一起学习 CC3200 系列教程之跑马灯

阿汤哥

序：

能力有限，英语不怎么好，难免有错，有问题请联系我，

QQ1519256298　　hytga@163.com

现在我们来学习 CC3200 的跑马灯的寄存器例程。我们将学习如何把 CC3200 的 IO 口设置成输出模式，实现跑马灯效果。

（一）CC3200 的 IO 口简介

（二）硬件设计

（三）软件设计

（一）CC3200 的 IO 口简介

学过其他单片机的人都知道，把 IO 口设置成输出，一般需要 4 个步骤

1、 开启时钟信号

2、 设置映射

3、 设置 IO 口的输入输出模式

4、 设置 IO 的电平

因此，我们将介绍 4 个部分涉及到的寄存器，

1、 时钟寄存器

CC3200 的 datasheet 有一个章节叫 Power Reset and　Clock manage(简称 PRCM)其中有一部分就是时钟的设置，PRCM 有以下的寄存器

15.6　PRCM Registers

Table 15-3 lists the memory-mapped registers for the ARCM. All register offset addresses not listed in Table 15-3 should be considered as reserved locations and the register contents should not be modified.

Table 15-3. PRCM Registers

Offset	Acronym	Register Name	Section
0h	CAMCLKCFG		Section 15.6.1.1
4h	CAMCLKEN		Section 15.6.1.2
8h	CAMSWRST		Section 15.6.1.3
14h	MCASPCLKEN		Section 15.6.1.4
18h	MCASPSWRST		Section 15.6.1.5
20h	SDIOMCLKCFG		Section 15.6.1.6
24h	SDIOMCLKEN		Section 15.6.1.7
28h	SDIOMSWRST		Section 15.6.1.8
2Ch	APSPICLKCFG		Section 15.6.1.9
30h	APSPICLKEN		Section 15.6.1.10
34h	APSPISWRST		Section 15.6.1.11
48h	DMACLKEN		Section 15.6.1.12

这里我们只关心

50h	GPIO0CLKEN	Section 15.6.1.14
54h	GPIO0SWRST	Section 15.6.1.15
58h	GPIO1CLKEN	Section 15.6.1.16
5Ch	GPIO1SWRST	Section 15.6.1.17
60h	GPIO2CLKEN	Section 15.6.1.18
64h	GPIO2SWRST	Section 15.6.1.19
68h	GPIO3CLKEN	Section 15.6.1.20
6Ch	GPIO3SWRST	Section 15.6.1.21
70h	GPIO4CLKEN	Section 15.6.1.22
74h	GPIO4SWRST	Section 15.6.1.23

CC3200 的 GPIO 分成了 4 组，分别为 GPIO A0 ，A1，A2，A3，其中 GPIO0CLKEN 寄存器就是管理 A0 的时钟使能问题，依次类推。

GPIO0CLKEN

Figure 15-17. GPIO0CLKEN Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							DSLPCLKEN
R-0h							R/W-0h
15	14	13	12	11	10	9	8
NU1							SLPCLKEN
R-0h							R/W-0h
7	6	5	4	3	2	1	0
NU2							RUNCLKEN
R-0h							R/W-0h

Table 15-17. GPIO0CLKEN Register Field Descriptions

Bit	Field	Type	Reset	Description
31-17	RESERVED	R	0h	
16	DSLPCLKEN	R/W	0h	GPIO_A_DSLP_CLK_ENABLE 0h = Disable GPIO_A clk during deep-sleep mode 1h = Enable GPIO_A clk during deep-sleep mode
15-9	NU1	R	0h	
8	SLPCLKEN	R/W	0h	GPIO_A_SLP_CLK_ENABLE 0h = Disable GPIO_A clk during sleep mode 1h = Enable GPIO_A clk during sleep mode
7-1	NU2	R	0h	
0	RUNCLKEN	R/W	0h	GPIO_A_RUN_CLK_ENABLE 0h = Disable GPIO_A clk during run mode 1h = Enable GPIO_A clk during run mode

这里我们可以看到有 3 个位可以设置，当芯片是 run 模式，我们就设置成 0x01，睡眠模式 0x10，深度睡眠 0x100，

2、 引脚的映射成 GPIO，datasheet 有一章是 IO Pads and Pin Multiplexing

下面是引脚映射表的局部，表示了一个引脚能够被映射成什么功能，不同的引脚能够被映射成不同的功能，一般地有 GPIO，I2C，UART 等，需要用到什么引脚就去看这个引脚的映射表

Table 16-7. Pin Multiplexing

General Pin Attributes						Function					Pad States		
Pkg Pin	Pin Alias	Use	Select as Wakeup Source	Config Addl Analog Mux	Muxed with JTAG	Dig. Pin Mux Config Reg	Dig. Pin Mux Config Mode Value	Signal Name	Signal Description	Signal Direction	LPDS[1]	Hib[2]	nRESET = 0
1	GPIO10	I/O	No	No	No	GPIO_PAD_CONFIG_10 (0x4402 E0C8)	0	GPIO10	General-Purpose I/O	I/O	Hi-Z	Hi-Z	Hi-Z
							1	I2C_SCL	I2C Clock	O (Open Drain)	Hi-Z		
							3	GT_PWM06	Pulse-Width Modulated O/P	O	Hi-Z		
							7	UART1_TX	UART TX Data	O	1		
							6	SDCARD_CLK	SD Card Clock	O	0		
							12	GT_CCP01	Timer Capture	I	Hi-Z		

每一个引脚都有其对应的映射配置寄存器，如上图，其映射配置寄存器就是 GPIO_PAD_CONDIG_10,

映射配置寄存器：

16.8.1.1.1 GPIO_PAD_CONFIG_0 to GPIO_PAD_CONFIG_32

Table 16-11. GPIO_PAD_CONFIG_0 to GPIO_PAD_CONFIG_32 Register Description

Bit	Field	Type	Reset	Description
31-12	Reserved	R	0	
11-0	MEM_GPIO_PAD_CONFIG 后面两位目前不知道作用，但是发现设置成模拟输入，这两位设置成1	RW	0xC61	[3:0] CONFMODE,这三个bit是映射位,把管脚映射成对应的功能,譬如GPIO,UART,IIC 等等 Determines which functional signal will be routed to the pad. Please refer to the Pin Mux Table. [4] Enable Open Drain mode (eg: when used as I2C). [7:5] DRIVESTRENGTH 驱动强度，sdk里面电流最大6ma，需探究 111 = 14mA 110 = 12mA 101 = 10mA 100 = 8mA 011 = 6mA 010 = 4mA 001 = 2mA 000 = Output Driver Not Enabled [8] Enable internal weak pull up 启动内部弱上拉 [9] Enable internal weak pull down 启动内部弱下拉 [10] Pad output enable override value. level â€˜0â€™ enables the pad output buffer. Else output buffer is TriStated. This does not affect the internal pull-up/pull-down which are controlled independently by bit 8 and bit 9 above. [11] This enables over-riding of the pad output buffer enable. When this bit is set to logic '1', then the value in bit 4 above will control the state of the pad output buffer. When this bit is set to logic '0', then the state of the pad output buffer is directly controlled by the peripheral module to which the pad is functionally mux-ed.

根据其描述，我们设置成 GPIO 模式，一般可以设置成 0x20，

3、 设置成输出模式

因为 CC3200 有 4 组 GPIO，因此有 4 组寄存器，

- GPIO Port A0: 0x4000.4000
- GPIO Port A1: 0x4000.5000
- GPIO Port A2: 0x4000.6000
- GPIO Port A3: 0x4000.7000

每组 GPIO 都有以下的寄存器

Table 5-3. GPIO_REGISTER_MAP Registers

Offset	Acronym	Register Name	Section
0h	GPIODATA	GPIO Data	Section 5.5.1.1
400h	GPIODIR	GPIO Direction	Section 5.5.1.2
404h	GPIOIS	GPIO Interrupt Sense	Section 5.5.1.3
408h	GPIOIBE	GPIO Interrupt Both Edges	Section 5.5.1.4
40Ch	GPIOIEV	GPIO Interrupt Event	Section 5.5.1.5
410h	GPIOIM	GPIO Interrupt Mask	Section 5.5.1.6
414h	GPIORIS	GPIO Raw Interrupt Status	Section 5.5.1.7
418h	GPIOMIS	GPIO Masked Interrupt Status	Section 5.5.1.8
41Ch	GPIOICR	GPIO Interrupt Clear	Section 5.5.1.9

其中 GPIODIR 就是其输出输入配置寄存器

5.5.1.2 GPIODIR Register (offset = 400h) [reset = 0h]

GPIODIR is shown in Figure 5-5 and described in Table 5-5.

The **GPIODIR** register is the data direction register. Setting a bit in the **GPIODIR** register configures the corresponding pin to be an output, while clearing a bit configures the corresponding pin to be an input. All bits are cleared by a reset, meaning all GPIO pins are inputs by default.

Figure 5-5. GPIODIR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								DIR							
R-0h																								R/W-0h							

Table 5-5. GPIODIR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	
7-0	DIR	R/W	0h	GPIO Data Direction 0h = Corresponding pin is an input. 1h = Corresponding pins is an output.

4、 接下来就是 GPIO 的数据配置寄存器

Figure 5-4. GPIODATA Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED																								DATA							
R-0h																								R/W-0h							

Table 5-4. GPIODATA Register Field Descriptions

Bit	Field	Type	Reset	Description
31-8	RESERVED	R	0h	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7-0	DATA	R/W	0h	GPIO Data This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and written to the registers are masked by the eight address lines [9:2]. Reads from this register return its current state. Writes to this register only affect bits that are not masked by ADDR[9:2] and are configured as outputs.

这个寄存器配置起来有点麻烦，

GPIO A0 的 GPIODATA 的地址是 0x40004000，长度是 0x400，GPIODATA 寄存器占据 4 个字节，那么照理来说应该是 0x100 个 GPIODATA，但是在这个 0x400 只有 8 个字节，那么是如何分配的了？

TI 的 datasheet 给出了一张图

写例程：

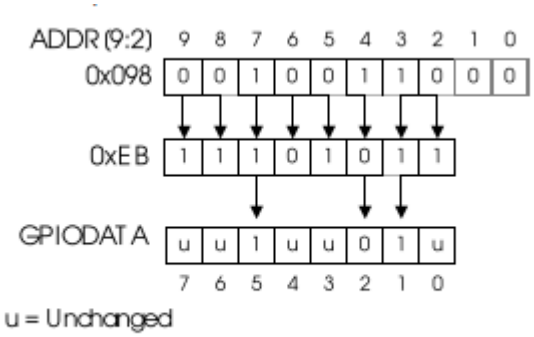


Figure 5-2. GPIODATA Write Example

读例程：

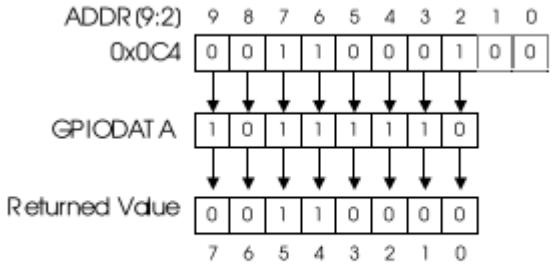
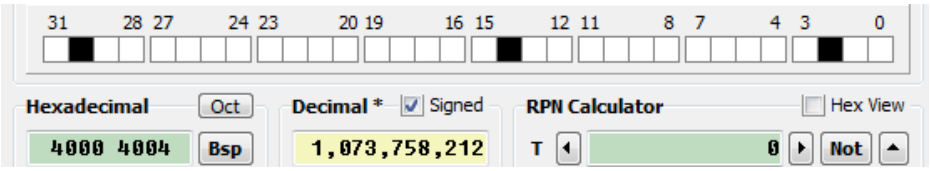


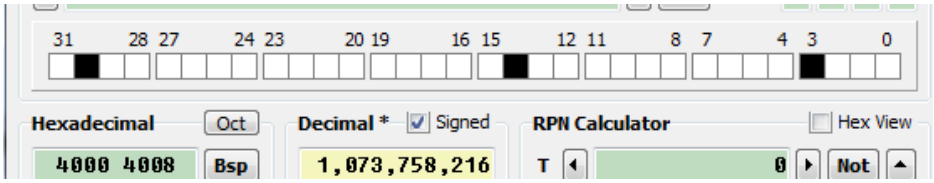
Figure 5-3. GPIODATA Read Example

下面是我自己写的（本来是没看到上面的图的，然后按照自己的想法写了下面的内容，后面就把 Ti 的图给补上了）。

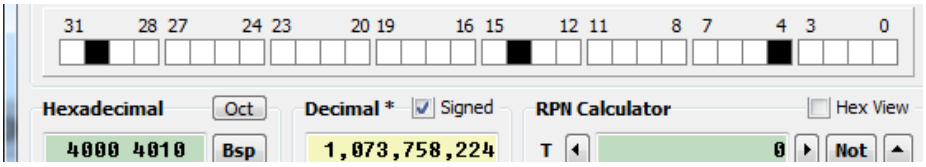
举例来说 GPIO A0 的第 0 个引脚的 GPIODATA 寄存器的地址是



GPIO A0 的第 1 个引脚的 GPIODATA 寄存器的地址是



GPIO A0 的第 2 个引脚的 GPIODATA 寄存器的地址是



其他引脚以此类推，

规律就是

地址线的[9 : 2] 就是决定是哪个寄存器

[2]为 1 决定的是第 0 个

[3]为 1 决定的是第 1 个

[4]为 1 决定的是第 2 个

[5]为 1 决定的是第 3 个

[6]为 1 决定的是第 4 个

[7]为 1 决定的是第 5 个

[8]为 1 决定的是第 6 个

[9]为 1 决定的是第 7 个

刚刚好分配完。

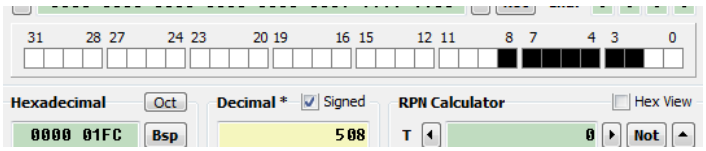
那么应该对 GPIODATA 写入什么值了.

规律

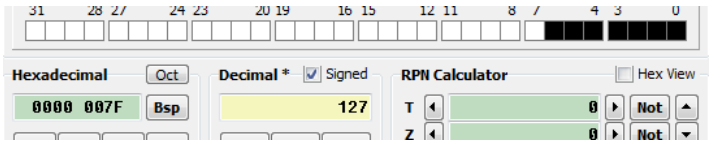
多个引脚的电平 = 地址线[9:2] &值。

譬如：

地址线：位表示



值：位表示



运行代码

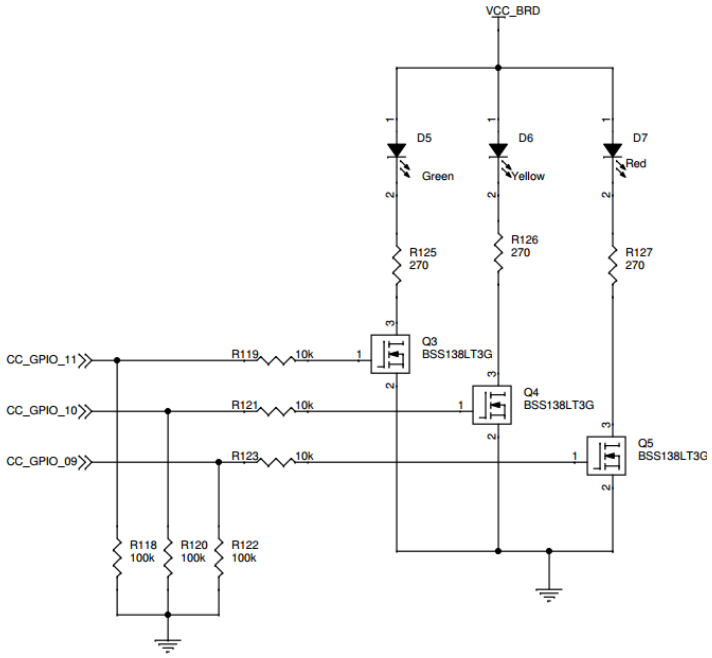
HWREG(0x1fc) = 0x7F

结果：

GPIO 除了第 7 个引脚（不受影响），其他全部输出高电平。

（二）硬件设计

采用的是 CC3200 的 launchxla 板子



引脚的映射配置寄存器的地址分别是

64	GPIO9	GPIO_PAD_CONFIG_9	0x4402 E0C4
1	GPIO10	GPIO_PAD_CONFIG_10	0x4402 E0C8
2	GPIO11	GPIO_PAD_CONFIG_11	0x4402 E0CC

三个引脚是对应于 GPIO 的第几组内的第几个引脚？：TI 给出了一张表格

Table 5-14. GPIO Mapping

GPIO Module Instance	GPIO Bit	GPIO #
GPIOA0	0	GPIO_00 (PM/Dig Mux)
GPIOA0	1	GPIO_01
GPIOA0	2	GPIO_02 (Dig/ADC Mux)
GPIOA0	3	GPIO_03 (Dig/ADC Mux)
GPIOA0	4	GPIO_04 (Dig/ADC Mux)
GPIOA0	5	GPIO_05 (Dig/ADC Mux)
GPIOA0	6	GPIO_06
GPIOA0	7	GPIO_07
GPIOA1	0	GPIO_08
GPIOA1	1	GPIO_09
GPIOA1	2	GPIO_10
GPIOA1	3	GPIO_11
GPIOA1	4	GPIO_12
GPIOA1	5	GPIO_13
GPIOA1	6	GPIO_14
GPIOA1	7	GPIO_15
GPIOA2	0	GPIO_16
GPIOA2	1	GPIO_17
GPIOA2	2	GPIO_18 (Reserved)
GPIOA2	3	GPIO_19 (Reserved)
GPIOA2	4	GPIO_20 (Reserved)
GPIOA2	5	GPIO_21 (Reserved)
GPIOA2	6	GPIO_22
GPIOA2	7	GPIO_23
GPIOA3	0	GPIO_24
GPIOA3	1	GPIO_25

GPIO_11 对应于 GPIO A1 的第 3 引脚
GPIO_10 对应于 GPIO A1 的第 2 引脚
GPIO_9 对应于 GPIO A1 的第 1 引脚

GPIO PORT A1 的时钟地址是（0x44025000 + 0x58）这个地址我在 datasheet 没有看到，是在例程找到的

GPIO PORT A1 的寄存器基地址是 0x4000 5000
现在我们都找到了所有的寄存器地址
（三） 软件设计

```
void delay(int temp) {
    int i = 0;
    for (i = 0;i < temp;i++) {

    }

}
void main(void) {
    //初始化板子，我看了一下大概是中断向量表的映射，和其他的一些东西，这里我们不关心
    BoardInit();
    //使能GPIO A1 时钟
    HWREG(0x44025000 + 0x58) = 0x01;

    //设置GPIO 9的映射配置寄存器：GPIO模式
    HWREG( 0x4402E0C4) = 0x20;
    //设置GPIO 9的输入输入寄存器，输出
    HWREG(0x40005000 + 0x400) |= 0x2;




    //设置GPIO 10的映射配置寄存器：GPIO模式
    HWREG( 0x4402E0C8) = 0x20;
    //设置GPIO 10的输入输入寄存器，输出
    HWREG(0x40005000 + 0x400) |= 0x4;

    //设置GPIO 11的映射配置寄存器：GPIO模式
    HWREG( 0x4402E0CC) = 0x20;
    //设置GPIO 11的输入输入寄存器，输出
    HWREG(0x40005000 + 0x400) |= 0x8;
```

```
    //这里我多设置了一个GPIO，用于验证程序
    //设置GPIO 8的映射配置寄存器：GPIO模式
    HWREG( 0x4402E0C0) = 0x20;
    //设置GPIO 8的输入输入寄存器，输出
    HWREG(0x40005000 + 0x400) |= 0x1;

#if 1 //设置GPIO 8 输出1
    HWREG(0x40005000 + 4 ) = 1;
#else//设置GPIO 8 输出0
    HWREG(0x40005000 + 4 ) = 0;
#endif

    while(1)  {
#if 0//全闪，后全灭
        HWREG(0x40005000 + (16+32+8 )) = 4 + 8 + 2 ;

        delay(0xfffff);
        HWREG(0x40005000 +(16+32+8)) = 0;
        delay(0xfffff);
#else //跑马灯现象，闪的速度有点慢，自己调速度，
            HWREG(0x40005000 + (8)) = 2;
            delay(0xffffff);
            HWREG(0x40005000 +(8)) = 0;
            delay(0xffffff);


        HWREG(0x40005000 + (16)) = 4;
        delay(0xffffff);
        HWREG(0x40005000 +(16)) = 0;
        delay(0xffffff);

        HWREG(0x40005000 + (32))= 8;
        delay(0xfffffff);
        HWREG(0x40005000 +(32)) = 0;
        delay(0xffffff);
    #endif
    }
}
```