity / enable

Command:

```
cd pwm0
```

```
echo xxxx > period
```

In our driver implementation, xxxx indicates output frequency.

For example, echo 2000 > period will generate 2KHz waveform.

```
echo xx > duty_cycle
```

For example, echo 25 > duty_cycle will generate 25% duty cycle.

```
echo inversed > polarity
```

Inverse output waveform, default is **normal**.

```
echo 1 > enable
```

Enable output waveform.

Operations at user mode:

Open a node;

Write a node;



4.2. 在 User mode Console 下控制 Motor

1. Motor hierarchy

Group 0

PWM 0

PWM 1

PWM 2

PWM 3

Group 1

PWM 4

PWM 5

PWM 6

PWM 7

Group 2

PWM 8

PWM 9

PWM 10

2. Cd 馬達控制路徑

Command:

```
cd /sys/devices/virtual/mstar/motor
```

3. Set mode/period(frequency) / Begin/End / round number/enable/hold/stop

- mode

Command:

```
echo PWM_ID enable > group_mode
```

ex : echo 0 1 > group_mode # 設定 PWM0 為馬達模式

ex : echo 0 0 > group_mode # 取消 PWM0 為馬達模式

- period

Command:

```
echo PWM_ID period > group_period
```

In our driver implementation, xxxx indicates output frequency

ex: echo 0 2000 > group_period # PWM0 will generate 2KHz waveform

- begin

Command:

```
echo PWM_ID begin > group_begin
```

ex: echo 0 100 > group_begin # PWM0 will generate duty_cycle starting from 100/1000 of



the period

- end

Command:

`echo PWM_ID end > group_end`

ex: `echo 0 250 > group_end` # PWM0 will generate duty_cycle ending at 250/1000 of the period

- Round mode

Command:

`echo GROUP_ID round > group_round`

ex: `echo 0 10000 > group_round` # Group 0 will generate 10000 period of waveform.

If need to **continue** set a new round after last round completed, set new arguments before the end of the round, than it will continue create a new round.

ex: `echo 0 10000 > group_round`

(During 10000 rounds)

`echo 0 2000 > group_period`

`echo 0 100 > group_begin`

`echo 0 250 > group_end`

`echo 0 20000 > group_round #`

Group 0 will generate 10000 period of waveform first, and continue generate 20000 period of waveform after that.

- enable

Command:

`echo GROUP_ID enable > group_enable`

ex: `echo 0 1 > group_enable` # Group 0 start generating the waveform

ex: `echo 0 0 > group_enable` # Group 0 stop generating the waveform

- Hold mode

Command:

`echo GROUP_ID > group_hold`

ex: `echo 0 > group_hold` # Group 0 hold the last complete waveform



After enable, if set new arguments **before set group hold**, it will generate new waveform rather than hold the last complete waveform.

Ex:



```
echo 0 1 > group_enable
```

```
echo 0 2000 > group_period
```

```
echo 0 100 > group_begin
```

```
echo 0 250 > group_end
```

```
echo 0 > group_hold
```



- stop

Command:

```
echo GROUP_ID > group_stop
```

ex: echo 0 > group_stop # Group 0 immediately stop the waveform