数字电源开发平台

基于 STM32F334 的同步降压转换器 BU4805S

使用手册 V1.3

桂林安合科技有限公司

2020 年 02 月 29 日

目录

简介

开关电源采用数字控制是目前工业产品的发展趋势。数字控制可以提升开关电源性能，提高产品的附加值，通过软件编程就可以灵活地实现各种各样的功能。数字电源由 DSP/MCU 控制输出闭环反馈、故障保护、软启动和通信等，具备高可靠性和灵活性能。

本手册介绍如何使用 STM32F334 实现同步降压（BUCK）拓扑电路的数字化控制；分别从原理到硬件设计、程序实现给出说明，希望对数字电源开发从业者有一个良好的参考价值。

关于安合科技

安合科技致力于电力电子开发平台的软硬件研发、销售和技术推广，涉及有数字电源、双向 DC/DC 变换器、光伏逆变器和电机驱动及控制等产品，同时也提供软硬件定制、技术支持等服务。

联系电话：17526712502

电子邮箱：690705980@qq.com

淘宝店铺：shop190345163.taobao.com

WARNING

此开发板仅限在实验室环境中工作，这不是常规消费品用途的最终产品。

此开发板只能由知悉高压电气、电子系统的工程师或技术人员使用。

此开发板使用时，如果处理或应用不当，可能引发电击、火灾或人身伤害。使用时务必小心，并采用适当的防护措施以避免人身伤害和财产损失。

数字电源开发板介绍

本数字电源开发板是安合科技基于 STM32F334C8 的同步 BUCK 降压转换器。STM32F334C8 是基于 Coetex-M4 内核，运行主频 72MHZ，并且支持 FPU 和 DSP 指令集。其强大的运算性能满足 PID、ADRC、智能控制及数字滤波处理等算法，满足数字电源闭环控制。该 MCU 还配置了高分辨定时器 HRTIM，可输出分辨率达 217ps 的 PWM，满足电压电流高精度控制。本套件采用全数字闭环控制，灵活实现无缝降压控制、电流双向控制，适用于直流稳压电源、LED 恒流电源、太阳能 MPPT 控制、电池充电器、直流 UPS 等应用。实物如图 1 所示。

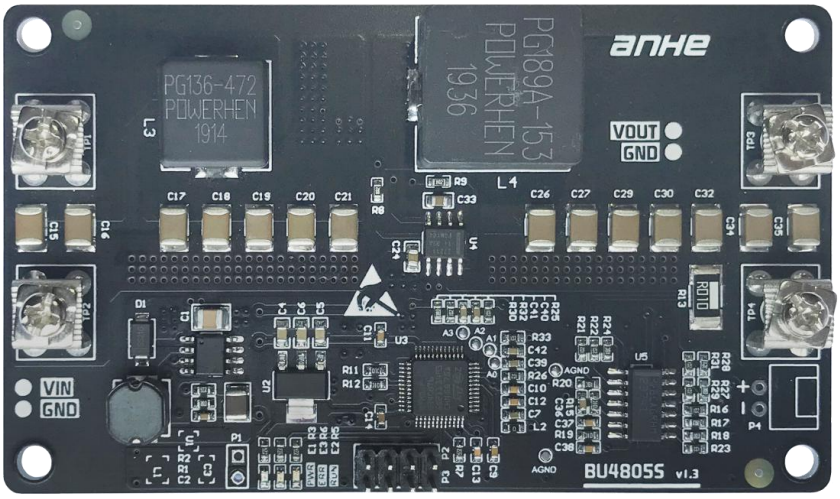

图 1 同步 BUCK 降压转换器实物图

参数

- 输入电压：10 至 48VDC
- 输出电压：0 至 48VDC
- 输出电流：0 到 8A
- 额定功率：240W
- 开关频率：200KHZ
- 控制模式：电压电流双闭环

特点

- 同步 BUCK 降压转换器
- 全数字闭环控制
- Debug 接口开放
- 恒压恒流无缝切换
- 支持 Modbus-rtu 通信
- 全 C 语言代码开源

1. 同步降压转换器原理

1.1. 工作原理

降压转换器正如其名，仅能提供比输入电压低的平均输出电压。由于其很高的效率，因而比线性稳压器具有优势。图 1-1 给出了同步降压转换器的基本拓扑结构。开关管 Q1 和 Q2 由一组对称互补频率固定且占空比可变的方波信号控制。由于开关管使用 MOSFET，因此可用于正向降压 BUCK/反向升压 BOOST 应用。

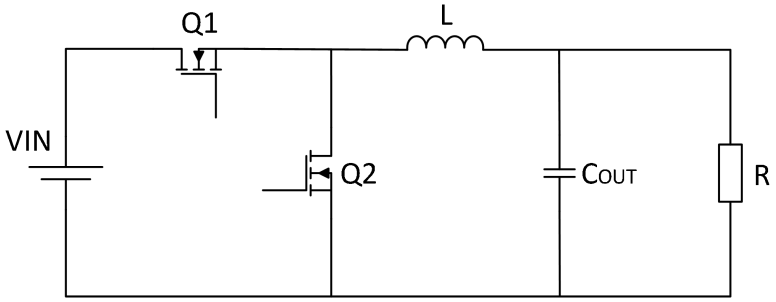

图 1-1 同步降压转换器电路拓扑图

在分析输入、输出电压与占空比关系时，可以把 BUCK 电路的电感 L 和输出电容 C_{OUT} 理解为一个 LC 低通滤波电路。当开关管 Q1 导通时（Q2 关断），V_{IN} 通过 L 给 C_{OUT} 充电，输出电压缓慢上升，如图 1-2 所示。当开关管 Q1 关断时（Q2 导通），电感 L 存储的能量继续对电容和负载提供电流，如图 1-3 所示。由于电感的能量慢慢变小，输出电压会缓慢下降。

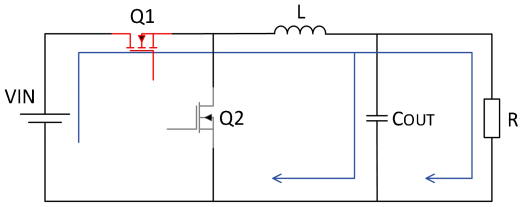

图 1-2 开关管 Q1 导通电流流向

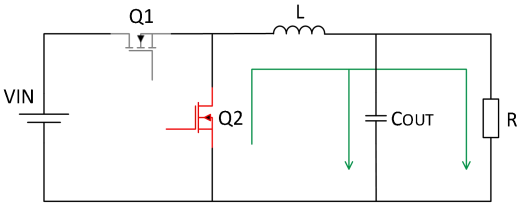

图 1-3 开关管 Q1 关断电流流向

传递公式可以从电感 L 在开关管 Q1 导通和关断情况下，由电压-时间的乘积公式来推导。基于能量守恒定理原则，两种情况下的电压-时间乘积相等（伏秒平衡）。

开关管 Q1 导通（ON）：$(V_{IN} - V_{OUT}) \times t_{ON}$ （式 1-1）

开关管 Q1 关断（OFF）：$V_{OUT} \times t_{off}$ （式 1-2）

式 1-1 与式 1-2 相减后得：

$$(V_{IN} - V_{OUT}) \times t_{ON} = V_{OUT} \times t_{OFF} \quad \text{（式 1-3）}$$

整理得：

$$V_{OUT} = \left(\frac{t_{ON}}{t_{ON} + t_{OFF}} \right) \times V_{IN} \quad \text{（式 1-4）}$$

定义 $D = T_{ON} / (T_{ON} + T_{OFF})$，称之为占空比，可将式 1-4 改写为：

$$V_{OUT} = D \times V_{IN} \quad \text{（式 1-5）}$$

在稳态情况下，输出电压为占空比和输入电压的乘积，其中 D 介于 0 和 1 之间。

1.2. 开关方案

开关对 PWM1A 和 PWM1B 是对称互补的，使得降压转换器的 Q1 和 Q2 交替导通。图 1-4 给出同步降压开关时序图。

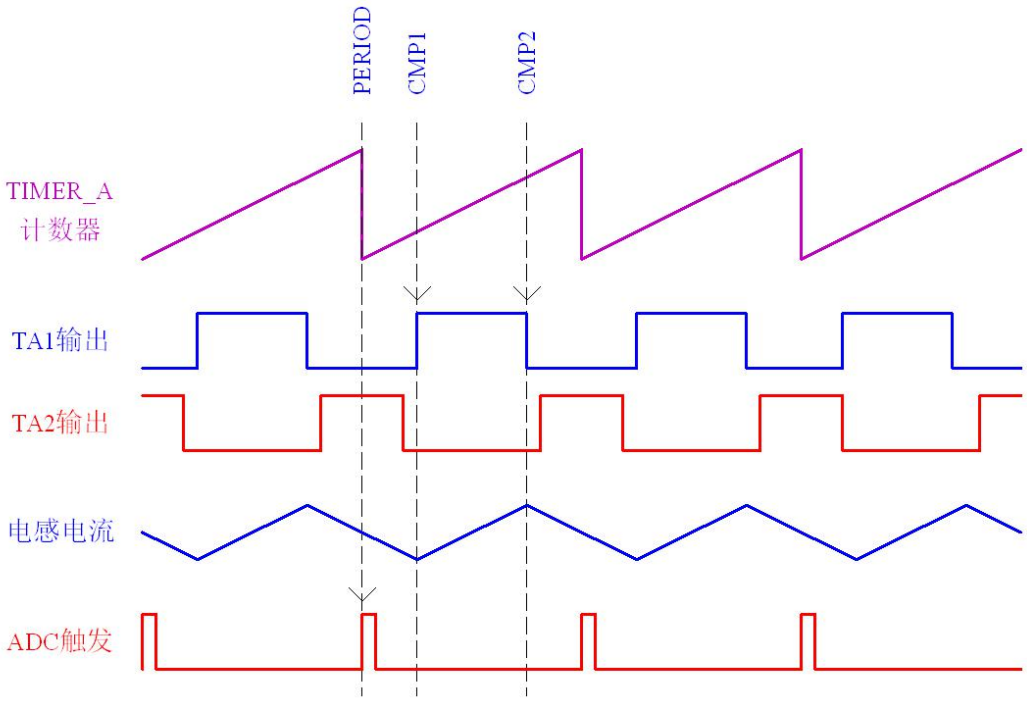

图 1-4 同步降压开关波形图

2. 硬件设计

本节将分别从功率级电路、MOSFET 驱动电路、信号调理电路、STM32 最小系统和辅助电源电路讨论数字式同步降压转换器的设计。如图 2-1 所示为数字式同步降压转换器的系统框图。

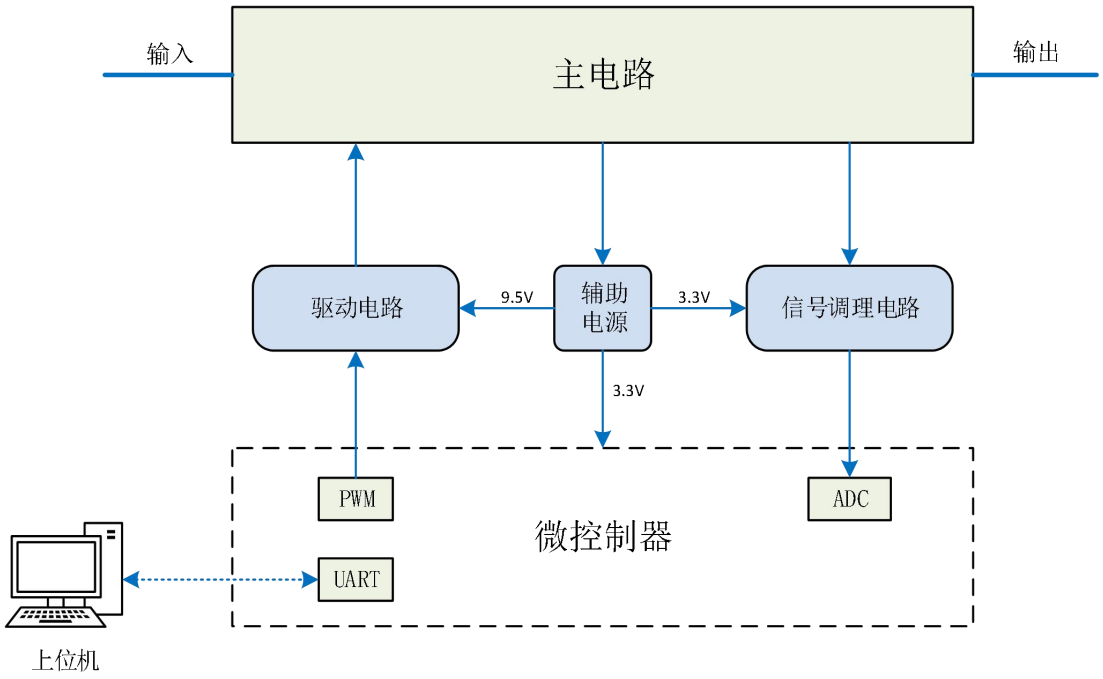

图 2-1 同步降压转换器系统图

2.1. 功率电路

本部分用于确定实现连续模式降压转换器相关元件参数计算公式，并结合实际给出计算过程。首先给出本设计的功率部分的电路原理图，如图 2-2 所示。

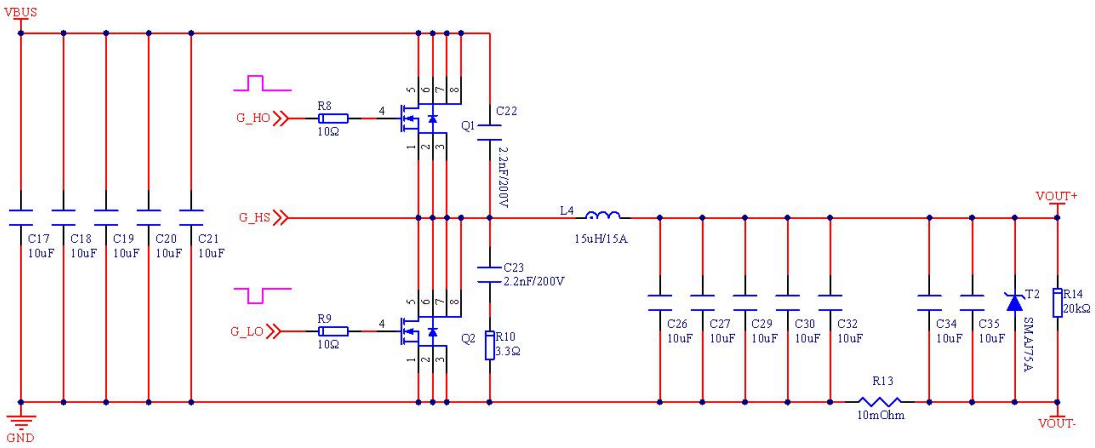

图 2-2 同步降压转换器功率电路原理图

在计算之前，我们确定额定设计要求：

- 输入电压：V_{IN} = 10V~48V
- 输出电压：V_{OUT} = 5V
- 输出电流：I_{OUT} = 8A
- 开关频率：F_{SW} = 200KHZ
- 电流纹波：ΔIL = 1.6A（输出电流的±10%）
- 输出纹波：$V_{p\text{-}p,OUT}$ = 50mV
- 输入纹波：$V_{p\text{-}p,IN}$ = 200mV

准备工作，计算出相关参数：

开关周期 $$Ts = \frac{1}{Fsw} = \frac{1}{200\times10^3} = 5us$$

最小占空比 $$D_{\min} = \frac{V_{OUT}}{V_{IN,\max}} = \frac{5}{48} = 0.11$$

最大占空比 $$D_{\max} = \frac{V_{OUT}}{V_{IN,\min}} = \frac{5}{10} = 0.5$$

最小输入电流 $$I_{IN,\min} = \frac{V_{OUT}\times I_{OUT}}{V_{IN,\max}} = \frac{5\times8}{48} = 0.83A$$

最大输入电流 $$I_{IN,\max} = \frac{V_{OUT}\times I_{OUT}}{V_{IN,\min}} = \frac{5\times8}{10} = 4A$$

注：输入电流按无损耗计算。

◆ 电感的选定

平均最小电流（$I_{O,av,min}$）设置为临界模式下的平均输出电流。当输出电流大于 $I_{O,av,min}$ 时，转换器将运行在连续模式（电感电流始终连续）。本设计最小平均电流取输出电流的±10%，计算公式如下：

$$L_{\min} = \frac{(V_{IN,\max} - V_{OUT})}{\Delta I_L}\times\frac{1}{F_{SW}}\times D \qquad \text{（式 2-1）}$$

根据式 2-1，使系统运行于连续模式，所需的电感值为：

$$L = \frac{48-5}{1.6}\times\frac{1}{200\times10^3}\times\frac{5}{48} = 14uH$$

电感最大电流计算如下：

$$I_{L,peak} = I_{OUT} + \frac{V_{OUT} \times (V_{IN} - V_{OUT})}{2 \times V_{IN} \times F_{SW} \times L} \quad \text{（式 2-2）}$$

根据式 2-2，求电感最大工作电流：

$$I_{L,peak} = 8 + \frac{5 \times (48 - 5)}{2 \times 48 \times 200 \times 10^3 \times 15 \times 10^{-6}} = 8.7A$$

根据上述结果，选择 14uH 以上且饱和电流大于 8.7A 的电感可满足需求。本设计采用电感量为 15uH，饱和电流为 12A 的电感，这给提升输出电流成为可能。事实上，输入电压并不恒定，所以应根据输入电压的最大最小值分别计算出所需电感值，并选取较大值，才能在整个输入电压范围内使系统进入连续模式。

◆ 输出电容计算

输出电容起到恒定输出电压的作用，但由于开关电源是周期性工作，所以不可避免的存在纹波电压。纹波电压有两部分组成，第一部分是电容本身的充放电引起的，另一部分是由于电容的串联等效电阻引起的。

第一部分：

$$\Delta V_{o1} = \frac{\Delta Q}{C} = \frac{\Delta I_L}{8 \times F_{sw} \times C} = \frac{V_{out} \times (1 - D) \times Ts^2}{8 \times L \times C} \quad \text{（式 2-3）}$$

第二部分：

$$\Delta V_{o2} = R_{ESR} \times \Delta I_L \quad \text{（式 2-4）}$$

总纹波电压：

$$\Delta V_o = \Delta V_{o1} + \Delta V_{o2} \quad \text{（式 2-5）}$$

我们通常会选择的电容是低 ESR 的或者采用电容并联方式以降低总 ESR，所以在计算纹波电压时，忽略 ESR 对纹波的影响。

$$\Delta V_o = \Delta V_{o1} \quad \text{（式 2-6）}$$

所以:

$$C_{\min} = \frac{V_{out} \times (1 - D_{\min}) \times Ts^2}{8 \times L \times \Delta V_o} \quad \text{（式 2-7）}$$

根据式 2-7 计算最小电容：

$$C_{\min}=\frac{5\times(1-\frac{5}{48})\times5^2}{8\times15\times0.05}=19uF$$

大于或等于 19uF 的输出电容即可满足 50mV 的输出纹波电压要求，我们这里采用 5 颗封装为 2812，容值为 10uF 的 MLCC 电容并联，总容值为 50uF。

◆ **输入电容**

输入电容在 TON 期间提供输出电流，并且其压降不得大于最大允许输入纹波电压。由于电容非常大，因此可将指数放电近似认为是线性放电。从电容汲取的电流为平均输出电流（Iin）。根据电荷守恒,输入电压纹波表达式：

$$\Delta V_i=\frac{\Delta Q}{C_{in}}=\frac{I_{in}\times T_{on}}{C_{in}}=\frac{I_{in}\times D}{C_{in}\times F_{sw}} \qquad (式\ 2\text{-}8)$$

所以：

$$C_{in,\min}=\frac{I_{in,\max}\times D_{\max}}{\Delta V_i\times F_{sw}} \qquad (式\ 2\text{-}9)$$

根据式 2-9 计算最小输入电容：

$$C_{IN}=\frac{4\times0.5}{0.2\times200\times10^3}=50uF$$

大于或等于 50uF 的输入电容即可满足 200mV 的输入纹波电压要求，我们这里采用 5 颗封装为 2812，容值为 10uF 的 MLCC 电容并联，总容值为 50uF。

◆ **开关管 MOSFET**

平均电流和最大电压是选择 MOSFET 的关键参数。本设计的输入电压范围是 10 至 48V，所以要求在最大电压 48V 时，系统能够安全运行。为躲避尖峰电压的影响，建议按 40%以上的安全裕量进行计算 MOSFET 的安全电压：

$$V_{Q,MAX}=V_{IN,MAX}+0.4\times V_{IN,MAX}=48\times1.4=67.2V$$

根据最大安全电压 $V_{Q,MAX}$ 为 67.2V，可选 VDS 为 80V 或者 100V 的 MOSFET。

本设计要求输出 240W，最大输出电流为 8A，意味输出电压为 30V 时，恰好最大功率输出。假设 $V_{IN}=36V$，能输出设计要求的最大功率 240W，效率按 95%计算输入端的平均电流为：$I_{IN,ave}=\frac{P_{OUT}}{0.95\times V_{IN}}=\frac{240W}{0.95\times36V}=7A$

开关管 Q1 平均电流：$I_{Q1,ave} = I_{IN,ave} = 7A$

同步管 Q2 平均电流：$I_{Q2,ave} = I_o \times (1-D) = 8 \times \left(1 - \frac{30}{36}\right) = 1.3A$

开关管的峰值电流为：$I_{Q1,peak} = I_{Q2,peak} = I_{L,peak} = 8.7A$

根据上述结果，所选的 MOSFET 为英飞凌的 BSC070N10NS3G，其 V_{DS} = 100V，I_D = 90A，R_{ON} = 7mΩ，$V_{GS(th),max}$=3.5V，$Q_{g,max}$=55nC。当然在实际中，还应根据系统特性计算出 MOSFET 的损耗，以保证温升在安全范围内，有必要进行散热设计，该过程本设计不讨论。

2.2. MOSFET 驱动电路

在开关电源中，场效应管是工作在饱和导通状态的，$R_{DS(ON)}$达到极小值，此时导通损耗也很小，系统整体效率才会提高。当 V_{GS} 大于 $V_{GS(th)}$（$V_{GS(th)}$可从场效应管的数据手册查询），场效应管都会处于饱和导通状态。由于控制器输出的 PWM 信号电压低、驱动能力弱、不能耐高压，所以需要增加一个信号放大电路以便于驱动场效应管。如图 2-3 所示为 MOSFET 自举驱动方案电路。

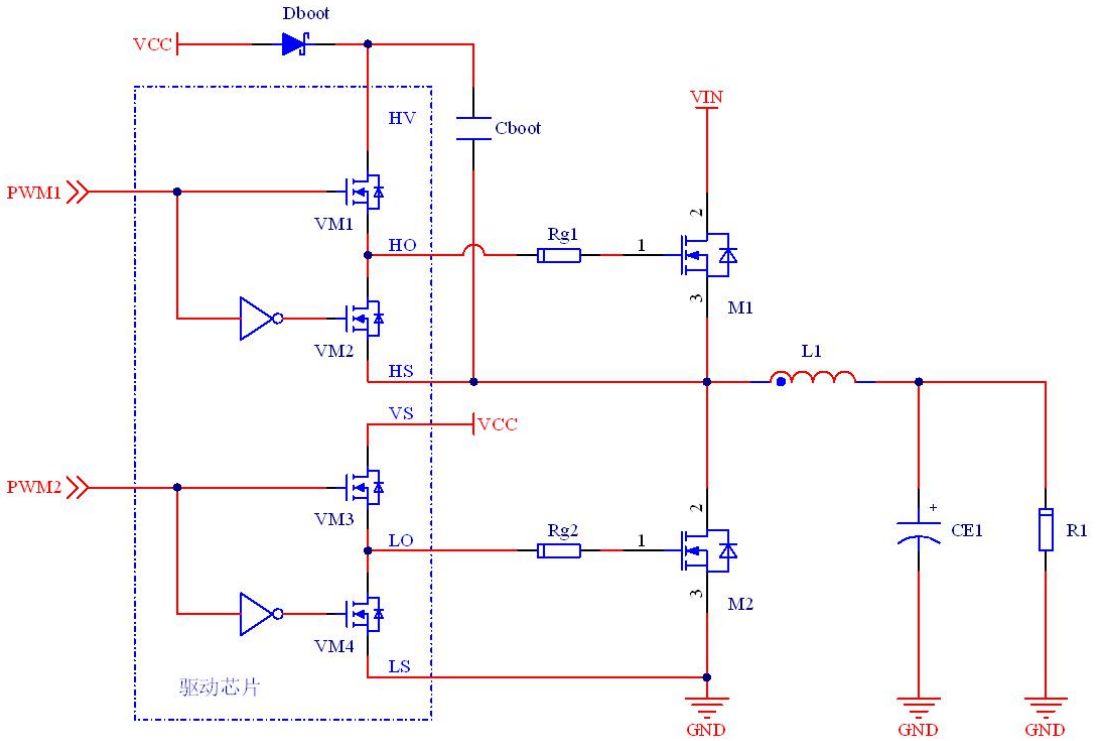

图 2-3 自举驱动电路

当下桥臂导通时，驱动电压 VCC 通过二极管 Dboot、下桥臂 M2 回路给自举电容 Cboot 充电；当上桥臂导通时，为整个高侧驱动电路提供浮动电源。由于自举电容不断地被消耗，所以应用这个电路时上桥臂不能 100%导通，需要定时开通下桥臂给自举电容充电。

自举电容能量的消耗，主要分为两大部分：第一部分，转移给了栅极电容 Cg；另一部分是整个高端电路的工作电流。根据电荷守恒原则：

$$C_{boot} * \Delta V_{boot} = Q_g + I_{\text{ave}} * T_{on} \qquad \text{（式 2-10）}$$

其中：

C_{boot} = 自举电容值

Δ_{Vboot} = 自举电容纹波电压

Q_g = 场效应管栅极电荷，可从数据手册中获取

I_{ave} = 上管开通时的平均电流

T_{on} = 上管开通时间

实际上，在上管开通时高侧电路的平均电流很小，有 $Q_g >> I_{ave} \times T_{on}$，因此在确定自举电容值时忽略其影响，有：

$$C_{boot} = \frac{Q_g}{\Delta V_{boot}} \qquad \text{（式 2-11）}$$

只要保证电容最低压降大于驱动阈值电压（$V_{GS(th)}$）,开关管能正常开通。本设计的驱动电压 VCC=9.5V，MOSFET 的 $V_{GS(th),max}$=3.5V，$Q_{g,max}$=55nC，自举二极管压降 $V_{D,boot}$=0.7V，可求出最大纹波电压：

$$\Delta V_{boot,\max} = VCC - V_{GS(th)} - V_{D,boot} = 9.5 - 3.5 - 0.7 = 5.3V$$

纹波电压取值低于 5.3V 可满足要求，假设 $\Delta Vboot = 0.5V$，代入（式 2-11）有：

$$C_{boot} = \frac{Q_g}{\Delta V_{boot}} = \frac{55}{0.5} = 110nF$$

考虑到计算时忽略了漏电流的影响，在最终取值上选择比计算值稍大，这里去自举电容值为 220nF。若要求自举电容的纹波电压更小，电容值可以适当取更大值。

自举二极管是一个重要的自举器件，它应能阻断直流干线上的高压，承受得了最大驱动电流。为了减少电荷损失，应选择反向漏电流小的快恢复二极管。

本设计的 MOSFET 驱动电路采用 TI 的半桥驱动芯片 UCC27211，驱动电路如图 2-4 所示。UCC27211 内部集成自举二极管，耐压 120V，驱动能力达 4A，满足大多数应用。特别注意，该驱动芯片内部不带死区时间功能，为避免上下桥臂同时导通，死区时间必须在微控制器的 PWM 模块中实现。

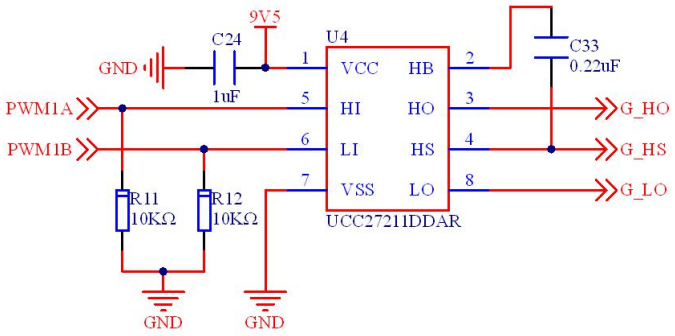

图 2-4 同步降压转换器 MOSFET 驱动电路

2.3. 信号调理电路

数字电源需要采样必要的电压电流信号，才能进行闭环控制。信号调理电路可将输入信号调整至匹配模数转换器（ADC）的范围，并进行滤波处理。本设计采集了输出电压、输出电流、输入电压和 MOSFET 温度等信号。

◆ 输出电压采样电路

使用图 2-5 所示电路采集直流输出电压。该差分放大电路将输出电压衰减至 ADC 模块输入电压水平。

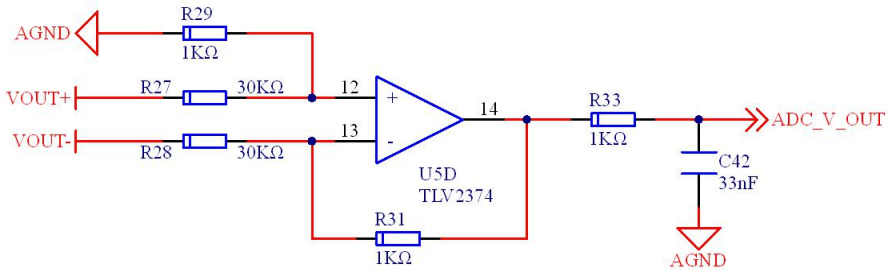

图 2-5 输出电压采样电路

当 R27 = R28 且 R29 = R31 可按（式 2-12）计算放大倍数：

$$Gain = \frac{R29}{R27} = \frac{R31}{R28} = \frac{1}{30} \qquad \text{（式 2-12）}$$

电阻 R33 和电容 C42 构成低通滤波器，其截止频率为：

$$f_{\text{cut-off}} = \frac{1}{2 \bullet \pi \bullet R33 \bullet C42} = \frac{1}{2 \times 3.14 \times 1000 \times 33 \times 10^{-9}} KHZ = 4.8KHZ$$

◆ 输出电流采样电路

使用图 2-6 所示电路采集直流输出电流。该采样电路采用分流器与差分放大电路结合的方案。其中 R13 为 10mΩ 的分流器，连接在板子和负载之间的地线上（该电路也叫低端采样）。电流流过分流器会有一个压降，并与负载电流成线性关系。该压降非常小，所以需要 R20、R21、R22、R23 和运放 U5A 构成的差分电路进行放大到 ADC 电压水平。

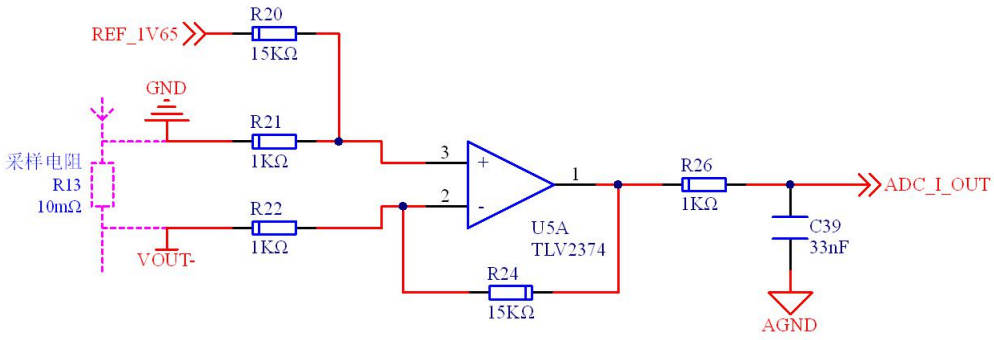

图 2-6 输出电流采样电路

当 R20 = R24 且 R21 = R22 可按公式 2-14 计算放大倍数：

$$Gain = \frac{R24}{R22} = \frac{R20}{R21} = \frac{15}{1} = 15 \qquad \text{（式 2-14）}$$

本设计中要求可以采样正负电流，所以在电路中强制增加一个 1.65V 的偏置电压。当 $V_{ADC\_I\_OUT}$ 高于 1.65V 时输出电流为正，低于 1.65V 为负。电压与电流 I_{OUT} 的关系可按公式 2-15 计算：

$$V_{ADC\_I\_OUT} = I_{OUT} \times R13 \times Gain + V_{ref} \qquad \text{（式 2-15）}$$

参数代入式 2-15 得：

$$V_{ADC\_I\_OUT} = 0.01 \times 15 \times I + 1.65 = 0.15 \times I + 1.65$$

电阻 R26 和电容 C39 构成低通滤波器，其截止频率为：

$$f_{\text{cut-off}} = \frac{1}{2 \bullet \pi \bullet R26 \bullet C39} = \frac{1}{2 \times 3.14 \times 1000 \times 33 \times 10^{-9}} KHZ = 4.8KHZ$$

◆ **输入电压采样电路**

使用图 2-7 所示电路采集直流输入电压。由 R30 和 R32 构成的电阻分压电路按比例将输入电压降低到 ADC 模块的输入电压水平，即范围 0V 到 3.3V 之间。电容 C41 用于对信号进行滤波，可使信号相对平稳的送入 ADC。

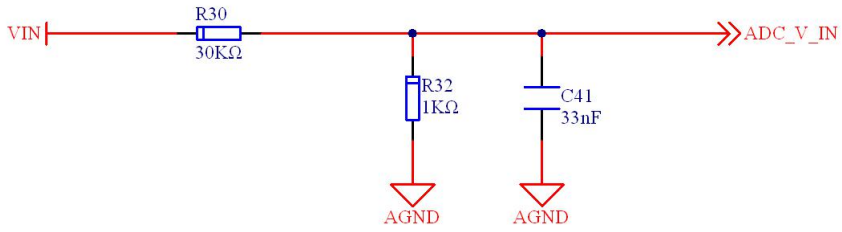

图 2-7 输入电压采样电路

R30 = 30K，R32 = 1K，衰减比可按公式 14 计算：

$$Gain = \frac{R32}{R30 + R32} = \frac{1}{30 + 1} = \frac{1}{31} \qquad \text{（式 2-26）}$$

◆ **MOSFET 温度采样电路**

使用图 2-8 所示电路采集 MOSFET 温度。由热敏电阻 RT1 和 R4 对 3.3V 分压后送入 ADC 模块，C8 为滤波电容。MCU 通过 V_{ADC_NTC} 电压值换算对应的温度值。

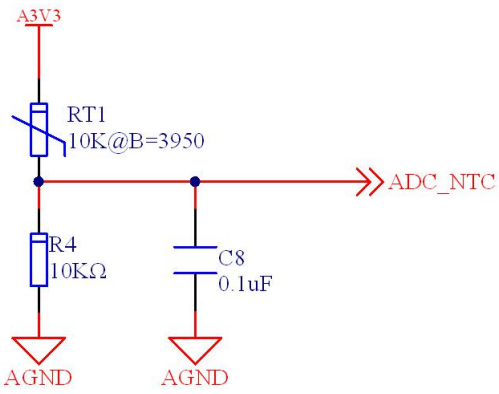

图 2-8 MOSFET 温度采样电路

2.4. 辅助电源电路

辅助电源从输入端获取电压，并向同步降压转换器控制系统的所有电子元件提供电能。

设计要求：

- 输入电压：10V-60V
- 输出 1： 9.5V@100mA
- 输出 2： 3.3V@50mA

由于输入电压范围较大以及要求辅助电源效率足够高，使用开关式降压转换器从 10V-60V 电压产生 9.5V。对于 3.3V 电压，所使用的对象为微控制器和运放，对电压纹波要求较高，所以使用简单价格低廉的线性稳压器。降压转换器选型为上海芯龙的 XL7005A，线性稳压器采用 SPX1117M3-L-3-3/TR，将两个稳压器串联，分别获取 9.5V 和 3.3V。如图 2-9 所示。

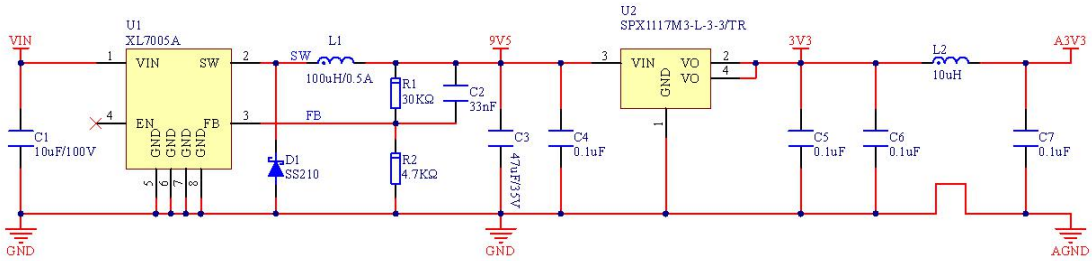

图 2-9 辅助电源电路

2.5. STM32 最小系统

如图 2-9 所示为 STM32 的最小系统图，也是本设计的核心控制器。STM32 控制器主要负责电压电流的采样及计算、PWM 波的产生以驱动功率电路、运行数字闭环控制和处理通信以便通过上位机控制等功能。如表 2-1 所示列出 STM32 的资源。

表 2-1 STM32 资源分配

序号	引脚名称	信号名称	说明
1	PA0	ADC_I_OUT	输出电流采样
2	PA1	ADC_V_OUT	输出电压采样
3	PA2	ADC_V_AIN	模拟输入信号
4	PA3	ADC_V_IN	输入电压采样
5	PA4	ADC_NTC	MOS 温度采样
6	PA8	PWM1A	上桥臂 PWM

表 2-1 续　STM32 资源分配

序号	引脚名称	信号名称	说明
7	PA9	PWM1B	下桥臂 PWM
8	PA13	SWDAT	仿真器信号
9	PA14	SWCLK	仿真器时钟
10	PB3	LED2	故障指示灯
11	PB4	LED1	运行指示灯
12	PB6	USART1_TX	串口通信发送
13	PB7	USART1_RX	串口通信接收

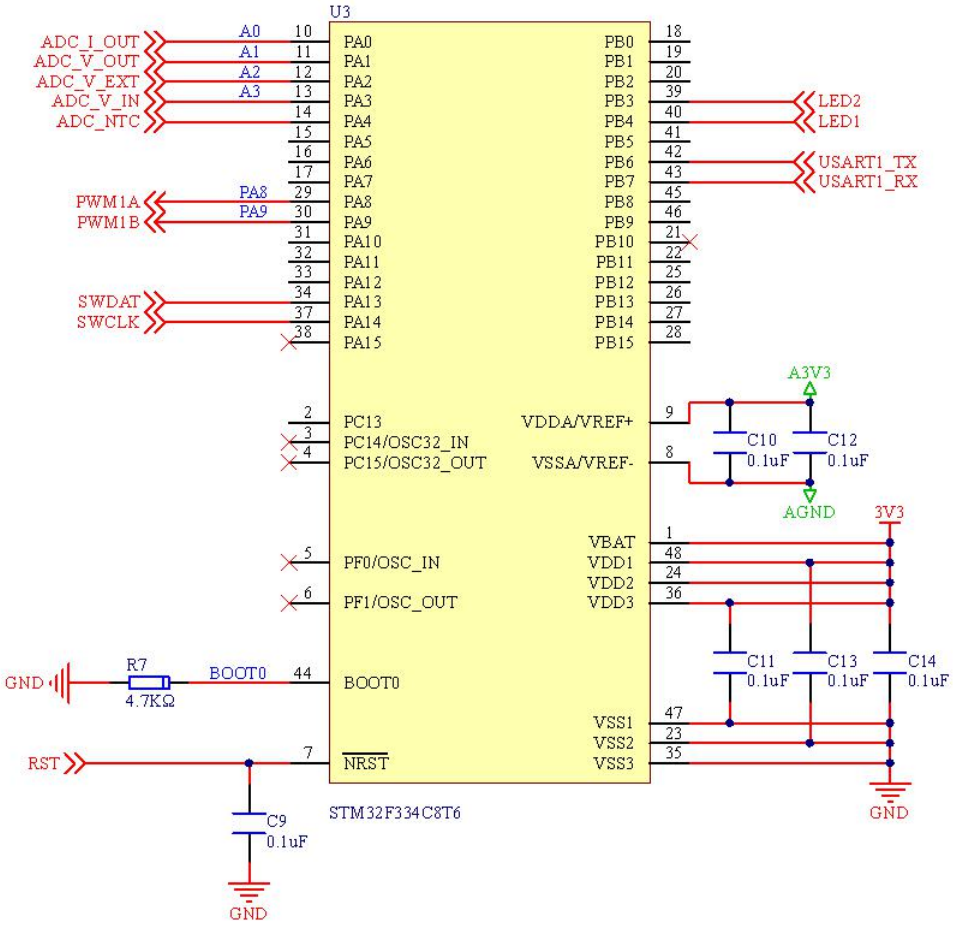

图 2-9 STM32 最小系统电路图

3. PID 控制原理

3.1. 模拟 PID 原理

在闭环控制系统中，PID 控制器是应用得最广泛的自动控制算法。PID 控制器由比例(P)、积分(I)和微分(D)通过线性组合构成，工作原理是基于误差而消除误差；它不依赖被控对象具体的数学模型，所以可以应用于多种场合。

如图 3-1 所示为 PID 控制器系统结构，期望(给定)输出 r(t)与实际(反馈)输出 y(t)比较，得到的误差值 $e(t)=r(t)-y(t)$；e(t)送入 PID 控制器进行运算后输出控制量 u(t)；u(t)经过功率放大控制被控对象输出，使得误差 e(t)不断减小直到为零。

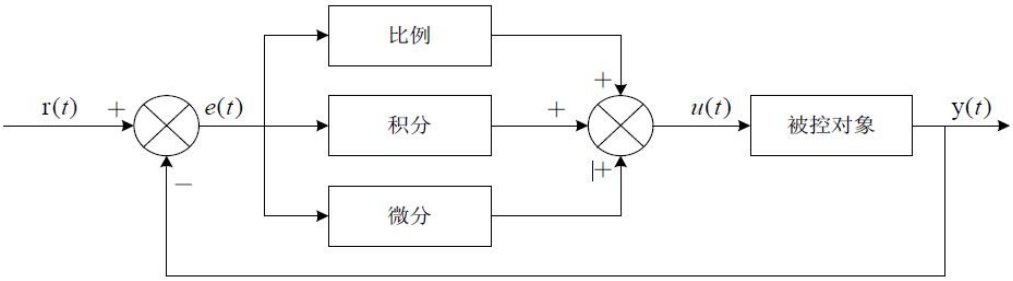

图 3-1 PID 控制系统

在模拟系统中，PID 控制器的表达为：

$$u(t)=Kp\left[e(t)+\frac{1}{Ti}\int_0^T e(t)dt+Td\frac{de(t)}{dt}\right] \qquad \text{（式 3-1）}$$

其中：

u(t) —— 控制器的输出量；

e(t) —— 误差信号，即给定值与反馈值之差；

Kp —— 比例系数；

Ti —— 积分时间；

Td —— 微分时间。

PID 控制器各个环节的作用如下：

1)比例环节(P)：在 PID 控制中，比例环节对系统偏差立即做出反应，保证系统的快速性。比例系数 Kp 的大小决定控制作用的强弱，Kp 越大作用越强，系统响应越快；Kp 越小，作用越弱，系统响应变慢。但值得注意的是，Kp 也不是越

大越好，过大的 Kp 会引起系统的振荡。

2)积分环节(I)：积分环节主要用于消除系统静态误差，提高系统的控制精度。积分作用的强度取决于积分时间 Ti，Ti 越大，积分作用越弱，反之越强。需要注意的是，Ti 在式 3-1 中处于分母位置，故 Ti 不能为 0，当我们不需要投入积分作用时应设 Ti 为较大值。

3)微分环节(D)：微分环节能预测误差信号的变化趋势，故当加入微分环节可以加入提前控制量，加快系统响应速度。当设定 $\mathrm{Td}=0$ 时，PID 控制器即变成 PI 控制，广泛应用于有惯性、滞后的场合。

3.2. 位置式数字 PID 控制器

计算机系统是一种离散化系统，只能处理离散时间的信息，不能像模拟控制那样连续输出控制量，进行连续控制。所以计算机系统不能直接使用（式 3－1）进行系统控制，必须将其进行离散化处理。常用的离散化方法有：欧拉方法、零极点匹配等效法、Z 变换法和保持器等效法等。由于计算机控制系统对实时性要求较高，所以必须采用简单、可靠和有效的方法。这里介绍欧拉法中的后向差分法进行离散化处理：以 T 作为采样周期，k 作为采样序号。当采用周期 T 足够小，我们可近似认为：

$$\left.\begin{aligned} &\mathrm{u}(t)\approx u(k)\\ &e(t)\approx e(k)\\ &\int_0^t e(t)dt\approx\sum_{i=0}^{k}e(i)\times\mathrm{T}\\ &\frac{de(t)}{dt}\approx\frac{e(k)-e(k-1)}{T}\end{aligned}\right\} \qquad \text{（式 3-2）}$$

将（式 3-2）代入（式 3-1）中得：

$$\mathrm{u}(k)=Kp\left[e(k)+\frac{1}{Ti}\times\sum_{i=0}^{k}e(i)\times T+Td\times\frac{e(k)-e(k-1)}{T}\right] \qquad \text{（式 3-3）}$$

整理可得：

$$\mathrm{u}(k)=Kp\times e(k)+Ki\times\sum_{i=0}^{k}e(i)+Kd\times(e(k)-e(k-1)) \qquad \text{（式 3-4）}$$

（式 3-4）即为数字 PID 控制器的位置式 PID 控制算法，其中：

k = 1,2,3,4,5…；

e(k)表示第 k 次误差值；

e(k-1)表示第 k-1 误差值；

U(k)表示第 k 次系统输出值；

Ki 表示积分环节系数，$\mathrm{Ki} = \mathrm{Kp} * \mathrm{T}/\mathrm{Ti}$ （式 3-5）

Kd 表示微分环节系数，$\mathrm{Kd} = \mathrm{Kp} * \mathrm{Td}/\mathrm{T}$ （式 3-6）

3.3. 增量式数字 PID 控制器

增量式 PID 是指控制器的输出只是控制量的增量 $\Delta \mathrm{u}(k)$。当执行机构需要的控制量是增量，而不是全量输出时，可以使用增量式 PID 控制算法进行控制。由（式 3-4）可知第 k-1 次的控制量输出为：

$$\mathrm{u}(k\text{-}1) = Kp\left[e(k\text{-}1) + \frac{1}{Ti} \times \sum_{i=0}^{k\text{-}1} e(i) \times T + Td \times \frac{e(k\text{-}1) - e(k-2)}{T} \right] \quad \text{（式 3-7）}$$

将（式 3-4）与（式 3-7）相减可得：

$$\begin{aligned}
&\Delta \mathrm{u}(k) = \mathrm{u}(k) - u(k-1) \\
&= Kp\left[\mathrm{e}(\mathrm{k}) - e(k-1) + \frac{\mathrm{T}}{Ti} e(k) + Td \frac{e(k) - 2e(k-1) + e(k-2)}{T} \right] \\
&= \mathrm{Kp}(1 + \frac{\mathrm{T}}{Ti} + \frac{\mathrm{Td}}{T}) e(k) - Kp(1 + \frac{2Td}{T}) e(k-1) + \frac{Kp \times Td}{T} e(k-2) \\
&= \mathrm{a0} \times \mathrm{e(k)} - \mathrm{a1} \times \mathrm{e(k\text{-}1)} + \mathrm{a2} \times \mathrm{e(k\text{-}2)} \qquad \text{(式3-8)}
\end{aligned}$$

式中，a0、a1、a2 定义如下：

$$\left.\begin{aligned}
&\mathrm{a0} = \mathrm{Kp}(1 + \frac{\mathrm{T}}{Ti} + \frac{\mathrm{Td}}{T}) \\
&a1 = Kp(1 + \frac{2Td}{T}) \\
&a2 = Kp\frac{Td}{T}
\end{aligned}\right\} \qquad \text{(式3-9)}$$

将 (式3-8) 改写成 $\mathrm{u}(k) = u(k-1) + \Delta u(k)$ 得：

$$\mathrm{u}(k) = u(k-1) + \mathrm{a0} \times \mathrm{e(k)} - \mathrm{a1} \times \mathrm{e(k\text{-}1)} + \mathrm{a2} \times \mathrm{e(k\text{-}2)} \qquad \text{(式3-10)}$$

通过上一次的输出加上本次增量即可实现全量式 PID 的计算，该方法比直接位置式（式 3-4）计算的工作量要少一些，是工程上广泛使用的方法。

3.4. 基于 C 语言的数字 PID 程序

需要注意的是，程序中的 a0、a1、a2 的值根据(式 3-9)计算得出。更新 Kp、Ti、Td 时只需事先计算一次 a0、a1、a2 存放在变量中，后面计算时直接调用，以节省 CPU 运行开支。

1）PID参数初始化：

```
void pid_init( PID_STRUCT *p )
{
    p->a0 = p->Kp * ( 1.0f + p->T / p->Ti + p->Td / p->T );
    p->a1 = p->Kp * ( 1.0f + p->Td / p->T * 2.0f );
    p->a2 = p->Kp * p->Td / p->T;
}
```

2）PID增量式计算：

```
void pid_calc( PID_STRUCT *p)
{
    /*      求误差      */
    p->Ek_0 = p->Ref - p->Fdb;
    /*      求增量      */
    p->Increm = ( p->a0 * p->Ek_0 - p->a1 * p->Ek_1 + p->a2 * p->Ek_2 );
    /*      计算输出值      */
    p->Output += p->Increm;
    /*      保存误差      */
    p->Ek_2 = p->Ek_1;
    p->Ek_1 = p->Ek_0;
}
```

4. 底层驱动

本数字电源设计，功率电路与模拟控制方案的相同，区别在于数字式的闭环控制。使用模拟数字转换器（ADC）将电压和电流等模拟信号数字化，然后反馈至数字补偿器，最后对脉冲宽度调制（PWM）进行调制以获得所需控制。可见数字电源中 ADC 和 PWM 是极为重要的功能模块，其分辨率决定系统的精度。

4.1. PWM 波形配置

PWM 技术使用非常广泛，尤其在工业电力电子系统的控制，如电机控制系统、开关电源及其他电力变换设备。PWM 的分辨率决定系统的控制精度：

$$PWM\text{分辨率} = \frac{\text{时钟频率}}{\text{PWM频率}} \qquad \text{（式 4-1）}$$

PWM 发生器须能够在高工作频率下具有良好分辨率，能够动态控制占空比、周期和相位等参数，并同步控制所有 PWM，还具有故障处理功能以及 CPU 负载交错以执行多个控制环。本设计所选择的微控制器 STM32F334 的 HRTIM 高分辨定时器模块满足所需条件。

HRTIM 高分辨率定时器可以产生 10 路独立输出通道，可达 217ps 高精度的 PWM，可满足开关电源的高精度和灵活控制。如图 4-1 所示为本设计需要输出的 PWM 时序，由从定时器 A 的 TA1 和 TA2 输出上产生。定时器 A 配置在连续工作模式，TA1 在 CMP1 置高，在 CMP2 复位；TA2 利用死区发生器，与 TA1 互补，并插入死区；ADC 转换在 PERIOD 触发，达到同步采样目的。

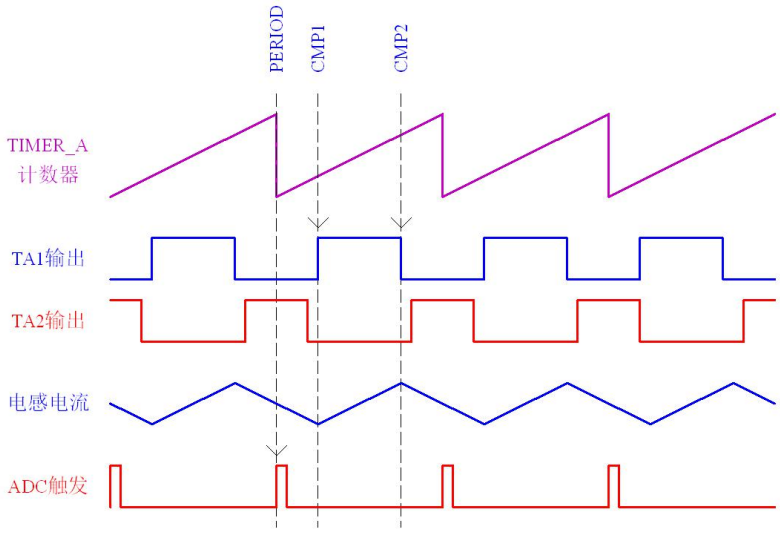

图 4-1 PWM 时序图

HRTIM 高分辨率定时器功能强大，使用的复杂度也较高，建议预先阅读 RM0364 参考手册（STM32F334xx 参考手册）。本节将从实际应用出发，说明高级定时器模块基于寄存器操作的配置。

◆ **系统时钟初始化**

为了实现高分辨率，HRTIM 需要由 PLL 高频输出直接馈送。高速内部（HSI）振荡器，能够提供 128MHz 的频率（8MHz 由 PLL 倍频 16 倍）。这种情况下，高分辨率步长为 244ps（128MHz 时钟周期的 1/32）。在系统初始化后，立即在主程序中使用以下函数，完成系统时钟初始化：

```
/*  系统时钟初始化  */
SystemInit( );
```

◆ **PWM 初始化**

1）使能 HRTIM、GPIOA 时钟：

```
/*  HRTIM1 时钟源为 PLL VCO  */
RCC_HRTIM1CLKConfig(RCC_HRTIM1CLK_PLLCLK);
/*  使能 GPIOA 时钟  */
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
/*  使能 HRTIM1 时钟  */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_HRTIM1, ENABLE);
```

2）配置 PA8、PA9 为输出模式并复用为 HRTIM 输出通道

```
GPIO_InitStructure.GPIO_Pin          = GPIO_Pin_8 | GPIO_Pin_9;
GPIO_InitStructure.GPIO_Speed   = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode   = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_OType   = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd        = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
/*  PA8 复用为 TA1，PA9 复用为 TA2  */
GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_13);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_13);
```

3）校准 HRTIM 时钟：

```
/*      使能周期校准      */
HRTIM1->HRTIM_COMMON.DLLCR |= 0x02UL;
/*      启动校准功能      */
HRTIM1->HRTIM_COMMON.DLLCR |= 0x01UL;
/*      等待校准完成      */
while( (HRTIM1->HRTIM_COMMON.ISR & 0x010000UL) != 0x010000UL ){ }
```

4）初始化 TIMER A 的工作模式、频率和重复计数器：

```
/*   配置 TIMA 控制寄存器   */
HRTIM1_TIMA->TIMxCR        &= 0x00UL;          //清空 HRTIM_TIMACR
HRTIM1_TIMA->TIMxCR        |= 0x01UL << 27;    //采用预加载模式
HRTIM1_TIMA->TIMxCR        |= 0x01UL << 18;    //TIMA 重置更新使能
HRTIM1_TIMA->TIMxCR        |= 0x01UL << 3;     //连续工作模式
HRTIM1_TIMA->TIMxCR        |= 0x02UL << 0;     //8 倍频，fHRCK: 1.024 GHz
/*   设置 TIMA 周期寄存器，决定工作频率   */
HRTIM1_TIMA->PERxR         = 1024e6 / fsw;
/*   设置 TIMA 重复计数寄存器，决定中断频率   */
HRTIM1_TIMA->REPxR         = 0x07;
```

5）初始化 TIMER A 的工作模式：

```
/*   CMP1 触发输出高电平   */
HRTIM1_TIMA->SETx1R     = 0x01UL << 3;
/*   CMP2 触发输出低电平   */
HRTIM1_TIMA->RSTx1R     = 0x01UL << 4;
```

6）死区配置：

```
/*   使能死区模式   */
HRTIM1_TIMA->OUTxR    = 0x100;
```

```
/*  设置死区时间  */
HRTIM1_TIMA->DTxR  &= 0x00UL;              //清空 HRTIM_DTAR
HRTIM1_TIMA->DTxR  |= 0x01UL << 10;        //死区时间时钟预分频值
HRTIM1_TIMA->DTxR  |= 0x00UL << 31; //可改写
HRTIM1_TIMA->DTxR  |= 0x00UL << 30; //下降沿死区信号可改写
HRTIM1_TIMA->DTxR  |= 0x00UL << 25; //下降沿正死区时间
HRTIM1_TIMA->DTxR  |= 0x00UL << 15; //可改写
HRTIM1_TIMA->DTxR  |= 0x00UL << 14; //可改写
HRTIM1_TIMA->DTxR  |= 0x00UL << 9;   //上升沿正死区时间
HRTIM1_TIMA->DTxR  |= PWM_DEADTIME << 16; //设置下降沿死区时间
HRTIM1_TIMA->DTxR  |= PWM_DEADTIME << 0;   //设置上升沿死区时间
```

7）预设比较值

```
/*  预设比较值，都设置为周期寄存器的一半  */
HRTIM1_TIMA->CMP1xR  = 0.5f * HRTIM1_TIMA->PERxR;
HRTIM1_TIMA->CMP2xR  = 0.5f * HRTIM1_TIMA->PERxR;
HRTIM1_TIMA->CMP3xR  = 0.5f * HRTIM1_TIMA->PERxR;
HRTIM1_TIMA->CMP4xR  = 0.5f * HRTIM1_TIMA->PERxR;
```

8）配置 ADC 转换触发：

```
/*  HRTIM 触发 ADC 转换配置  */
HRTIM1->HRTIM_COMMON.ADC1R      = RTIM_ADC1R_AD1TAPER;
HRTIM1->HRTIM_COMMON.CR1        = 0x01UL << 16;    //TIMA 触发 ADC
```

9）配置 TIMA 中断：

```
/*  配置中断组优先级  */
NVIC_Init.NVIC_IRQChannel                    = HRTIM1_TIMA_IRQn;
NVIC_Init.NVIC_IRQChannelSubPriority         = 0;
NVIC_Init.NVIC_IRQChannelPreemptionPriority  = 0;
NVIC_Init.NVIC_IRQChannelCmd                 = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

```
/*  配置定时器中断源  */
HRTIM1_TIMA->TIMxDIER = 0x01UL << 4;  //TIMA 重复中断使能;
```

10）启动 HRTIM_TIMER_A：

```
/*  启动 HRTIM_TIMR_A  */
HRTIM1->sMasterRegs.MCR  |= 0x01UL << 17;
```

◆ **PWM 禁止输出**

在系统停止运行时，强制 PWM 的 TA1 和 TA2 都为低电平，通过写 TIMA 禁用寄存器实现：

```
/*  禁止 TA1、TA2 输出  */
HRTIM1->HRTIM_COMMON.DISR  |= 0x03UL;
```

◆ **PWM 使能输出**

系统运行时，调用一次该函数即可使能 PWM 输出，通过写 TIMA 输出使能寄存器解除 PWM 禁用：

```
/*  使能 TA1、TA2 输出  */
HRTIM1->HRTIM_COMMON.OENR  |= 0x03UL;
```

◆ **PWM 占空比更新**

在需要更新占空比时，调用该函数并在参数中设置占空比值（范围为 0 到 TIMA 周期寄存器值），PWM 可输出 0-100%占空比。函数内部做了限幅处理，实际占空比在 2%到 95%之间，为配合自举电容在上管占空比很大也能正常充电的需求。该函数还实现了软件模拟的基于二分之一周期处的中心对称，该方法为了触发 ADC 转换时，尽可能地采样到平均值位置，实现同步采样。

```
uint32 m_cmp_max = 0.95F * HRTIM1_TIMA->PERxR;/*  定义最大比较值  */
uint32 m_cmp_min = 0.02F * HRTIM1_TIMA->PERxR;/*  定义最小比较值  */
/*  限幅处理  */
if(m_comp_value > m_cmp_max)m_comp_value = m_cmp_max;
if(m_comp_value < m_cmp_min)m_comp_value = m_cmp_min;
/*  更新比较寄存器值  */
HRTIM1_TIMA->CMP1xR  = ( HRTIM1_TIMA->PERxR - m_comp_value)/2u;
HRTIM1_TIMA->CMP2xR  = ( HRTIM1_TIMA->PERxR + m_comp_value)/2u;
```

4.2. ADC 采样配置

◆ **ADC 模块配置**

电压、电流、温度等连续信号必须通过 ADC 模块将模拟信号转换成数字信号，系统才能进行数字控制。在数字电源中，分辨率和转换速率这两个特性决定 ADC 模块的选择。本设计采用 STM32F334 内部 ADC 实现，其中输出电流连接至 PA0，输出电压连接 PA1，模拟输入信号连接 PA2，输入电压连接 PA3 及 MOS 温度连接 PA4，共 5 个通道的模拟采样电路。其中 PA0~PA3 使用 ADC1 转换，PA4 使用 ADC2 转换。本设计的 ADC 模块采用主从模式的双通道同步采样模式，由 HRTIM 模块触发转换，其结果由 DMA 传输至 gAdcSampleBuf[4]缓冲区，实现同步 PWM 时钟信号采样。以下为配置过程：

1）变量定义

```
uint32_t    gAdcSampleBuf[4];     //ADC 采样值,通过 DMA 接收
uint8_t     SampleTime     = 4;   //采样时间 19.5 ADC clock cycles
```

2）时钟配置

```
RCC_AHBPeriphClockCmd( RCC_AHBPeriph_GPIOA, ENABLE);
RCC_ADCCLKConfig( RCC_ADC12PLLCLK_Div1);
RCC_AHBPeriphClockCmd( RCC_AHBPeriph_ADC12, ENABLE);
RCC_AHBPeriphClockCmd( RCC_AHBPeriph_DMA1, ENABLE);
RCC_APB2PeriphClockCmd( RCC_APB2Periph_SYSCFG, ENABLE);
```

3）ADC 的 GPIO 配置

```
/*   PA0 ~ PA4   */
GPIO_InitStructure.GPIO_Pin       = 0x001F;
GPIO_InitStructure.GPIO_Mode   = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd    = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed   = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_OType   = GPIO_OType_PP;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

4）DMA 配置

```
DMA1_Channel1->CCR    &= 0x00UL;            //CCR 寄存器清零
DMA1_Channel1->CCR    |= 0x00UL << 4;       //方向，从外设读
DMA1_Channel1->CCR    |= 0x01UL << 5;       //循环模式
DMA1_Channel1->CCR    |= 0x01UL << 7;       //内存地址自动增加
DMA1_Channel1->CCR    |= 0x02UL << 8;       //外设 32 bit  数据宽度
DMA1_Channel1->CCR    |= 0x02UL << 10;      //内存也是 32bit 数据宽度
DMA1_Channel1->CCR    |= 0x03UL << 12;      //优先级最高

DMA1_Channel1->CNDTR  = 0x04;                              //数据长度
DMA1_Channel1->CPAR     = (uint32_t)(0x5000030C);          //外设地址
DMA1_Channel1->CMAR   = (uint32_t)gAdcSampleBuf;       //转换缓存地址
DMA1_Channel1->CCR      |= 0x01UL << 0;                 //DMA 使能
```

5）ADC1、ADC2 模块唤醒

```
/*   ADC1   */
ADC1->CR   &= 0x00UL;        //先清空 ADC1_CR 寄存器，退出深度休眠
ADC1->CR   |= 0x01UL << 28; //ADC1 电压稳压器使能

/*   ADC2   */
ADC2->CR   &= 0x00UL;        //先清空 ADC2_CR 寄存器，退出深度休眠
ADC2->CR   |= 0x01UL << 28; //ADC2 电压稳压器使能
```

6）ADC 模块全局控制寄存器配置

```
ADC1_2->CCR   &= 0x0000;           //复位 CCR 寄存器
ADC1_2->CCR   |= 0x01UL << 16;     //同步时钟模式，adc_hclk divided by 1
ADC1_2->CCR   |= 0x02UL << 14;     //DMA 12bit  传输
ADC1_2->CCR   |= 0x01UL << 13;     //DMA 连续传输
ADC1_2->CCR   |= 0x06UL << 0;      //常规双通道同步采样模式
```

7）ADC1 配置

```
ADC1->CFGR  &= 0x00UL;            //重置 ADC1_CFGR
ADC1->CFGR  |= 0x01UL << 12;      //数据实时更新
ADC1->CFGR  |= 0x01UL << 10;      //上升沿触发
ADC1->CFGR  |= 0x07UL << 6;       //HRTIM_ADCTRG1 触发转换
ADC1->CFGR  |= 0x00UL << 5;       //右对齐
ADC1->CFGR  |= 0x00UL << 3;       //12-bit  分辨率

ADC1->SQR1  &= 0x00UL;            //重置 ADC2_SQR1
ADC1->SQR1  |= 0x03UL << 0;       //转换 3+1 个通道
ADC1->SQR1  |= 0x01UL << 6;       //第 1 次转换 ADC1_CH1(PA0)
ADC1->SQR1  |= 0x02UL << 12;      //第 2 次转换 ADC1_CH2(PA1)
ADC1->SQR1  |= 0x03UL << 18;      //第 3 次转换 ADC1_CH3(PA2)
ADC1->SQR1  |= 0x04UL << 24;      //第 3 次转换 ADC1_CH4(PA3)

ADC1->SMPR1  &= 0x00UL << 31;                    //重置 ADC1_SMPR1
ADC1->SMPR1  |= (uint32_t)SampleTime << 0;       //第 1 通道采样时间
ADC1->SMPR1  |= (uint32_t)SampleTime << 3;       //第 2 通道采样时间
ADC1->SMPR1  |= (uint32_t)SampleTime << 6;       //第 3 通道采样时间
ADC1->SMPR1  |= (uint32_t)SampleTime << 9;       //第 4 通道采样时间
```

8）ADC2 配置

```
ADC2->CFGR  &= 0x00UL;            //重置 ADC1_CFGR
ADC2->CFGR  |= 0x01UL << 12;      //数据实时更新
ADC2->CFGR  |= 0x01UL << 10;      //上升沿触发
ADC2->CFGR  |= 0x07UL << 6;       //HRTIM_ADCTRG1 触发转换
ADC2->CFGR  |= 0x00UL << 5;       //数据右对齐
ADC2->CFGR  |= 0x00UL << 3;       //12-bit  分辨率
```

```
ADC2->SQR1    &= 0x00UL;              //重置 ADC2_SQR1
ADC2->SQR1    |= 0x03UL << 0;         //转换 3+1 个通道
ADC2->SQR1    |= 0x01UL << 6;         //第 1 次转换 ADC2_CH1(PA4)
ADC2->SQR1    |= 0x01UL << 12;        //第 2 次转换 ADC2_CH2(PA4)
ADC2->SQR1    |= 0x01UL << 18;        //第 3 次转换 ADC2_CH3(PA4)
ADC2->SQR1    |= 0x01UL << 24;        //第 4 次转换 ADC2_CH4(PA4)

ADC2->SMPR1    &= 0x00UL << 31;                    //重置 ADC2_SMPR1
ADC2->SMPR1    |= (uint32_t)SampleTime << 0;       //第 1 通道采样时间
ADC2->SMPR1    |= (uint32_t)SampleTime << 3;       //第 2 通道采样时间
ADC2->SMPR1    |= (uint32_t)SampleTime << 6;       //第 3 通道采样时间
ADC2->SMPR1    |= (uint32_t)SampleTime << 9;       //第 4 通道采样时间
```

9）ADC 校准

```
/*   启动 ADC 校准   */
ADC1->CR    |= 0x01UL << 31;
ADC2->CR    |= 0x01UL << 31;
/*   等待 ADC1 校准完成   */
while( (ADC1->CR & ADC_CR_ADCAL) == ADC_CR_ADCAL){ }
while( (ADC2->CR & ADC_CR_ADCAL) == ADC_CR_ADCAL){ }
```

10）启动 ADC

```
ADC1->CR    |= 0x0001UL;   //使能 ADC1
ADC2->CR    |= 0x0001UL;   //使能 ADC2
while( (ADC1->ISR & 0x0001UL) != 0x0001UL ){} //等待 ADC1 就绪
while( (ADC2->ISR & 0x0001UL) != 0x0001UL ){} //等待 ADC2 就绪

ADC1->CR    |= 0x0004UL;   //启动 ADC1 转换
ADC2->CR    |= 0x0004UL;   //启动 ADC2 转换
```

4.3. 数据转换

ADC 转换完成后，通过读取 gAdcSampleBuf[4]缓冲区的数据获得结果值，根据调理电路的系数进一步转换成有名值，才能数据处理、闭环控制、故障保护和显示等。数据的转换我们通过程序实现：

1）首先读取输出电流的 ADC 值：

```
输出电流 ADC 值    =   gAdcSampleBuf[0].Channel.Adc1
输出电压 ADC 值    =   gAdcSampleBuf[1].Channel.Adc1
模拟输入 ADC 值    =   gAdcSampleBuf[2].Channel.Adc1
输入电压 ADC 值    =   gAdcSampleBuf[3].Channel.Adc1
MOS 温度 ADC 值    =   gAdcSampleBuf[0].Channel.Adc2
```

2）根据第 2 章定义相关参数的比例和偏移量：

```
#define DP_CURRENT_OFFSET          (1.65 /3.3 *4095)   //电流偏移
#define DP_VOLTAGE_IN_RATIO        (31*3.3 / 4095)     //输入电压比例
#define DP_VOLTAGE_OUT_RATIO       (30 *3.3 / 4095)    //输出电压比例
#define DP_CURRENT_OUT_RATIO       (3.3 /0.15 / 4095)  //输出电流比例
#define DP_VOLTAGE_AIN_RATIO       (10 * 3.3 / 4095)   //模拟输入比例
```

3）输出电压计算：

```
Vout   =   DP_VOLTAGE_OUT_RATIO * gAdcSampleBuf[2].Channel.Adc1;
```

4）输出电流计算

```
tmp   =   ( gAdcSampleBuf[0].Channel.Adc1 - DP_CURRENT_OFFSET );
Iout   =   DP_CURRENT_OUT_RATIO * tmp;
```

5）输入电压计算：

```
Vin   =   DP_VOLTAGE_IN_RATIO * gAdcSampleBuf[3].Channel.Adc1;
```

限于篇幅，本节只举例输出电压、输出电流、输入电压三个关键参数的计算，模拟量输入和 MOS 温度可同理求出。

5. 程序设计

本设计的程序全部采用 C 语言，实现系统所有的管理任务、决策制定、智能管理、通信处理以及与上位机的交互。功率级的数字闭环控制在实时中断服务程序实现，功能应用在主函数循环处理。

5.1. 关键 C 文件及函数说明

main/mian.c

此文件用于调用外设初始化程序、处理循环任务、运行及管理客户的应用；还包含运行环路控制的中断服务程序。

int main(void)

- 主函数
- 配置器件的工作频率。
- 调用用于配置 GPIO、UART、ADC 和 PWM 模块的函数。
- 处理 modbus 通信协议。
- 处理循环任务。
- 处理 LED 指示灯亮灭

void HRTIM1_TIMA_IRQHandler(void)

- 定时器 TIMA 中断服务函数
- 运行数字电源环路控制
- 运行频率与 PID 控制频率一致

void TIM1_UP_TIM16_IRQHandler(void)

- 定时器 TIM1 中断服务函数
- 提供循环任务的系统时钟
- 工作频率为 10KHZ

dp_buck/dp_buck.c

void dp_BuckCtrlInit(void)

- 电源相关数据初始化
- 电压电流参数初始化
- 数字滤波参数初始化
- ADC 模块初始化
- PWM 模块初始化
- PID 参数初始化

void dp_ExcursionCheck(void)

- 输出电流零漂检测
- 电源待机时运行

void dp_BuckCtrl(void)

- 电压电流计算及滤波
- 故障保护检测及处理
- 电源环路闭环控制
- 更新 PWM 占空比

dp_buck/dp_filter.c

void low_filter_init(LOW_FILTER_STRUCT *p)

- 低通滤波初始化

void low_filter_clc(LOW_FILTER_STRUCT *p)

- 低通滤波相关数据清除

void low_filter_calc(LOW_FILTER_STRUCT *p)

- 低通滤波算法计算

dp_buck/dp_pid.c

static void pid_reset(struct PID_STRUCT *p);

- PID 数据清空

static void pid_init (struct PID_STRUCT *p);

- PID 参数初始化

static void pid_clc (struct PID_STRUCT *p);

- PID 参数相关参数清除

static void pid_calc (struct PID_STRUCT *p, float fb, float ref);

- PID 算法运算

dp_driver/dp_pwm.c

void dr_pwm_stop(void);

- PWM 禁止输出

void dr_pwm_start(void);

- PWM 启动输出

void dr_pwm_init(uint32 fsw);

- PWM 模块初始化

void dr_pwm_update(uint32 CompareValue);

- PWM 占空比更新

dp_driver/dp_adc.c

void dr_AdcInit(void)

- ADC 模块初始化

5.2. 开环控制

通过直接给定 PWM 占空比 D 控制 BUCK 降压电路输出电压，实验现象通过万用表测量，也可通过上位机观察。电源控制环路 dp_BuckCmcCtrl 以 25KHZ 的频率运行（PWM 开关频率的十分之一）。读取 ADC 值，将其转换为有名值，由于开环控制不需要反馈值，所以电压电流信号仅做为故障保护。由于固定占空比，所以输出电压与输入电压和占空比值有关，其关系为 $Vout = Vin \times D$ 。开环控制框图如图 5-1 所示。

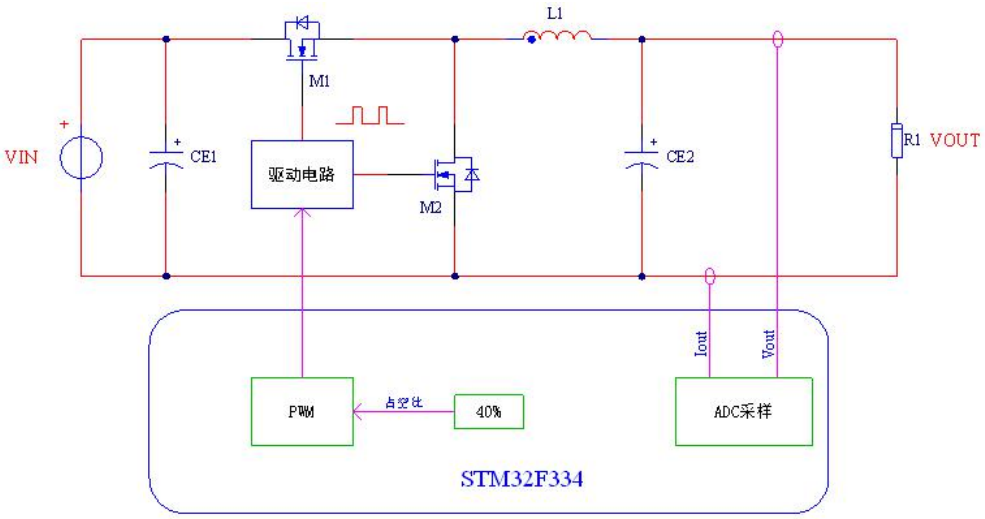

图 5-1 降压转换器的开环控制框图

程序上，电源控制函数 dp_BuckCmcCtrl 直接调用 PWM 占空比更新函数 dr_pwm_update 并设定一个固定占空比，完成开环控制，部分代码如下：

```
void dp_BuckCmcCtrl(void)
{
    .....
    dr_pwm_update( 0.40f * DP_PWM_PER );    // 固定 40%占空比
    dr_pwm_start();//使能 PWM 输出
    .....
}
```

*完整的程序为 bu4805s_open_loop_demo v1.3.0

5.3. 电压模式控制

在电压模式控制中，对输出电压（Vout）进行测量并将测量结果与参考值（Vref_out）进行比较求得误差值。PID 补偿算法将对误差进行处理，产生下一个控制量（占空比），达到维持输出电压稳定的目的。如图 5-2 所示为电压模式控制框图。

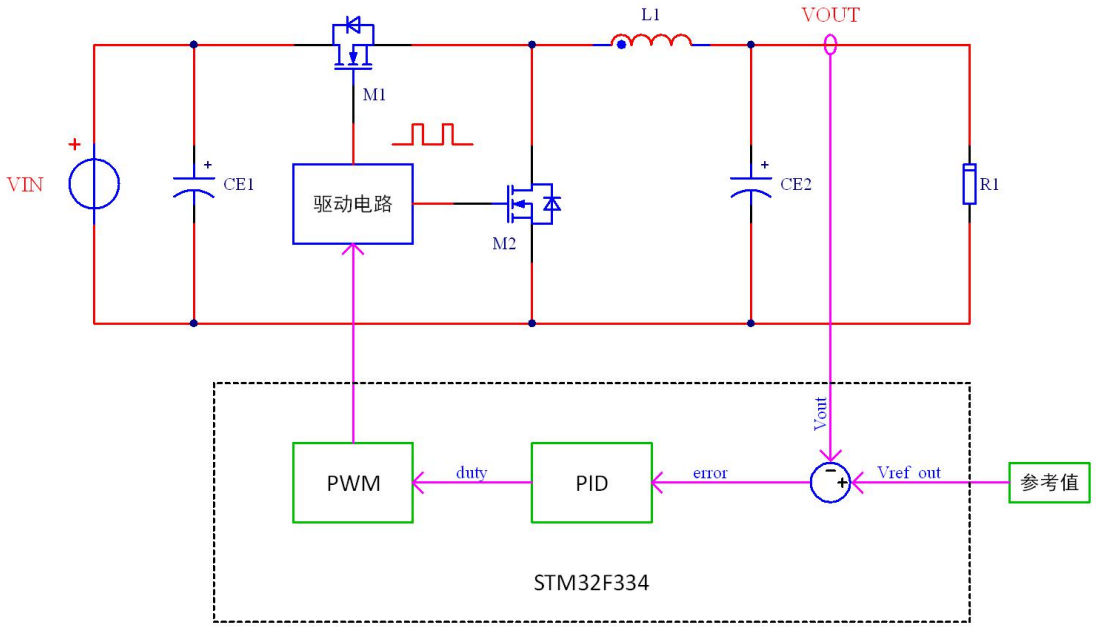

图 5-2 降压转换器的电压模式控制

电压模式控制是单闭环控制，其中只有一个 PID 控制器。PID 算法函数内部已包含求误差操作，程序中电压目标值和当前电压值赋值给 PID 的参数结构体；PID 函数完成控制量计算，最后将控制量直接更新 PWM 占空比。关键程序如下：

```
void dp_BuckCtrl(void)
{
    ......
    /*  目标电压值  */
    gPID_VoutLoop.Ref       =  gMainCmd.VoRef;
    /*  输出电压反馈值  */
    gPID_VoutLoop.Fdb       =  gVoutStr.Value;
    /*  输出电压环 PID 计算  */
    pid_calc( &gPID_VoutLoop );
```

```
    /*   更新 PWM 占空比   */
    dr_pwm_update( gPID_VoutLoop.Output );
    dr_pwm_start();//使能 PWM 输出
    .....
}
```

* 完整的程序见 bu4805s_vmc_demo v1.3.0

5.4. 电流模式控制

电流模式控制技术需要两个反馈环，如图 5-3 所示。在这一模式，需对输出电压和输出电流进行检测以实现控制目的。在电流模式控制中，首先将输出电压和参考电压（期望输出电压）进行比较，PID 电压补偿程序随后对该误差进行处理以产生电流环的参考信号。电流参考信号将与测量的电流进行比较，PID 电流补偿程序将对由电压补偿器产生的参考信号与实际输出电流进行比较得到的误差进行处理。这将产生所需占空比以保持输出电压在限定范围之内。由于电流模式控制对电路电流进行检测，因此任何输出负载电流或输入电压的变化在影响输出电压之前都会被校正。由于采用电流内环，因此阶跃负载响应和瞬态响应特性也得到改善。

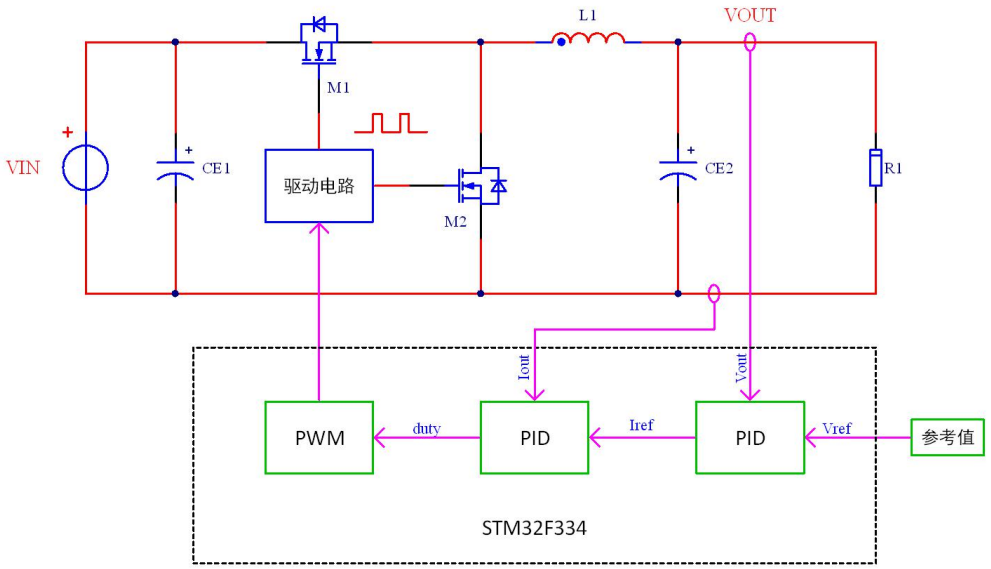

图 5-3 降压转换器的电流模式控制

电流模式控制是双闭环控制，一般电压环为外环，电流环为内环。本设计中内环的运行频率与 TIMA 中断频率一致，所以电压环频率为 25KHZ；外环运行频率要比内环低，这里为 5KHZ（内环频率的五分之一）。关键程序如下：

```
void dp_BuckCtrl(void)
{
    ......
```

```
  if( i == 5 )      /* 外环运行频率是内环的五分之一 */
  {
    /*  目标电压值  */
    gPID_VoutLoop.Ref          =  gMainCmd.VoRef;
    /*  输出电压反馈值  */
    gPID_VoutLoop.Fdb          =  gVoltOutStr.Value;
    /*  输出电压环 PID 计算  */
    pid_calc( &gPID_VoutLoop );
  }
  /*  外环的输出是内环的参考值  */
  gPID_IoutLoop.Ref          =  gPID_VoutLoop.Output;
  /*  输出电流反馈值  */
  gPID_VoutLoop.Fdb          =  gIoutStr.Value;
  /*  输出电压环 PID 计算  */
  pid_calc( &gPID_IoutLoop );
  /*  更新 PWM 占空比  */
  dr_pwm_update( gPID_IoutLoop.Output );
  dr_pwm_start();//使能 PWM 输出
  .....
}
```

* 完整的程序见 bu4805s_cmc_demo v1.3.0

2020-02-29