

实验 2 UART 监控 LED 亮度

一、实验目的：

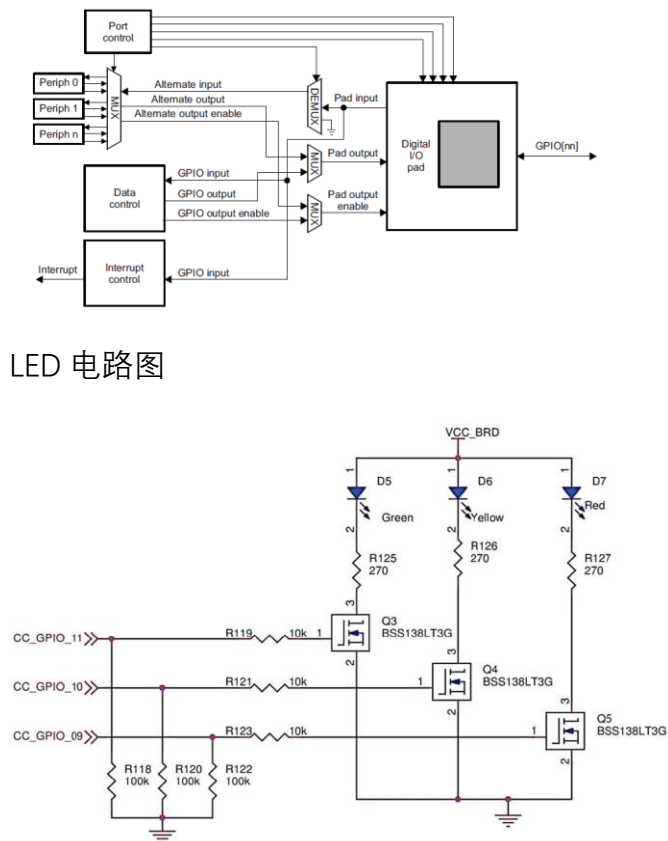
- 1、掌握定时器 PWM 控制原理及方法。
- 2、熟悉 IO 复用配置方法，熟悉中断系统应用。
- 3、熟悉串口通信配置，以及串口通信编程设计。

二、实验内容

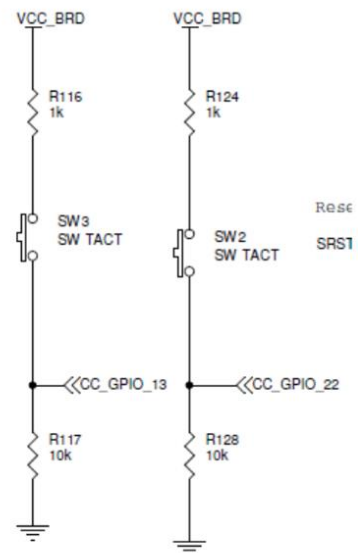
- 1、通过定时器 PWM 控制 LED 亮度
- 2、利用 UART 串口通信实现上位机对 CC3200 的控制
- 3、实现上位机监控 CC3200LED 亮度控制。

三、硬件方框图和电路图

可控 LED 闪烁灯电路图



按键电路图



四、实验原理

1、通过 PWM 控制 LED 显示程序，参考 CC3200SDK 的 PWM 工程。

在管脚复用设置中设定两个时钟，引脚映射成 GPIO 模式并设置为端口输出

```
PinMuxConfig(void)
{
    MAP_PRCMPeripheralClkEnable(PRCM_TIMER_A2, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_TIMER_A3, PRCM_RUN_MODE_CLK);
    MAP_PinTypeTimer(PIN_64, PIN_MODE_3);
    MAP_PinTypeTimer(PIN_01, PIN_MODE_3);
    MAP_PinTypeTimer(PIN_02, PIN_MODE_3);
}
```

Main 中初始化 PWM 模型，

```
void InitPWMModules()
{
    MAP_PRCMPeripheralClkEnable(PRCM_TIMER_A2, PRCM_RUN_MODE_CLK);
    MAP_PRCMPeripheralClkEnable(PRCM_TIMER_A3, PRCM_RUN_MODE_CLK);
    SetupTimerPWMMode(TIMER_A2_BASE, TIMER_B,
        (TIMER_CFG_SPLIT_PAIR | TIMER_CFG_B_PWM), 1);
    SetupTimerPWMMode(TIMER_A3_BASE, TIMER_A,
        (TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PWM | TIMER_CFG_B_PWM), 1);
    SetupTimerPWMMode(TIMER_A3_BASE, TIMER_B,
        (TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_PWM | TIMER_CFG_B_PWM), 1);
    MAP_TimerEnable(TIMER_A2_BASE, TIMER_B);
    MAP_TimerEnable(TIMER_A3_BASE, TIMER_A);
    MAP_TimerEnable(TIMER_A3_BASE, TIMER_B);
}
```

2、串口通信参考 uart_demo 例程

利用 MAP_UARTCharGet(CONSOLE)函数获得 console 中输入的字符，并回显

```
while(1)
{
    cCharacter = UartGetChar();
    g_iCounter++;
    if(cCharacter == '\r' || cCharacter == '\n' ||
        (iStringLength >= MAX_STRING_LENGTH - 1))
```

```

{
    if(iStringLength >= MAX_STRING_LENGTH - 1)
    {
        UartPutChar(cCharacter);
        cString[iStringLength] = cCharacter;
        iStringLength++;
    }
    cString[iStringLength] = '\0';
    iStringLength = 0;
    Report("\n\r cmd#%s\n\r cmd#", cString);
}
else
{
    UartPutChar(cCharacter);
    cString[iStringLength] = cCharacter;
    iStringLength++;
}
}

```

五、程序流程图和核心语句

```

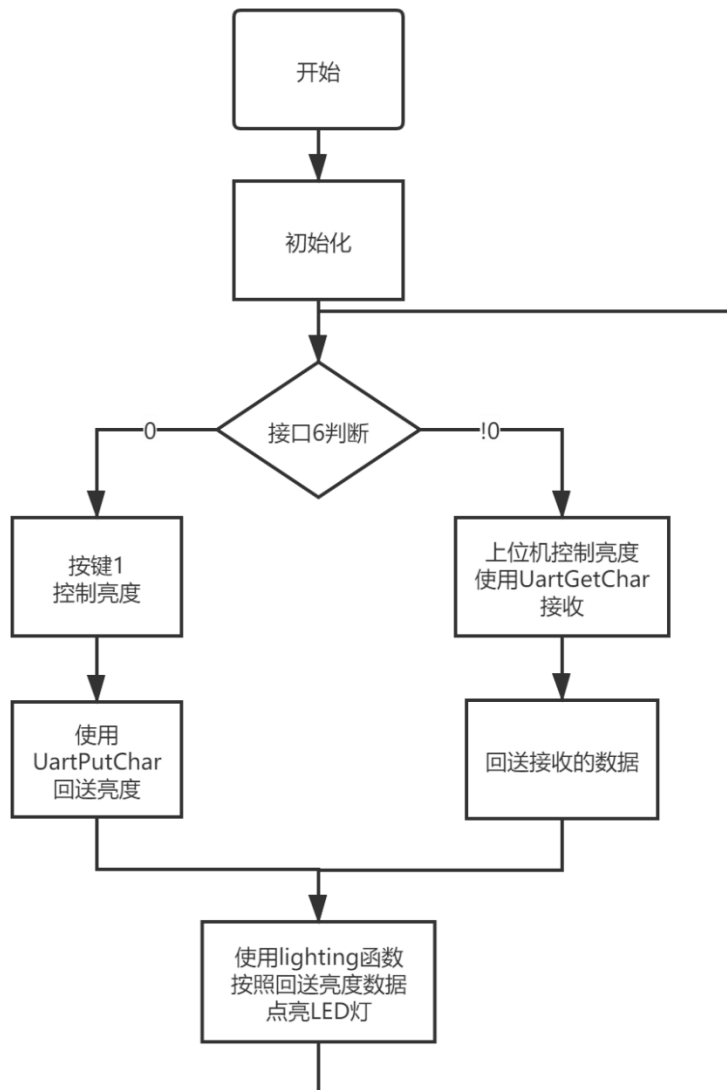
void main()
{
    int flag = 0;
    BoardInit();
    PinMuxConfig();
    InitPWMModules();
    while (1){
        if (MAP_GPIOPinRead(GPIOA2_BASE, GPIO_PIN_6) != 0)
            flag = 1;
        if (flag == 0){
            if (MAP_GPIOPinRead(GPIOA1_BASE, GPIO_PIN_5) != 0){
                brightness = (brightness + 64) % 256;
                UartPutChar(brightness);
            }
        }
        else{
            int getch = UartGetChar();
            UartPutChar(getch);
            if (getch == 255)
                flag = 0;
            else
                brightness = getch;
        }
    }
}

```

```

    }
    lighting(brightness);
    MAP_UtilsDelay(1000000);
}
}

```



六、设计过程中遇到的问题和解决方法

部分板卡还是有按键上的问题, 无法实现亮度控制中的部分功能, 只能更换板卡。

七、思考问题解答、收获和建议。

1、UART 的主要指标有哪些？

波特率

这是一个衡量符号传输速率的参数。它表示每秒钟传送的符号的个数。例如 300 波特表示每秒钟发送 300 个符号。当我们提到时钟周期时, 我们就是指波特率, 例如如果协议需要 4800 波特率, 那么时钟是 4800Hz。这意味着串口通信在数据线上的采样率为 4800Hz。通常电话线的波特率为 14400, 28800 和 36600。波特率可以远远大于这些值, 但是波特率和距离成反比。高波特率常常用于放置的很近的仪器间的通信, 典型的例子就是 GPIB 设备的通信。

数据位

这是衡量通信中实际数据位的参数。当计算机发送一个信息包, 实际的数据不会是 8 位的, 标准的值是 5、6、7 和 8 位。如何设置取决于你想传送的信息。比如, 标准的 ASCII 码是 0~127 (7 位)。扩展的 ASCII 码是 0~255 (8 位)。如果数据使用简单的文本 (标准 ASCII 码), 那么每个数据包使用 7 位数据。每个包是指一个字节, 包括开始/停止位, 数据位和奇偶校验位。由于实际数据位取决于通信协议的选取, 术语“包”指任何通信的情况。

停止位

用于表示单个包的最后一位。典型的值为 1, 1.5 和 2 位。由于数据是在传输线上定时的, 并且每一个设备有其自己的时钟, 很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束, 并且提供计算机校正时钟

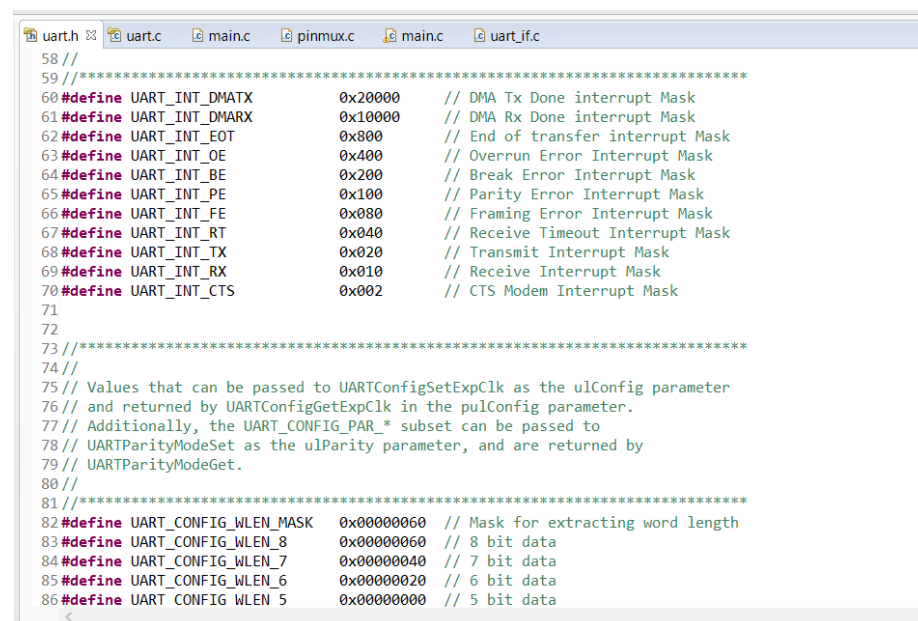
同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。

奇偶校验位

在串口通信中一种简单的检错方式。有四种检错方式：偶、奇、高和低。当然没有校验位也是可以的。对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。例如，如果数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。如果是奇校验，校验位为 1，这样就有 3 个逻辑高位。高位和低位不是真正的检查数据，简单置位逻辑高或者逻辑低校验。这样使得接收设备能够知道一个位的状态，有机会判断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。

2、UART 的参数如何设置？

在 uart.h 中设置



```
58 //
59 //*****
60 #define UART_INT_DMATX      0x20000    // DMA Tx Done interrupt Mask
61 #define UART_INT_DMARX      0x10000    // DMA Rx Done interrupt Mask
62 #define UART_INT_EOT        0x800      // End of transfer interrupt Mask
63 #define UART_INT_OE         0x400      // Overrun Error Interrupt Mask
64 #define UART_INT_BE         0x200      // Break Error Interrupt Mask
65 #define UART_INT_PE         0x100      // Parity Error Interrupt Mask
66 #define UART_INT_FE         0x080      // Framing Error Interrupt Mask
67 #define UART_INT_RT         0x040      // Receive Timeout Interrupt Mask
68 #define UART_INT_TX         0x020      // Transmit Interrupt Mask
69 #define UART_INT_RX         0x010      // Receive Interrupt Mask
70 #define UART_INT_CTS        0x002      // CTS Modem Interrupt Mask
71
72
73 //*****
74 //
75 // Values that can be passed to UARTConfigSetExpClk as the ulConfig parameter
76 // and returned by UARTConfigGetExpClk in the pulConfig parameter.
77 // Additionally, the UART_CONFIG_PAR_* subset can be passed to
78 // UARTParityModeSet as the ulParity parameter, and are returned by
79 // UARTParityModeGet.
80 //
81 //*****
82 #define UART_CONFIG_WLEN_MASK 0x00000060 // Mask for extracting word length
83 #define UART_CONFIG_WLEN_8    0x00000060 // 8 bit data
84 #define UART_CONFIG_WLEN_7    0x00000040 // 7 bit data
85 #define UART_CONFIG_WLEN_6    0x00000020 // 6 bit data
86 #define UART_CONFIG_WLEN_5    0x00000000 // 5 bit data
```