****

**电力电子综合设计报告书**

**设计题目：基于单片机的开关电源设计与制作**

**姓 名： 叶盛廷**

**班 级： 电气12201**

**学 号： 2203240107**

**指导教师： 韩谷静**

**成绩评定：**

**2025年6月20日**

目录

[第1章 设计任务与要求 3](#_Toc201744976)

[1.1 设计任务 3](#_Toc201744977)

[1.2 设计要求 3](#_Toc201744978)

[第2章 系统硬件设计 4](#_Toc201744979)

[2.1 电源板设计 4](#_Toc201744980)

[2.1.1 Boost电路设计 4](#_Toc201744981)

[2.1.2 采样电路设计 4](#_Toc201744982)

[2.1.3 接地与隔离 6](#_Toc201744983)

[2.2 控制板设计 7](#_Toc201744984)

[2.2.1 电源设计 7](#_Toc201744985)

[2.2.2 复位与时钟 7](#_Toc201744986)

[2.2.3 外部接口 8](#_Toc201744987)

[2.2.4 MCU系统设计 8](#_Toc201744988)

[2.3 PCB-Layout 9](#_Toc201744989)

[2.3.1 电源板布局 9](#_Toc201744990)

[2.3.2 控制板布局 12](#_Toc201744991)

[第3章 系统软件设计 12](#_Toc201744992)

[3.1 仿真建模分析 12](#_Toc201744993)

[3.1.1 开环分析 14](#_Toc201744994)

[3.1.2 闭环分析 14](#_Toc201744995)

[3.2 开环特性分析 15](#_Toc201744996)

[3.3 闭环算法设计 17](#_Toc201744997)

[3.3.1 系统初始化 17](#_Toc201744998)

[3.3.2 采样滤波以及控制 18](#_Toc201744999)

[第4章 系统调试与测试 20](#_Toc201745000)

[4.1 调试 20](#_Toc201745001)

[4.2 测试 26](#_Toc201745002)

[4.2.1 动态带载 27](#_Toc201745003)

[4.2.2 效率分析 28](#_Toc201745004)

[4.2.3 纹波 29](#_Toc201745005)

[4.2.4 温度分析 30](#_Toc201745006)

[4.2.5 EMI测试 30](#_Toc201745007)

[第5章 设计总结 31](#_Toc201745008)

[第6章 参考文献 31](#_Toc201745009)

[6.1 附录 32](#_Toc201745010)

# 设计任务与要求

## 设计任务

1. 题目：基于单片机的开关电源设计与制作
2. 设计并制作如图1所示的直流开关电源。



图1 直流开关电源框图

2）设计基本要求

（1）输出电压*U*O：35-40VDC（按36VDC额定电压设计）。

（2）最大输出电流*I*Omax：2A（实验室提供100Ω滑动变阻器）。

（3）具有电路保护功能（通过AD模块采样软件保护）。

1. 设计加分要求

（1）*I*O从0变到1A时，输出电压*U*O 波动不超过±10%。

（2）电源效率≥80%（*U*2=18VAC,*U*O=36VDC, *I*O=1A）。

## 设计要求

1. 基本要求
2. 采用单片机进行开环PWM升压控制。

2）根据题目要求进行方案论证，理论分析与计算，可重新或修改参考设计。

3）用Matlab等软件进行主电路原理仿真（参考：电气22级电力电子综合设计Matlab参考仿真模型word版）。

4）原理图及PCB采用立创EDA软件进行设计，然后申请免费打板，PCB上至少需丝印“武汉纺织大学+电气1220x班+“本人姓名”。

5）电源电路焊接、调试，并达到参数要求。

6）撰写设计报告。

# 系统硬件设计

## 电源板设计

### Boost电路设计

本次设计要求需要设计的电源板中升压斩波电路(boost)，采样电路和栅极驱动电路。

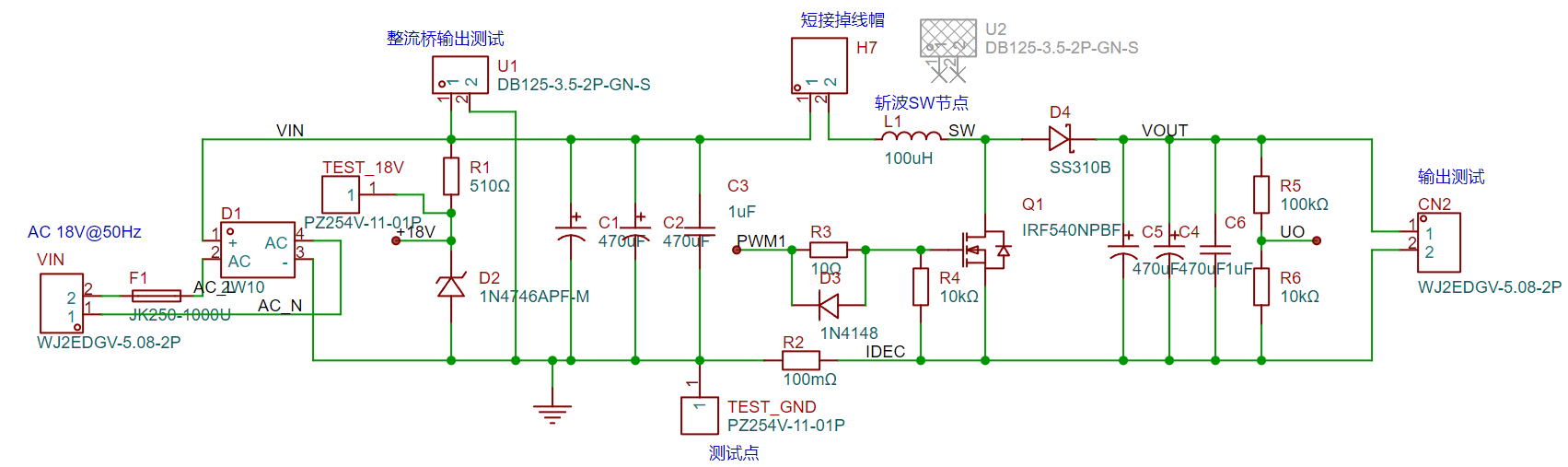


图 1 Boost电路设计

输入18V的交流电后经过整流桥整流，然后经过典型的升压斩波电路拓扑进行升压，并配备有电流采样电阻，滤波电容以及各种测试点。

### 采样电路设计

采样电路首先经过一个ADC采样前级的电压跟随器（Buffer），使用了一个运算放大器LM358，输入信号首先经过了一个的限流电阻，它同时可以抑制输入突变对后级的影响，与电容形成RC低通滤波器。经过计算，可以抑制输入噪声



的噪声。并且被配有一个快速二极管，用于钳位，当电压超过5V的时候保护信号输入端。随后信号到达运放，运放在这里配置为电压跟随器，输出电压紧跟随输入电压，提供低输出阻抗。

前级具体功能汇总由下表给出

表格 1 前级设计

|  |  |
| --- | --- |
| 原因 | 描述 |
| 降低信号源阻抗 | ADC输入通常具有较高的输入阻抗和较低的采样电容。如果信号源阻抗太高,会导致ADC采样电容无法及时充电,影响精度。电压跟随器具有极高的输入阻抗和极低的输出阻抗,能驱动ADC而不引起压降或响应延迟。 |
| 隔离功能 | 运放起到隔离作用,避免ADC的采样瞬间拉低信号源电压,防止干扰信号源稳定性。 |
| 稳定电压采样 | 运放输出能力强,能快速跟随输入变化并驱动ADC,因此提高采样精度和响应速度。 |
| 简单滤波配合 | 前级RC滤波器(R7+C7)配合电压跟随器,可有效滤除高频干扰,而不会因电阻导致压降。 |
| 过压保护(D5) | 当 UO意外高于+5V(如电感反冲或外部错误信号),D5导通将电压钳位保护在+5V附近,保护LM358和ADC输入。 |

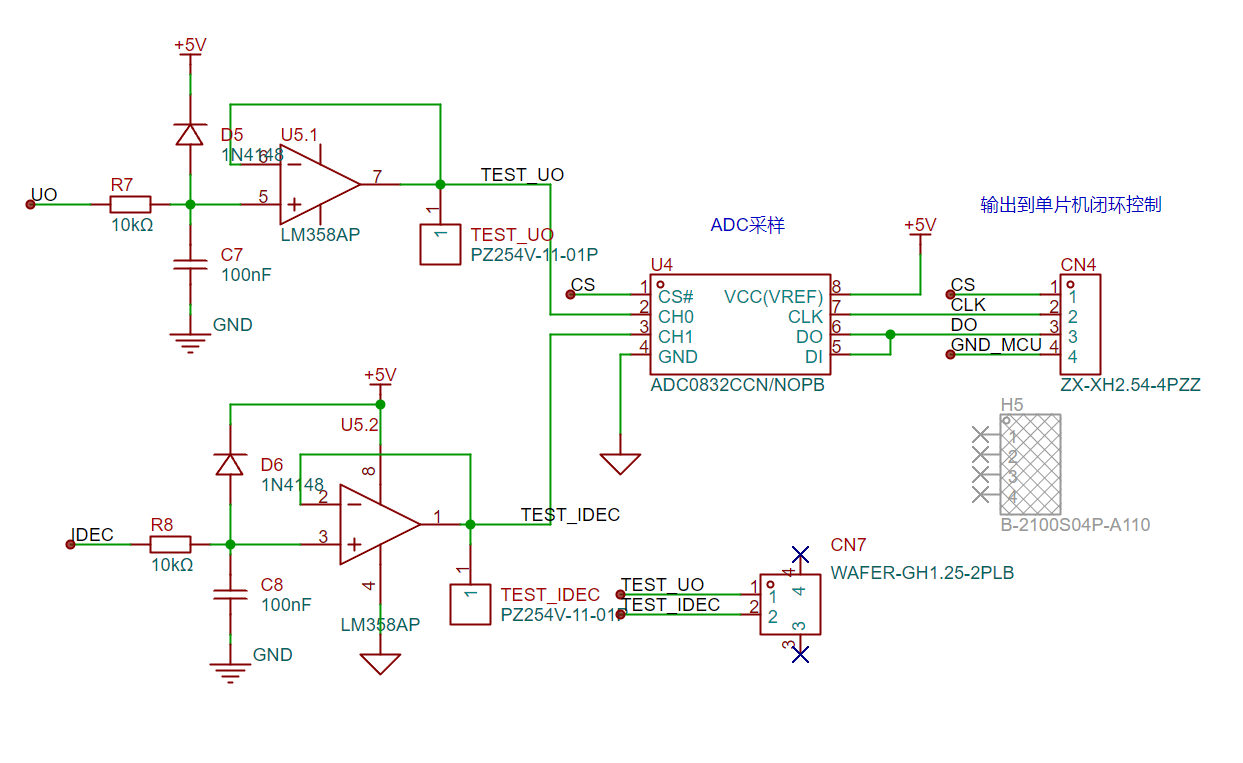


图 2 ADC采样及其前级

由于MCU的I/O口推挽能力不够，因此本次采样一颗带有高低侧驱动的IR2104栅极驱动

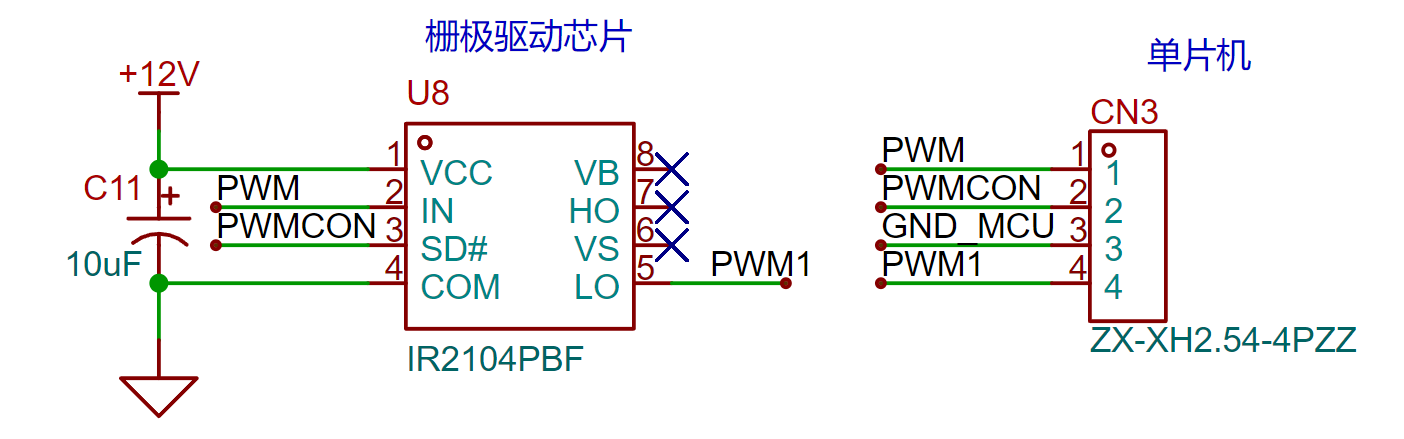


图 3 栅极驱动

该栅极驱动具有UVLO功能，因此在电感的DCM区会导致PWM占空比陡然减小，该现象将在第四章做详细说明，该栅极驱动的t\_{on/off}(typ.)为 680ns和150ns，加上死区时间520ns，预期的PWM频率上限可以设定为



为了流出足够的裕度以及根据实际波形质量测试结果最终采用100kHz的PWM频率。

### 接地与隔离

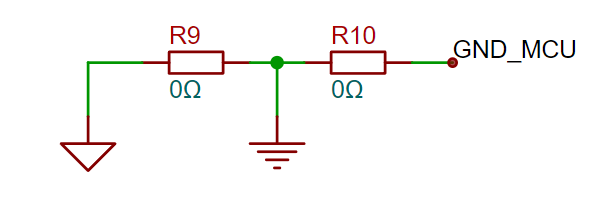


图 4 单端接地

采用0Ohm的电阻对功率地和数字地进行单端桥架，以减小回流路径，将高频噪声，高电流回流的地电流和精密信号回流路径物理隔离，用于改善EMC性能。

有的设计此处采用磁珠进行隔离，可以提供更优秀的高频阻抗，抑制高频噪声，改善EMI，此处因为成本以及方便调试采用0Ohm进行单端接地。

## 控制板设计

本次设计由于涉及ADC采样，各种硬件防护以及闭环控制等，因此需要配备有一个控制单元，采用MCU作为控制，具体型号为STM32F411CEU6，以下为具体的详细设计:

### 电源设计

Boost端为18V整流，首先需要将电压等级降至MCU系统常用的5V供电，由于压差较大，因此采用两级LDO降压，以防止压差过大造成的发热问题

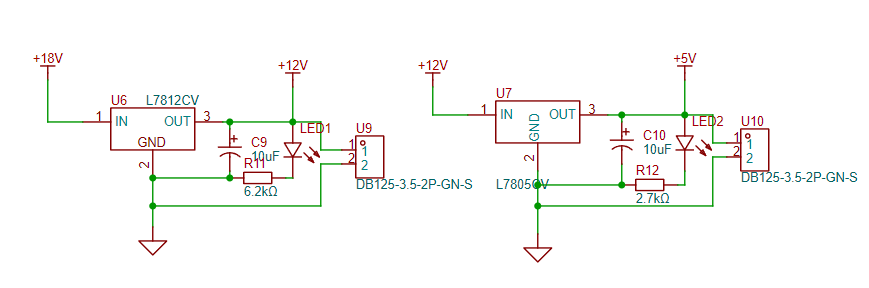


图 5 两级LDO降压

随后5V系统输入控制板，经过TLV76733单元进行欠压保护，过温/过流保护，并且考虑上位由整流桥整流输入可能具有较大的涌流采用一颗TVS管防止涌流输入，为MCU系统提供稳定的电平。

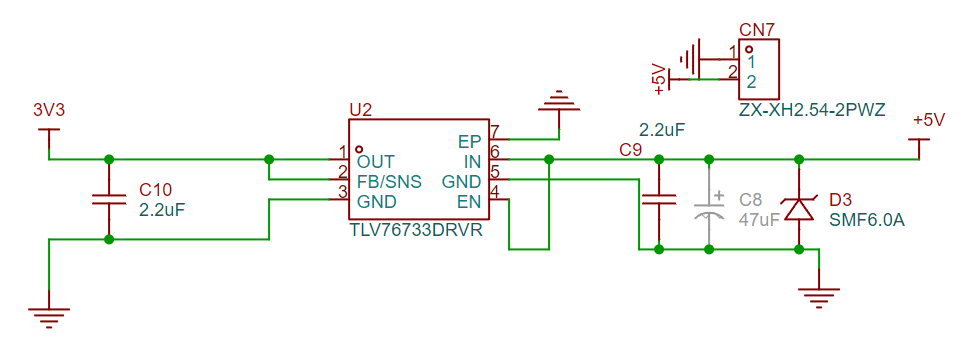


图 6 MCU供电

### 复位与时钟

采用25MHz无源晶振，拨码开关用于配置BOOT模式

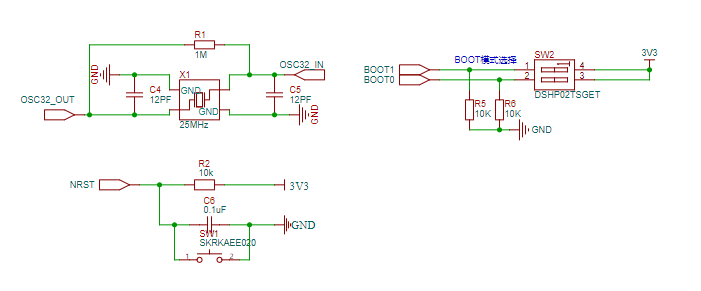
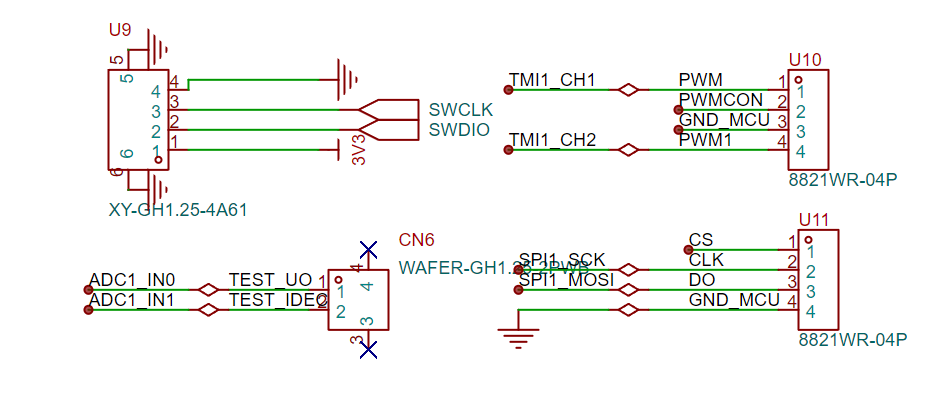


图 7 复位与时钟电路

### 外部接口

该部分和电源板的接口一一对应，使用XH和GH系列的冷压端子进行压接接线，并灌入UV胶增加稳定性，接口是ADC采样以及通讯可靠性的前提。



### MCU系统设计

将上述的电源输入进来，以及时钟.复位电路，连接上外部接口即可完成MCU的系统设计。



图 8 MCU系统设计

## PCB-Layout

### 电源板布局

电源板Boost部分采用U型布局，栅极驱动尽量靠近SW节点，并且SW节点回路尽量短，数字部分注意和功率部分分开布局，减小功率部分的各种串扰，功率部分注意表面开窗并走有厚锡，以增大载流能力以及更好的散热效果。

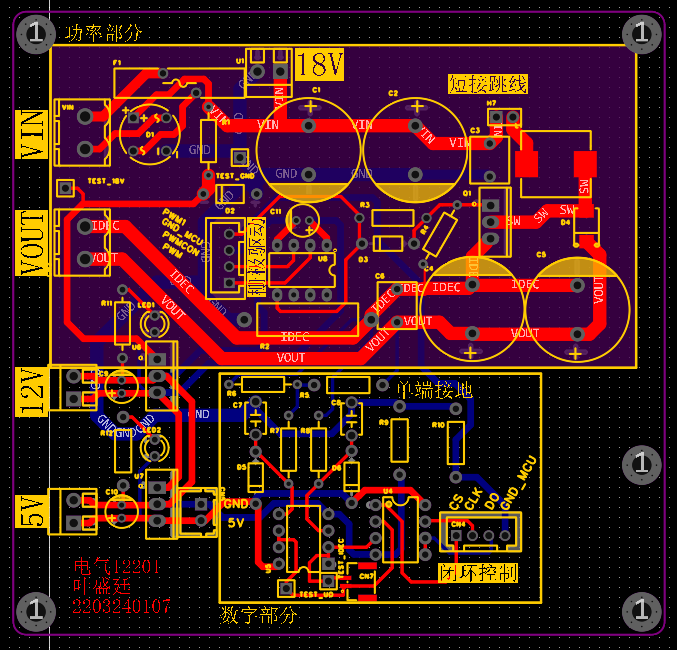


图 9 电源板布局

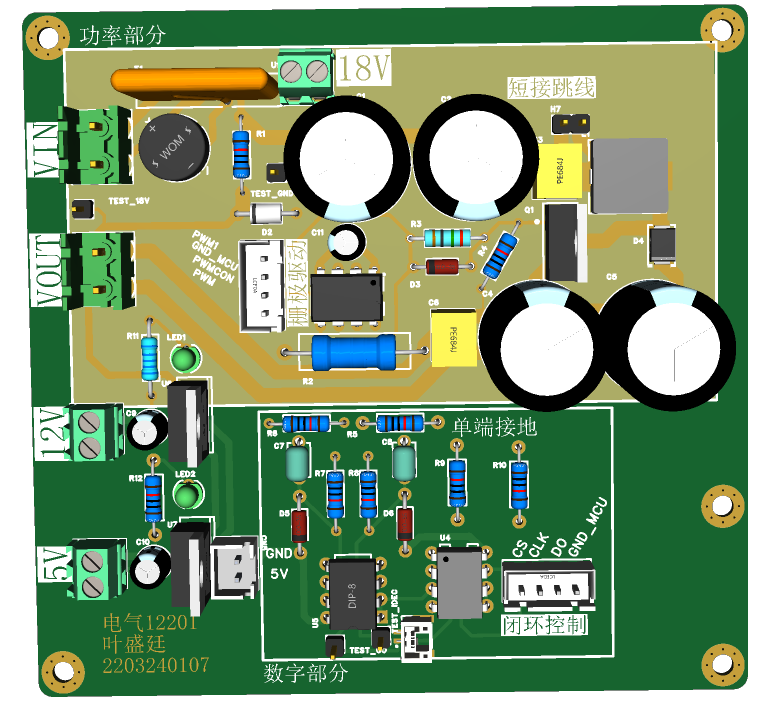


图 10 仿真图

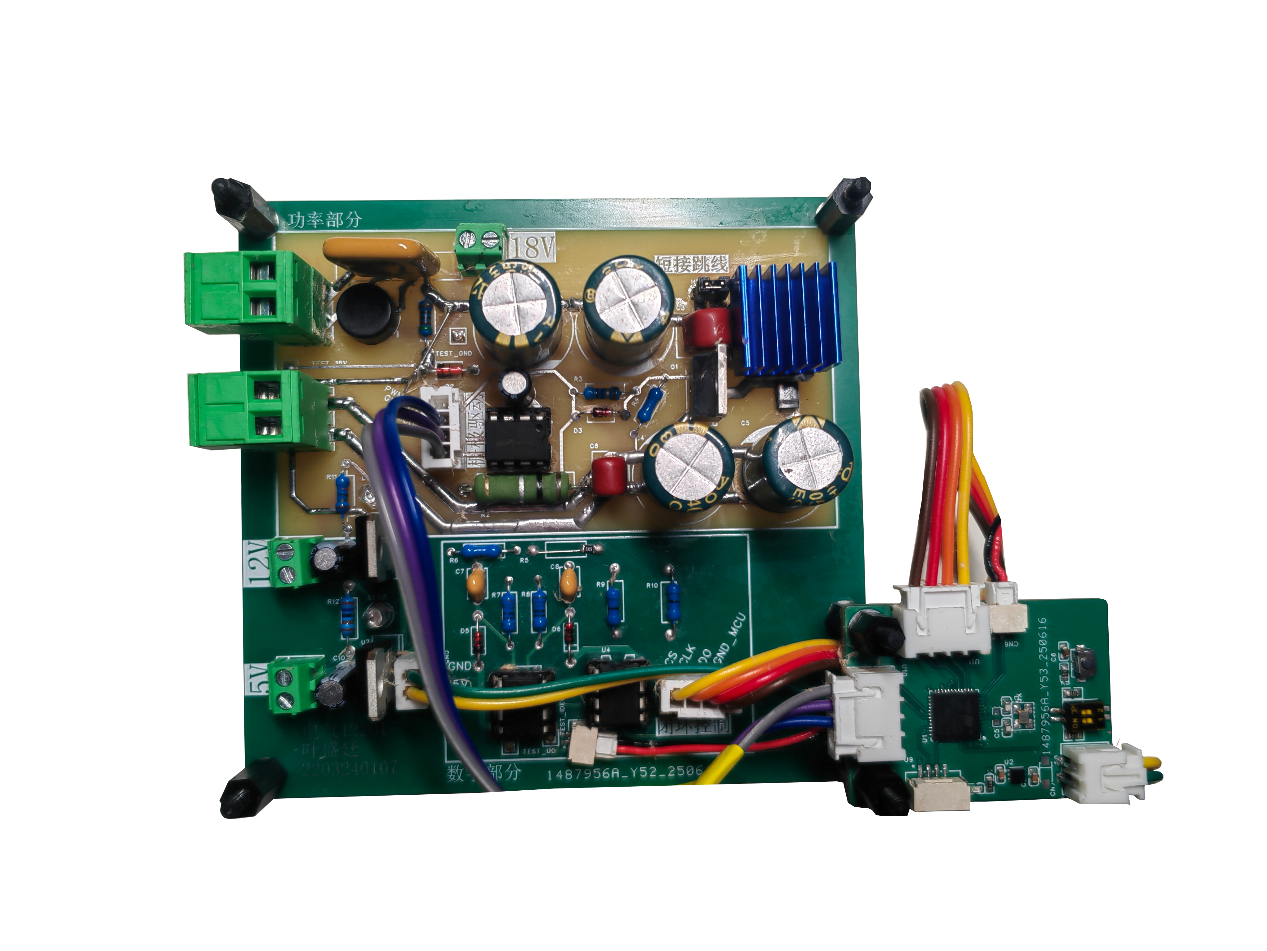


图 11 实物图

### 控制板布局

|  |  |
| --- | --- |
| 图 12 正面布局 | 图 13 背面局部 |

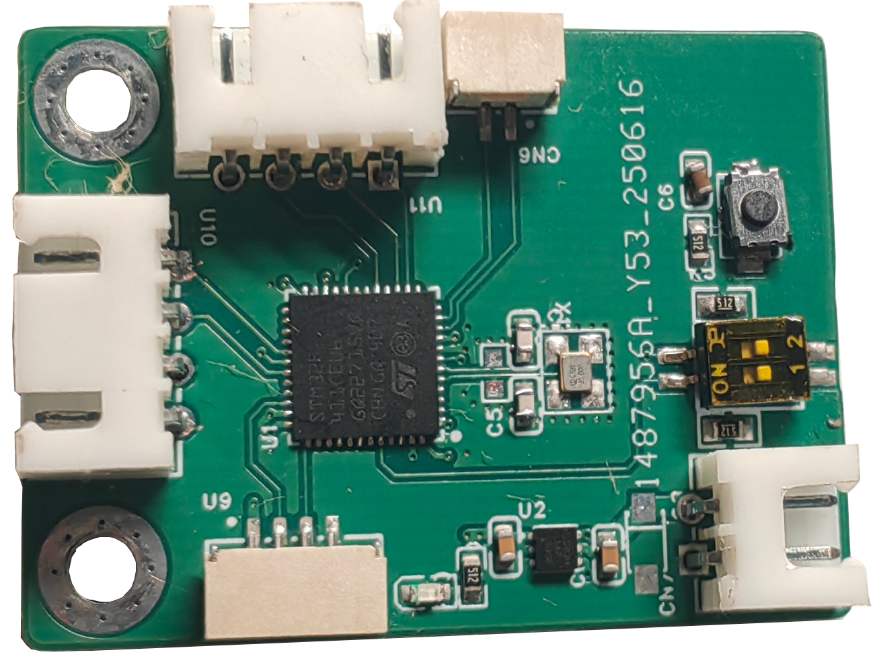


图 14 实物图

# 系统软件设计

## 仿真建模分析

根据题目参数搭建BOOST电路拓扑

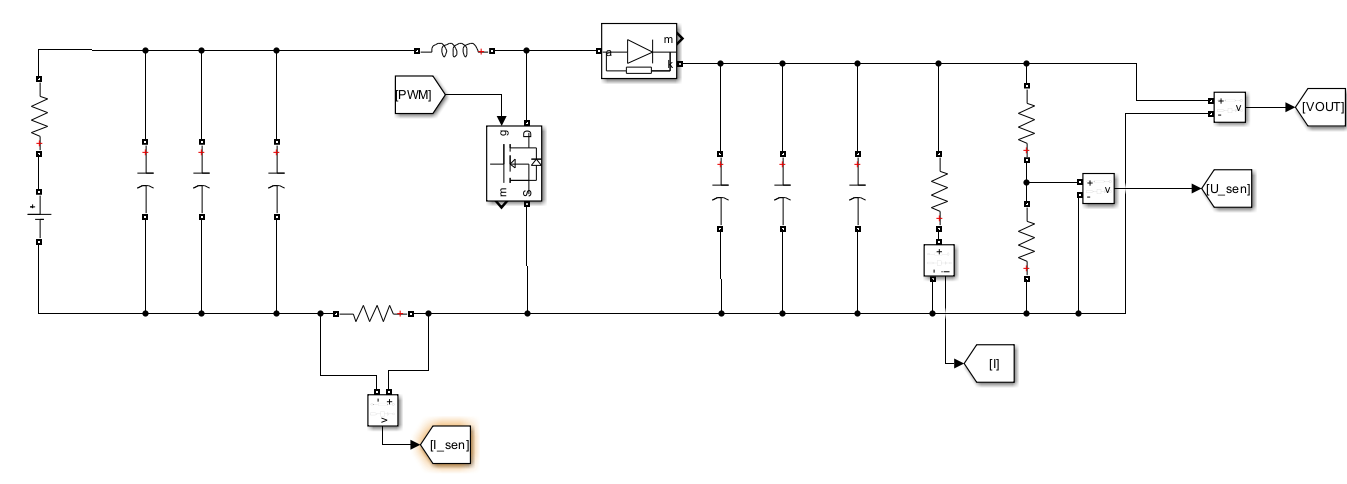


图 15 BOOST电路拓扑

分别设定开环控制和闭环控制进行不同的分析:

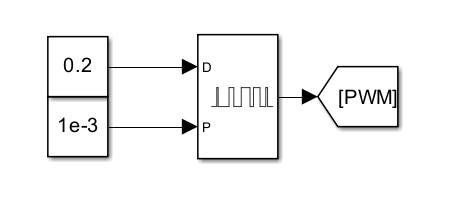


图 16 开环控制

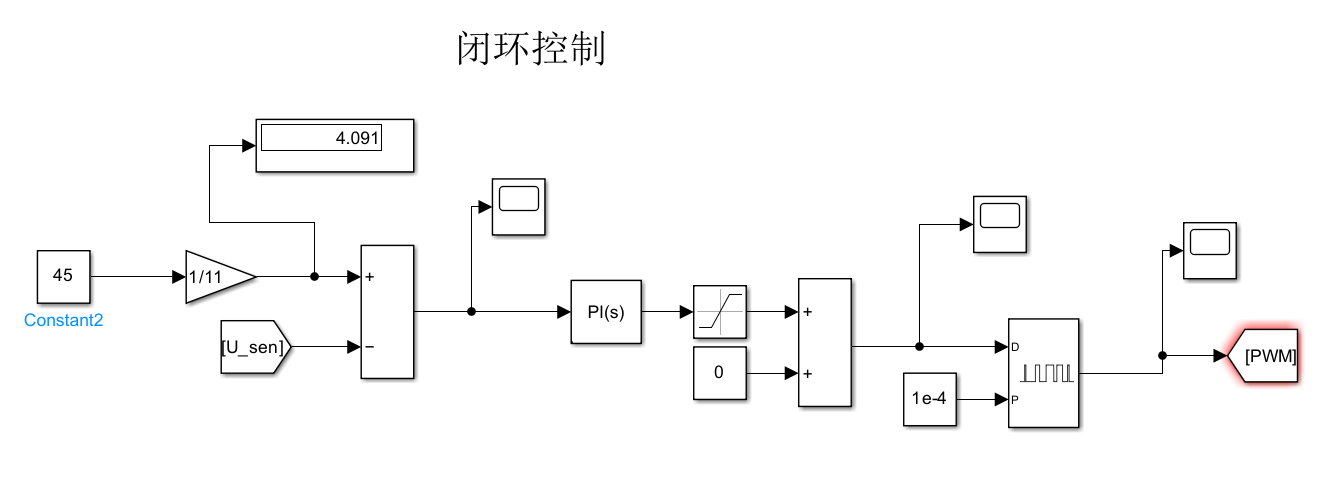


图 17 闭环控制

并设立有各种监测模块用于分析

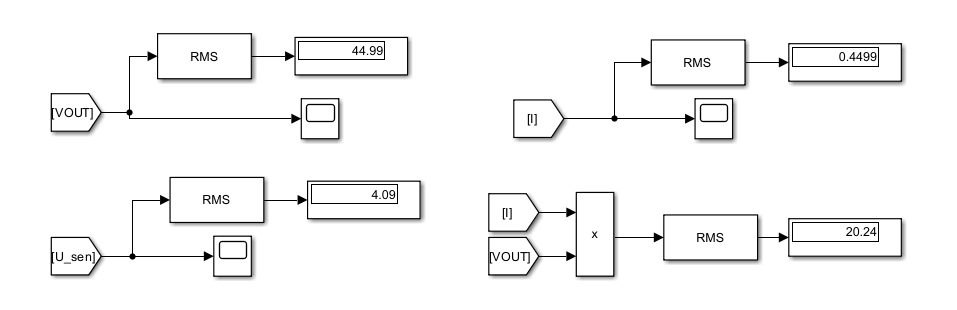


图 18 监测模块

### 开环分析

开分分析，我们的预期电压是36V，输入电压为18V，则根据稳态电压计算公式



则计算的到占空比应该为50%

当考虑负载的影响时



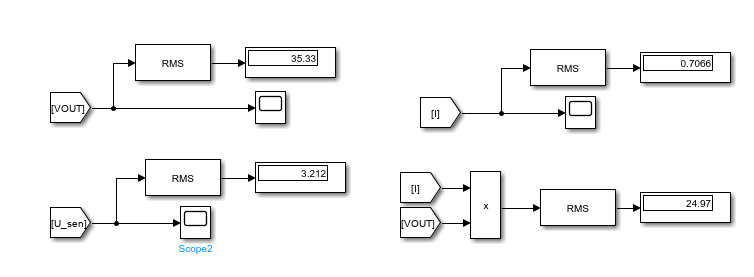


图 19 开环结果

### 闭环分析

通过调整PI参数，使偏差迅速趋于0

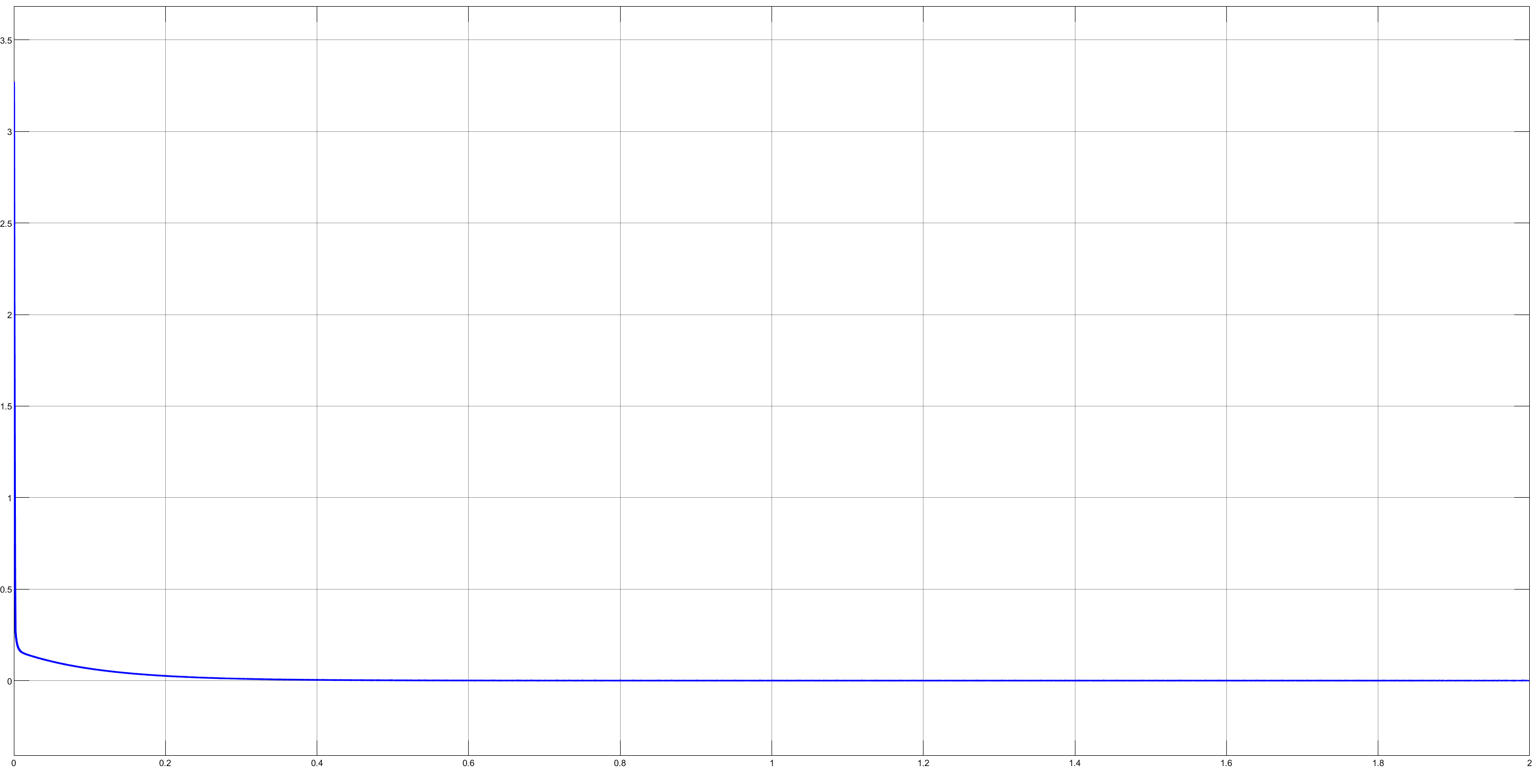


图 20 ERROR数值监测

这使得我们的输出电压迅速达到我们给定的设定值:

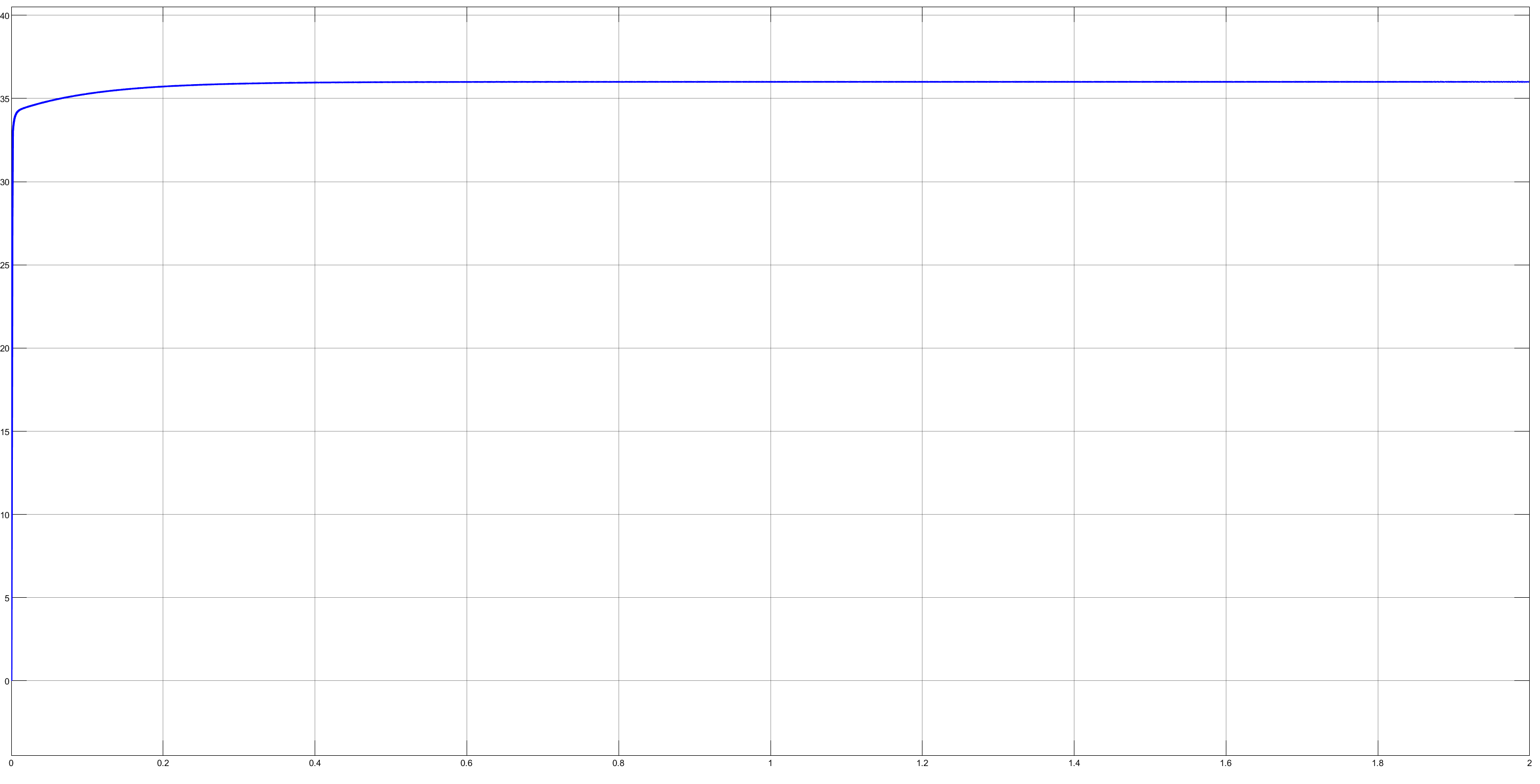


图 21 输出电压波形

## 开环特性分析

硬件设计完毕后，首先需要在开环条件下测定物理量的关系，用于拟合出ADC采样电压，占空比，以及输出电压的数据关系，随后拟合的结果可以用于闭环控制。

在开环条件下，100Ohm额定负载，100kHzPWM脉冲情况下测得如下数据用于拟合分析

表格 2 开环参数测定

|  |  |  |
| --- | --- | --- |
| Duty (%) | UO (V) | VADC (V) |
| 35% | 45.000 | 3.267 |
| 40% | 40.088 | 2.896 |
| 45% | 36.080 | 2.596 |
| 50% | 32.789 | 2.352 |
| 55% | 30.030 | 2.149 |
| 60% | 27.691 | 1.976 |
| 65% | 25.679 | 1.829 |
| 70% | 23.936 | 1.701 |

根据该开环参数可以得到拟合曲线，我们需要的到ADC电压和输出电压的关系用于设定目标值，我们还需要得到ADC电压和PWM波的关系，用于设定基础PWM。

为了得到比较准确的数据我们建立了ADC电压模型，使用三次多项式尝试拟合

|  |  |
| --- | --- |
|  |  |

非线性拟合关系(三次多项式)



由于高次项误差极小，优化为一次最小二乘拟合:



验证RMSE数值，其中线性拟合的RMSE: 0.0029 V，多项式拟合的RMSE: 0.0003 V，为了减小开销，并且线性拟合的RMSE值也不大，因此采用线性拟合的方式。

同样根据上述经验建立PWM和VADC的拟合关系，以提供基础占空比

|  |  |
| --- | --- |
|  |  |





其中最小而成得到的RMSE: 0.0907 V，三次多项式的RMSE: 0.0021 V，此时应当采用三次拟合以减小误差。根据该拟合关系球的逆拟合关系，通过VADC给出Duty



计算得到RMSE: 0.0005，在合理的精度范围以内。

综上所述得到如下关系



|  |
| --- |
| 计算代码 |
| /\*\*   \* @brief 电压拟合：根据vadc\_target目标电压返回基础占空比   \*/  static float BaseDuty\_FromFit(float vadc\_target)  {    // 方法一：三次多项式拟合（更精确）    float v2 = vadc\_target \* vadc\_target;    float v3 = v2 \* vadc\_target;    return 100.0f \* (-0.035971f \* v3 + 0.355110f \* v2 - 1.299556f \* vadc\_target + 2.059551f);    // // 方法二：线性拟合（更高性能）    // return 100.0f \* (-0.221493f \* vadc\_target + 1.044567f);  }  /\*\*   \* @brief 将输出电压转换为 ADC 目标电压（线性拟合）   \*/  float calculate\_vadc(float v\_target)  {    return 0.0743f \* v\_target - 0.0802f;  } |

## 闭环算法设计

闭环控制的软件主要由如下部分构成0. 系统初始化1. 双路ADC采样 2. DMA循环传输 3. TIM1产生PWM 4. TIM2产生特定频率的回调用于计算和控制频率 4. ADC数据滤波5. PI闭环控制算法 。

软件平台选择Cmake构建工程， 工具链选择GCC-ARM，IDE使用VSC，debug 基于SWD接口调用ARM-GDB。

### 系统初始化

系统初始化主要包含如下部分 1. 全局debug变量声明，各种句柄声明 2. 各种BSP初始化 3. 等待ADC数据稳定 4. 滤波器初始化

### 采样滤波以及控制

该部分代码主要包含ADC采样，滤波以及定时器回调闭环控制，软件主要流程如下图所示:

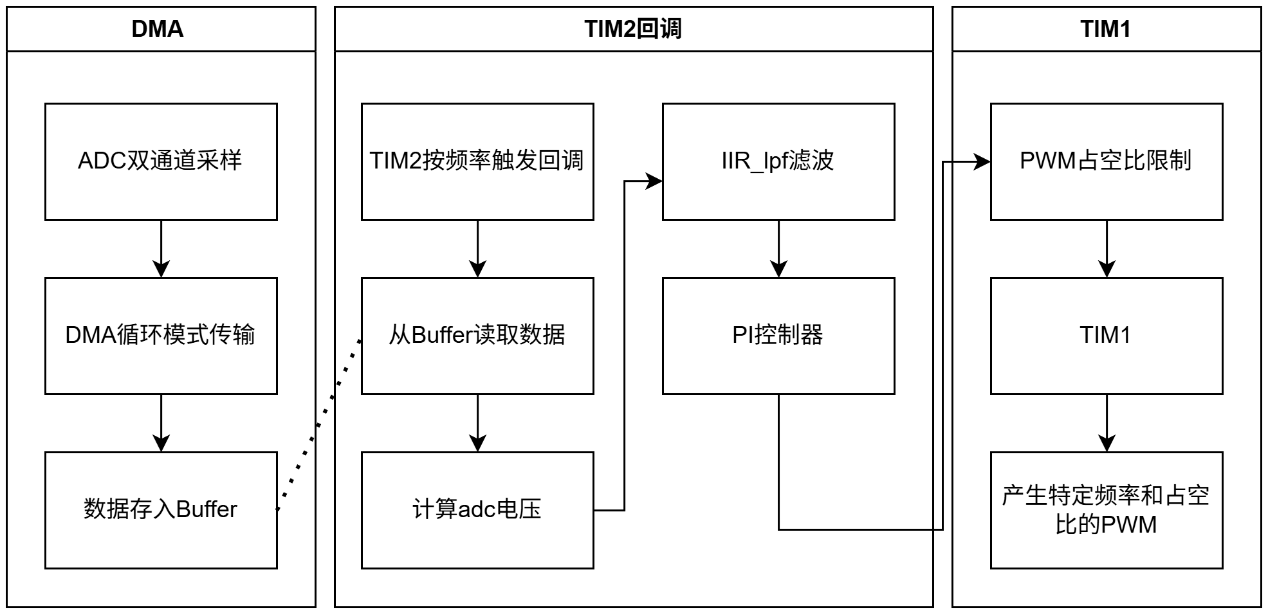


图 22 采样以及控制流程

ADC部分采用双通道，因此使用Scan Conversion Mode，由于开关频率在100kHz所以这里采样频率不用过高，从PCLK2时钟6分频即可。使能Continuous模式，一面再回调中重复触发中断，使能DMA Continuous并搭配DMA的Circular模式连续搬运数据，实现ADC的数据高效传输。

表格 3 ADC详细配置

|  |  |
| --- | --- |
| 配置项 | 值 |
| Clock Prescaler | PCLK2 divided by 6 |
| Resolution | 12 bits (15 ADC Clock cycles) |
| Data Alignment | Right alignment |
| Scan Conversion Mode | Enabled |
| Continuous Conversion Mode | Enabled |
| Discontinuous Conversion Mode | Disabled |
| DMA Continuous Requests | Enabled |
| End Of Conversion Selection | EOC flag at the end of single channel conversion |

完成采样后通过设定TIM2定时器，稳定触发回调函数从内存中读取ADC 存在Buffer中的数据，紧跟着计算ADC的电压，然后输入IIR低通滤波器，抑制高频噪声、平滑信号，然后输入PI控制器用于闭环控制。

|  |
| --- |
| 回调计算流程 |
| void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  {    if (htim->Instance == TIM2)    {      // ADC 原始采样      float v\_raw = adc\_buffer[0] \* 3.3f / 4096.0f;      float i\_raw = adc\_buffer[1] \* 3.3f / 4096.0f;      // 滤波处理      v\_adc = IIR\_LPF\_Update(&lpf\_vadc, v\_raw);      current = IIR\_LPF\_Update(&lpf\_current, i\_raw);      // PID 控制输出      delta = PI\_Update(&voltage\_pi, vadc\_target, v\_adc, CONTROL\_PERIOD\_S);      duty = base\_duty + delta;      // 限制输出占空比      if (duty < 0.0f)        duty = 0.0f;      if (duty > 100.0f)        duty = 100.0f;      // 应用 PWM      Set\_PWM\_Duty(&htim1, TIM\_CHANNEL\_1, duty);    }  } |

闭环控制器采用PI控制，经过积分限幅后输入TIM1模块，产生控制所需要的PWM频率和占空比。

|  |
| --- |
| PI控制器流程 |
| float PI\_Update(PI\_Controller\_t \*pi, float target, float feedback, float dt)  {      //UPdate 使用全局变量      pi\_error = -(target - feedback);      // 更新积分项      pi->integral += pi\_error \* dt;      // 防止积分项过大，限幅在 [-output\_limit/ki, +output\_limit/ki]      if (pi->ki > 0.0f)      {          float int\_max = pi->output\_limit / pi->ki;          if (pi->integral > int\_max)              pi->integral = int\_max;          else if (pi->integral < -int\_max)              pi->integral = -int\_max;      }      // PI 计算      float output = pi->kp \* pi\_error + pi->ki \* pi->integral;      // 输出限幅      if (output > pi->output\_limit)          output = pi->output\_limit;      else if (output < -pi->output\_limit)          output = -pi->output\_limit;      return output;  } |

详细的控制算法和代码见附录

# 系统调试与测试

## 调试

调试平台为了保证实时性和最简化处理，通过SWD接口，以及从全局变量空间直接读取内存数据的方式进行调试分析，PI参数的标定，搭配CubeMonitor等实时检测数据变化和响应。

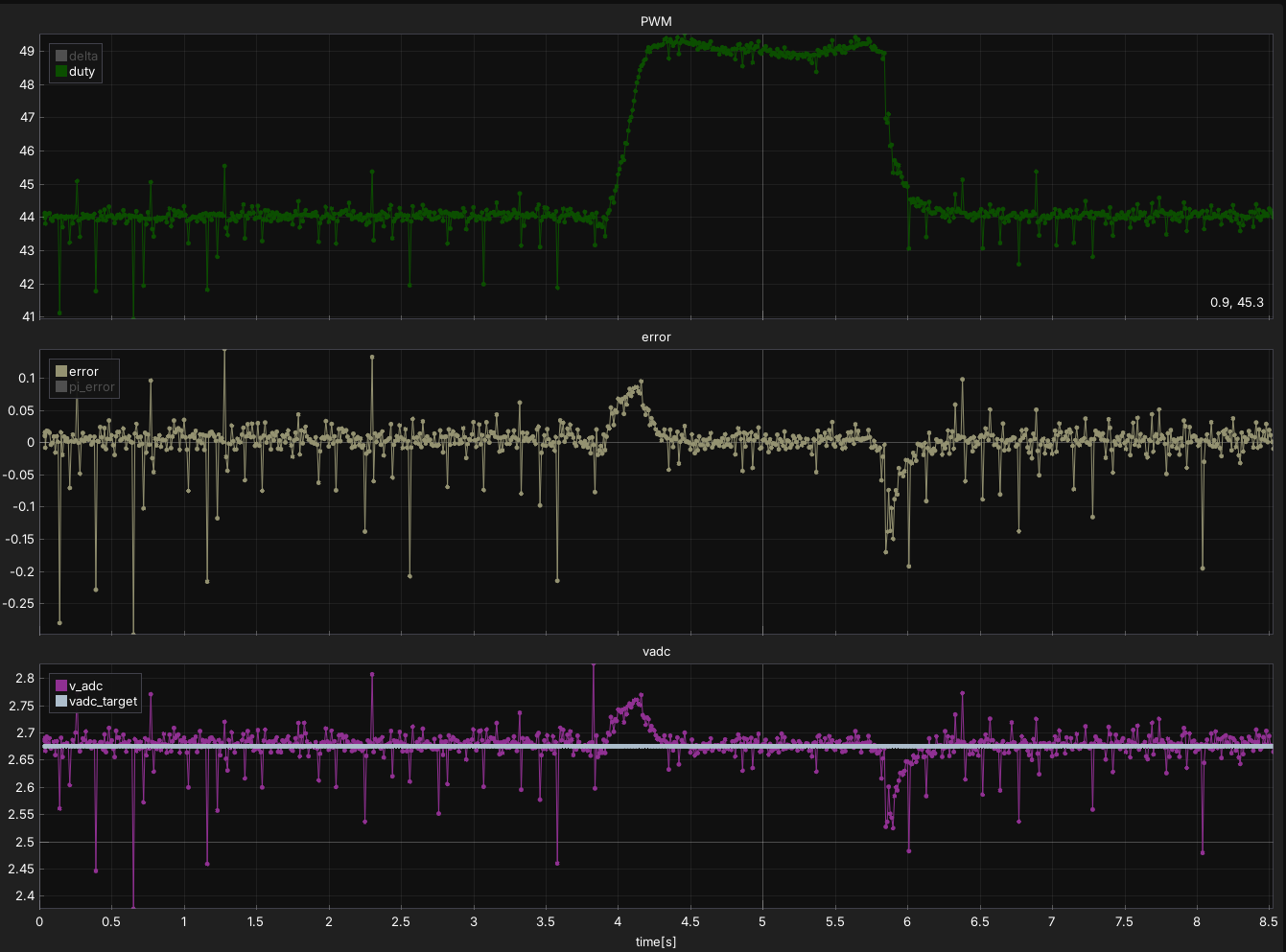


图 23 MCU Viewer波形

根据MCUviewer波形调整PI参数

表格 4 MCU系统常见Debug方式

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 方法 | 实时性 | 输出速度 | 使用复杂度 | 占用资源 | 优点 | 缺点 |
| printf 串口重定向 | 中等 | 慢(几百B/s) | 低 | 占用UART资源 | 易用、格式化输出、通用 | 阻塞性强、影响实时性、速度慢 |
| SWD+变量暂停 观察 | 低(暂停) | 低 | 中 | 占用调试器和 IDE | 支持断点、单步 调试,适合静态 分析 | 程序必须暂停,不能 动态跟踪变量 |
| SWD + Live Watch | 中高 | 中 (~1-10Hz) | 中 | 仅占用SWD | 程序运行时查看 变量,适合中低 速控制调试 | 不适合高速信号、变 量受限于编译优化 |
| SEGGER RTT | 高 | 快(>100KB/s) | 中 | J-Link + SRAM | 非阻塞、无需串 口、可视化好、 速度极快 | 需 SEGGER J- Link、嵌入RTT 库,配置略复杂 |
| SWO | 高 | 快 (10~100KB/s) | 中 | 占用SWO引 脚 | 低资源、高速、 可视化输出变 量、兼容STM Studio 等工具 | 配置复杂、需MCU 和调试器支持SWO |

这里由于回调频率并不高，而且无需RTOS等队列复杂debug，因此采用SWD结合内存LiveWatch是本次任务的最佳调试方案。

书写debug规则:

|  |
| --- |
| Debug规则(Launch.json) |
| {      "version": "0.2.0",      "configurations": [          {              "name": "Build & Debug Microcontroller - ST-Link",              "cwd": "${workspaceFolder}",              "type": "cortex-debug",              "executable": "${command:cmake.launchTargetPath}",              // Let CMake extension decide executable: "${command:cmake.launchTargetPath}"              // Or fixed file path: "${workspaceFolder}/path/to/filename.elf"              "request": "launch",              "servertype": "stlink",              "device": "STM32F411CEUx", //MCU used              "interface": "swd",              "serialNumber": "", //Set ST-Link ID if you use multiple at the same time              "runToEntryPoint": "main",              "svdFile": "${config:STM32VSCodeExtension.cubeCLT.path}/STMicroelectronics\_CMSIS\_SVD/STM32F411.svd",              "v1": false, //Change it depending on ST Link version              "serverpath": "${config:STM32VSCodeExtension.cubeCLT.path}/STLink-gdb-server/bin/ST-LINK\_gdbserver",              "stm32cubeprogrammer": "${config:STM32VSCodeExtension.cubeCLT.path}/STM32CubeProgrammer/bin",              "stlinkPath": "${config:STM32VSCodeExtension.cubeCLT.path}/STLink-gdb-server/bin/ST-LINK\_gdbserver",              "armToolchainPath": "${config:STM32VSCodeExtension.cubeCLT.path}/GNU-tools-for-STM32/bin",              "gdbPath": "${config:STM32VSCodeExtension.cubeCLT.path}/GNU-tools-for-STM32/bin/arm-none-eabi-gdb",              "serverArgs": [                  "-m",                  "0",              ],              "liveWatch": {                  "enabled": true,                  "samplesPerSecond": 4              }              //"preLaunchTask": "Build + Flash"              /\* If you use external loader, add additional arguments \*/              //"serverArgs": ["--extload", "path/to/ext/loader.stldr"],          },          {              "name": "Attach to Microcontroller - ST-Link",              "cwd": "${workspaceFolder}",              "type": "cortex-debug",              "executable": "${command:cmake.launchTargetPath}",              // Let CMake extension decide executable: "${command:cmake.launchTargetPath}"              // Or fixed file path: "${workspaceFolder}/path/to/filename.elf"              "request": "attach",              "servertype": "stlink",              "device": "STM32F411CEUx", //MCU used              "interface": "swd",              "serialNumber": "", //Set ST-Link ID if you use multiple at the same time              "runToEntryPoint": "main",              "svdFile": "${config:STM32VSCodeExtension.cubeCLT.path}/STMicroelectronics\_CMSIS\_SVD/STM32F411.svd",              "v1": false, //Change it depending on ST Link version              "serverpath": "${config:STM32VSCodeExtension.cubeCLT.path}/STLink-gdb-server/bin/ST-LINK\_gdbserver",              "stm32cubeprogrammer": "${config:STM32VSCodeExtension.cubeCLT.path}/STM32CubeProgrammer/bin",              "stlinkPath": "${config:STM32VSCodeExtension.cubeCLT.path}/STLink-gdb-server/bin/ST-LINK\_gdbserver",              "armToolchainPath": "${config:STM32VSCodeExtension.cubeCLT.path}/GNU-tools-for-STM32/bin",              "gdbPath": "${config:STM32VSCodeExtension.cubeCLT.path}/GNU-tools-for-STM32/bin/arm-none-eabi-gdb",              "serverArgs": [                  "-m",                  "0",              ],              /\* If you use external loader, add additional arguments \*/              //"serverArgs": ["--extload", "path/to/ext/loader.stldr"],          }      ]  } |

绘制调试数据流:

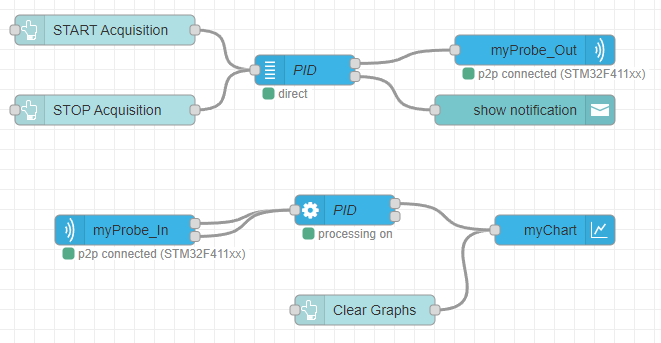


图 24 Debug数据流

建立完数据流之可以根据Chart进行误差分析

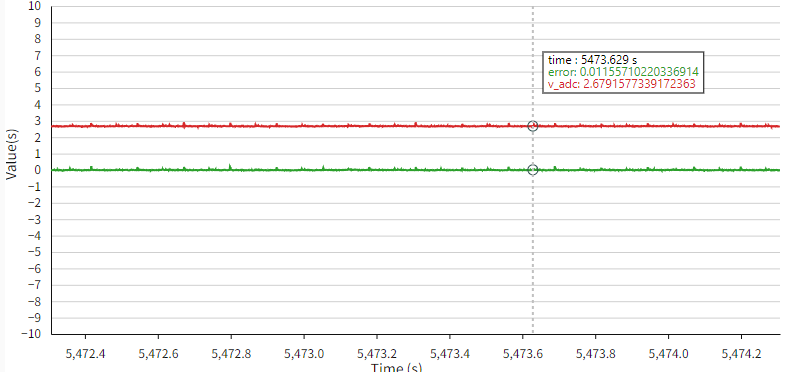


图 25 误差分析

开启MCUgragh 可视化调整PI参数

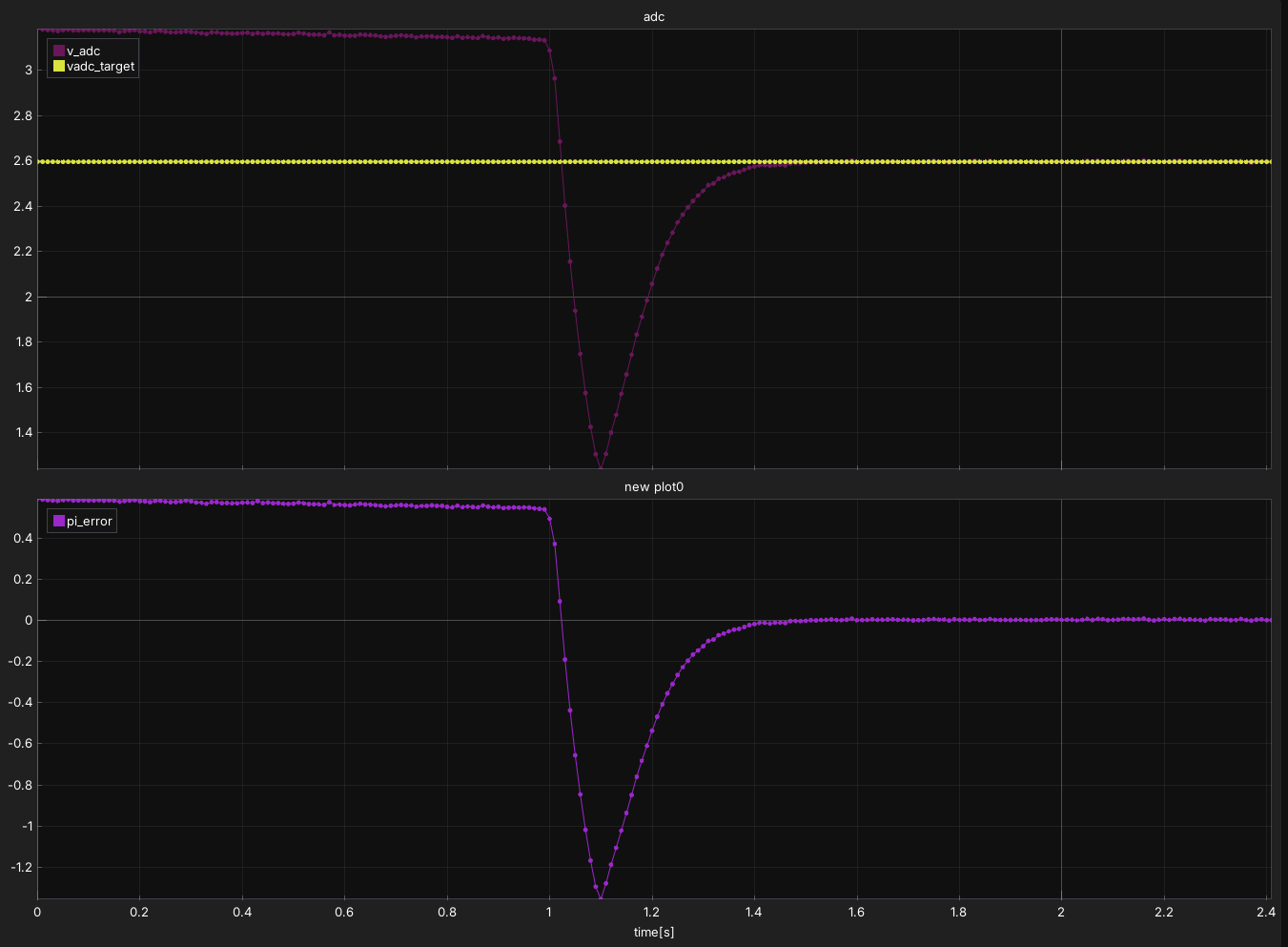


图 26 Error偏差图

## 测试

表格 5 课程设计设计要求

|  |
| --- |
| 设计要求 |
| 输出电压*U*O：35-40VDC（按36VDC额定电压设计）。 |
| 最大输出电流*I*Omax：2A（实验室提供100Ω滑动变阻器）。 |
| 具有电路保护功能（通过AD模块采样软件保护）。 |
| *I*O从0变到1A时，输出电压*U*O 波动不超过±10%。 |
| 电源效率≥80%（*U*2=18VAC,*U*O=36VDC, *I*O=1A）。 |

开关电源的测试指标如下表所示：

表格 6 电源常见测试项目

|  |  |
| --- | --- |
| 测试类别 | 测试指标 |
| 输入输出参数 | 输入电压范围 (VBBIN) |
| 输出电压范围 (VOUT) |
| 输出电压精度(VOUT\_ACC) |
| 最大输出电流(IOMAX) |
| 动态性能 | 负载调整率(AV\_BOOST\_LOAD) |
| 电压调整率 (AV\_BUS) |
| Phase点Jitter (jitter) |
| 动态响应时间(T\_LOAD) |
| 保护机制 | 过电压保护(VBUSOVP) |
| 欠压保护(V\_BOOSTUV) |
| 热保护(T\_SHUTDOWN) |
| 电流限制(I\_LIM) |
| 效率与损耗 | 总功率损耗(P\_tot) |
| MOSFET导通损耗(R\_DS(ON)) |
| 输出电压纹波(Vpp) |
| 其他关键指标 | 启动时间(TSTART) |
| 静态电流(ISHUT) |
| 栅极驱动峰值电流(IGC1 Peak) |

本次主要测试动态指标，效率，纹波以及温度

### 动态带载

负载跳变0.2A~1.2A,电压超调4.75V，恢复时间180ms

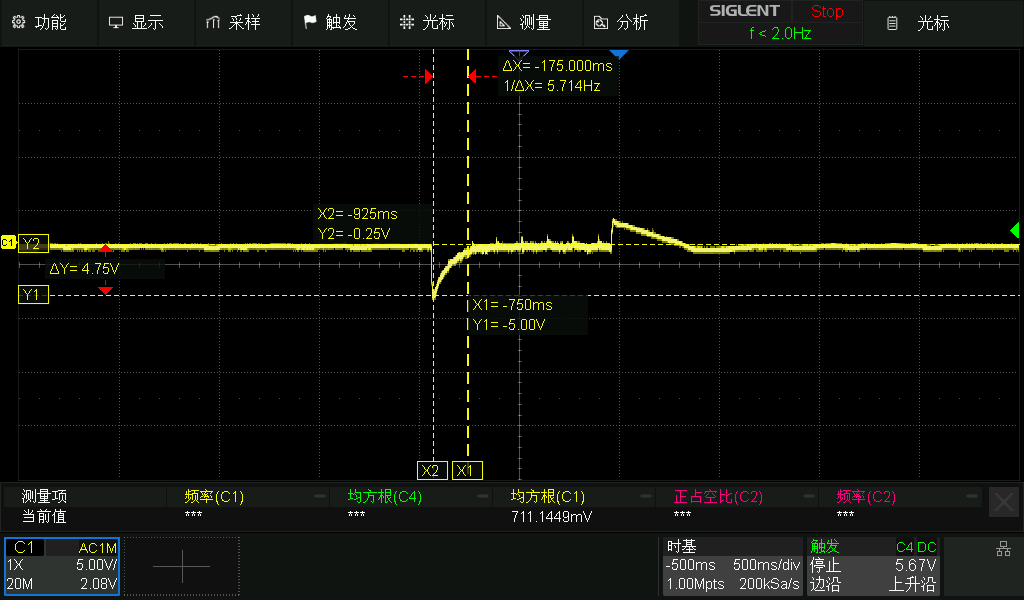


图 27 动态带载测试

### 效率分析

|  |  |
| --- | --- |
| Pi | Po |
| 8.657 | 7.58 |
| 12.70 | 11.28 |
| 16.7 | 14.93 |
| 20.5 | 18.5 |
| 24.5 | 22.0 |
| 28.3 | 25.5 |
| 32.2 | 28.9 |
| 36.0 | 32.23 |
| 39.9 | 35.5 |

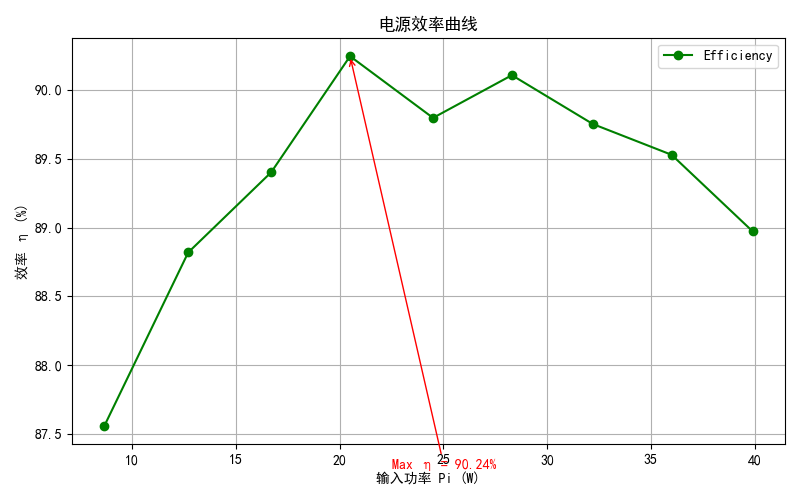


图 28 电源效率分析

效率区间为87.5%~90%

### 纹波

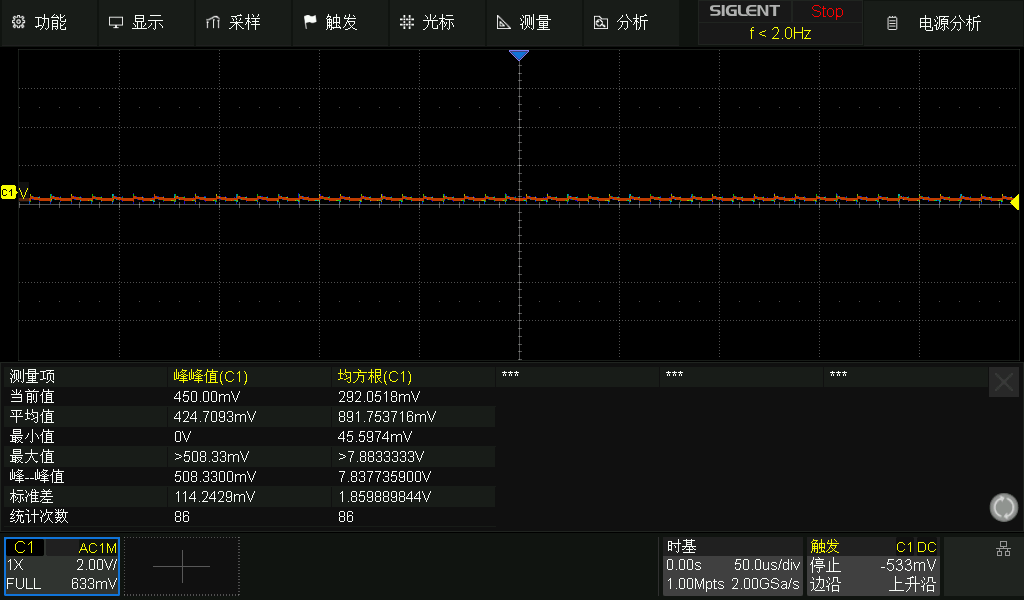


图 29 纹波分析

### 温度分析

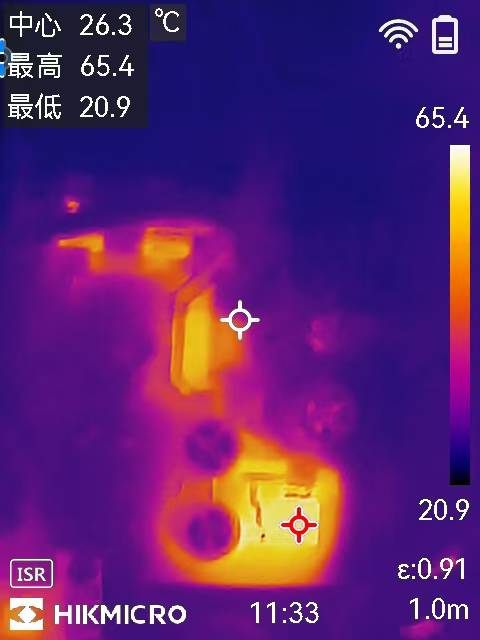


图 30 温度分析

### EMI测试

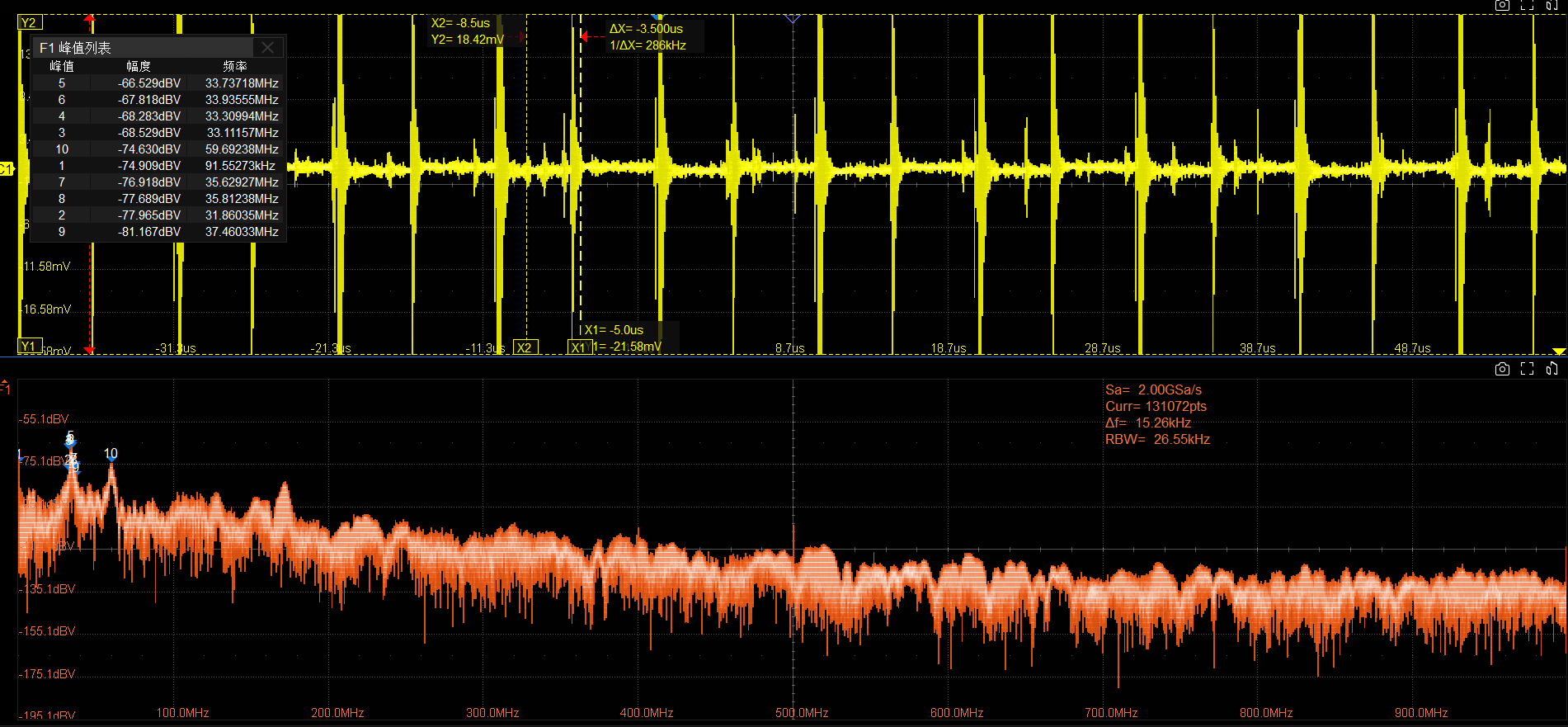


图 31 全频段EMI

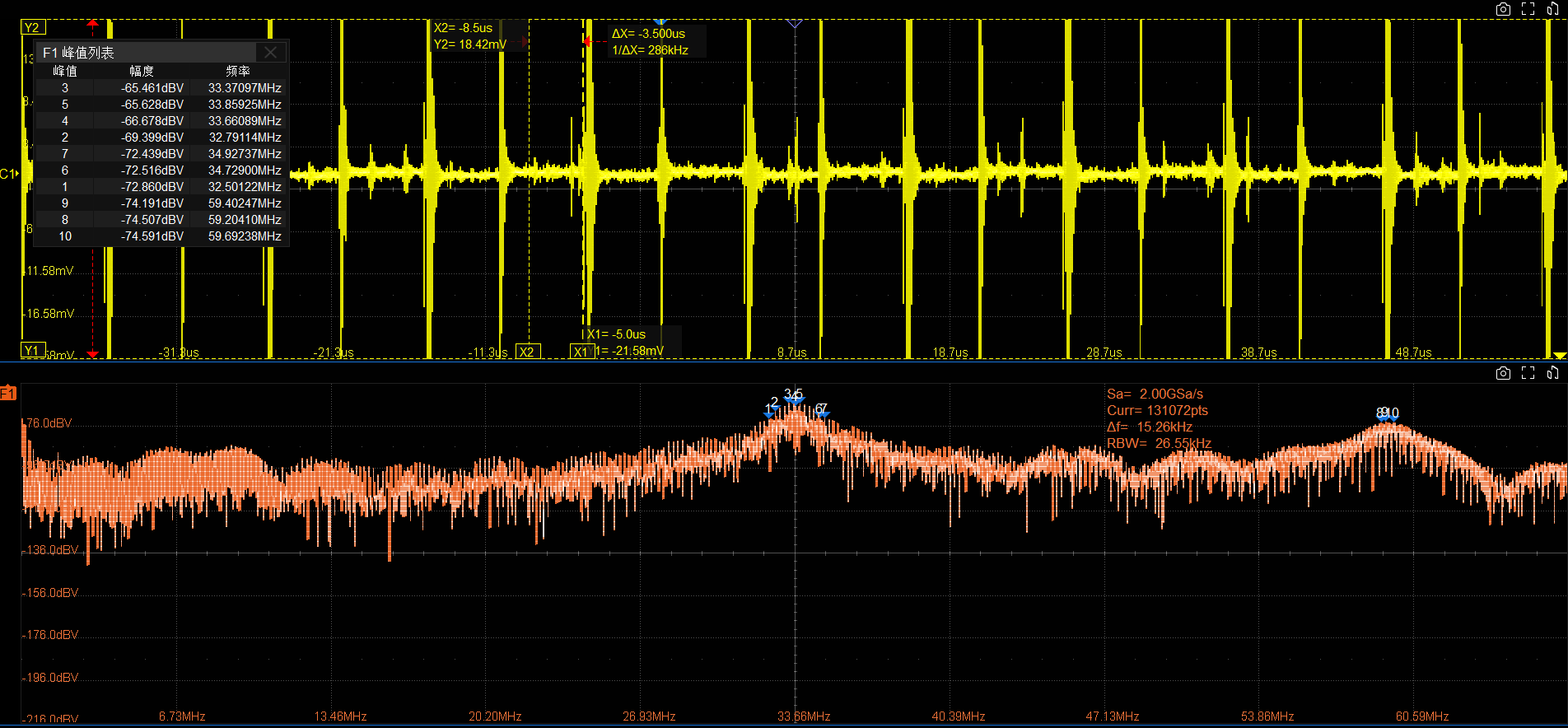


图 32 峰值

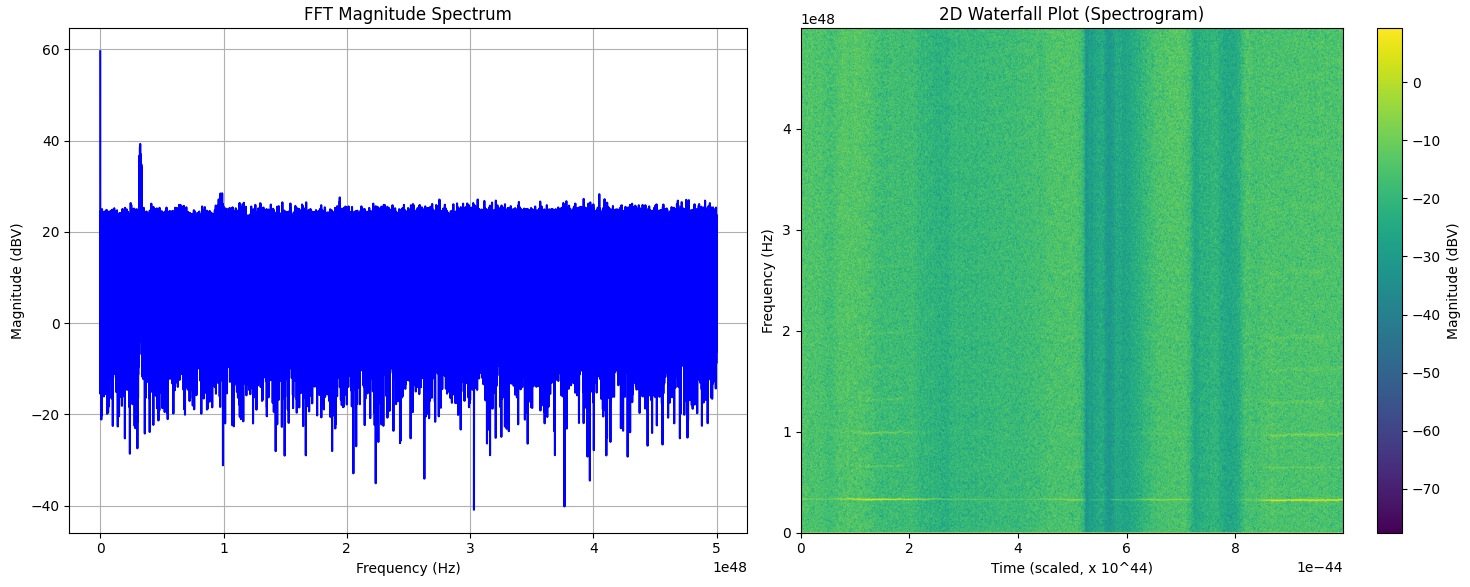


图 33 瀑布图

# 设计总结

此次设计，即使ERROR偏差非常接近0，让ADC的采样值稳定不变，但是输出电压依然会随着负载的变化而产生波动，这是因为输出电压为40V的高压，因此ADC需要使用FB分压电阻(14变比)到ADC，次数输出电压的波动在FB上会变得非常小，ADC的精度不足以跟踪。尝试引入电流前馈补偿，但是收效仍然甚微，因此后续可以考虑提高ADC精度，调整运放参数等优化电路输出电压精度。

# 参考文献

无参考，自行设计

## 附录

|  |
| --- |
| Main.c |
| /\* USER CODE BEGIN Header \*/  /\*\*   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \* @file           : main.c   \* @brief          : Main program body   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \* @attention   \*   \* Copyright (c) 2025 STMicroelectronics.   \* All rights reserved.   \*   \* This software is licensed under terms that can be found in the LICENSE file   \* in the root directory of this software component.   \* If no LICENSE file comes with this software, it is provided AS-IS.   \*   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \*/  /\* USER CODE END Header \*/  /\* Includes ------------------------------------------------------------------\*/  #include "main.h"  #include "adc.h"  #include "dma.h"  #include "tim.h"  #include "gpio.h"  /\* Private includes ----------------------------------------------------------\*/  /\* USER CODE BEGIN Includes \*/  /\* USER CODE END Includes \*/  /\* Private typedef -----------------------------------------------------------\*/  /\* USER CODE BEGIN PTD \*/  /\* USER CODE END PTD \*/  /\* Private define ------------------------------------------------------------\*/  /\* USER CODE BEGIN PD \*/  #define CONTROL\_PERIOD\_S (0.001f) // 控制周期 1 ms  /\* USER CODE END PD \*/  /\* Private macro -------------------------------------------------------------\*/  /\* USER CODE BEGIN PM \*/  /\* USER CODE END PM \*/  /\* Private variables ---------------------------------------------------------\*/  /\* USER CODE BEGIN PV \*/  \_\_attribute\_\_((aligned(4))) uint16\_t adc\_buffer[2];  float v\_adc = 0.0f;  float current = 0.0f;  float v\_target = 36.00f;  float vadc\_target = 0.0f;  float delta = 0.0f;  float base\_duty = 0.0f;  float duty = 0.0f;  PI\_Controller\_t voltage\_pi;  PIController v\_pi;  /\* 定义滤波器句柄 \*/  static IIR\_LPF\_HandleTypeDef lpf\_vadc;  static IIR\_LPF\_HandleTypeDef lpf\_current;  /\* USER CODE END PV \*/  /\* Private function prototypes -----------------------------------------------\*/  void SystemClock\_Config(void);  /\* USER CODE BEGIN PFP \*/  /\* USER CODE END PFP \*/  /\* Private user code ---------------------------------------------------------\*/  /\* USER CODE BEGIN 0 \*/  /\*\*   \* @brief 电压拟合：根据vadc\_target目标电压返回基础占空比   \*/  static float BaseDuty\_FromFit(float vadc\_target)  {    // 方法一：三次多项式拟合（更精确）    float v2 = vadc\_target \* vadc\_target;    float v3 = v2 \* vadc\_target;    return 100.0f \* (-0.035971f \* v3 + 0.355110f \* v2 - 1.299556f \* vadc\_target + 2.059551f);    // // 方法二：线性拟合（更高性能）    // return 100.0f \* (-0.221493f \* vadc\_target + 1.044567f);  }  /\*\*   \* @brief 将输出电压转换为 ADC 目标电压（线性拟合）   \*/  float calculate\_vadc(float v\_target)  {    // 直接拟合@100Ohm Load    // return 0.0743f \* v\_target - 0.0802f;    // 电流补偿    return 0.0743f \* v\_target - 1.0553 \* current;  }  void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)  {    if (htim->Instance == TIM2)    {      // ADC 原始采样      float v\_raw = adc\_buffer[0] \* 3.3f / 4096.0f;      float i\_raw = adc\_buffer[1] \* 3.3f / 4096.0f;      // 滤波处理      v\_adc = IIR\_LPF\_Update(&lpf\_vadc, v\_raw);      current = IIR\_LPF\_Update(&lpf\_current, i\_raw);      // 不滤波      //  v\_adc = v\_raw;      //  current = i\_raw;      // PID 控制输出      // delta = PI\_Update(&voltage\_pi, vadc\_target, v\_adc, CONTROL\_PERIOD\_S);      delta = PIController\_Update(&v\_pi, vadc\_target, v\_adc);      duty = base\_duty + delta;      // 限制输出占空比      if (duty < 0.0f)        duty = 0.0f;      if (duty > 100.0f)        duty = 100.0f;      // 应用 PWM      Set\_PWM\_Duty(&htim1, TIM\_CHANNEL\_1, duty);    }  }  /\* USER CODE END 0 \*/  /\*\*    \* @brief  The application entry point.    \* @retval int    \*/  int main(void)  {    /\* USER CODE BEGIN 1 \*/    /\* USER CODE END 1 \*/    /\* MCU Configuration--------------------------------------------------------\*/    /\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/    HAL\_Init();    /\* USER CODE BEGIN Init \*/    /\* USER CODE END Init \*/    /\* Configure the system clock \*/    SystemClock\_Config();    /\* USER CODE BEGIN SysInit \*/    /\* USER CODE END SysInit \*/    /\* Initialize all configured peripherals \*/    MX\_GPIO\_Init();    MX\_DMA\_Init();    MX\_ADC1\_Init();    MX\_TIM1\_Init();    MX\_TIM2\_Init();    /\* USER CODE BEGIN 2 \*/    // PI\_Init(&voltage\_pi, 2.0f, 120.0f, 500.0f);    PIController\_Init(&v\_pi, 10.0f, 300.0f, 0.001f, -50.0f, 50.0f); // kp, ki, dt(1ms), min, max    vadc\_target = calculate\_vadc(v\_target);    base\_duty = BaseDuty\_FromFit(vadc\_target);    HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_1);    Set\_PWM\_Duty(&htim1, TIM\_CHANNEL\_1, 45); // 初始占空比    /\* 首次读取 ADC，获得初始值 \*/    HAL\_ADC\_Start\_DMA(&hadc1, (uint32\_t \*)adc\_buffer, 2);    HAL\_Delay(10); // 等待 DMA 首次采样完成    float v\_init = adc\_buffer[0] \* 3.3f / 4096.0f;    float curr\_init = adc\_buffer[1] \* 3.3f / 4096.0f;    /\* 初始化滤波器，α 推荐 0.6 \*/    IIR\_LPF\_Init(&lpf\_vadc, 0.6f, v\_init);    IIR\_LPF\_Init(&lpf\_current, 0.60f, curr\_init);    HAL\_TIM\_Base\_Start\_IT(&htim2); // 启动中断模式定时器    /\* USER CODE END 2 \*/    /\* Infinite loop \*/    /\* USER CODE BEGIN WHILE \*/    while (1)    {      /\* USER CODE END WHILE \*/      /\* USER CODE BEGIN 3 \*/    }    /\* USER CODE END 3 \*/  }  /\*\*    \* @brief System Clock Configuration    \* @retval None    \*/  void SystemClock\_Config(void)  {    RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};    RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};    /\*\* Configure the main internal regulator output voltage    \*/    \_\_HAL\_RCC\_PWR\_CLK\_ENABLE();    \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);    /\*\* Initializes the RCC Oscillators according to the specified parameters    \* in the RCC\_OscInitTypeDef structure.    \*/    RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;    RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;    RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;    RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;    RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;    RCC\_OscInitStruct.PLL.PLLM = 8;    RCC\_OscInitStruct.PLL.PLLN = 100;    RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;    RCC\_OscInitStruct.PLL.PLLQ = 4;    if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)    {      Error\_Handler();    }    /\*\* Initializes the CPU, AHB and APB buses clocks    \*/    RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK                                |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;    RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;    RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;    RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;    RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;    if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_3) != HAL\_OK)    {      Error\_Handler();    }  }  /\* USER CODE BEGIN 4 \*/  /\* USER CODE END 4 \*/  /\*\*    \* @brief  This function is executed in case of error occurrence.    \* @retval None    \*/  void Error\_Handler(void)  {    /\* USER CODE BEGIN Error\_Handler\_Debug \*/    \_\_disable\_irq();    while (1)    {    }    /\* USER CODE END Error\_Handler\_Debug \*/  }  #ifdef  USE\_FULL\_ASSERT  /\*\*    \* @brief  Reports the name of the source file and the source line number    \*         where the assert\_param error has occurred.    \* @param  file: pointer to the source file name    \* @param  line: assert\_param error line source number    \* @retval None    \*/  void assert\_failed(uint8\_t \*file, uint32\_t line)  {    /\* USER CODE BEGIN 6 \*/    // printf("Wrong parameters value: file %s on line %d\r\n", file, line);    /\* USER CODE END 6 \*/  }  #endif /\* USE\_FULL\_ASSERT \*/ |

|  |
| --- |
| Adc.c |
| /\* USER CODE BEGIN Header \*/  /\*\*    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \* @file    adc.c    \* @brief   This file provides code for the configuration    \*          of the ADC instances.    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \* @attention    \*    \* Copyright (c) 2025 STMicroelectronics.    \* All rights reserved.    \*    \* This software is licensed under terms that can be found in the LICENSE file    \* in the root directory of this software component.    \* If no LICENSE file comes with this software, it is provided AS-IS.    \*    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \*/  /\* USER CODE END Header \*/  /\* Includes ------------------------------------------------------------------\*/  #include "adc.h"  /\* USER CODE BEGIN 0 \*/  /\* USER CODE END 0 \*/  ADC\_HandleTypeDef hadc1;  DMA\_HandleTypeDef hdma\_adc1;  /\* ADC1 init function \*/  void MX\_ADC1\_Init(void)  {    /\* USER CODE BEGIN ADC1\_Init 0 \*/    /\* USER CODE END ADC1\_Init 0 \*/    ADC\_ChannelConfTypeDef sConfig = {0};    /\* USER CODE BEGIN ADC1\_Init 1 \*/    /\* USER CODE END ADC1\_Init 1 \*/    /\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)    \*/    hadc1.Instance = ADC1;    hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV4;    hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;    hadc1.Init.ScanConvMode = ENABLE;    hadc1.Init.ContinuousConvMode = ENABLE;    hadc1.Init.DiscontinuousConvMode = DISABLE;    hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;    hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;    hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;    hadc1.Init.NbrOfConversion = 2;    hadc1.Init.DMAContinuousRequests = ENABLE;    hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;    if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)    {      Error\_Handler();    }    /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.    \*/    sConfig.Channel = ADC\_CHANNEL\_0;    sConfig.Rank = 1;    sConfig.SamplingTime = ADC\_SAMPLETIME\_480CYCLES;    if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)    {      Error\_Handler();    }    /\*\* Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.    \*/    sConfig.Channel = ADC\_CHANNEL\_1;    sConfig.Rank = 2;    if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)    {      Error\_Handler();    }    /\* USER CODE BEGIN ADC1\_Init 2 \*/    /\* USER CODE END ADC1\_Init 2 \*/  }  void HAL\_ADC\_MspInit(ADC\_HandleTypeDef\* adcHandle)  {    GPIO\_InitTypeDef GPIO\_InitStruct = {0};    if(adcHandle->Instance==ADC1)    {    /\* USER CODE BEGIN ADC1\_MspInit 0 \*/    /\* USER CODE END ADC1\_MspInit 0 \*/      /\* ADC1 clock enable \*/      \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE();      \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();      /\*\*ADC1 GPIO Configuration      PA0-WKUP     ------> ADC1\_IN0      PA1     ------> ADC1\_IN1      \*/      GPIO\_InitStruct.Pin = GPIO\_PIN\_0|GPIO\_PIN\_1;      GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;      GPIO\_InitStruct.Pull = GPIO\_NOPULL;      HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);      /\* ADC1 DMA Init \*/      /\* ADC1 Init \*/      hdma\_adc1.Instance = DMA2\_Stream0;      hdma\_adc1.Init.Channel = DMA\_CHANNEL\_0;      hdma\_adc1.Init.Direction = DMA\_PERIPH\_TO\_MEMORY;      hdma\_adc1.Init.PeriphInc = DMA\_PINC\_DISABLE;      hdma\_adc1.Init.MemInc = DMA\_MINC\_ENABLE;      hdma\_adc1.Init.PeriphDataAlignment = DMA\_PDATAALIGN\_HALFWORD;      hdma\_adc1.Init.MemDataAlignment = DMA\_MDATAALIGN\_HALFWORD;      hdma\_adc1.Init.Mode = DMA\_CIRCULAR;      hdma\_adc1.Init.Priority = DMA\_PRIORITY\_LOW;      hdma\_adc1.Init.FIFOMode = DMA\_FIFOMODE\_DISABLE;      if (HAL\_DMA\_Init(&hdma\_adc1) != HAL\_OK)      {        Error\_Handler();      }      \_\_HAL\_LINKDMA(adcHandle,DMA\_Handle,hdma\_adc1);    /\* USER CODE BEGIN ADC1\_MspInit 1 \*/    /\* USER CODE END ADC1\_MspInit 1 \*/    }  }  void HAL\_ADC\_MspDeInit(ADC\_HandleTypeDef\* adcHandle)  {    if(adcHandle->Instance==ADC1)    {    /\* USER CODE BEGIN ADC1\_MspDeInit 0 \*/    /\* USER CODE END ADC1\_MspDeInit 0 \*/      /\* Peripheral clock disable \*/      \_\_HAL\_RCC\_ADC1\_CLK\_DISABLE();      /\*\*ADC1 GPIO Configuration      PA0-WKUP     ------> ADC1\_IN0      PA1     ------> ADC1\_IN1      \*/      HAL\_GPIO\_DeInit(GPIOA, GPIO\_PIN\_0|GPIO\_PIN\_1);      /\* ADC1 DMA DeInit \*/      HAL\_DMA\_DeInit(adcHandle->DMA\_Handle);    /\* USER CODE BEGIN ADC1\_MspDeInit 1 \*/    /\* USER CODE END ADC1\_MspDeInit 1 \*/    }  }  /\* USER CODE BEGIN 1 \*/  /\* USER CODE END 1 \*/ |

|  |
| --- |
| Dma.c |
| /\* USER CODE BEGIN Header \*/  /\*\*    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \* @file    dma.c    \* @brief   This file provides code for the configuration    \*          of all the requested memory to memory DMA transfers.    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \* @attention    \*    \* Copyright (c) 2025 STMicroelectronics.    \* All rights reserved.    \*    \* This software is licensed under terms that can be found in the LICENSE file    \* in the root directory of this software component.    \* If no LICENSE file comes with this software, it is provided AS-IS.    \*    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    \*/  /\* USER CODE END Header \*/  /\* Includes ------------------------------------------------------------------\*/  #include "dma.h"  /\* USER CODE BEGIN 0 \*/  /\* USER CODE END 0 \*/  /\*----------------------------------------------------------------------------\*/  /\* Configure DMA                                                              \*/  /\*----------------------------------------------------------------------------\*/  /\* USER CODE BEGIN 1 \*/  /\* USER CODE END 1 \*/  /\*\*    \* Enable DMA controller clock    \*/  void MX\_DMA\_Init(void)  {    /\* DMA controller clock enable \*/    \_\_HAL\_RCC\_DMA2\_CLK\_ENABLE();    /\* DMA interrupt init \*/    /\* DMA2\_Stream0\_IRQn interrupt configuration \*/    HAL\_NVIC\_SetPriority(DMA2\_Stream0\_IRQn, 0, 0);    HAL\_NVIC\_EnableIRQ(DMA2\_Stream0\_IRQn);  }  /\* USER CODE BEGIN 2 \*/  /\* USER CODE END 2 \*/ |

|  |
| --- |
| Iir\_lpf.c |
| /\* iir\_lpf.c \*/  #include "iir\_lpf.h"  void IIR\_LPF\_Init(IIR\_LPF\_HandleTypeDef \*h, float alpha, float init)  {      if (alpha < 0.0f)          alpha = 0.0f;      if (alpha > 1.0f)          alpha = 1.0f;      h->alpha = alpha;      h->prev\_out = init;  }  float IIR\_LPF\_Update(IIR\_LPF\_HandleTypeDef \*h, float sample)  {      // y[n] = α⋅y[n-1] + (1-α)⋅x[n]      h->prev\_out = h->alpha \* h->prev\_out + (1.0f - h->alpha) \* sample;      return h->prev\_out;  } |