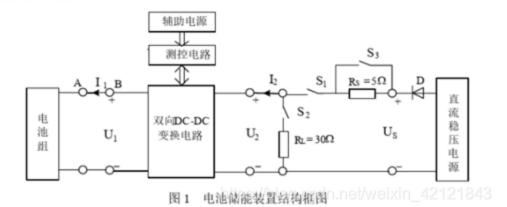


https://blog.csdn.net/weixin\_42121843

# 1.硬件部分

#### 1.1理论

#### 题目示意图



#### Boost升压电路

boost升压电路(boost converter or step-up converter)是一种常见的开关直流升压电路,它通过开关管导通和关断来控制电感储存和释放能量,从而使输出电压比输入电压高。

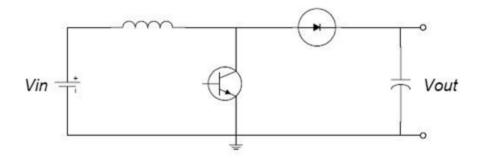


Figure 1: The Boost Converter

分析: 电感和电容的值都很大, 因此在充放电过程中, 电感电流和电容电压变化缓慢

#### 充电过程中三极管导通,等效电路如下:

此时输入电压流过电感,电感电流增加。二极管防止电容对地放电。由于输入是直流电,所以电感上的电流以一定的比率线性增加,这个比率跟电感大小有关。随着电感电流增加,电感能量增加。

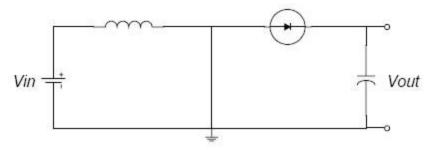


Figure 2: Boost Converter Charge Phase

#### 放电时三极管截止,等效电路如下:

由于电感的电流保持特性,流经电感的电流不会马上变为0,而是缓慢的由充电完毕时的值变为0。而原来的电路已断开,于是电感只能通过新电路放电,即电感开始给电容充电,电容两端电压升高,此时电压已经高于输入电压了。由于开关速度很快,电容放电后又被电感充电,电压变化很小,可以近似看做直流。

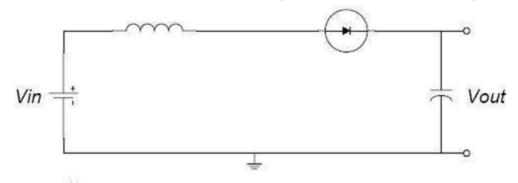
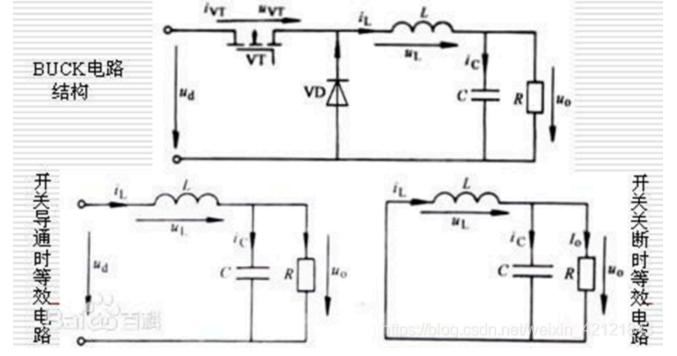


Figure 3: Boost Converter Discharge Phase And 42121843

因此,电感能量与电容能量相互转换。这个开关的过程不断重复,输出端就能一直保持比输入端高的电压,大小由三极管或MOS管的输入的占空比决定;电容量足够大的话,输出端就会保持一个持续的电流。

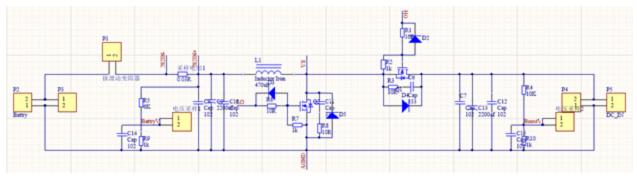
#### BUCK降压电路



与BOOST升压原理相似,MOS管导通时给LC充电,断开时LC给R供电,由于电感电流和电容电压不能突变,电压变化小,开关速度快的时候可以近似看做直流。由于直流经MOS管变成方波,其能量减少,于是输出的电压小于输入。

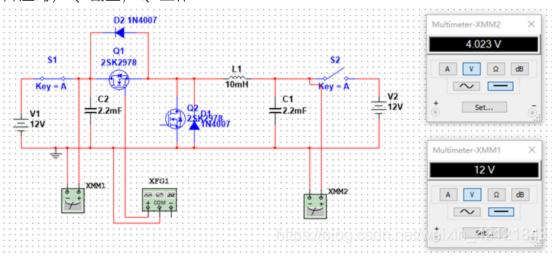
# 1.2原理图和PCB

## 主电路

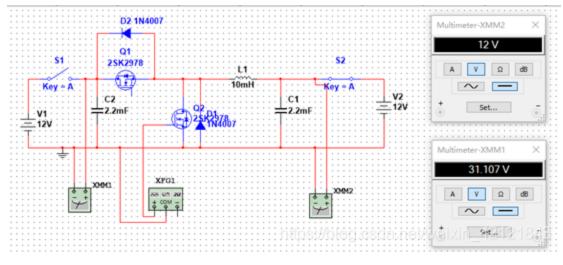


主电路仿真 (升压和降压)

## 降压时, Q2截止, Q1工作:

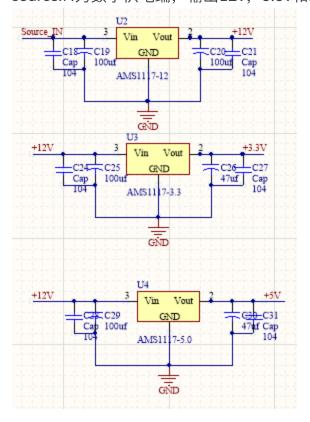


升压时, Q1截止, Q2工作

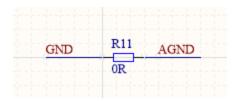


## 电源部分

sourceIN为数字供电端,输出12v, 3.3v和5v供其他模块使用

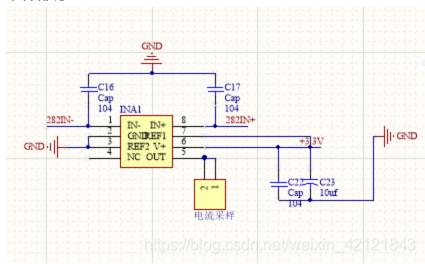


# 接地部分

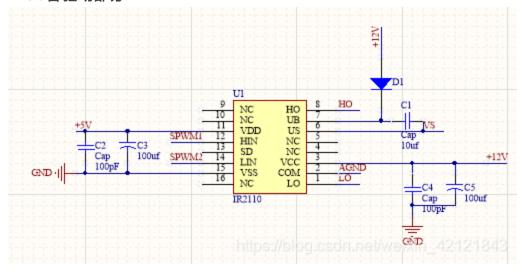


AGND为模拟地,GND为数字地,两者用OR电阻(仍有阻抗)相连。为了防止数字地干扰模拟地。 注意各部分GND的区分,最后在总电源处汇聚,一点接地。

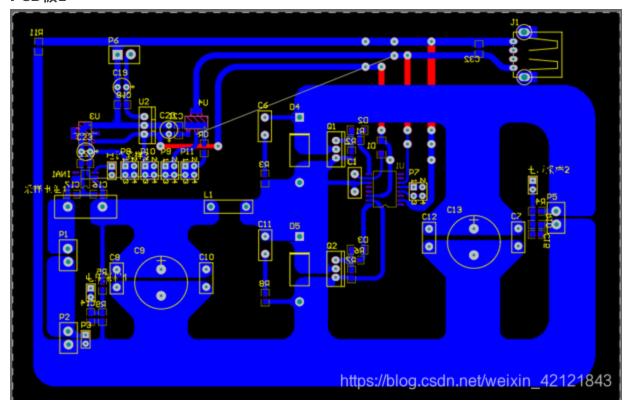
## 采样部分



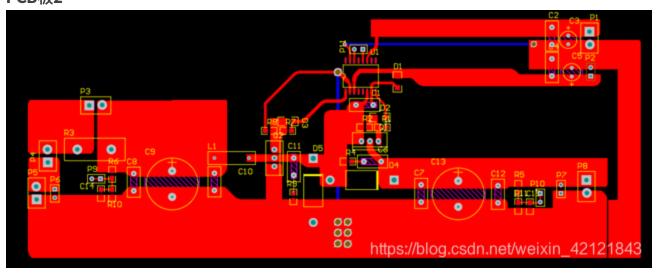
#### MOS管驱动部分



# PCB板1



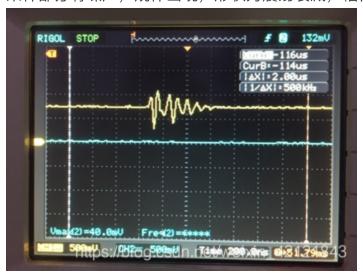
# PCB板2



# 1.3 实验数据和波形

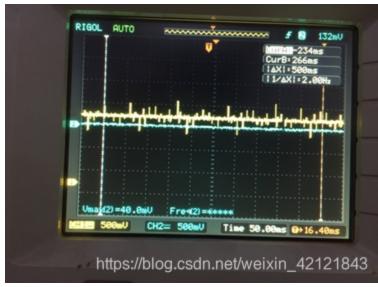
PCB板1波形及分析:

采样部分有噪声,规律出现,形状为震荡衰减,幅值约1V,原因不明:



#### 时间延长后:





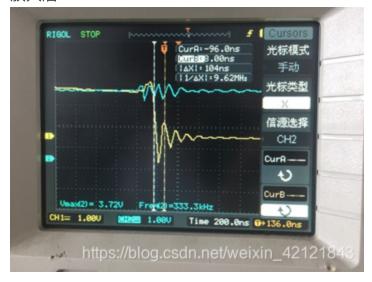
#### PCB板1:噪声原因分析

因为噪声的幅值在输出端显得很小,所以不明显。但是经过采样后噪声幅值没有变化,因此相对于采样电 压很明显。

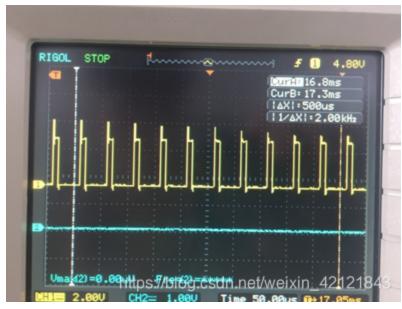
原因是MOS的下降沿,噪声频率约10MHZ:



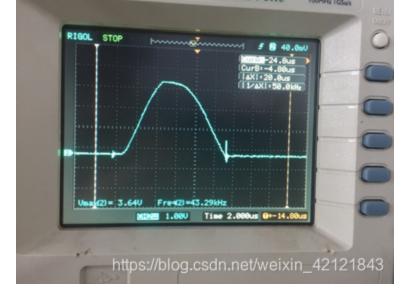
#### 放大后:



经排查,发现是单片机输出的波形有尖峰,导致mos输出震荡,更换单片机后问题得到改善:单片机波形有尖峰,且频率越高尖峰越大:

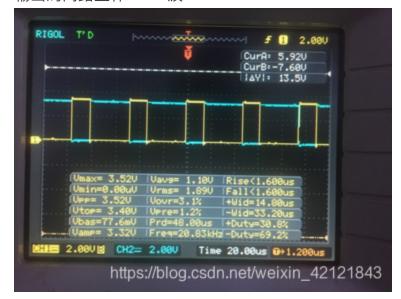


更换单片机后,波形得到改善:

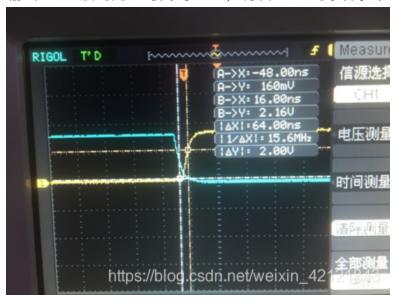


#### PCB板2波形及分析

输出的两路互补PWM波



输出PWM波的死区时间约64ns,符合MOS的条件。(ir2110没有内置死区)



# 采样GND实验记录:

当GND选得离采样点从远及近时:

# Wolt2 =1.2794 V Wolt2 =1.2705 V

原因分析: GND也有电阻, 当GND越长, 电阻越大, 其产生的误差越大。当乘一个比例时, 误差就会被放大, 因此GND务必选在采样点最近的地, 然后直接连到单片机的GND上。

#### 采样增益的测量:

电流增益根据INA282增益以及采样铜丝估计,而后实际测量多次,求平均值。 电压增益根据电阻分压大致估算,测量方法同上。

实验时测量数据:

10 inst	Task P.
JMA282供电=3.96V.	
# DUT I	the OUTZ
1.56 V. 0.004A.	数注: 1.56 V. 0.004A
1.54v. 0.045A	1.47Y 0.(9)A
1-381. 0.436A.	1-36V 0.372A
129V V.508A.	1-29/ 0.5/48
126V 055 5A	1.22V 0.63/1A
1.23V 0.616A	1-15Y 0.764A
1-16V. 0.735A	1-11V D.846A
1-11V 0-844A	₿',
(-04U. 0.97.9A	1.05 V 0.95/A
0.95V. 1-145A	0.921 1.1922
0.87V 1.285A	0-814, 1,3962
0-78V. 1.4453A	
6.69V. 1.615A	
11.	0.537 0.541
$\bigcup_2$ :	0.548 0.540
# #AT 15V 0 013+.	06-25V 0.561V. 0.569 0.5421
878V 0-078V	7.03V 0.631V. 11. 0.569 0.5421
1. 57V 0.23V. (1.17	8.35Y. 0.748Y 1150 576
.87V 0-347V 11.15	9-70 V 8-8714 054)
	https://blog.csdn.net/weixin_42121843

经分析,测量值与理想值误差原因有很多,主要为电阻的误差,其次是电路布局,焊接时的状态等等。

# 输出波形

输入28v时输出20v,看其FFT可知,纹波非常小。

# 2.软件部分

2.1代码(STM32控制器)

#### 位置PID部分:

```
543
     函数名称:Pid_modulate
           能:PID调节实现函数
544
     功
           数: Set: 预期值
545
              Act:实际值
i:模式选择: 0: 调节电压U; 1: 调节电流I1
546
547
     accuracy: 精度控制
返 回 值: 无
548
549
550
     -*********
551
     void Pid_modulate(float Set, float Act, int i, float accuracy)
552 □ {
553
       float tempI1, tempU1, tempU2, tempI2;
554
       float PID_con;
555
       float temp_p, temp;
       //ADC_get(&tempI1, &tempI2, &tempU1, &tempU2);
556
557
       temp = Act;
       if (i==1)
558
559 E
         if ((Act-Set)/Set>accuracy||(Act-Set)/Set<-accuracy)</pre>
560
561 E
562
           PID_con = temp;
           while((Act-Set)/Set>accuracy||(Act-Set)/Set<-accuracy)</pre>
563
564 E
565
             LED1=0:
566
             TFT_Show():
567
             PID_con = PID_realize(Set, Act, i);
             PID_val=PID_con/temp;
568
569
             ADC_get(&tempI1, &tempI2, &tempU1, &tempU2);
             if (status=='1')
570
571 🖨
               printf("A %f\r\nU2 %f\r\nU1 %f\r\n", duty*PID_val, tempU2*K_Volt2, tempU1*K_Volt1)
572
573
               Act=To_Current(tempI1, Volt0, K_Current);
574
575
             if (status=='1'&&tempU1*K_Volt1>24.5)
576 E
577
               status='0';
578
               break;
579
           3.
580
```

```
582
          else
583 🖹
            LED1=1;
584
585
            if (status=='1'&&tempU1*K_Volt1>24.5)
586 白
587
                status='0';
588
589
        }
590
591
        else if (i==0)
592 🖨
593
          if ((Act-Set)/Set>accuracy||(Act-Set)/Set<-accuracy)</pre>
594 🖨
595
            PID_con = temp;
            while((Act-Set)/Set>accuracy||(Act-Set)/Set<-accuracy)</pre>
596
597 🖨
598
              LED1=0;
599
              TFT_Show();
              PID_con = PID_realize(Set, Act, i);
600
              PID_val = 1/duty - PID_con/temp;
601
              ADC_get(&tempI1,&tempI2,&tempU1,&tempU2);
if(status=='2'||status=='3')
602
603
604 🖨
605
                printf("A %f\r\nU2 %f\r\n", duty*PID_val, tempU2*K_Volt2);
606
                Act=tempU2*K_Volt2;
607
608
            }
609
610
          else
611 🗎
612
            LED1=1:
613
614
        }
615
616
     }
617
618
不工作模式
函数名称: MODEO_NOT_WORKING
      能:模式封装: NOT WORKING模式
数:无
功
   回 值:无
void MODEO_NOT_WORKING(void)
1 {
  float tempI1, tempI2, tempU1, tempU2;
```

/\*\*/ printf("Cur\_set = %.1f A\t\tstatus = %s\t\tduty = %f\r\n", Cur\_set, STATUSs[status-48], duty\*PID\_val);
ADC\_get(&tempI1,&tempI2,&tempU1,&tempU2);

printf("VOLT+: %f\t\tCURR1: %f A\r\n", tempI1, To\_Current(tempI1, Volt0, K\_Current));

//显示屏数据更新

TFT\_Show();

}

#### 充电模式

```
******************************
 函数名称: MODE1_CHARGING
           能:模式封装: CHARGING模式
 参数:无返回值:无
 void MODE1_CHARGING (void)
    float tempI1, tempI2, tempU1, tempU2;
/**/ printf("Cur_set = %.1f A\t\tstatus = %s\t\tduty = %f\r\n", Cur_set, STATUSs[status-48], duty*PID_val);
ADC_get(&tempI1, &tempI2, &tempU2, &tempU2);
/**/ printf("VOLI+: %f\t\tCURR1: %f A\r\n", tempI1, To_Current(tempI1, Volt0, K_Current));
/**/ printf("A %f\r\nI %f\r\n", duty*PID_val, To_Current(tempI1, Volt0, K_Current));
IFI_Show(); //显示屏数据更新
    if (Cur_set==2.0)
       ADC_get(&tempI1,&tempI2,&tempU1);
Pid_modulate(Cur_set,To_Current(tempI1,Volt0,K_Current),1,&CCURACY/2);

printf("A %f\r\nI %f\r\n",duty*PID_val,To_Current(tempI1,Volt0,K_Current));
if(status=='0')
          Gui_DrawFont_GBK16(32, 0, BLACK, WHITE, STATUSs[status-48]);
          Current_Decline(1);
   }
    else
       ADC_get (&tempI1, &tempI2, &tempU1, &tempU2)
      Pid_modulate(Cur_set, To_Current(tempI1, Volt0, K_Current), 1, ACCURACY);
/ printf("A %f\r\n" %f\r\n", duty*PID_val, To_Current(tempI1, Volt0, K_Current));
if(status=="0")
          Gui_DrawFont_GBK16(32, 0, BLACK, WHITE, STATUSs[status-48]);
          Current_Decline(1);
   }
}
```

#### 放电模式(与自动模式共有,显示时判断键值即可)

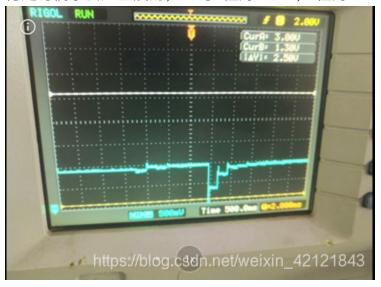
# 2.2数据与波形分析

#### 充电模式波形

位置式PID调节电流I1时所产生的波形,震荡较大,采用增量式PID和自适应调节,可能效果会好一些

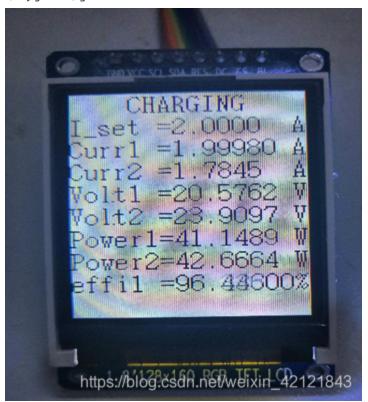


稳定时偶尔会产生震荡,此时P值为0.001, I值为0.05,。将铅蓄电池更换为锂电池之后,波形得到改善。

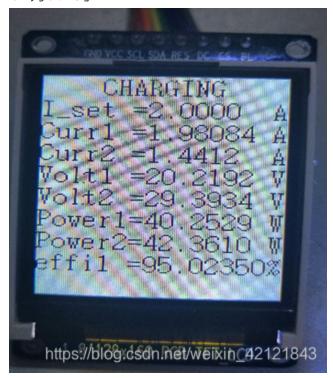


充电模式数据

# U2为24V时



U2为30V时



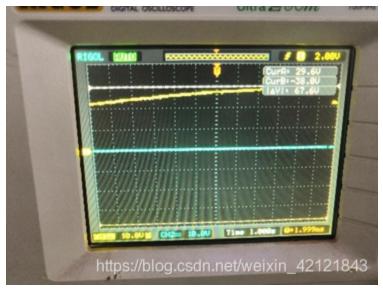
U2为36V时

```
CHARGING
I_set =2.0000 A
Curr1 =1.99680 A
Curr2 =1.2554 A
Volt1 =20.7249 V
Volt2 =35.9991 V
Power1=41.3841 W
Power2=45.1932 W
effil =91.57147%

https://blog.csdn.net/weixin_42121843
```

#### 放电模式波形

电压调节过程,基本较稳定(PID参数为Kp=0.001,Ki=0.03,故电源调节不太需要PID,平稳上升,波动小是最重要的



#### 放电模式数据

电压在一段时间后稳定下来,满足30(±0.5)的要求,但效率在90%-92%之间波动(显示最高在94%), 未达到要求。

```
DISCHARGING
I_set =1.0000 A
Curr1 =-2.0473 A
Curr2 =-0.9783 A
Volt1 =15.8095 V
Volt2 =30.2640 V
Power1=-32.3662W
Power2=-29.6087W
effi2 =91.4815 %
```



#### 原因分析

效率问题可能出在二极管和MOS管上

第一次测的放电效率为91.3%-91.6%(取十次串口返回的效率值的平均值)。

更换MOS管之后效率为92.1%-92.5%,效率依然没有提上去多少。

二极管,选择最大反向电压为60/42的SR260二极管,压降已经是比较低

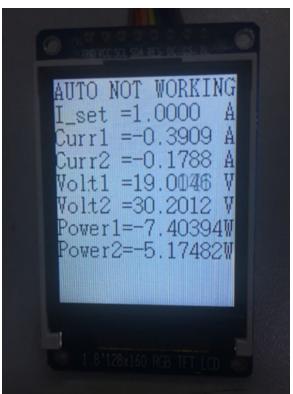
0.12	<b>洲屋明为</b>   MIZ		
②:二杯草:	CPENKY RA	(5) 像大声电流的	正阵(U)
SRIGO	60 1/42	24	0.70
plzoT	1000/700	2	1.1
11/100	50/35	1	11
W5399	1000/700	1.5	14
FR107	1000/700	1	12
FR207	1000/700	2	12
11400]	100/700	11	(-).
1 N 5819 1 N 4004	to/28	-	0.3440.9

注:显示效率与实际效率依然有偏差,实际效率并没有测,可能大也可能小,偏差在于ADC采样的偏差和计算时带入的之前测的比例系数。

# 自动充放电模式

电压U2在30V(±0.5V)稳定。

```
AUTO NOT WORKING
I_set =1.0000 A
Curr1 =-1.2213 A
Curr2 =-0.6633 A
Volt1 =18.5089 V
Volt2 =30.1474 V
Power1=-22.6053W
Power2=-19.9959W
```



# 总结

2015年电源题应该是比较简单的,拓扑较基础,程序也不复杂。 比较有技巧的地方是减少直流输出的纹波,PCB布局,器件参数的计算和型号选择