

电气工程学院

课程设计说明书

设计题目： 电流断续状态下 Buck 变换器设计

系 别： 电气工程学院

年级专业： 15 级应电 1-4 班

学生姓名： 杨德浩 王云兆 项兴 王雷 方盼想

段伟康 刘港 李瑞彪 庄严

指导教师： 王立乔 沈虹 张金龙 吴俊娟

时 间： 2018 年 12 月 7 日

电气工程学院《课程设计》任务书

课程名称：计算机控制综合课程设计

基层教学单位：电气工程及其自动化系

指导教师：沈虹

学号	150103020081 150108040055 150103030112 130103030144 140103030092 150103030103 150103030104 150103030106 150103030120	学生姓名	杨德浩 王云兆 庄严 项兴 王雷 方盼想 段伟康 刘港 李瑞彪	(专业) 班级	15 应电 1-4 班
设计题目	Buck 变换器设计				
设计技术参数	设计 Buck 变换器。 1、主电路设计：（同电力电子课程设计，已完成） 2、控制系统设计： 采用数字控制器 STM32F103RCT6 完成电压闭环控制系统设计以实现稳压输出，具体包括： 2.1 调节器及其参数设计，算法仿真及编程调试； 2.2 反馈采样、调理电路设计，AD 采样程序编写与调试； 2.3 基于 MCU 实现 PWM 控制，实现开环及闭环调节。 2.4 数字 PI 的性能测试，验证 PI 参数及采样周期对系统性能的影响。 3、过电流保护设计（保护主功率开关器件不因过载或桥臂短路而损坏） 在完成上述工作的基础上，开展仿真及硬件试验。				
设计要求	所设计系统应达到如下性能指标： 开关频率：50kHz 输入电压：24V 输出电压：5~15V 最大输出功率：100W 电感电流纹波：低于 20% 输出电压纹波：低于 5% 闭环系统要求：稳态无静差，输出电压超调量<20%，过渡过程时间<50 个开关周期				
参考资料	1、STM32F103RCT6 应用手册 2、陈伯时主编 电力拖动自动控制系统（第 3 版） 机械工业出版社 3、王兆安 张明勋 电力电子设备设计 and 应用手册 机械工业出版社				
周次	第一周		第二周		
应完成内容	完成全部方案设计： 周一、二：查阅参考资料 完成模拟到数字系统的方案论证 周三：数字系统参数设计与仿真 周四：熟悉 STM32 开发板 周五：PWM/PID/AD 程序编写		周一、二：实验及软件调试 周三、四：总结，完成设计说明书 周五：答辩考核		
指导教师签字	沈虹		基层教学单位主任签字	孙孝峰	

任务分解明细

课程名称：计算机控制课程设计

专业班级：15 应电 1-4 班

序号	周次	主要任务	设计内容	任务要求 (工作不能及时完成可顺延)
1 2	1	1.1 从模拟控制电源到数字控制电源	1) 电力电子课程设计任务及其完成情况说明，包括：任务要求，主电路及参数，主电路数学模型，调节器参数，系统闭环传递函数，系统稳态响应参数及动态响应参数，系统控制芯片及控制电路图，控制芯片调节器部分的实现， 实验调试情况汇总。 2) 从模拟调节器到数字调节器，系统保留哪些电路，单片机系统替换哪一部分电路；如果原设计为计算机系统，说明已完成工作。	周二晚 23 点前将团队报告发送至任务邮箱，团队报告给出执笔人姓名。
3 4	1	1.2 模拟离散化计算与系统仿真 1-仿真框架 1.3 模拟离散化计算与系统仿真 2-参数优化	1) 设计连续域调节器，并离散化；若原设计为数字控制系统， 需进行说明。 2) 用 m 函数，替换模拟调节器，完成 matlab 仿真。 3) 参数优化，根据系统暂稳态参数要求，优化调节器参数，达到设计要求	周四晚 23 点前将团队报告发送至任务邮箱，团队报告给出执笔人姓名。
5	1	1.4 单片机系统设计-最小系统及检测、驱动电路	1) 提供单片机最小系统电路图 2) 电压电流检测电路及驱动电路设计，与模拟系统对比 3) 元器件清单，器件价格及采购商 团队讨论：电商对开发速度的正影响；互联网对开发的影响；以高精度电源和民用车载电源价格对比分析产品附加值	模拟与数字系统元器件成本对比、参数修改等调试成本对比，将经济决策思想应用到成本核算。 任务要求：周日 23:00 前将团队报告发送至任务邮箱。
6	2	2.1 单片机系统设计-基础软件调试	1) 系统控制流程分析与设计 2) 结合 CDIO 程序，修改为本课题程序，注明你修改的部分。 3) 软件调试，基于 STM32 平台，测试 PWM 输出等数字系统基础特性。	
7	2	2.2 单片机系统设计-信号检测及开环调试	1) 控制板与主电路连接 2) 系统开环调试 3) 电压电流检测调试	
8 9	2	2.3 单片机系统设计-闭环调试	1) 闭环实验调试与优化 2) 团队报告 ppt 制作 3) 将已完成个人资料汇总整理成为设计说明书。绘制大图	任务要求：周四 23:00 前将团队报告发送至任务邮箱，实验及调试总结。
10	2	2.4 分析总结与答辩	团队 ppt 制作及答辩	周五中午 12:00 前将 ppt 发送任务邮箱

摘要

随着科技的进步，开关电源广泛应用于各个行业。本文主要对电流断续状态下直流 24V 输入的 Buck 变换器进行设计，使其输出电压能稳定在 10V。本文在模拟电路的基础上，采用数字控制器，首先采用 m 函数对数字控制器进行编程，用 MATLAB 仿真确定调节器的 PI 参数；然后用 32 单片机编程，采用了单电压闭环来控制输出电压，展开实验。

关键字：Buck 变换器；电流断续；MATLAB；数字控制

目录

1 绪论	4
1.1 DC/DC 变换器的发展概况和趋势	4
1.2 本课题的目的与任务	4
2 主电路设计	5
2.1 主电路	5
2.2 参数设计	5
2.2.1 电感	5
2.2.2 电容	6
2.2.3 稳态工作点	6
2.3 硬件设计	7
2.3.1 电感	7
2.3.2 电容	8
2.3.3 开关管	9
3 开环仿真	10
4 单电压闭环	11
4.1 DCM 模式下的建模	11
4.2 控制器的设计	12
4.3 m 函数编写	13
4.4 仿真	15
4.4.1 参数设置	15
4.4.2 仿真波形	16
5 硬件部分	18
5.1 STM32F103RCT6 控制电路	18
5.1.1 MCU 部分的原理图	18
5.1.2 PCB	19
5.1.3 3D 效果图	19
5.1.4 JTAG 部分电路	20
5.1.5 按键	20
5.1.6 USB、电源	21
5.2、驱动电路	22
5.2.1 驱动电路原理图	22
5.2.2 PCB	22
5.2.3 3D 效果图	23
5.2.4 BOM 报表	23
5.2.5 电压检测电路	24
5.2.6 驱动电路	24
5.2.6 与模拟系统对比	25
5.3 主电路	26
5.3.1 主电路原理图	26
5.3.2 PCB	26
5.3.3 3D 效果图	27
5.3.4 BOM 报表	27

5.4 UART 串口屏幕	27
5.4.1 产品特点	28
5.4.2 功能简介	28
5.4.3 原理图	29
6 软件部分	29
6.1 FreeRTOS 简介	29
6.2 FreeRTOS 特点	30
6.3 程序流程图	31
6.4 代码解析	32
6.4.1 mian.c (主函数)	32
6.4.2 PID.c (PID 调节代码)	37
6.4.3 pid.h (PID 调节代码)	38
6.4.4 timer.c (定时器与 PWM 代码)	39
6.4.5 adc.c (ADC 采样代码)	40
6.4.6 HMI.c (串口屏幕显示代码)	42
6.4.7 key.c (按键代码)	43
7 实验部分	45
7.1 开环实验	45
7.2 闭环实验	47
8 结论	48
参考文献	48

1 绪论

1.1 DC/DC 变换器的发展概况和趋势

随着电力电子技术的高速发展，电力电子设备与人们的工作、生活的关系日益密切，而电子设备都离不开可靠的电源。进入 80 年代，计算机电源全面实现了开关电源化，率先完成计算机的电源换代。进入 90 年代，开关电源相继进入各种电子、电器设备领域，程控交换机、通讯、电力检测设备电源、控制设备电源均广泛采用了开关电源，促进了开关电源技术的迅速发展。

开关电源可分为 AC/DC 和 DC/DC 两大类。以 AC/DC 变换为例，与传统采用工频变换技术的相控电源及线性电源相比，采用大功率开关管的高频整流电源，在技术上是一次飞跃，它不但可以方便地得到不同的电压等级，更重要的是甩掉了体积大笨重的工频变压器。采用高频功率变换，显著减小了电源装置体积和重量，而有可能和设备的主机体积相协调，并且使电性能得到进一步提高。

而随着对节能技术的呼声越来越高，随着电子设备小型化的要求，随着对环境保护的更高要求，更高效率，更小体积，更少电磁污染，更可靠地工作的开关电源几乎每个月都在推陈出新。

1.2 本课题的目的与任务

当前国内的 DC/DC 变换技术广泛应用于远程及数据通讯、计算机、办公自动化设备、工业仪器仪表、军事、航天等领域，涉及到国民经济的各行各业。但随着对节能技术的呼声越来越高，随着电子设备小型化的要求，随着对环境保护的更高要求，开关电源技术也在飞速地发展着。更高效率，更小体积，更少电磁污染，更可靠地工作的开关电源几乎每个月都在推陈出新。随着计算机控制技术的发展，数字控制相较于模拟控制而言，具有控制算法灵活，可靠性高，生产安全等优点，拥有广阔的发展前景。

本课题正是运用数字控制对断续状态的 Buck 变换器进行闭环设计。而在设计电流断续下的 Buck 电路的过程中，也可以将我在大学期间所学的理论知识和实际有机结合起来，从而提高我们分析问题，解决问题的能力。

2 主电路设计

2.1 主电路

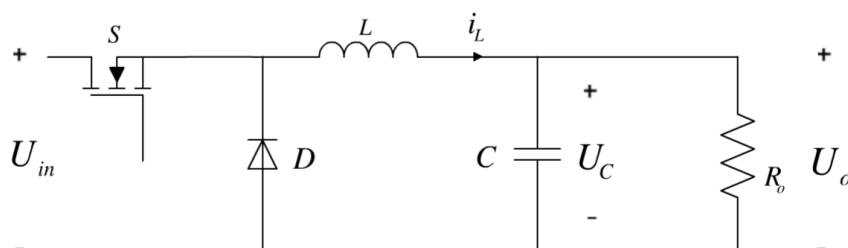


图 1 Buck 主电路

Buck 型变换器是一种单开关非隔离变换器。其电路组成如上图所示，它由一个电子开关 S ，二极管 D ，电感 L ，电容 C 和一个基本负载电阻 R 构成。如果让开关 S 周期性通断，对输入电压进行斩波，在二极管的两端可以得到一连串方波电压。经过串联电感滤波电路的滤波，在输出端就可以得到平稳的直流输出电压 V_o 。控制开关 S 开通和断开的比例，就可以对输出电压的高低进行控制。

2.2 参数设计

2.2.1 电感

电流连续时，电流波动 $\Delta I_L = \frac{V_o}{L} t_{off}$ ，输出的负载电流 I_o 等于电感电流的平均值，脉动成分就是滤波电容的充放电电流。

$$I_o = I_L$$

$$I_c = I_L - I_o$$

由临界电感的定义（使电感电流处于连续与断续的分界点的电感值），当

$$\frac{\Delta I_L}{2} = I_o$$

时电路处于临界状态。由此可得：

$$L_c = \frac{R(1-D)}{2f_s}$$

此次我们研究的是电感电流断续时的情况。它出现在电感值过小、负载电阻较大、占空比较小的时候。电感电流断续时的波形如下图。所谓电流断续是指在一个周期尚未结束时电感电流就已经衰减到零的情况。由于仅在开关断开期间 t_{off} 的一部分时间存在电感电流，其余时间没有电流，所以必须同时使用二极管导通比 D_2 来描述问题。

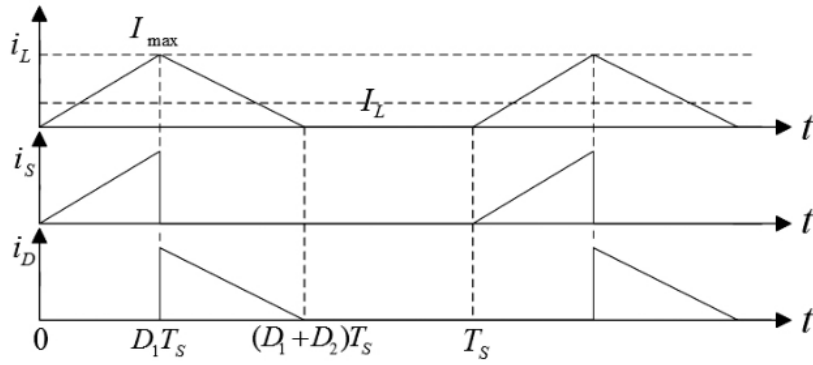


图 2 Buck 电路电感电流断续波形

根据题目要求我们选定输出电压 $V_o=10V$ ，根据实验条件，负载电阻 $R=20\Omega$ ，开关频率 $f_s = 50kHz$ ，滤波电感 $L=100\mu H$ 。若是在连续时 $D = \frac{V_o}{V_i} = \frac{10}{24}$ ，可以求出临界电感 $L_c = 116.7\mu H > 100\mu H$ ，所以稳态工作点处于电流断续状态。从而求得系统的标么时间常数 $\tau_L = \frac{L}{RT_s} = 0.25$ 。

2.2.2 电容

在电感电流断续时，变换器的电压增益为

$$\frac{V_o}{V_i} = \frac{2}{1 + \sqrt{1 + \frac{8\tau_L}{D_1^2}}}$$

将 $V_o=10V$, $V_i=24V$ 带入上式可得开关管导通比 $D_1=0.386$ ，题目要求电压纹波 $\gamma < 5\%$ ，再根据公式

$$D_2 = \frac{D_1}{2} \left(\sqrt{1 + \frac{8L}{RT_s D_1^2}} - 1 \right)$$

$$\gamma = \frac{(D_1 + D_2)}{2RCf_s} \left(\frac{D_2}{\tau_L} + \frac{\tau_L}{D_2} - 2 \right) < 5\%$$

可得 $D_2=0.54$ ， $C=10\mu f$ 。

2.2.3 稳态工作点

从上文我们选取的滤波器结构，我们已知

输入电压 $V_i=24V$

输出电压 $V_o=10V$

电感 $L=100\mu H$ ，电容 $C=10\mu f$ ，负载电阻 $R=20\Omega$

系统的标么时间常数 $\tau_L = 0.25$

开关管导通比 $D_1=0.386$ ，续流管导通比 $D_2=0.54$

2.3 硬件设计

2.3.1 电感

我们设计的电感为 $100\mu\text{H}$ ，通过电流为 $I_L=0.5\text{A}$ ，工作在电流断续状态，电流脉动 $\Delta I=1\text{A}$ ，开关频率 $f_s=50\text{KHz}$ ，最高温度限制不超过 90° 。

因为磁芯工作时会有温升。当温度比较高时，对磁性材料的磁性会产生影响，应该考虑温升对磁感应强度的影响。故应适当低取 B_w 值比较稳妥。今采用磁芯为 TDK 公司的 PC40 磁芯，型号选择为 EE 型。

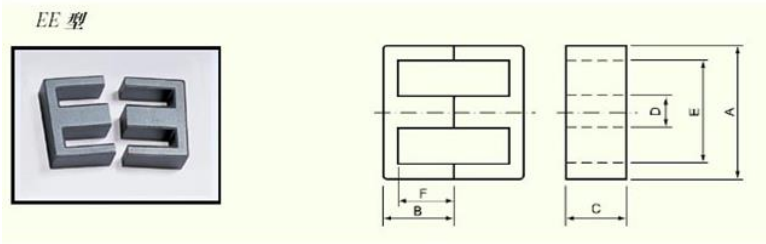


图 3 EE 型磁芯

查 PC40 磁芯，BS 在 25° 时为 510mT ，在 60° 时为 450mT ，而在 100° 时为 390mT 。显然，我们应该取 390mT 。而剩余磁感应强度 B_r 的数值为： 25° 时为 95mT ，在 60° 时为 65mT ，而在 100° 时为 55mT 。综合考虑，按 60% 低取 B_w 为 $0.6 \times (390 - 60) = 198\text{mT}$ ，最后确定 B_w 取为 0.2T 。绕线载流量为 $J=4\text{A}/\text{mm}^2$ ，窗口利用系数 $K_0=0.2$ 估算所需磁芯尺寸如下：

$$A_w A_e = \frac{LI^2}{B_w J K_0} = \frac{100 * 0.5^2}{0.2 * 4 * 0.2} = 156.25 \text{mm}^4$$

TYPE	MATERIAL	Dimensions (mm)	A_p	A_e	A_w	A_L
		$A * B * C$	(cm^4)	(mm^2)	(mm^2)	(mH/N^2)
EE05	PC40	5.25*2.65*1.95	0.0013	2.63	5.00	285.00
EE6.3	PC40	6.1*2.85*7.95	0.0015	3.31	4.46	405.00
EE8	PC40	8.3*4.0*3.6	0.0091	7.00	13.05	590.00
EE10/11	PC40	10.2*5.5*4.75	0.0287	12.10	23.70	850.00
EE13	PC40	13.0*6.0*6.15	0.0570	17.10	33.35	1130.00
EE16	PC40	16*7.2*4.8	0.0765	19.20	39.85	1140.00
EE19	PC40	19.1*7.95*5.0	0.1243	23.00	54.04	1250.00
EE19/16	PC40	19.29*8.1*4.75	0.1191	22.40	53.15	1350.00
EE20/20/5	PC40	20.15*10*5.1	0.1572	31.00	50.70	1460.00
EE22	PC40	22*9.35*5.75	0.1590	41.00	38.79	2180.00
EE2329S	PC40	23*14.7*6	0.4368	35.80	122.00	1250.00
EE25/19	PC40	25.4*9.46*6.29	0.3128	40.00	78.20	2000.00
EE25.4	PC40	25.4*9.66*6.35	0.3173	40.30	78.73	2000.00
EE2825	PC40	28*12.75*10.6	0.8525	86.90	98.10	3300.00
EE30	PC40	30*13.15*10.7	0.7995	109.00	73.35	4690.00
EE30/30/7	PC40	30.1*15*7.05	0.7455	59.70	124.87	2100.00

图 4 磁芯参数

通过查表，我们选择磁芯 EE16，查得 $A_w A_e = 747.84 \text{mm}^4$ 。大于计算值，且有充分裕量，满足要求。再查 EE16 磁芯 $A_e=19.2\text{mm}^2$ ，有气隙时的电感系数 $A_L=1140\text{nH/N}^2$ 。

可以算出匝数：

取为整数 10 匝。考虑直流分量设置的气隙为

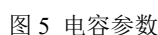
由于工作频率是 50KHz，考虑高频电流导线的集肤效应，需要计算穿透深度：

式中 ω 是工作角频率, ρ 是导线电阻率, 带入具体数值计算结果为

可选 $\varphi = 0.59$ 的导线，两股并绕。

安装在整流电路两端用以降低交流脉动波纹系数提升高效平滑直流输出的一种储能器通常把这种器件称其为滤波电容。当整流电压高于电容电压时电容充电，当整流电压低于电容电压时电容放电，在充放电的过程中，使输出电压基本稳定。

选取电容型号为 CL31B106KBHNNNE，如下图所示。



2.3.3 开关管

此次开关频率 $f_s = 50\text{kHz}$ ，我们选用 IGBT 作为功率开关管。

1、额定电压的选择

开关管导通期间无需承受电压。在开关管关断续流二极管导通时，开关管承受电压为 $V_i=24\text{V}$ ；而当续流二极管关断，电感电流断续时开关管承受电压为 $V_i-V_o=14\text{V}$ ，开关管的额定电压一般要求高于承受电压的两倍，根据 IGBT 规格的电压等级，选择 600V 电压等级的 IGBT。

2、额定电流的选择

开关管导通期间流过负载的平均电流为 $I_{av} = \frac{V_o}{R} = 0.5\text{A}$ ，由于电感电流断续，流过开关管的最大电流 $I_{max} > 2I_{av} = 1\text{A}$ 。开关管的额定电流一般要求留有 1.5 倍的裕量，选择 75A 电流等级的 IGBT。

综上，选取型号为 CT75AM-12 的 IGBT 开关管，如下图所示。

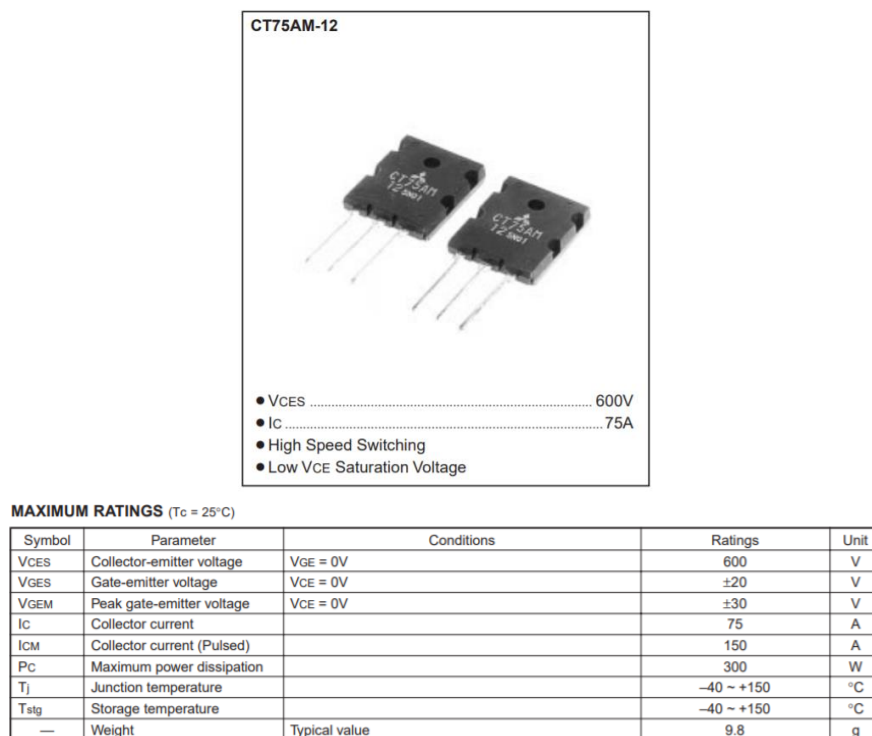


图 6 开关管参数

3 开环仿真

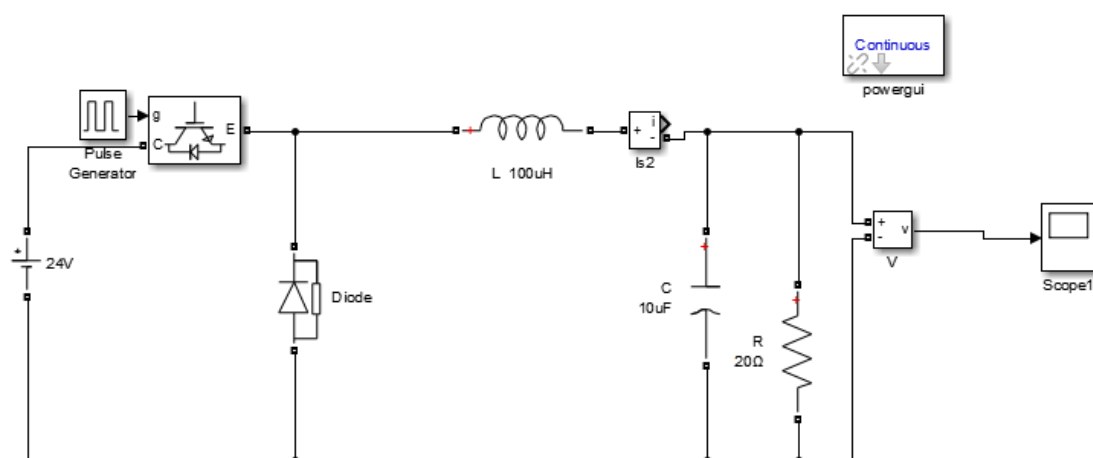


图 7 开环电路仿真

驱动采用 PWM 脉冲驱动，占空比为 38.6%，周期为 0.02ms。滤波电感取 $L=100\ \mu\text{H}$ ，滤波电容取 $C=10\ \mu\text{F}$ ，负载电阻 $R_L=20\ \Omega$ 。

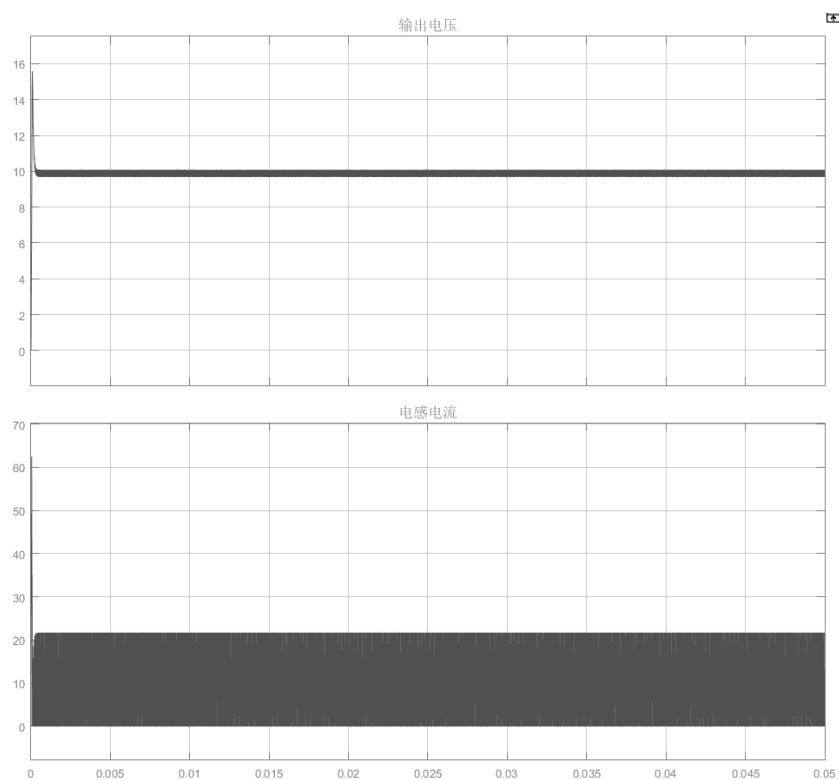


图 8 输出电压与电感电流波形

开环仿真图与仿真波形如上。可以看到输出电压在 10V 左右，下面来查看指标要求。

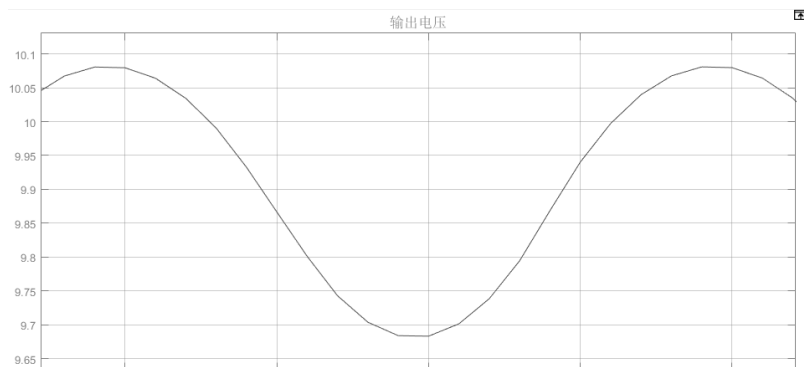


图9 输出电压纹波

电压纹波 $\gamma = \frac{10.08-9.68}{10} * 100\% = 4\%$ ，符合要求。

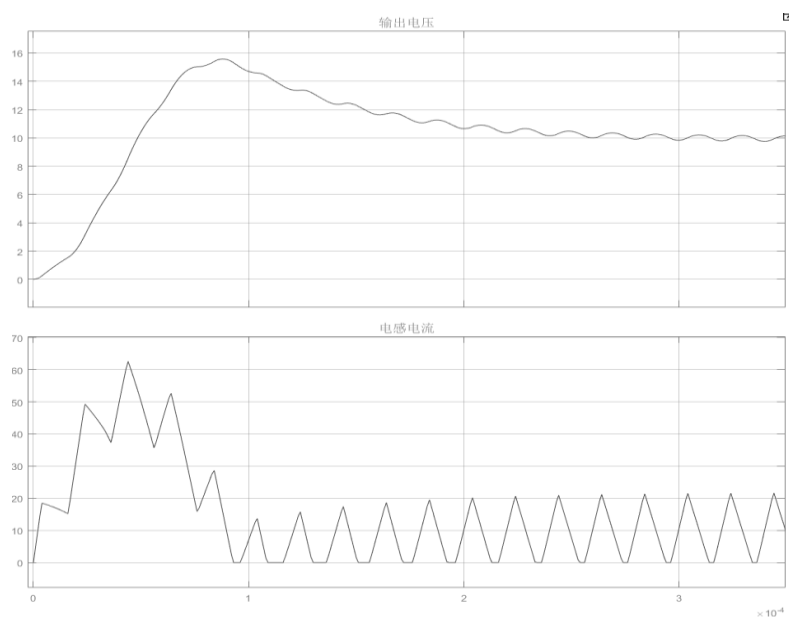


图10 输出电压与电感电流波形

从上图我们看出电感电流断续，电压超调为 $\frac{15.5-10}{10} * 100\% = 55\%$ ，不符合要求，所以我们接下来要用闭环来降低超调。

4 单电压闭环

4.1 DCM 模式下的建模

首先列写各阶段状态方程和电感电流平均变量方程，然后求静态工作点和 D_2 ，列写理想 Buck 变换器的小信号状态方程与输出方程为

$$\begin{bmatrix} \dot{i}(t) \\ \dot{\hat{v}}(t) \end{bmatrix} = \begin{bmatrix} 0 & -\frac{D_1 + D_2}{L} \\ \frac{D_1 + D_2}{C} & -\frac{1}{RC} \end{bmatrix} \begin{bmatrix} \dot{i}(t) \\ \dot{\hat{v}}(t) \end{bmatrix} + \begin{bmatrix} \frac{D_1}{L} \\ 0 \end{bmatrix} \hat{v}_g(t) + \left(\begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix} \begin{bmatrix} I \\ V \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} V_g \right) \hat{d}_1(t) \\ + \begin{bmatrix} 0 & -\frac{1}{L} \\ \frac{1}{C} & 0 \end{bmatrix} \begin{bmatrix} I \\ V \end{bmatrix} \hat{d}_2(t)$$

$$\hat{i}_g(t) = [D_1 \ 0] \begin{bmatrix} \dot{i}(t) \\ \dot{\hat{v}}(t) \end{bmatrix} + [1 \ 0] \begin{bmatrix} I \\ V \end{bmatrix} \hat{d}_1(t)$$

电感电流提供的辅助方程为

$$\frac{d\hat{i}(t)}{dt} = 0$$

$$\hat{i}(t) = -\frac{D_1 T_s}{2L} \hat{v}(t) + \frac{D_1 T_s}{2L} \hat{v}_g(t) + \frac{T_s}{2L} (V_g - V) \hat{d}_1(t)$$

上面四个式子为理想 Buck 变换器的交流小信号方程组，整理可得输出 $\hat{v}(s)$ 对控制变量 $\hat{d}(s)$ 的传递函数 $G_{vd}(s)$ 为

$$G_{vd}(s) = \frac{\hat{v}(s)}{\hat{d}_1(s)} \big|_{\hat{v}_g(s)=0} = \frac{2(1-M)}{M(2-M)} \sqrt{\frac{1-M}{K}} V \frac{1}{1 + s \frac{(1-M)RC}{2-M}}$$

式中，M 为电压增益， $M=V_o/V_i$

带入所求得稳态工作点可得主电路数学模型为

$$G_{vd}(s) = \frac{\hat{v}(s)}{\hat{d}_1(s)} \big|_{\hat{v}_g(s)=0} = \frac{19.1}{1 + 7.37 * 10^{-5} s}$$

锯齿波的幅值为 0.9V-3.3V，传递函数为 $G_m(s) = 1/2.4$

$$G_0 = G_m(s) G_{vd}(s) = \frac{7.99}{1 + 7.37 * 10^{-5} s}$$

4.2 控制器的设计

我们已知原系统传递函数为

$$G_0 = G_m(s) G_{vd}(s) = \frac{7.99}{1 + 7.37 * 10^{-5} s}$$

采用 PI 补偿网络，来消除稳态误差，达到动态要求。传递函数为

$$G_c(s) = K \frac{(1 + \frac{s}{\omega_z})}{s}$$

确定穿越频率为开关频率的 1/20， $f_c=1/5f_s=2500\text{Hz}$ 。

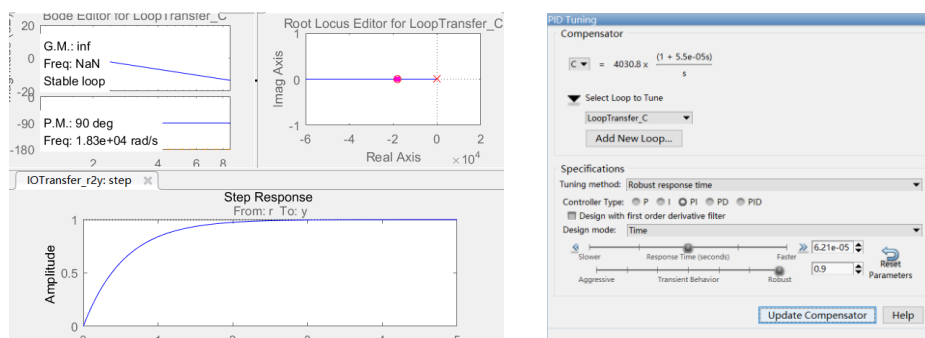


图 11 siso 工具箱

利用 MATLAB 工具箱设计出调节器传递函数为

$$G_c(s) = 4030.8 \frac{(1 + 5.5e - 5 * s)}{s}$$

根据选择的穿越频率，可得补偿器增益为 K=2291，画出 Bode 图。

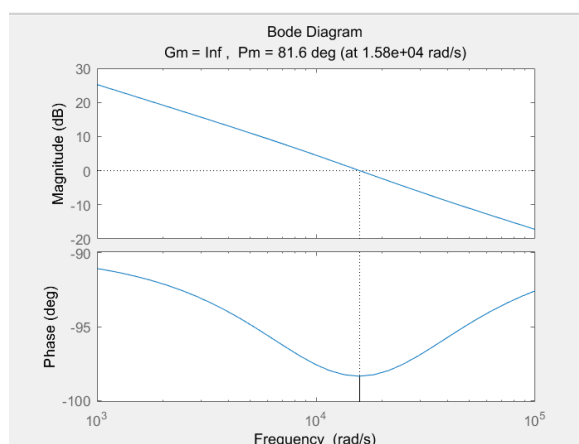


图 12 bode 图

得到调节器

$$G_c(s) = 2291 * \frac{(1 + 5.5 * 10^{-5} * s)}{s}$$

根据香农采样定理 $T \leq \frac{\pi}{\omega_{max}} = 10^{-5}$ ，选取采样时间为 $T=10^{-5}s$

将调节器离散化得

$$G_c(s) = \frac{0.1925 - 0.1696z^{-1}}{1 - z^{-1}}$$

列写调节器的差分方程：

$$u(k)=u(k-1) +0.1925e(k)-0.1696e(k-1)$$

4.3 m 函数编写

运用 s 函数进行编程。

```

function [sys, x0, str, ts, simStateCompliance] = PID1(t, x, u, flag, P, I, D)%声明变量

switch flag,

    case 0,
        [sys, x0, str, ts, simStateCompliance]=mdlInitializeSizes;

    case 2,
        sys=mdlUpdate(t, x, u, P, I, D);

    case 3,
        sys=mdlOutputs(t, x, u, P, I, D);

    case{1 4 9}
        sys=[];

    otherwise
        DASTudio.error('Simulink:blocks:unhandledFlag', num2str(flag));

end

```

```

function [sys, x0, str, ts, simStateCompliance]=mdlInitializeSizes%初始化参数

sizes = simsizes;

sizes.NumContStates = 0;
sizes.NumDiscStates = 2;
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1; % at least one sample time is needed

sys = simsizes(sizes);

x0 = [0;0];

str = [];

ts = [1e-5 0];

simStateCompliance = 'UnknownSimState';

```

```
sys(2,1)=u(1);
```

```
sys = x(1);
```

Parameters

Sample time (-1 for inherited):

1e-5

4.4.2 仿真波形

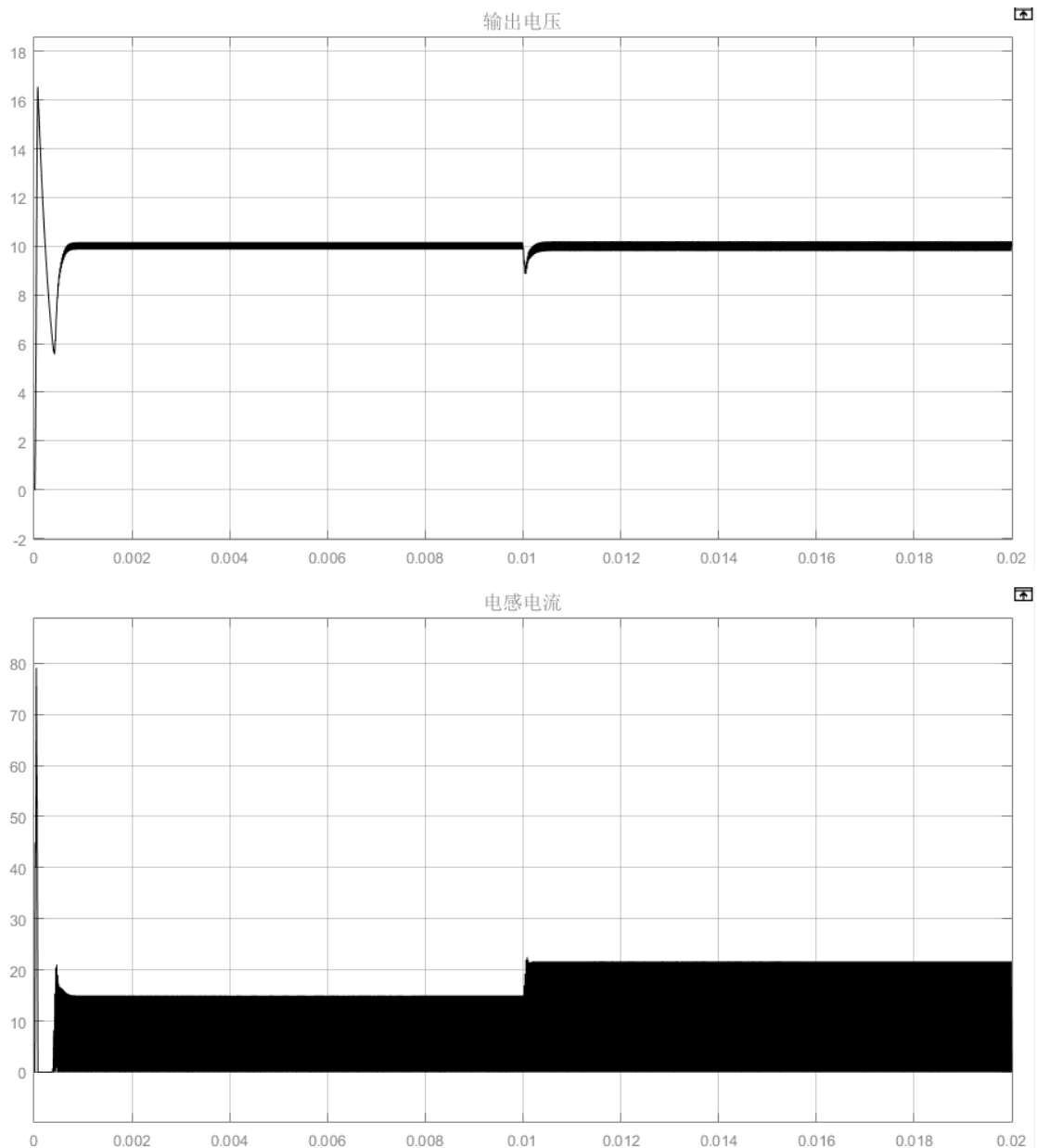
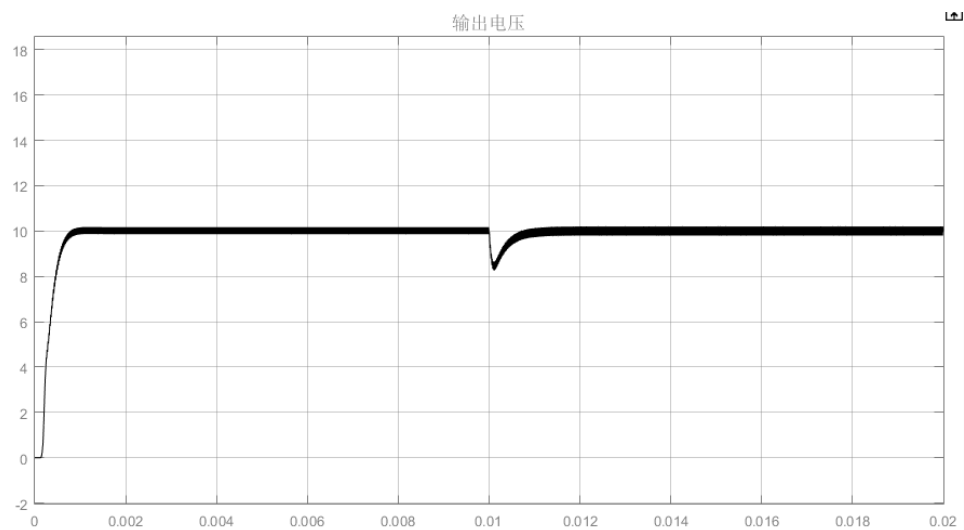


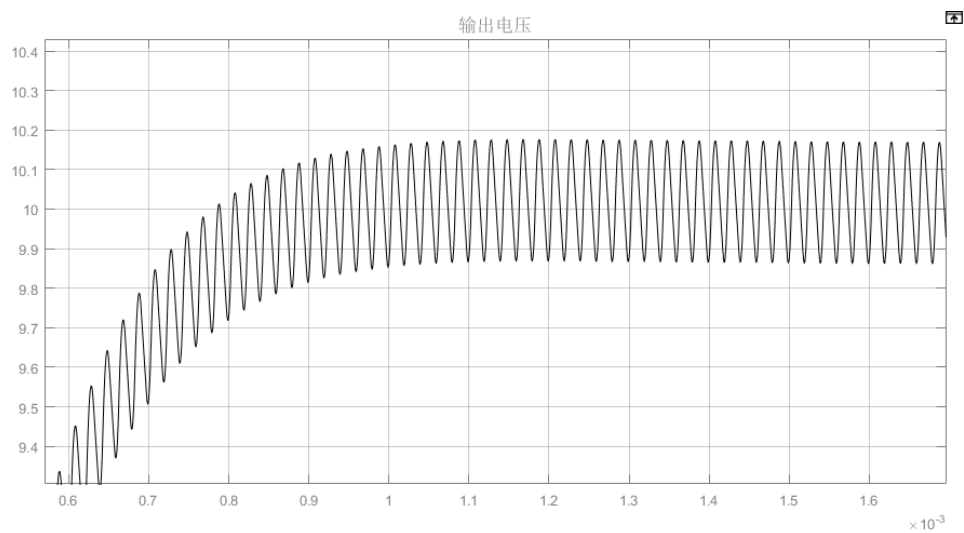
图 14 输出电压与电感电流波形图

0.01s 加入扰动，可以实现稳压。但超调过大，所以需要进行参数整定。

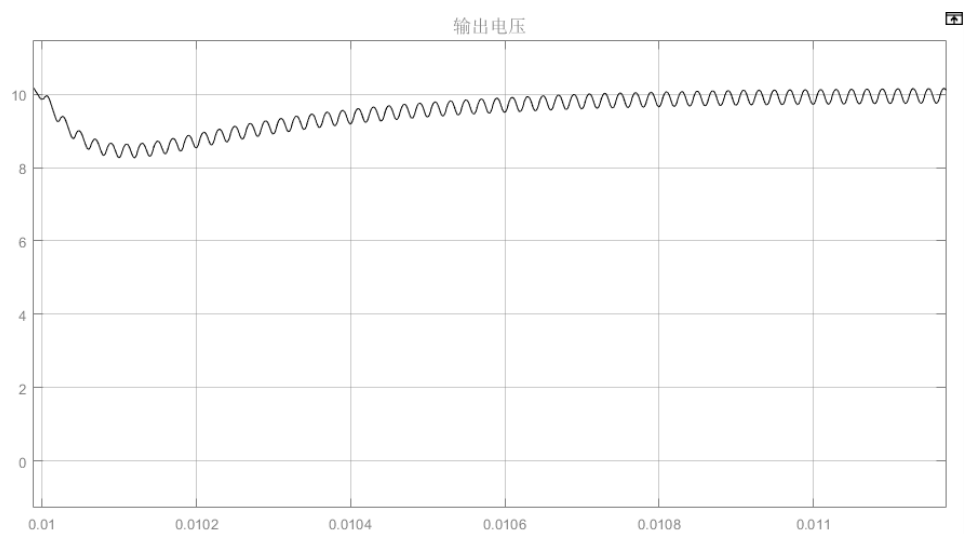
采用试凑法整定 PI 参数。整定时，首先将比例系数 K_p 由小变大，观察系统响应，得到反应快、超调小的响应曲线。再将 K_p 适当减小， T_i 的初始值取较大些，然后减小积分时间常数，是系统在保持良好动态性能的情况下，静差得以消除。在此过程中，应根据对响应曲线的满意程度反复修改比例系数和积分时间常数，以期得到满意的响应过程。



0.01s 加入扰动，可以实现稳压。



电压无超调，电压纹波为 $(10.15 - 9.85) / 10 * 100\% = 3\%$



过渡过程时间 1ms，约为 50 个开关周期。

5 硬件部分

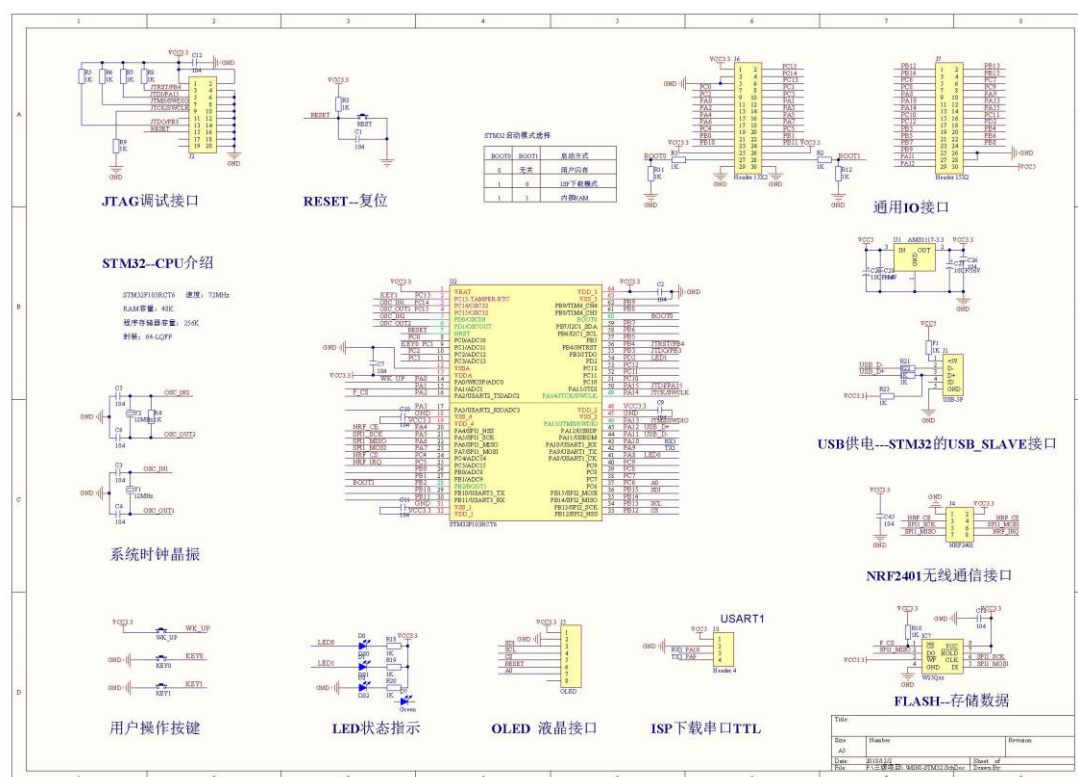
5.1 STM32F103RCT6 控制电路

Cortex-M3 采用目前主流 ARM V7-M 架构，相比曾风靡一时的 ARM V4T 架构拥有更加强健的性能，更高的代码密度，更高的性价比。Cortex-M3 处理器结合多种突破性技术，在低功耗、低成本、高性能三方面具有突破性的创新，使其在这几年迅速在中低端单片机市场异军突起。

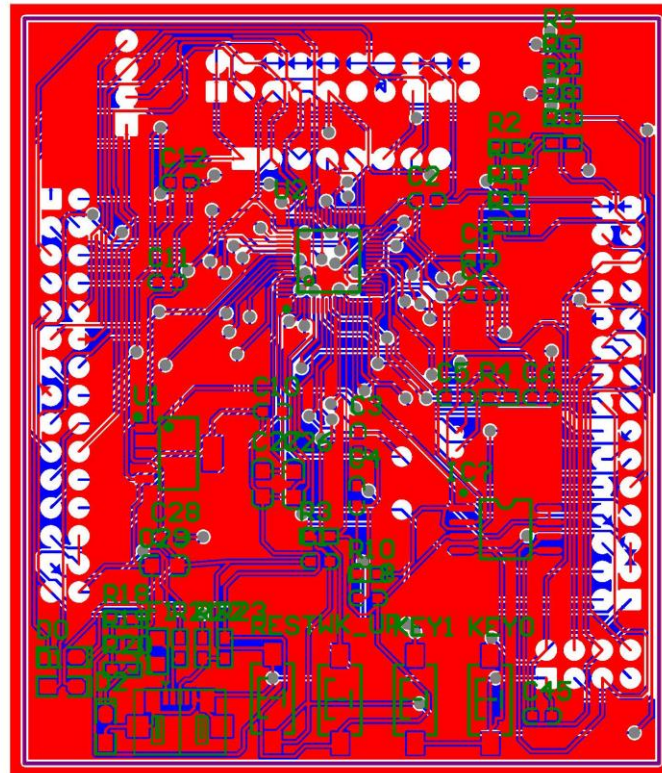
国内 Cortex-M3 市场，ST（意法半导体）公司的 STM32 无疑是最大赢家，ST 无论是在市场占有率，还是在技术支持方面，都是远超其他对手。在 Cortex-M3 芯片的选择上，STM32 无疑是大家的首选。所以自从 ST 推出 STM32 之后，一股强劲的 STM32 学习开发风潮扑面而来。

开发板选择的是 STM32F103RCT6 作为 MCU，它拥有的资源包括：48KBSRAM、256KBFLASH、2 个基本定时器、4 个通用定时器、2 个高级定时器、2 个 DMA 控制器（共 12 个通道）、3 个 SPI、2 个 IIC、5 个串口、1 个 USB、1 个 CAN、3 个 12 位 ADC、1 个 12 位 DAC、1 个 SDIO 接口及 51 个通用 IO 口。该芯片性价比极高。

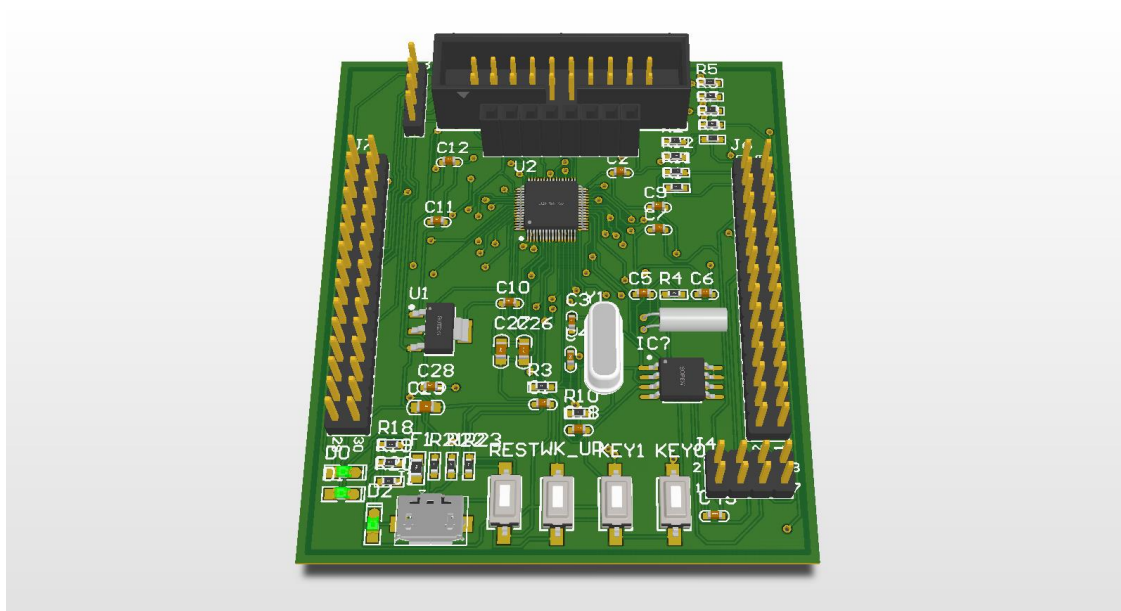
5.1.1 MCU 部分的原理图



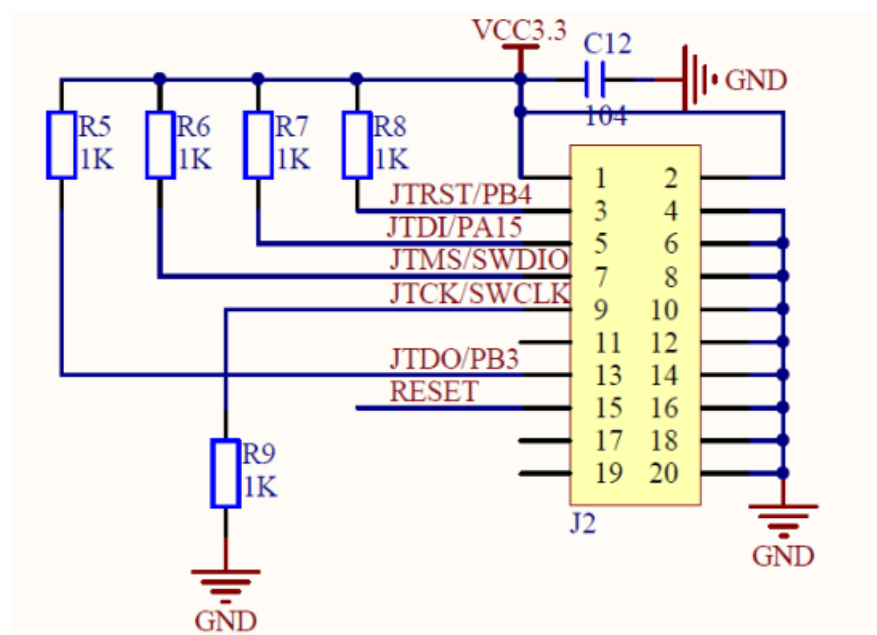
5.1.2 PCB



5.1.3 3D 效果图



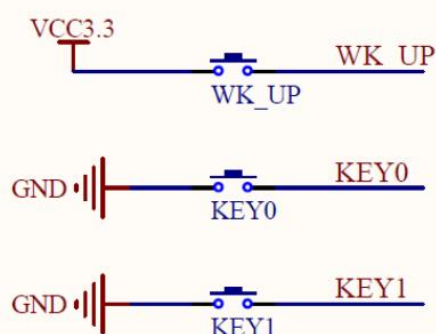
5.1.4 JTAG 部分电路



这里采用的是标准的 JTAG 接法，但是 STM32 还有 SWD 接口，SWD 只需要最少 2 根线（SWCLK 和 SWDIO）就可以下载并调试代码了，这同我们使用串口下载代码差不多，而且速度更快，能调试。所以建议在设计产品的时候，可以留出 SWD 来下载调试代码，而摒弃 JTAG。STM32 的 SWD 接口与 JTAG 是共用的，只要接上 JTAG，你就可以使用 SWD 模式了（其实 SWD 并不需要 JTAG 这么多线），JLINK V8/JLINKV7ULINK2 以及 STLINK 等都支持 SWD。这里，我们使用 SWD 模式。

5.1.5 按键

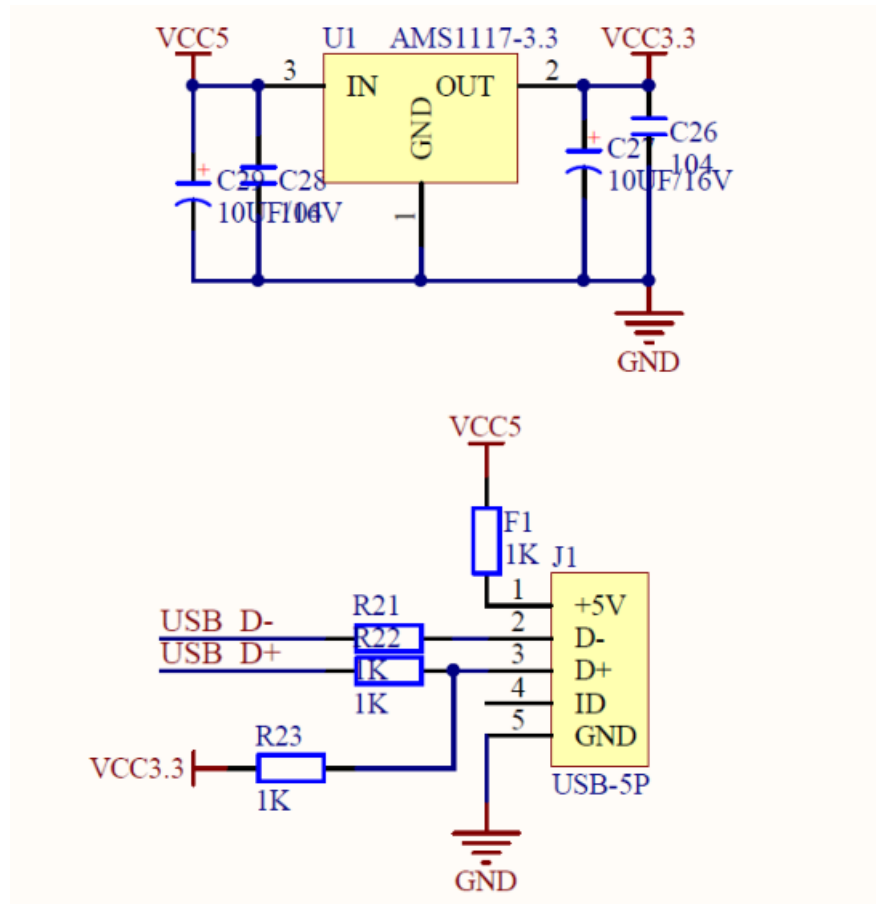
STM32 开发板总共有 3 个按键，其原理图如下：



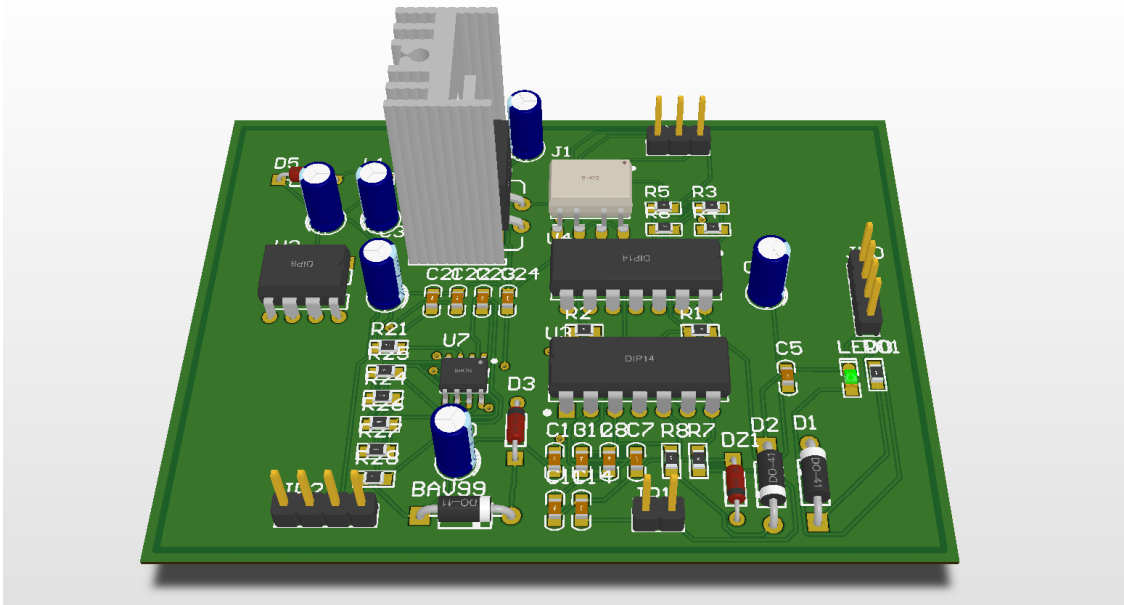
KEY0 和 KEY1 用作普通按键输入，分别连接在 PC1 和 PC13 上，WKUP 按键连接到 PA0（STM32 的 WKUP 引脚），它除了可以用作普通输入按键外，还可以用作 STM32 的唤醒输入。

5.1.6 USB、电源

开发板的供电部分还引出了 5V (VOUT2) 和 3.3V (VOUT1) 的排针，可以用来为外部设备提供电源或者从外部引入电源，这在很多时候是非常有用的，有时候你突然要一个 3.3V 的电源，但找半天就是没这样的电源，而我们的板子则可直接向外部提供 3.3V 电源，有了它，你就可以给外部设备提供 3.3V、5V 电源了。注意电流不能太大！开发板的 USB 接口 (USB) 通过独立的 MiniUSB 头引出，不和 USB 转串口 (USB232) 共用，这样不但可以同时使用，还可以给系统提供更大的电流。这几个部分的原理图如下：



5.2.3 3D 效果图



5.2.4 BOM 报表

Comment	Description	Designator	Footprint	LibRef	Quantity
1N4007	整流二极管	BAV99, D1, D2	DO-41	1N4007	3
100uF/35V	直插电解电容	C1	CAP 1.5*4*8	C-CAP	1
0.1uF	无极性贴片电容	C2, C3, C21, C22, C24	C 0805	C	5
470uF/25V	直插电解电容	C4	CAP 1.5*4*8	C-CAP	1
100pF	直插电解电容	C05	CAP 1.5*4*8	C-CAP	1
104	无极性贴片电容	C5	C 0805	C	1
100uF/25V	直插电解电容	C6, C10	CAP 1.5*4*8	C-CAP	2
0.47uF	无极性贴片电容	C7, C8, C11, C12, C13, C14	C 0805	C	6
2uF/10V	直插电解电容	C9	CAP 1.5*4*8	C-CAP	1
20nF	无极性贴片电容	C23	C 0805	C	1
1N4007	整流二极管	D3, D5	DO-35	1N4007	2
1N4106	12V/0.5W稳压管	DZ1	DO-35	1N4106	1
TLP521-2	2路开关光耦	J1	SOP8-W2.54 N	TLP521-2	1
HDR-1X4	4P接插件	JP0, JP2	HDR-1X4	Header 4	2
HDR-1X2	2P接插件	JP1	HDR-1X2	Header 2	1
HDR-1X3	3P接插件	JP3	HDR-1X3	Header 3	1
330uH	小功率贴片电感	L1	L 0805	L	1
Red	贴片LED	LED0	LED 0805G	LED-SMD	1
1K	贴片电阻	R1, R2, R5, R6, R7, R8, R28	R 0805	R	7
20K	贴片电阻	R01, R24, R25	R 0805	R	3
5.1K	贴片电阻	R3, R4	R 0805	R	2
2.8K	贴片电阻	R21	R 0805	R	1
2K	贴片电阻	R26, R27	R 0805	R	2
散热片	散热片	S1	散热片	S1	1
LM2575	DC降压芯片	U1	TO220-5B	LM2576	1
LT1054		U2	DIP8-300	LT1054	1
IR2110	High and Low Side Driver	U3	DIP14-300	IR2110	1
SN74LS07D	Hex Buffer / Driver with Open-Collector High-Voltage Outputs	U4	DIP14-300	SN74LS07D	1
LM358	双路运放	U7	SOP8_N	LM358	1

5.2.5 电压检测电路

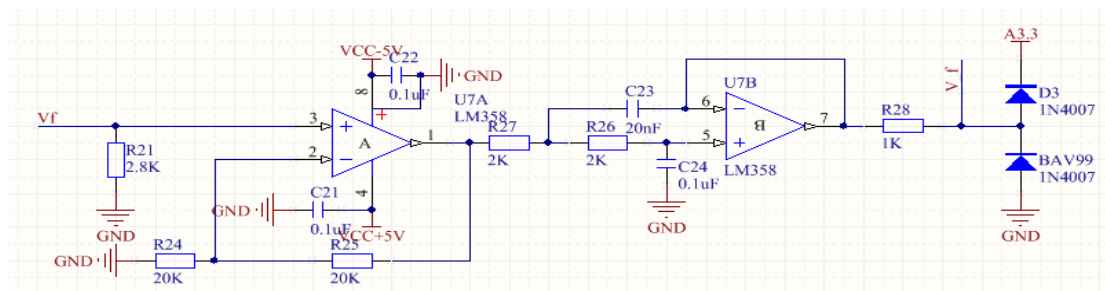


图 15 电压检测电路

将输出电压经过运放 358 放大 2 倍，送入滤波电路后，进行电压限幅后送入单片机 AD 通道。因为输出电压一直为正，运算放大器不会产生负压，所以 358 采用单电源供电，减少了 -5V 电源的产生电路。

5.2.6 驱动电路

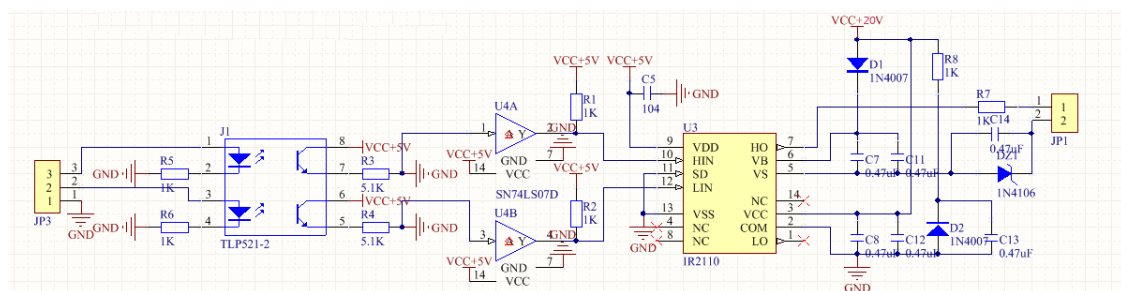


图 16 驱动电路

我们在单片机的 AD 输出端口与 IR2110 外围模拟电路之间添加了 TLP521 光耦隔离，增强电路的安全性，减小电压的干扰。

另外，IR2110 的不足是不能产生负偏压。在大功率 IGBT 驱动场合，各路驱动电源独立，集成驱动器一般都能产生 -5V 负压，用于增强 IGBT 关断的可靠性，防止由于密勒效应而造成误导通。IR2110 器件内部虽不能产生负压，但可通过外加无源器件产生负压。

在驱动电路中增加由电容和 5V 稳压管组成的负压电路。其工作原理为：电源电压 Vcc 为 20V。在上电期间，电源通过 R8 为 C13 充电，C13 保持 5V 电压。HIN 输入高电平时，HO 输出 20V，加在 VG1 的电压为 15V。当 HIN 为低电平时，HO 输出 0V，VG1 电压为 -5V。选择的 C13，C14 要大于 IGBT 栅极输入寄生电容 Ciss。自举电容充电电路中的二极管 D1 必须是快恢复二极管，以保证在有限时间内快速导通。

5.2.6 与模拟系统对比

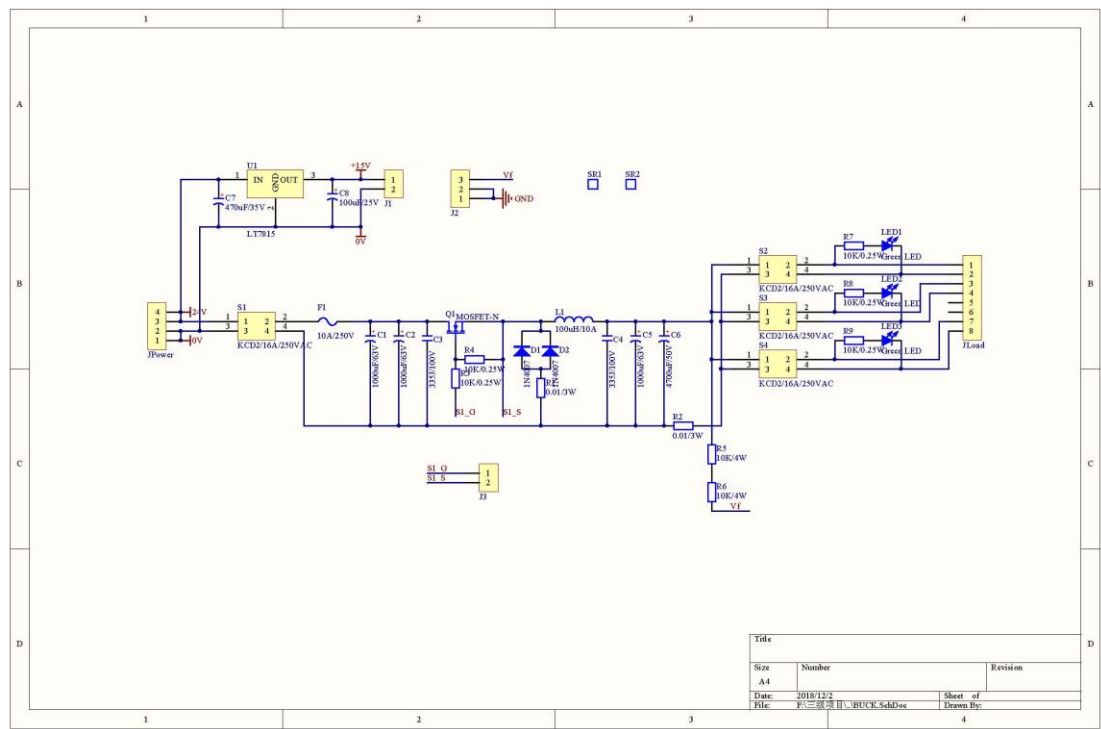
数字系统元器件清单如下：

器件	单价	数目	总价	供应商
贴片电容	0.01	18	0.18	深圳市深南紫光科技有限公司
电解电容	0.28	12	3.36	东莞市迪森电子有限公司
贴片电阻	0.01	28	0.28	振宝佳电子源头厂家
开关管	0.37	1	0.37	恒东信电子企业店
二极管	0.02	9	0.18	安久素源头厂家
电感	0.01	2	0.02	一芯钛佑品牌专营店
LM7815	0.3	1	0.3	晟睿芯电子科技
LM358	0.05	2	0.1	深圳市成宏微电子科技有限公司
IR2110	2	2	4	深圳市康胜世纪电子企业店
LM2575	0.57	1	0.57	深圳昇源科技有限公司
LT1054	4	1	4	深圳市亨源电子商行
74LS07	0.94	1	0.94	义胜电子有限公司
熔断器	2	1	2	正泰置优专卖店
LED	0.75	4	3	王子百货
单片机最小系统	18.34	1	18.34	risym 旗舰店

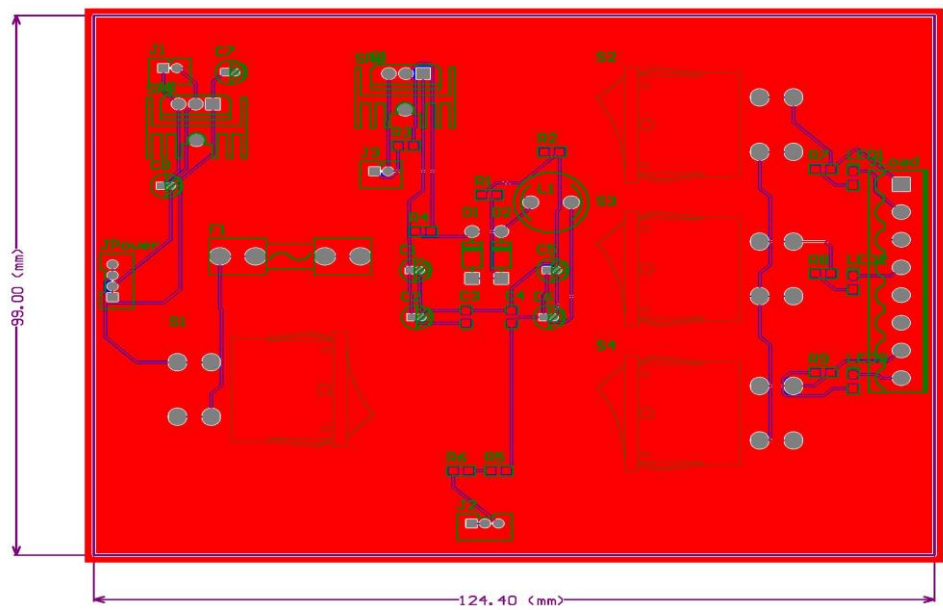
共计 37.64 元，因为数字系统采用单片机控制，电路需要添加隔离电路，相比较模拟系统的 3525 芯片及其外围电路，成本较高；而在闭环的 PI 调试环节，数字系统只需要更改程序中的 PI 参数，相比较模拟系统更改电位器的电阻值而言，更加的方便快捷。

5.3 主电路

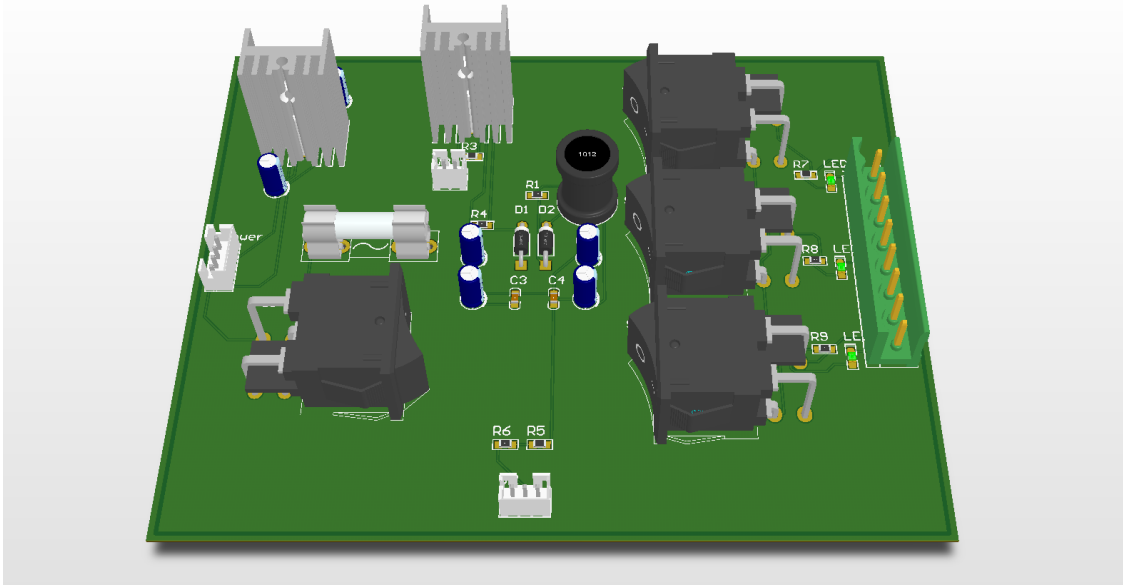
5.3.1 主电路原理图



5.3.2 PCB



5.3.3 3D 效果图



5.3.4 BOM 报表

Comment	Description	Designator	Footprint	LibRef	Quantity
1000uF/63V	直插电解电容	C1, C2, C5	CAP 1.5*4*8	C-CAP	3
335J/100V	无极性贴片电容	C3, C4	C 0805	C	2
4700uF/50V	直插电解电容	C6	CAP 1.5*4*8	C-CAP	1
470uF/35V	直插电解电容	C7	CAP 1.5*4*8	C-CAP	1
100uF/25V	直插电解电容	C8	CAP 1.5*4*8	C-CAP	1
1N4007	整流二极管	D1, D2	DO-41	1N4007	2
10A/250V	5*20保险丝	F1	FU 5x20	FU 5x20	1
HDR-1X2	2P接插件	J1, J3	PH2.0 D-L-2P	Header 2	2
HDR-1X3	3P接插件	J2	PH2.0 D-L-3P	Header 3	1
HDR-1X8	8P接插件	JLoad	KF2EDGK5.08-8P	Header 8	1
HDR-1X4	4P接插件	JPower	PH2.0 D-L-4P	Header 4	1
100uH/10A	小功率贴片电感	L1	L 0805	L	1
Green LED	贴片LED	LED1, LED2, LED3	LED 0805G	LED-SMD	3
MOSFET-N	N-Channel MOSFET	Q1	TO220A	MOSFET-N	1
0.01/3W	贴片电阻	R1, R2	R 0805	R	2
10K/0.25W	贴片电阻	R3, R4, R7, R8, R9	R 0805	R	5
10K/4W	贴片电阻	R5, R6	R 0805	R	2
KCD2/16A/250VAC		S1, S2, S3, S4	KCD1-104-W	KCD2/16A/250VAC	4
散热片	散热片	SR1, SR2	散热片	S1	2
LT7815		U1	TO220A	LT7815	1

5.4 UART 串口屏幕

串口屏定义就是，带串口控制的液晶屏就是串口屏了

详细定义：一套由单片机或 PLC 带控制器的显示方案，显示方案中的通讯部分由串口通讯，UART 串口或者 SPI 串口等；它由显示驱动板、外壳、LCD 液晶显示屏三部分构成。接收用户单片机串口发送过来的指令，完成在 LCD 上绘图的所有操作。

5.4.1 产品特点

1、使用字符串指令

字符串指令比十六进制指令更方便开发，提升工程师的工作效率，使用字符串指令的源代码更易读，使用字符串指令在以后查阅单片机代码时会更清楚代码含义。

2、数据结构精简

产品使用指令结构如下：字符串指令+结束符

3、产品使用“C 语言”指令

产品直接使用“C 语言”指令，同行其他产品使用“汇编”指令。C 语言指令比汇编语言更易读写，开发更方便。

4、控件属性赋值支持简易运算

比如 j0 控件的 val 属性赋值通常情况下是这么写 `j0.val=10` 使用运算方式可以这么写：`j0.val=j0.val+10` 也可以这么写：`j0.val=j0.val/2+10` 还可以这么写：`j0.val=j1.val-j0.val"2+dim`（dim 是系统变量）

5、屏幕固件自动升级

上位软件编译出来的资源文件就包含了最新的显示模组固件。任何功能性的升级或者 bug 修复，只需要使用最新的上位软件，编译出资源文件后下载到显示模组，显示模组固件立刻自动升级。上位软件有自动更新功能，只要电脑网络畅通，电脑防火墙没有隔离的软件，每次启动软件的时候软件检测到新版本会提示是否立即升级。

5.4.2 功能简介

I/O 接口：通过软件配置可以吧 I/O 配置成输入状态或输出状态。另外我们的产品/O 可以绑定控件使用。

支持通讯接口：TTL、RS232、RS485、CAN 女触摸类型：电容触摸、电阻触摸、无触摸

通电进入工作状态无需任何初始化设置

提供用户数据存储空间（EEPROM）产品结构可根据客户需求进行定制

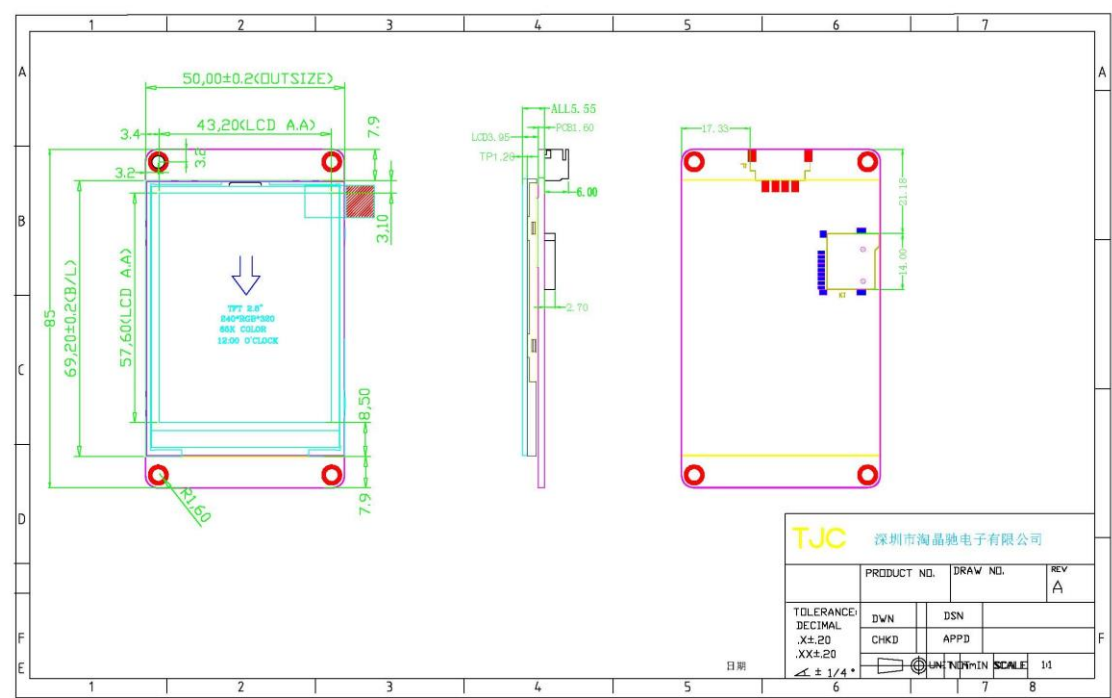
可通过串口指令调整背光

可通过串口指令画图

支持 RTC 功能

横竖可控

5.4.3 原理图



6 软件部分

6.1 FreeRTOS 简介

我们看一下 FreeRTOS 的名字，可以分为两部分：Free 和 RTOS，Free 就是免费的、自由的、不受约束的意思，RTOS 全称是 Real Time Operating System，中文名就是实时操作系统。可以看出 FreeRTOS 就是一个免费的 RTOS 类系统。这里要注意，RTOS 不是指某一个确定的系统，而是指一类系统。比如 UCOS，FreeRTOS，RTX，RT-Thread 等这些都是 RTOS 类操作系统。

操作系统允许多个任务同时运行，这个叫做多任务，实际上，一个处理器核心在某一时刻只能运行一个任务。操作系统中任务调度器的责任就是决定在某一时刻究竟运行哪个任务，任务调度在各个任务之间的切换非常快！这就给人们造成了同一时刻有多个任务同时运行的错觉。

操作系统的分类方式可以由任务调度器的工作方式决定，比如有的操作系统给每个任务分配同样的运行时间，时间到了就轮到下一个任务，Unix 操作系统就是这样的。RTOS 的任务调度器被设计为可预测的，而这正是嵌入式实时操作系统所需要的，实时环境中要求操作系统必须对某一个事件做出实时的响应，因此系统任务调度器的行为必须是可预测的。像

FreeRTOS 这种传统的 RTOS 类操作系统是由用户给每个任务分配一个任务优先级，任务调度器就可以根据此优先级来决定下一刻应该运行哪个任务。

FreeRTOS 是 RTOS 系统的一种，FreeRTOS 十分的小巧，可以在资源有限的微控制器中运行，当然了，FreeRTOS 不仅局限于在微控制器中使用。但从文件数量上来看 FreeRTOS 要比 UCOSII 和 UCOSII 小的多。

6.2 FreeRTOS 特点

FreeRTOS 是一个可裁剪的小型 RTOS 系统，其特点包括：

- FreeRTOS 的内核支持抢占式，合作式和时间片调度。
- SafeRTOS 衍生自 FreeRTOS，SafeRTOS 在代码完整性上相比 FreeRTOS 更胜一筹。
- 提供了一个用于低功耗的 Tickless 模式。
- 系统的组件在创建时可以选择动态或者静态的 RAM，比如任务、消息队列、信号量、软件定时器等等。
- 已经在超过 30 种架构的芯片上进行了移植。
- FreeRTOS-MPU 支持 Corex-M 系列中的 MPU 单元，如 STM32F103。
- FreeRTOS 系统简单、小巧、易用，通常情况下内核占用 4k-9k 字节的空间。
- 高可移植性，代码主要 C 语言编写。
- 支持实时任务和协程（co-routines 也有称为合作式、协同程序，本教程均成为协程）。
- 任务与任务、任务与中断之间可以使用任务通知、消息队列、二值信号量、数值型信号量、递归互斥信号量和互斥信号量进行通信和同步。
- 创新的事件组（或者事件标志）。
- 具有优先级继承特性的互斥信号量。
- 高效的软件定时器。
- 强大的跟踪执行功能。
- 堆栈溢出检测功能。
- 任务数量不限。
- 任务优先级不限。

6.3 程序流程图

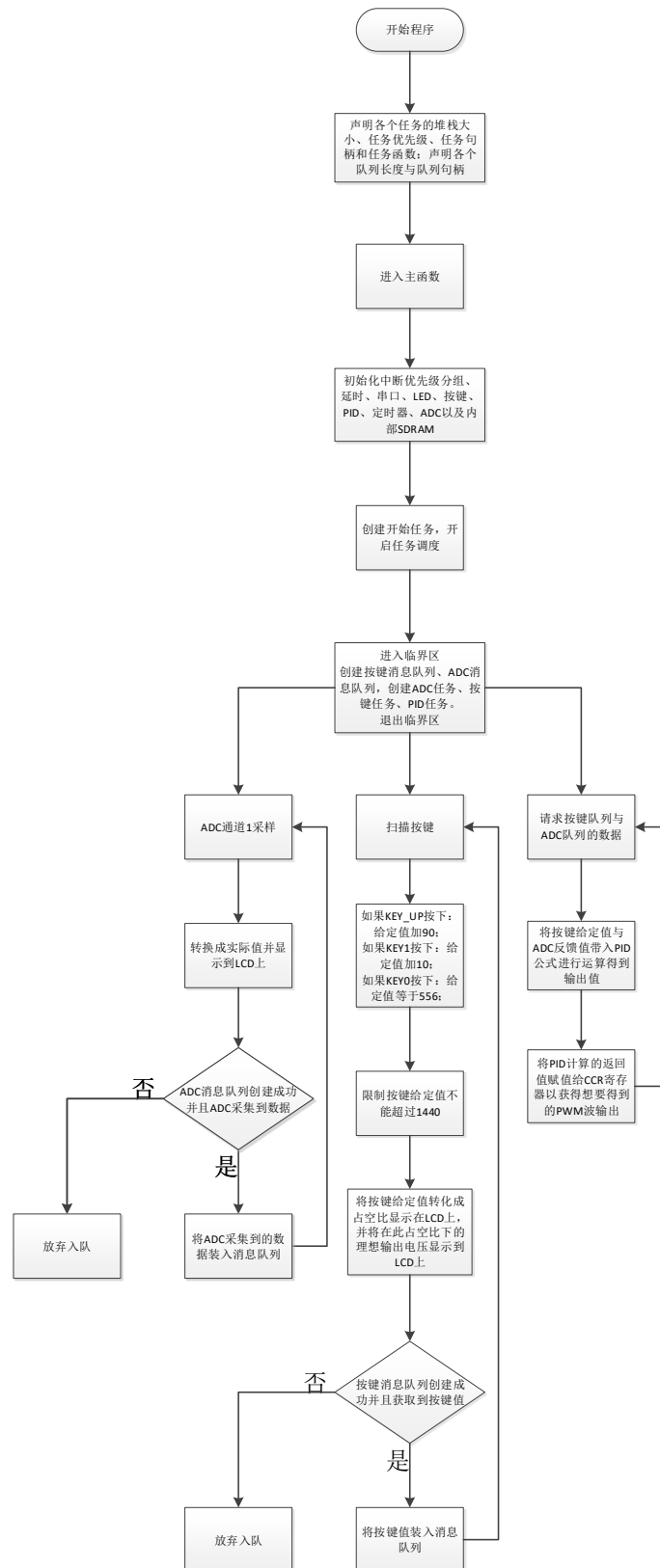


图 17 程序流程图

6.4 代码解析

6.4.1 mian.c（主函数）

```
// 引用头文件
#include "sys.h"
#include "delay.h"
#include "usart.h"
#include "led.h"
#include "key.h"
#include "HMI.h"
#include "timer.h"
#include "adc.h"
#include "pid.h"
#include <math.h>
#include "malloc.h"
#include "string.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"

// 开始任务参数设置
// 任务优先级
#define START_TASK_PRIO    1
// 任务堆栈大小
#define START_STK_SIZE     256
// 任务句柄
TaskHandle_t StartTask_Handler;
// 任务函数
void start_task(void *pvParameters);

// ADC 任务参数设置
// 任务优先级
#define ADC_TASK_PRIO      3
// 任务堆栈大小
#define ADC_STK_SIZE       256
// 任务句柄
TaskHandle_t ADCTask_Handler;
// 任务函数
void adc_task(void *pvParameters);

// 按键任务参数设置
// 任务优先级
```

```

#define KEYPROCESS_TASK_PRI0 2
//任务堆栈大小
#define KEYPROCESS_STK_SIZE 128
//任务句柄
TaskHandle_t Keyprocess_Handler;
//任务函数
void Keyprocess_task(void *pvParameters);

//PID 调节任务参数设置
//任务优先级
#define PID_TASK_PRI0 4
//任务堆栈大小
#define PID_STK_SIZE 256
//任务句柄
TaskHandle_t PID_Handler;
//任务函数
void pid_task(void *pvParameters);

//消息队列参数设置, 按键与ADC 值的队列都设置为20 长度
#define KEYMSG_Q_NUM 20 //按键消息队列的数量
#define MESSAGE_Q_NUM 20 //发送数据的消息队列的数量
QueueHandle_t Key_Queue; //按键值消息队列句柄
QueueHandle_t Message_Queue; //信息队列句柄

int main(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4); //设置系统中断优先
    级分组4
    delay_init(); //延时函数初始化
    uart_init(9600); //初始化串口
    LED_Init(); //LED 初始化
    PID_init(); //PID 初始化
    KEY_Init(); //按键初始化
    TIM3_PWM_Init(1439,0); //TIM3 初始化, 不分频。PWM 频率=7
    2000000/1440=50Khz
    Adc_Init(); //ADC 初始化
    my_mem_init(SRAMIN); //初始化内部内存池

    //创建开始任务
    xTaskCreate((TaskFunction_t )start_task, //任务函数
                (const char* )"start_task", //任务名称
                (uint16_t )START_STK_SIZE, //任务堆栈大小
                (void* )NULL, //传递给任务函数的
    参数

```

```

        (UBaseType_t )START_TASK_PRI0,      // 任务优先级
        (TaskHandle_t* )&StartTask_Handler); // 任务句柄

vTaskStartScheduler();      // 开启任务调度
}

// 开始任务任务函数
void start_task(void *pvParameters)
{
    taskENTER_CRITICAL();      // 进入临界区
    // 创建消息队列
    Key_Queue=xQueueCreate(KEYMSG_Q_NUM,sizeof(u16));      // 创建消息
    // 创建消息队列
    Message_Queue=xQueueCreate(MESSAGE_Q_NUM,sizeof(u16)); // 创建消息
    // 创建消息队列
    // 创建消息队列, 队列项长度是串口接收缓冲区长度

    // 创建 TASK1 任务
    xTaskCreate((TaskFunction_t )adc_task,
        (const char* )"adc_task",
        (uint16_t )ADC_STK_SIZE,
        (void* )NULL,
        (UBaseType_t )ADC_TASK_PRI0,
        (TaskHandle_t* )&ADCTask_Handler);

    // 创建 TASK2 任务
    xTaskCreate((TaskFunction_t )Keyprocess_task,
        (const char* )"keyprocess_task",
        (uint16_t )KEYPROCESS_STK_SIZE,
        (void* )NULL,
        (UBaseType_t )KEYPROCESS_TASK_PRI0,
        (TaskHandle_t* )&Keyprocess_Handler);

    // 创建 TASK3 任务
    xTaskCreate((TaskFunction_t )pid_task,
        (const char* )"pid_task",
        (uint16_t )PID_STK_SIZE,
        (void* )NULL,
        (UBaseType_t )PID_TASK_PRI0,
        (TaskHandle_t* )&PID_Handler);
    vTaskDelete(StartTask_Handler); // 删除开始任务
    taskEXIT_CRITICAL();      // 退出临界区
}

// ADC 任务函数
void adc_task(void *pvParameters)

```

```

{
    u16 adcx,flo;

    BaseType_t err;                                     //定义错误返回接收句柄

    while(1)
    {
        adcx=Get_Adc_Average(ADC_Channel1_1,10);      //ADC 通道1 采样10次并取平均值
        flo = (int)adcx*134.3/4096;                   //串口屏要显示的实际电压值
        CurveCommand(1,0,flo);                         //将电压值通过串口发送到串口屏幕显示
        if ((Message_Queue!=NULL) && (adcx))           //如果队列创建成功并且ADC 采集到数据
        {
            err=xQueueSend(Message_Queue,&adcx,10);    //将数据装入队列,如果队列已满最多等10个时钟节拍,否则放弃本次入队操作
            if(err==errQUEUE_FULL)                     //如果入队失败
            {
                //printf("队列Message_Queue 已满, 数据发送失败!\r\n");
            }
        }
        vTaskDelay(10);                                //延时10ms, 也就是10个时钟节拍
    }
}

//Keyprocess_task 函数
void Keyprocess_task(void *pvParameters)
{
    u8 key;
    u16 pwm=0;
    BaseType_t err;
    while(1)
    {
        key=KEY_Scan(0);                                //扫描按键
        switch(key)
        {
            case WKUP_PRES:                             //KEY_UP 控制pwm 值加90
                pwm += 90;
                break;
            case KEY1_PRES:                             //KEY1 控制pwm 值加180
                pwm += 10;

```

```

        break;
    case KEY0_PRES:    //KEY0 控制pwm 值减 90
        pwm = 556;
        break;
    }
    if (pwm>1440)      //限制pwm 的值不能超过 1440
    {
        pwm = 1440;
    }
    NumberCommand(0,(int)pwm*100/1440);    //串口屏幕显示当前输出
电压的值
    NumberCommand(1,(int)pwm*24/1440);    //串口屏幕显示当前输出
占空比的值
    if((Key_Queue!=NULL)&&(pwm!=0))        //消息队列Key_Queue 创
建成功,并且按键被按下
    {
        err=xQueueSend(Key_Queue,&pwm,10);
        if(err==errQUEUE_FULL)            //如果入队失败
        {
            //printf("队列Key_Queue 已满, 数据发送失败!\r\n");
        }
    }
    vTaskDelay(10);                        //延时10ms, 也就是10
个时钟节拍
}

//PID_task 函数
void pid_task(void *pvParameters)
{
    u16 adcx,pwm;
    extern PID V_PID;
    while(1)
    {
        if(Message_Queue!=NULL && Key_Queue!=NULL)    //如果按键队列和AD
C 消息队列都创建成功
        {
            if(xQueueReceive(Message_Queue,&adcx,portMAX_DELAY) && xQ
ueueReceive(Key_Queue,&pwm,portMAX_DELAY))//请求消息Message_Queue 和K
ey_Queue
            {
                //闭环
                V_PID.setpulse = pwm*4096/1440;    //按键给定值
            }
        }
    }
}

```

```

        V_PID.backpulse = adcx/2;           //电压反馈值
        PWM_VAL1=V_PIDCalc(&V_PID);
        PWM_VAL2=V_PIDCalc(&V_PID);
        //开环
        //PWM_VAL1=pwm;
        //PWM_VAL2=pwm;
    }
}
vTaskDelay(10);    //延时10ms, 也就是10 个时钟节拍
}
}

```

首先采用上述介绍的 FreeRTOS 操作系统，设置了开始任务，按键任务，PID 调节任务函数，并进行初始化参数，然后进行任务函数之间的消息队列设置。主函数进行函数初始化，并对三个任务函数进行编写。

6.4.2 PID.c（PID 调节代码）

```

#include "pid.h"
PID V_PID;
void PID_init(void)
{
    V_PID.setpulse = 0 ;           //电压设定值
    V_PID.backpulse = 0 ;          //电压反馈值
    V_PID.last_error = 0 ;
    V_PID.pre_error = 0 ;
    V_PID.P = Pv;
    V_PID.I = Iv;
    V_PID.D = Dv;
    V_PID.motorout = 0 ;           //控制输出值
}

unsigned int V_PIDCalc( PID *pp ) //按照课本给定的PID 公式进行操作
{
    int error;

    error = pp->setpulse - pp->backpulse;

    pp->motorout +=( int) (pp->P*(error-pp->last_error) + pp->I*error
+ pp->D*(error-2*pp->last_error+pp->pre_error));

    pp->pre_error = pp->last_error;
    pp->last_error = error;

    if( pp->motorout >= D_MAX)

```

```

        pp->motorout = D_MAX;
    else if( pp->motorout <= D_MIN) //电压PID, 防止调节最低溢出
        pp->motorout = D_MIN;

    return (pp->motorout);          // 返回预调节占空比
}

```

6.4.3 pid.h (PID 调节代码)

```

#ifndef __PID_H
#define __PID_H

#include "delay.h"
#include "stm32f10x.h"

//////////PID 调节////////////////////////////////////////
//////////
#define Pv 0.1    //0.6//0.1          振荡 0.05      振荡至稳定 0.1

#define Iv 0.5    //0.08              0.04          0.03

#define Dv 0

#define D_MAX 620 //占空比输出最大值
#define D_MIN 0   //占空比输出最小值
//定义PID
typedef struct PID
{
    int setpulse;          //设定值
    int backpulse;         //反馈值
    int last_error;
    int pre_error;
    float P;
    float I;
    float D;
    int motorout;          //控制输出值
}PID;

void PID_init(void);
unsigned int V_PIDCalc( PID *pp );

#endif

```

此次课设我们做的是电压单闭环，在计控 CDIO 程序的基础上，删去了一个增量式 PID 函数。

6.4.4 timer.c（定时器与 PWM 代码）

```
#include "timer.h"
#include "led.h"
#include "usart.h"

//TIM3 PWM 部分初始化
//PWM 输出初始化
//arr: 自动重装值
//psc: 时钟预分频数
void TIM3_PWM_Init(u16 arr,u16 psc)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_OCInitTypeDef TIM_OCInitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);    //使能定时
器3 时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE); //使能GP
IOA 模块时钟

    //设置该引脚为复用输出功能,输出 TIM3 CH1 的PWM 脉冲波形    GPIOA.6
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7; //TIM_CH2
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //复用推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);//初始化 GPIO

    //初始化 TIM3
    TIM_TimeBaseStructure.TIM_Period = arr; //设置在下一个更新事件装入
活动的自动重载寄存器周期的值
    TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置用来作为TIMx 时钟
频率除数的预分频值
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TCTS =
Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TI
M 向上计数模式
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //根据 TIM_TimeBas
eInitStruct 中指定的参数初始化 TIMx 的时间基数单位
```

```

//初始化 TIM3 Channel2 PWM 模式
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //选择定时器模式:TIM 脉冲宽度调制模式 2
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable; //比较输出使能
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High; //输出极性:TIM 输出比较极性高
TIM_OC1Init(TIM3, &TIM_OCInitStructure); //根据T 指定的参数初始化外设 TIM3 OC1

TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable); //使能 TIM3 在 CR1 上的预装载寄存器
//上面两句中的 OC1 确定了是 channel 几, 要是 OC2 则是 channel 2
TIM3->CCR1 = 1440; //初始化占空比

TIM_OC2Init(TIM3, &TIM_OCInitStructure); //根据T 指定的参数初始化外设 TIM3 OC1
TIM_OC2PreloadConfig(TIM3, TIM_OCPreload_Enable); //使能 TIM3 在 CR1 上的预装载寄存器
TIM3->CCR2 = 1440;

TIM_Cmd(TIM3, ENABLE); //使能 TIM3
}

```

通过主函数中的 PID 任务的按键队列和 ADC 消息队列来实现中断函数功能, 所以我们删除了 timer.c 中的中断函数。

6.4.5 adc.c (ADC 采样代码)

```

#include "adc.h"
#include "delay.h"

//初始化 ADC
//这里我们仅以规则通道为例
//我们默认将开启通道 0~3

void Adc_Init(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_ADC1, ENABLE ); //使能 ADC1 通道时钟
}

```

```
RCC_ADCCLKConfig(RCC_PCLK2_Div6); //设置ADC 分频因子6 72M/6=12,ADC 最大时间不能超过 14M
```

```
//PA1 作为模拟通道输入引脚
```

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1|GPIO_Pin_2;
```

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; //模拟输入引脚
```

```
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
ADC_DeInit(ADC1); //复位ADC1,将外设 ADC1 的全部寄存器重设为缺省值
```

```
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //ADC 工作模式:ADC1 和 ADC2 工作在独立模式
```

```
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //模数转换工作在单通道模式
```

```
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //模数转换工作在单次转换模式
```

```
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //转换由软件而不是外部触发启动
```

```
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right; //ADC 数据右对齐
```

```
ADC_InitStructure.ADC_NbrOfChannel = 1; //顺序进行规则转换的ADC 通道的数目
```

```
ADC_Init(ADC1, &ADC_InitStructure); //根据ADC_InitStruct 中指定的参数初始化外设ADCx 的寄存器
```

```
ADC_Cmd(ADC1, ENABLE); //使能指定的ADC1
```

```
ADC_ResetCalibration(ADC1); //使能复位校准
```

```
while(ADC_GetResetCalibrationStatus(ADC1)); //等待复位校准结束
```

```
ADC_StartCalibration(ADC1); //开启AD 校准
```

```
while(ADC_GetCalibrationStatus(ADC1)); //等待校准结束
```

```
}
```

```
//获得ADC 值
```

```
//ch: 通道值 0~3
```

```
u16 Get_Adc(u8 ch)
```

```
{
```

```
//设置指定ADC 的规则组通道, 一个序列, 采样时间
```

```
ADC-RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5
```

```
); //ADC1,ADC 通道, 采样时间为 239.5 周期
```

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);    //使能指定的ADC1 的软件
转换启动功能
```

```
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
```

```
return ADC_GetConversionValue(ADC1);    //返回最近一次ADC1 规则组的
转换结果
}
```

```
u16 Get_Adc_Average(u8 ch,u8 times)      //采样times 次并取平均值
{
    u32 temp_val=0;
    u8 t;
    for(t=0;t<times;t++)
    {
        temp_val+=Get_Adc(ch);
        delay_ms(5);
    }
    return temp_val/times;
}
```

adc 采样部分与计控的 CDIO 程序一样，最后在程序中将 ADC 通道 1 采样 10 次并取平均值，将数据送入多任务之间的消息队列中。

6.4.6 HMI.c（串口屏幕显示代码）

```
#include "HMI.h"
#include "usart.h"
```

```
void CurveCommand(int ID, int channel, int value)    //串口屏幕绘图
曲线指令
```

```
{
    USART_SendData(USART1, 0XFF);//向串口1 发送数据
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);//等待发送
    结束
    USART_SendData(USART1, 0XFF);//向串口1 发送数据
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);//等待发送
    结束
    USART_SendData(USART1, 0XFF);//向串口1 发送数据
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);//等待发送
    结束
    printf("add %d,%d,%d",ID,channel,value);
    while(USART_GetFlagStatus(USART1,USART_FLAG_TC)!=SET);//等待发送
}
```

```

    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
}

void NumberCommand(int ID, int num) //串口屏幕绘制数字指令
{
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    printf("n%d.val=%d", ID, num);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
    USART_SendData(USART1, 0XFF); //向串口1 发送数据
    while(USART_GetFlagStatus(USART1, USART_FLAG_TC) != SET); //等待发送
    结束
}

```

6.4.7 key.c（按键代码）

```

#include "stm32f10x.h"
#include "key.h"

```

```

#include "sys.h"
#include "delay.h"

//按键初始化函数
void KEY_Init(void) //IO 初始化
{
    GPIO_InitTypeDef GPIO_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOC,
    ENABLE); //使能 PORTA, PORTE 时钟

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1|GPIO_Pin_13; //KEY0-KEY1
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU; //设置成上拉输入
    GPIO_Init(GPIOC, &GPIO_InitStructure); //初始化 GPIOE4, 3

    //初始化 WK_UP-->GPIOA.0 下拉输入
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD; //PA0 设置成输入，默
    认下拉
    GPIO_Init(GPIOA, &GPIO_InitStructure); //初始化 GPIOA.0
}

//按键处理函数
//返回按键值
//mode:0, 不支持连续按;1, 支持连续按;
//0, 没有任何按键按下
//1, KEY0 按下
//2, KEY1 按下
//3, KEY3 按下 WK_UP
//注意此函数有响应优先级, KEY0>KEY1>KEY_UP!!
u8 KEY_Scan(u8 mode)
{
    static u8 key_up=1; //按键按松开标志
    if(mode)key_up=1; //支持连按
    if(key_up&&(KEY0==0 || KEY1==0 || WK_UP==1))
    {
        delay_ms(10); //去抖动
        key_up=0;
        if(KEY0==0)return KEY0_PRES;
        else if(KEY1==0)return KEY1_PRES;
        else if(WK_UP==1)return WKUP_PRES;
    }else if(KEY0==1&&KEY1==1&&WK_UP==0)key_up=1;
    return 0; //无按键按下
}

```

按键部分与计控的 CDIO 程序一样，最后在程序中用扫描函数将按键返回值送入按键任务中。

7 实验部分

7.1 开环实验

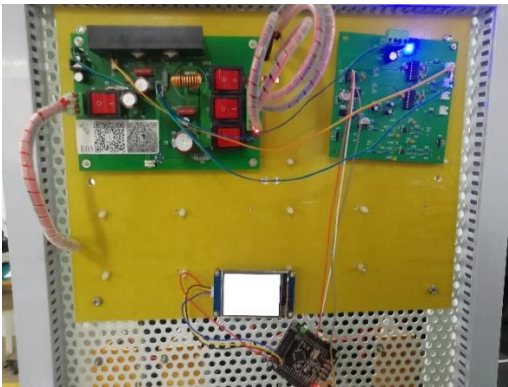


图 18 开环实验电路

在计算的稳态工作点上展开实验，更改单片机的程序让占空比为 38.6%，从单片机的 PA6,PA7 进行 PWM 输出，观察 IR2110 输出的驱动脉冲。



图 19 IR2110 驱动脉冲

在显示屏和示波器上观察输出电压为 10V。



图 20 输出电压

切换到交流档，观察输出电压纹波 $\gamma = \frac{0.06}{10} = 0.6\%$

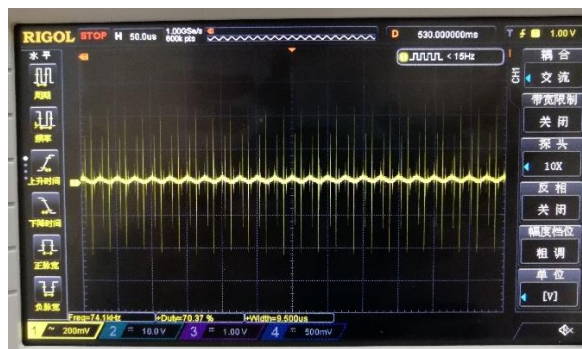


图 21 电压纹波

观察电感电压，在开关管导通时电感电压为正，续流管导通时电感电压为负，电流断续状态下电感电压为 0。

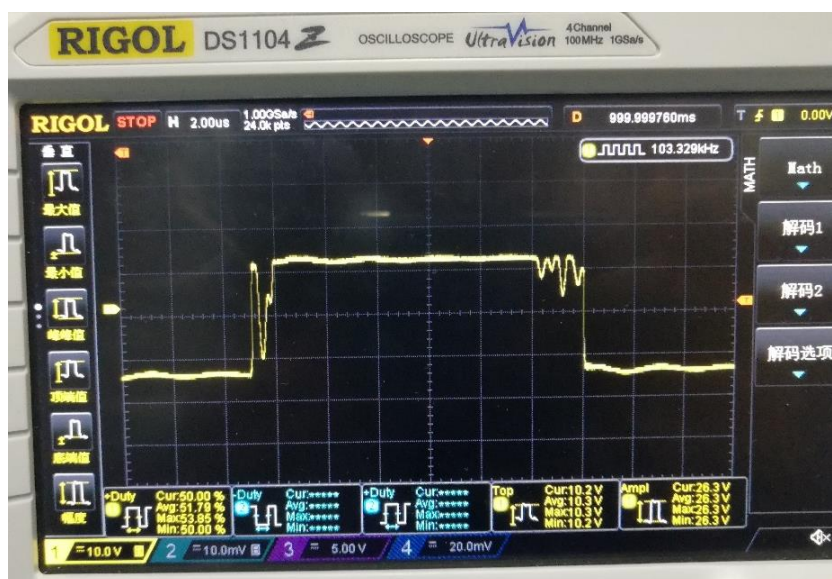


图 22 电感电压

开环状态下，突增负载，输出电压发生变化，也可以说明此时电感电流断续。



图 23 输出电压

7.2 闭环实验



图 24 闭环实验电路

输出电压经过 $20\text{K}\Omega$ 和 $2.8\text{K}\Omega$ 电阻分压后，放大两倍并进行滤波，送入单片机的输入端口 PA1。

突减负载时，经过 100ms 重新回到原来的输出电压。

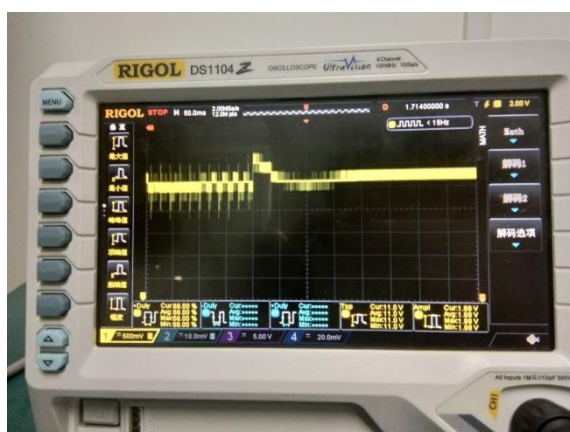


图 25 加扰动后的输出电压

输出电压启动波形，无超调，调整时间为 20ms。

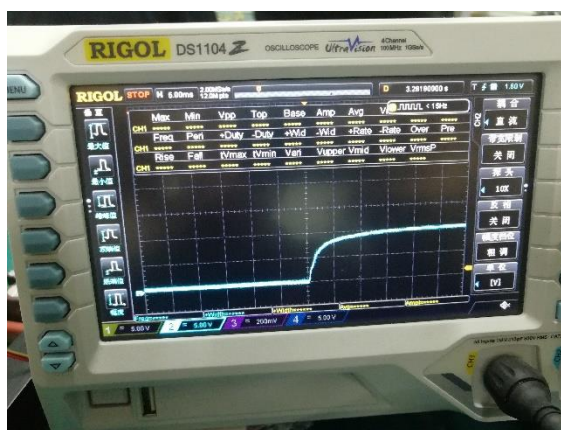


图 26 输出电压启动波形

8 结论

随着现代电力电子技术和计算机等技术更加高速的发展，Buck 变换器不仅在家用电器、空间技术、计算机、通讯、舰船设备中运用越来越多，还将在其他相关领域占据了越来越多的市场份额，应用范围会变的更加广泛。

本文设计了断续状态下 Buck 变换器的控制电路。该系统可以实现从高压到低压的电压变换，并可以通过闭环使输出电压恒定。同时，我们初步学习了基于计算机控制的电压闭环系统，并展开实验，获益匪浅。

同时，由于时间仓促，我们也有许多不足之处，比如并没有使实验过程中闭环电压的波形与 MATLAB 仿真的波形一致，希望在以后的学习过程中能继续深入学习。

参考文献

- 1 王兆安 电力电子技术（第 5 版） 机械工业出版社，2010：
- 2 郑颖楠 电源技术
- 3 张卫平 开关变换器的建模与控制 中国电力出版社，2006：190-205

计算机控制综合课程设计组内自评表

题目：电流断续状态下 Buck 变换器设计		组别：8B	
班级：15 级应电 1-4 班			
工作质量排名	姓名	组内分配工作内容	本人签名
A	王云兆	程序编写，实验调试	
A	杨德浩	器件选型，撰写报告	
B	庄严	查资料	
B	李瑞彪	查资料	
B	方盼想	查资料	
B	刘港	查资料	
B	王雷	查资料	
C	段伟康	查资料	
D	项兴	查资料	