摘 要

ADC 和 DAC 在数字化时代中扮演着极其重要的角色，是现实的模拟世界与虚拟的数字世界的桥梁，所以研究单片机应用系统中模拟通道（模数转换和数模转换通道）的设计与应用就显得十分重要。

本课题的主要任务有两项，其一是完成了以 STM32F103ZET6 微处理器为主控芯片的实验板的设计与制作。其中包括实验板的原理电路设计，实验板的 PCB 板设计及实验板的焊接装配与调试等任务。其二是利用该实验板完成了若干个 ADC 和 DAC 通道实验课题的开发任务。

论文介绍了 STM32 微处理器的技术要点，说明了 STM32 实验板设计方法，重点讨论了 STM32 的 ADC 和 DAC 的实验课题的开发过程。本论文所给出的实验课题是针对本科生嵌入式系统实验而设计的，实验内容的难度和任务量适当，一般以 2 个小时或 4 个小时为一个单元，实验目的明确，实验原理正确且具有代表性。实验课题的设计思想是由易到难、环环相扣，系统而全面的展现了 STM32 的 ADC 和 DAC 技术的特点及应用方法，对广大学生学习掌握嵌入式系统具有指导意义。

关键字：单片机 STM32 ADC DAC 模拟通道 PCB

ABSTRACT

ADC and DAC play an extremely important role in the digital age, they are the bridge of the real analog world and the virtual digital world. So the study of the design and application of SCM application system analog channels (analog to digital conversion and digital to analog conversion channel) is very important.

The main task of this project has two, one is to complete the design and fabrication of experimental board with STM32F103ZET6 microprocessor as the main control chip. It include the design of principle circuit, PCB board and the welding assembly and debugging of the experimental board. The second is to use the experimental board to complete the development of several ADC and DAC channel experimental subjects.

This thesis introduces the key technology of STM32 microprocessor, explains the design method of the STM32 experimental board, discusse the development process of STM32 ADC and DAC experimental subject. The experimental subjects of this thesis are designed for the undergraduate student's embedded system experiment, the content of the experiment and the difficulty of the experiment are appropriate, generally 2 hours or 4 hours as a unit, the purpose of the experiment is explicit, and the principle of the experiment is correct and representative. Design idea of the experiment subject is from easy to difficult, interlocking, systematic and comprehensive display characteristics and application method of STM32 ADC and DAC technology, it has guiding significance for the majority of the students to master the embedded system.

Keywords: SCM STM32 ADC DAC Analog Channels PCB

目 录

第一章 绪论

1.1 课题的研究背景与意义

多年来西安电子科技大学通信工程学院本科生微机原理课程一直以 INTEL X86 系统为主进行理论课教学，与其配套的微机原理实验均以 X86 系统为主进行教学，以至于影响后续课程，如课程设计，综合开发应用实验，工程设计等大多以 51 单片机等系统为平台，严重制约了学生对新技术的了解和掌握。

基于以上原因，通信工程学院计划从 2015 年开始将本科生微机原理实验课升级为嵌入式系统实验，拟开发一套以 STM32 为主体的实验平台，从根本上提升学生的实验训练环境。此计划对学生迅速掌握和跟进当今最新技术有关键性作用，对提高学校的办学水平，增强学校学生的竞争力具有有很重要的意义。

本课题就是上述计划的一个子课题，其主要任务有两项，其一是设计一套以 STM32 微处理器为核心的实验平台，其中包括实验平台的原理图设计，PCB 板设计，实验平台的焊接装配及调试等。其二是为该平台开发和配置 ADC 及 DAC 相关实验。

1.2 STM32F103ZET6 简介

STM32 系列微处理器基于专为要求高性能、低成本、低功耗的嵌入式应用专门设计的 ARM Cortex-M3 内核，其按性能分成两个不同的系列：STM32F103 增强型系列和 STM32F101 基本型系列。

以 STM32 微处理器为核心的实验平台的 STM32 微处理器实验板设计采用的 STM32F103ZET6 属于增强型系列，它的各参数如下[1]：

1. 内核：STM32F103ZET6 微处理器以 ARM 32 位的 Cortex™-M3 为内核，具有高达 72MHz 的工作频率。
2. 存储器：STM32F103ZET6 微处理器内部具有 512K 字节的闪存程序存储器和高达 64K 字节的 SRAM，并带 4 个片选的静态存储器控制器。同时，STM32F103ZET6 微处理器支持 CF 卡、SRAM、PSRAM、NOR 和 NAND 存储器，具有并行 LCD 接口，兼容 8080/6800 模式。

3. 时钟、复位和电源管理：STM32F103ZET6 微处理器支持 2.0～3.6 伏供电和 I/O 引脚，支持上电/断电复位(POR/PDR)、可编程电压监测器(PVD)，具有 4～16MHz 晶体振荡器，内嵌经出厂调校的 8MHz 的 RC 振荡器，内嵌带校准的 40kHz 的 RC 振荡器，具有带校准功能的 32kHz RTC 振荡器。
4. 低功耗：STM32F103ZET6 微处理器具有睡眠、停机和待机模式，其 VBAT 可以为 RTC 和后备寄存器供电。
5. 3 个 12 位模数转换器：STM32F103ZET6 微处理器内含 3 个 12 位模数转换器，其转换范围为 0V 至 3.6V，具有三倍采样和保持功能，内置温度传感器。
6. 2 通道 12 位 D/A 转换器。
7. DMA：STM32F103ZET6 微处理器内涵 12 通道 DMA 控制器，它支持的外设有定时器、ADC、DAC、SDIO、I2S、SPI、I2C 和 USART。
8. 调试模式：STM32F103ZET6 微处理器具有串行单线调试(SWD)和 JTAG 接口，具有 Cortex-M3 内嵌跟踪模块(ETM)。
9. 多达 112 个快速 I/O 端口：STM32F103ZET6 微处理器具有 112 个多功能双向的 I/O 口，所有 I/O 口可以映像到 16 个外部中断，几乎所有端口均可容忍 5V 信号。
10. 多达 11 个定时器：STM32F103ZET6 微处理器内含 4 个 16 位定时器，2 个 16 位带死区控制和紧急刹车，2 个看门狗定时器(独立的和窗口型的) ，2 个 16 位基本定时器用于驱动 DAC，1 个系统时间定时器。
11. 多达 13 个通信接口：STM32F103ZET6 微处理器内含 2 个 I2C 接口，5 个 USART 接口， 3 个 SPI 接口，2 个可复用为 I2S 接口 CAN 接口。同时，支持 USB 2.0 全速接口和 SDIO 接口

可见 STM32F103ZET6 增强型系列微处理器是一个功能强大的微处理器，对于进行 AD 转换和 DA 转换的设计和应用有着得天独厚的优势。

1.3 本课题任务与工作内容

本论文主要完成的是 STM32 微处理器实验平台设计和 PCB 绘制，以及 STM32 微处理器实验平台当中，有关于 STM32 内部的 ADC 和 DAC 外围电路设计，并为 STM32 微处理器实验平台开发和配置了 STM32 内部的 ADC 和 DAC 相关的应用实验。

本论文为 STM32 微处理器实验平台设计了相应的 ADC 和 DAC 外围电路，比如模拟电压采集电路，电平抬升电路等等。课题的另一个关键问题主要集中在相关应用实验的设计上，因为只有合理的实验设计，才能全面而完整的展现出 STM32 内部 ADC 和 DAC 的强大功能和使用方法。

本文一共为 STM32 微处理器实验平台开发和配置了五个 ADC 和 DAC 相关应用实验，表 1.1 列出了相关实验的名称。

表 1.1 STM32 内部 ADC 和 DAC 实验列表

实验编号	实验题目
实验一	STM32 微处理器 A/D 单通道采集电压实验
实验二	STM32 微处理器 A/D 通道采集正弦波实验
实验三	STM32 微处理器 D/A 通道产生模拟电压实验
实验四	STM32 微处理器 D/A 通道产生三角波实验
实验五	STM32 微处理器 A/D 与 D/A 通道的联合实验

表 1.1 给出的实验课题是针对本科生嵌入式系统实验而设计的，实验内容的难度和任务量适当，一般以 2 个小时或 4 个小时为一个单元，实验目的明确，实验原理正确且具有代表性。实验课题的设计思想是由易到难、环环相扣，系统而全面的展现了 STM32 的 ADC 和 DAC 技术的特点及应用方法。

“STM32 微处理器 A/D 单通道采集电压实验”完成的是 ADC 单通道采集一个模拟电压，实验内容虽然比较简单，但是基本原理明确，软硬件原理清楚，便于学生快速掌握嵌入式开发的基本方法，达到快速入门的目的。

“STM32 微处理器 A/D 通道采集正弦波实验”完成 ADC 单通道采集一个正弦波的实验，在实验一的基础上增加了 DMA 的操作，难度有所增加，体现了 STM32 的技术优点。

“STM32 微处理器 D/A 通道产生模拟电压实验”实现了 DAC 输出幅度可调的模拟电压，该实验的输出结果可以用示波器测量显示，也可以通过 ADC 通道进行

采样，将 DAC 输出值显示到 LCD 屏幕上，实验原理清楚，实验效果明了。

“STM32 微处理器 D/A 通道产生三角波实验”完成利用 DAC 输出一定幅度一定频率的三角波，该实验的输出结果可以用示波器测量显示，也可以通过 ADC 通道进行采样，将 DAC 输出的波形显示到 LCD 屏幕上，这里 ADC 的采样程序是在实验二 ADC 相关代码的基础上增加功能改进而来的，其中增加了 ADC 的 DMA 中断功能，增加定时器控制功能来周期性的处理，实验难度进一步加大。

“STM32 微处理器 A/D 与 D/A 通道的联合实验”是一个综合实验，该实验利用 ADC 采样一个方波信号，并将采集到的数据送到 DAC 通道输出，此时 DAC 通道的输出与 ADC 通道的输入是幅值相同频率相同的方波信号，将前面实验当中涉及到的知识系统的综合起来，能使学生全面掌握 STM32 的 ADC 和 DAC 的使用。

本文所要进行的具体工作为：

1. 设计相应的硬件实现电路。
2. 绘制 STM32 实验板 PCB，制板、焊接、调试。
3. 完成 STM32 微处理器内部 A/D 通道的实验设计与验证。

   利用微处理器内部的 ADC1 的一个通道，采集外部模拟电压信号、低频正弦波信号，通过 A/D 转换器转化为数字量，并显示在 LCD 显示屏上或通过串口传输到上位机。
4. 完成 STM32 微处理器内部 D/A 通道的实验设计与验证。

   将给定的数字量通过 D/A 转换器转化为模拟输出电压，利用 D/A 转换器输出连续的模拟三角波信号，并经 ADC 转化后在显示到 LCD 显示屏上。
5. 完成 STM32 微处理器内部 A/D 和 D/A 通道联合实验设计与验证。

   利用 STM32 内部的 ADC 采集方波信号，并将获得的数字量在 LCD 屏上显示，再通过 DA 转换器将获得的数字量转化为方波输出。

1.4 论文章节安排

本论文完成了 STM32 微处理器实验板的设计和制作，为 STM32 的 ADC 和 DAC 通道设计了相应的硬件电路，并完成了 STM32 内部的 ADC 和 DAC 的应用实验的设计工作，论文的具体章节如下：

第一章是绪论，介绍课题的研究背景和意义，以及本课题的任务和所做的工作。

第二章简单的介绍了 STM32 内部 ADC 和 DAC 的相关知识以及以 STM32 微处理器实验平台当中与 ADC 和 DAC 有关的硬件原理图的设计。

第三章简单的介绍了以 STM32 微处理器为核心的试验平台的设计步骤和相关问题。

第四章详细介绍本课题设计的相关实验的目的原理内容和验证分析。

第五章是对所做工作的总结与展望。

第二章 STM32 微处理器的模拟通道

2.1 STM32 内部 ADC 概述

STM32F103ZET6 微处理器中的 ADC 是一种 12 位的逐次逼近型模拟数字转换器，其主要特性如下[2]：

1. 12 位分辨率。
2. 转换结束、注入转换结束和发生模拟看门狗事件时产生中断。
3. 单次和连续转换模式。
4. 从通道 0 到通道 n 的自动扫描模式。
5. 自校准。
6. 带内嵌数据一致性的数据对齐。
7. 采样间隔可以按通道分别编程。
8. 规则转换和注入转换均有外部促发选项。
9. 间断模式、双重模式。
10. ADC 转换时间：STM32F103ZET6 的时钟为 56MHz 时为 1μs（时钟为 72MHz 时为 1.17μs）
11. ADC 供电要求：2.4V~3.6V。
12. ADC 输入范围：$V_{REF-} \le V_{IN} \le V_{REF+}$。
13. 规则通道转换期间有 DMA 请求产生。

表 2.1 对 STM32F103ZET6 内部 ADC 引脚进行了说明[2]。

表 2.1 STM32F103ZET6 内部 ADC 引脚说明

名称	信号类型	注解
V_{REF+}	输入，模拟参考正极	ADC 使用的高端/正极参考电压， $2.4V \le V_{REF+} \le V_{DD}$
V_{DDA}(1)	输入，模拟电源	等效于 VDD 的模拟电源且： $2.4V \le V_{DDA} \le V_{DD}$ (3.6V)
V_{REF-}	输入，模拟参考负极	ADC 使用的低端/负极参考电压， $V_{REF-} = V_{SSA}$
V_{SSA}(1)	输入，模拟电源地	等效于 V_{SS} 的模拟电源地
ADCx_IN[15:0]	模拟输入信号	16 个模拟输入通道

1．V_{DDA} 和 V_{SSA} 应该分别连接到 V_{DD} 和 V_{SS}。

接下来简单的介绍一下STM32F103ZET6内部三个ADC的通道和管脚的关系，STM32F103ZET6内部有3个ADC共含有多达18个转换通道，每个通道以及ADC所对应管脚如表2.2所示[3]。

表2.2 各ADC通道管脚分布

	ADC1	ADC2	ADC3
通道0	PA0	PA0	PA0
通道1	PA1	PA1	PA1
通道2	PA2	PA2	PA2
通道3	PA3	PA3	PA3
通道4	PA4	PA4	PF6
通道5	PA5	PA5	PF7
通道6	PA6	PA6	PF8
通道7	PA7	PA7	PF9
通道8	PB0	PB0	PF10
通道9	PB1	PB1	
通道10	PC0	PC0	PC0
通道11	PC1	PC1	PC1
通道12	PC2	PC2	PC2
通道13	PC3	PC3	PC3
通道14	PC4	PC4	
通道15	PC5	PC5	
通道16	温度传感器		
通道17	内部参照电压		

STM32F103ZET6微处理器最多有16个外部通道，可用来测量16个外部模拟信号源，各各通道的A/D转换可以以单次、连续、扫描或不连续的模式执行A/D转换，转换的结果可以以左对齐或者右对齐的方式存储在16位数据寄存器中[2]。

ADC也可以使用DMA功能，同时处理器提供的ADC模拟看门狗允许非常精确的监视一路、多路或者所有选中的通道，当被监视的信号超出预置的阈值时将产生中断，由标准定时器（TIMx）和高级控制定时器（TIM1）产生的事件可以分内部级联到ADC的开始促发、外部促发和DMA促发，使应用程序能同步A/D转换和时钟。

2.2 ADC 通道实验硬件设计

为了配合下面实验的设计，A/D 通道采用 STM32F103ZET6 微处理器内部的 ADC1 进行 AD 转换，采集模拟电压信号、正弦波信号等，进行 AD 转化，把最后转化结果显示到 LCD 屏幕或者是通过串口传输到上位机显示。

相应的硬件电路包括：3.3V 高精度基准电压电路、模拟电压采集实验电压产生电路、8 路 ADC 预留接口、正弦波采集电平抬升电路等。

2.2.1 3.3V 高精度基准电压电路

由 2.1 节的介绍可知，ADC 的输入模拟参考电压正极 V_{REF+}的范围为 $2.4V \le V_{REF+} \le V_{DD}$，在这里我们为它提供 3.3V 的参考电压。为了提供稳定且高精度的参考电压[4]，我们设计了为 ADC 提供 3.3V 高精度基准电压的电路，3.3V 高精度基准电压电路的原理图如图 2.1 所示。

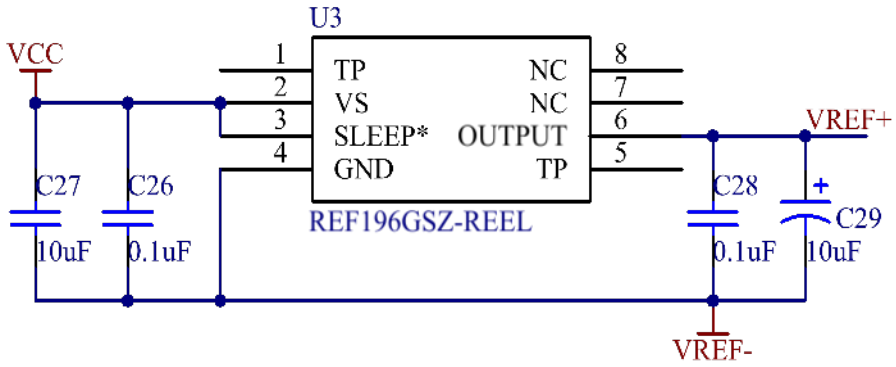

图 2.1 3.3V 高精度基准电压原理图

3.3V 高精度基准电压电路使用一个 3.3V 基准电压芯片：REF196[5]。REF196 是精密、微功耗、低压差、低压基准电压源。采用温度漂移曲率校正专利电路，并对高稳定性薄膜电阻进行激光调整，从而实现极低的温度系数和高初始精度[6]。

2.2.2 模拟电压采集实验电压产生电路

模拟电压采集实验电压产生电路原理图如图 2.2 所示。

模拟电压采集实验电压产生电路使用一个 10K 精密可调电阻，调节其大小可以从 3.3V 分压得到所要采集的模拟电压值，将这个模拟电压值送到 PC0，即 ADC1 的通道 10，就可以做 ADC 模拟电压的采样实验。

另外电路中还分别接了一个光敏电阻和热敏电阻，用于以后扩展温度采集实验

和光强采集实验。

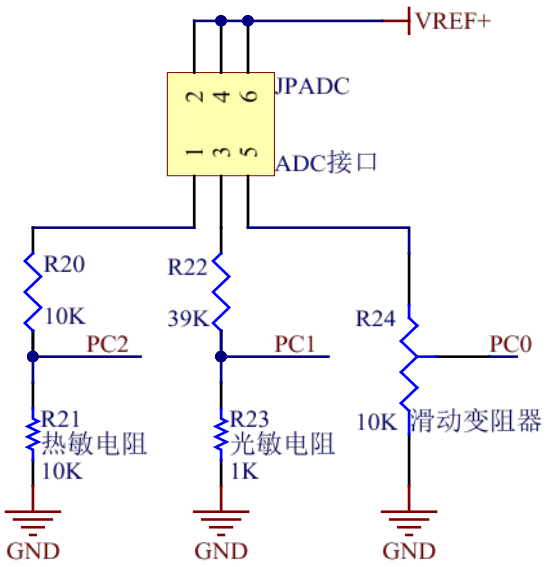

图 2.2 模拟电压采集实验电压产生电路

2.2.3 正弦波采集电平抬升电路

由表 2.1 可知，STM32 的 ADC 是不能直接测量负电压的，而且其输入的电压信号的范围为：$V_{REF-} \le V_{IN} \le V_{REF+}$。当需要测量负电压、测量的电压信号超出范围或测量的量为交变信号的话，要先经过运算电路进行电平抬升或利用电阻分压。

ADC 采集正弦波的实验所要采集的正弦电压信号为交变信号，为了达到 ADC 对输入电压的要求，需要集成一个电平抬升电路，将输入的正弦电压信号抬升到 0V 到 3.3V 范围内，然后再进行 A/D 转换，电平抬升电路原理图如图 2.3 所示。

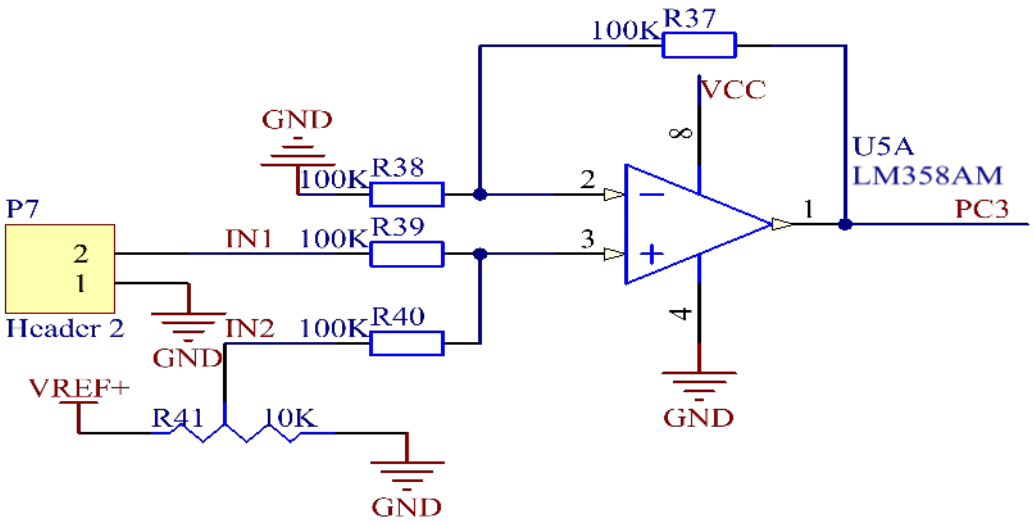

图 2.3 电平抬升电路原理图

电平抬升电路采用一个同相相加器，经电平抬升电路抬升后，信号输出端 PC3 即 ADC1 的通道 13 的输入电压 u_{PC3} 的计算公式为[7]：

$$u_{PC3} = (1 + \frac{R_{37}}{R_{38}}) \bullet (\frac{R_{40}}{R_{39} + R_{40}} u_{IN1} + \frac{R_{39}}{R_{39} + R_{40}} u_{IN2})$$

$$= u_{IN1} + u_{IN2} \quad (2\text{-}1)$$

电平抬升电路中的放大器使用可以单电源供电的 LM358[8]，用 5V 电源进行单电源供电。

LM358 内部包括有两个独立的、高增益、内部频率补偿的双运算放大器，适合于电源电压范围很宽的单电源使用，也适用于双电源工作模式，在推荐的工作条件下，电源电流与电源电压无关。它的使用范围包括传感放大器、直流增益模块和其他所有可用单电源供电的使用运算放大器的场合。

2.2.4 8 路 ADC 预留接口

为了方便进行 STM32 内部 ADC 与 DAC 相关实验以及后续扩展实验的方便，STM32 内部 ADC123_IN0 - ADC123_IN7，即 GPIO 口 PA0 - PA7 被引出到板子上的独立 8 针的单排针 P3 上，8 路 ADC 预留接口所对应的引脚定义图如图 2.4 所示。

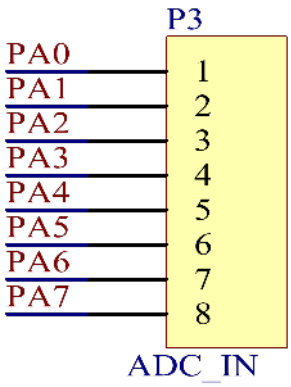

图 2.4 预留接口引脚定义图

因为本文并没有设计相应的滤波、隔离和电压跟随等电路，若做扩展 8 通道采集实验，需在外部扩展相应电路进行信号处理，再输入开发板进行数据采集，才会达到更好的效果。

2.3 STM32 内部 DAC 概述

在 STM32F103ZET6 微处理器中，含有一个 DAC，它是 12 位数字输入电压输出的 D/A 转换器。其主要特征如下[2]：

1. 2 个 DAC 转换器：每个转换器对应 1 个输出通道
2. 8 位或者 12 位单调输出
3. 12 位模式下数据左对齐或者右对齐
4. 同步更新功能
5. 噪声波形生成
6. 三角波形生成
7. 双 DAC 通道同时或者分别转换
8. 每个通道都有 DMA 功能
9. 外部触发转换
10. 输入参考电压 V_{REF+}

接下来简单的介绍一下 STM32F103ZET6 内部 DAC 的通道和管脚的对应关系，STM32F103ZET6 内部有一个共含两个转换通道，每个通道所对应管脚如表 2.3 所示[2]。

表 2.3 DAC 各通道管脚分布

通道	管脚
DAC_Channel_1	PA4
DAC_Channel_2	PA5

表 2.4 中给出了 STM32 内部 DAC 的管脚的说明[2]。

表 2.4 DAC 引脚说明

名称	型号类型	注释
V_{REF+}	输入，正模拟参考电压	DAC 使用的高端/正极参考电压 $2.4V \le V_{REF+} \le V_{DDA}$ (3.3V)
V_{DDA}	输入，模拟电源	模拟电源
V_{SSA}	输入，模拟电源地	模拟电源的地线
DAC_OUTx	模拟输出信号	DAC 通道 x 的模拟输出

一旦使能 DACx 通道，相应的 GPIO 引脚(PA4 或者 PA5)就会自动与 DAC 的模拟输出相连(DAC_OUTx)。为了避免寄生的干扰和额外的功耗，引脚 PA4 或者 PA5

在之前应当设置成模拟输入(AIN)。

STM32F103ZET6 微处理器内置的 DAC 可以配置为 8 位或 12 位模式，也可以与 DMA 控制器配合使用。DAC 工作在 12 位模式时，数据可以设置成左对齐或右对齐。

DAC 模块有 2 个输出通道，每个通道都有单独的转换器。在双 DAC 模式下，2 个通道可以独立地进行转换，也可以同时进行转换并同步地更新 2 个通道的输出。DAC 可以通过引脚输入参考电压 V_{REF+}以获得更精确的转换结果。

2.4 DAC 通道实验硬件设计

为了配合下面实验的设计，D/A 通道采用 STM32F103ZET6 微处理器内部的 DAC 的通道 1 和通道 2 进行模拟电压输出，三角波的输出等，所以应该将 DAC 两个通道所对应的管脚引出来，在 STM32 微处理器实验板上留出相应的接口，该接口在 STM32 微处理器实验板上为独立的单排针 P6，DAC 实验接口所对应的引脚的定义如图 2.5 所示。

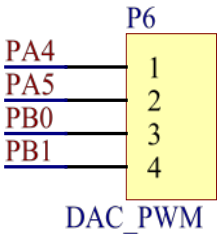

图 2.5 DAC 实验接口引脚定义图

因本文针对 DAC 的实验主要是让 DAC 产生模拟电压信号，产生三角波，再同 ADC 进行联合实验，所以 DAC 电路提供了 STM32F103ZET6 的内部 DAC 的两个通道的接口（分别连接 PA4 和 PA5），另外也同时预留了 PWM 的接口以供后续实验开发。

因为本课题仅是做相关输出实验，并没有过高的对输出信号的干扰提出要求，故没有设计相关的平滑滤波电路，若要做与 DAC 相关的扩展实验的话，需要根据 DAC 输出信号的频率设计相匹配的低通滤波器起到平滑滤波作用。

第三章 STM32 微处理器实验板的设计与制作

本论文要完成的任务是 STM32 微处理器实验平台的设计（与他人合作完成），所以先要为该平台设计一个 STM32 微处理器实验板，完成包括 STM32 微处理器最小系统、4×4 矩阵键盘、8 位 LED 灯、蜂鸣器、I2C-EEPROM 电路、串口电路、LCD 显示接口电路、USB 电路、电源电路等电路的设计，绘制出 PCB 板并做出 STM32 微处理器实验板实物。

本论文的原理图和 PCB 板使用 Altium Designer 制作[9][10]。

3.1 STM32 微处理器实验板原理图设计

本课题所做工作为 STM32 微处理器实验板的一部分，STM32 微处理器实验板涉及到的电路除了第二章提到的电路外，还包括 STM32 微处理器最小系统、4×4 矩阵键盘、8 位 LED 灯、蜂鸣器、I2C-EEPROM 电路、串口电路、LCD 显示接口电路、USB 电路、电源电路等。

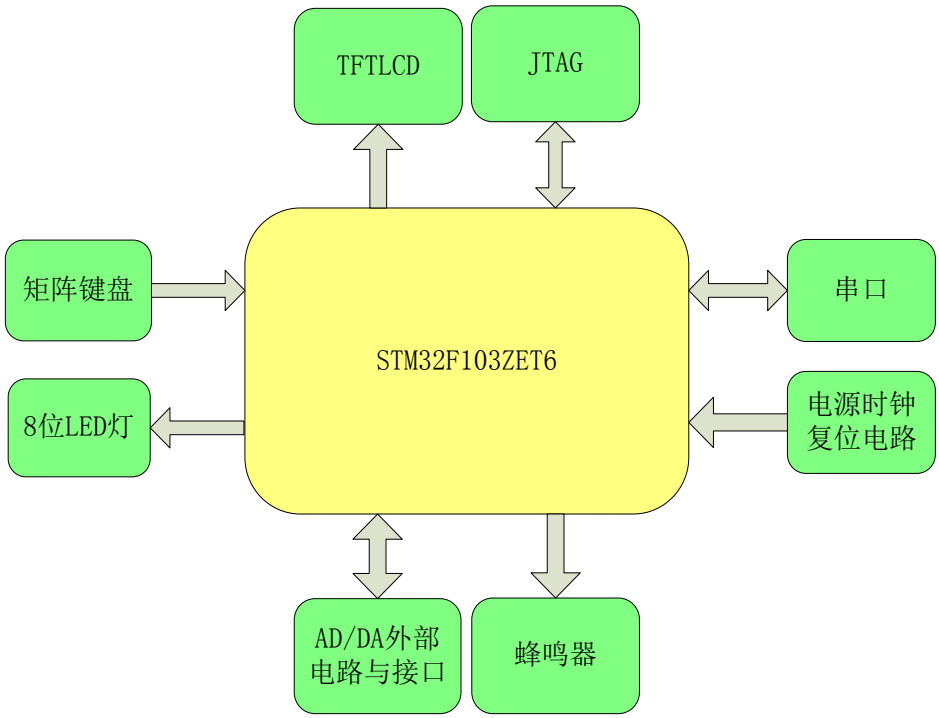

图 3.1 STM32 微处理器实验板的整体框图

STM32 微处理器实验板可以提供进行矩阵键盘实验，蜂鸣器实验，流水灯实验，串口实验，LCD 显示实验，I2C-EEPROM 实验，GPIO 口实验、AD 和 DA 转

换实验等，实验板使用 JTAG 进行程序的烧写还有调试工作，实验板设计采用如下三种供电方式：

1. DC9V 通过 7805 转化为 5V 再经过 AMS1117[11]转化到 3.3V。
2. JTAG 提供 5V 电压，再经过 AMS1117 转化到 3.3V。
3. USB 提供 5V 电压，再经过 AMS1117 转化到 3.3V。

STM32 微处理器实验板的整体框图如图 3.1 所示。

为完成 STM32 微处理器实验板的设计，须将各同学所设计的实验模块集中到同一个板子上，统一原理图的各个网络，完成整体原理图的统合设计，合理引用管脚，尽量避免管脚的过多复用，这些工作是在我的协调下完成的。

在 STM32 微处理器实验板原理图的整体设计当中，使用方框将各各单元电路部分区分开，并加以文字说明，结构清晰明了。

设计完成后对整体的原理图进行编译来进行电气连接检测，无误后方可进行 PCB 的设计。

3.2 STM32 微处理器实验板 PCB 绘制

绘制 PCB 前，首先要对原理图当中所有的元件进行封装选择，然后再将编译通过的原理图导入到建好的 PCB 文件，接着设定布线规则，然后进行布线工作，最后进行铺铜完成 PCB 板的绘制[12]。

3.2.1 元件封装选择和布线规则设定

首先在元件封装库中寻找相应的封装形式，没有的封装在网上下载相应的库或自己绘制相应的封装，然后使用封装管理器进行封装的统一管理。

为了尽量减小板子的大小，同时考虑手工焊接操作的方便性，STM32 微处理器实验板的元件多采用相对比较大号的表面贴装的元件封装形式，部分元件的封装形式如表 3.1 所示。

进行布线规则的设定是绘制 PCB 板的重要一环，布线规则将会影响所设计板子的印制成本还有板子的整体性能。好的布线不仅能让板子看起来美观，更能提升板子的抗干扰性。

设定好布线规则后，每当对 PCB 当中的元件进行移动或者布线操作时，如果出现违反规则的操作，系统都会自动显示出违反规则的地方和错误的类型，即错误

标志，这就是所谓的在线规则检查(online DRC)。

表 3.1 部分原件封装形式

元件类型	封装形式
STM32F103ZET6	LQFP114_N
电容	0805 1206(极性电容)
电阻	0805
7805、1117	D2PAK_N
USB	USB-B
LED 发光二极管	SMT LED 0805
ST24C02	SO8
三极管	TO92
键盘开关	AN66
MAX232	WSO16_L
LM358	M08A_L

结合当前的制板工艺和板子的布局等因素，对制板的部分规则做如表 3.2 所示的设定。

表 3.2 部分布线规则

类型	规则
焊盘过孔与导线间距	最小 8mil
线宽	最大 10mil，最小 8mil，默认 8mil
过孔	外径最小 28mil，内径最小 14mil
孔与孔间距	最小 10mil

另外设定关闭对丝印层规则的检查，因为这些规则检查对板子的设计影响较小，可以忽略，其他的规则使用默认规则。

3.2.2 PCB 板布局布线铺铜

PCB 板布局的时候，按电路模块进行布局，电路模块中的元件采用就近集中原则，合理布局元件位置，尽量减少导线过多交织，布线时，顶层尽量走水平线，底层尽量走竖直线，线的拐角不可出现 90 度角，布线结束后进行规则检查看有没有错误，如果没有，对板子的顶层和底层铺铜，将电源地填充满整个 PCB 板以起到屏蔽作用，减少干扰的影响。

对于 STM32 微处理器实验板的 PCB 布线，在布线时尽量将模拟电路与数字电路部分分隔开，以免引起相互的干扰，提升板子的抗干扰性能。

STM32 微处理器实验板铺铜后的整体 PCB-2D 图如图 3.2 所示。

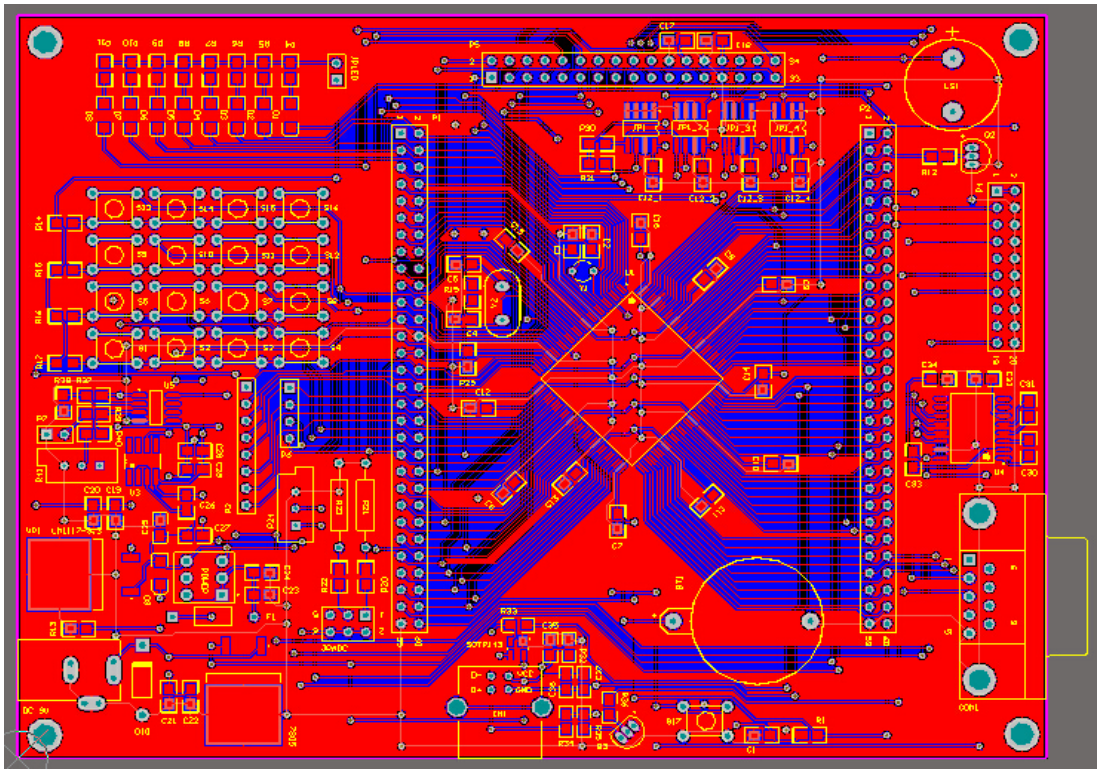

图 3.2　STM32 微处理器实验板铺铜后的整体 PCB

STM32 微处理器实验板铺铜后的整体 PCB-3D 图如图 3.3 所示。

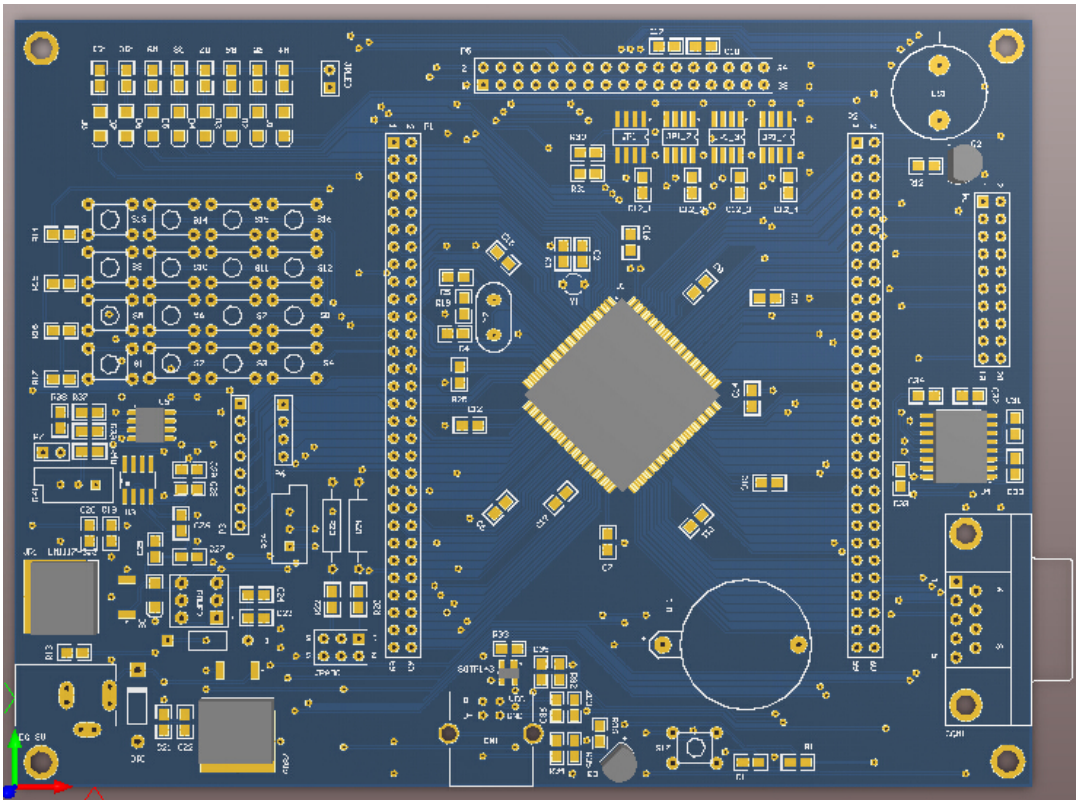

图 3.3　STM32 微处理器实验板铺铜后的整体 PCB-3D 图

3.3 PCB 板印制焊接成型

铺铜后可以把 PCB 发给工厂印刷，待板子印好元件配齐以后，就开始焊接工作，焊接工作最大的难点在于 STM32F103ZET6 微处理器芯片的焊接，需要使用刀头烙铁进行焊接。另外每焊接一个电路模块，要对该模块部分进行功能检查，以确保整体的每一部分都不出现焊接问题，并对有出现问题的部分进行补救处理。

焊接完成后的 STM32 微处理器实验板的实物图如图 3.4 和图 3.5 所示。

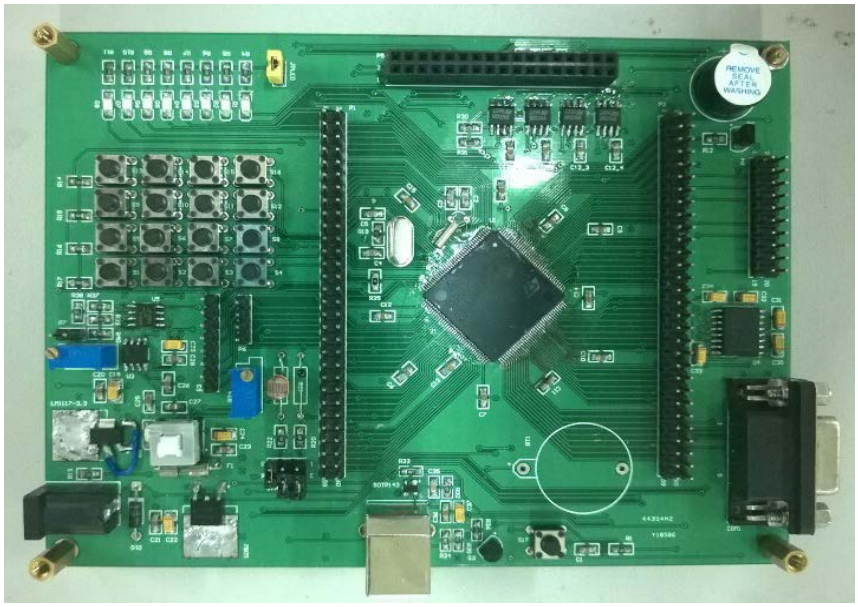

图 3.4 STM32 微处理器实验板的实物图 A

图 3.5 STM32 微处理器实验板的实物图 B

第四章 STM32 微处理器模拟通道的实验设计

4.1 建立 Keil MDK 模版工程

Keil MDK 开发工具是 ARM 公司推出的针对各种嵌入式处理器的软件开发工具。Keil MDK 为基于 Cortex-M、Cortex-R4、ARM7、ARM9 处理器设备提供了一个完整的开发环境，不仅易学易用，而且功能强大，能满足大多苛刻的嵌入式应用[13]。

为了实验方便，本文建立了 Keil MDK 模版工程，要建立模版工程需要完成以下几个工作：

一． 在将要建立工程的文件夹下创建 7 个文件夹，要创建的文件夹的名称和相应的作用如下表 4.1 所示。

表 4.1 要创建的文件夹的名称和说明

文件夹	说明
USER	需要添加的文件为： main.c、 stm32f10x.h、stm32f10x_conf.h、stm32f10x_it.c、 stm32f10x_it.h、system_stm32f10x.c、system_stm32f10x.h
HARDWARE	相关的外设操作代码存放在这里
SYSTEM	自己封装好的有关于 STM32 操作的代码存放在这里， 比如 delay.c、delay.h、sys.c、sys.h、usart.c、usart.h
CORE	存放 STM32 的启动代码： core_cm3.c、core_cm3.h、startup_stm32f10x_hd.s
STM32F10x_FWLib	存放 STM32 固件库的头文件与源文件
PROJECT	存放工程文件
OBJ	存放生成的 hex 文件

二． 文件都准备好以后，用 Keil MDK 在工程目录的文件夹 PROJECT 下建立新的工程，使用 Manage Components 将工程名称改为所建立的工程名，然后把上一节的创建的文件夹添加到 Groups 里面，并将里面的文件添加到相应的 Groups 下面[14]。例如添加好 STM32F10x_FWLib 所需文件后的情况如图 4.1 所示。

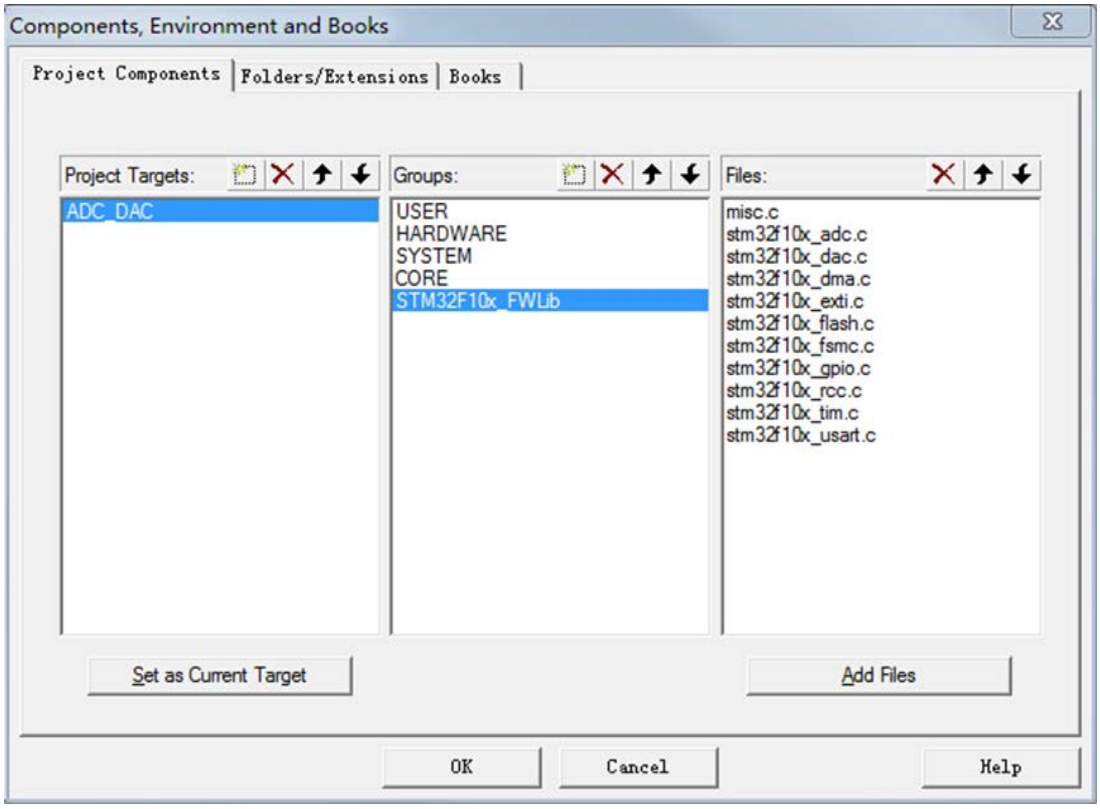

图 4.1 向 STM32F10x_FWLib 组里添加文件

三． 下面需要告诉 MDK 在哪些路径之下搜索相应的文件。在 Target Options 里的 C/C++选项卡里面，将“STM32F10X_HD,USE_STDPERIPH_DRIVER”输入到 Define 输入框里，并在 Include Path 里添加文件夹 CORE、SYSTEM、USER、STM32F10x_FWLib、HARDWARE，路径要到最后一节文件目录[14]，例如图 4.2 所示的情况。

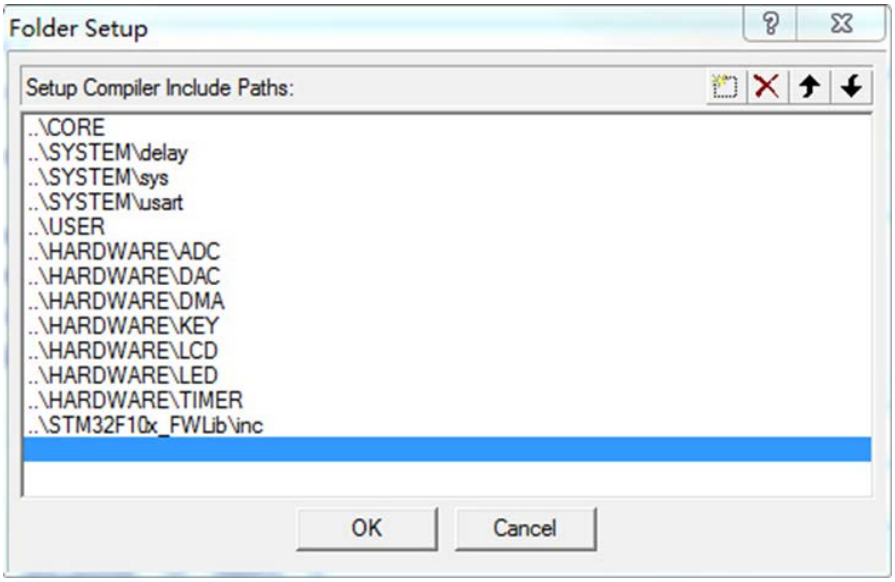

图 4.2 指定所包括的路径

四． 在 Target Options 里的 Output 选项卡里面，勾选 Create HEX File 复选框，并用 Select Folder Object 选择 HEX 文件要存放的位置[14]，这里选择之前创建的文件夹 OBJ，HEX 文件是程序编译生成的可执行文件。

这样一个完整的工程基本上就建完了，接下来的任务就是编写代码，完成各外设模块的功能。

4.2 STM32 微处理器 A/D 单通道采集电压实验

4.2.1 实验目的

掌握 STM32 内部 ADC 的基本操作。

利用 STM32 内部的 ADC1 的通道 10 进行单次转换模式的模拟电压采样。

4.2.2 实验原理

被采样的电压是通过滑动变阻器从 3.3V 电压分出的，这样可以防止 ADC 输入电压过大烧坏 ADC。

实验原理图如图 4.3 所示。

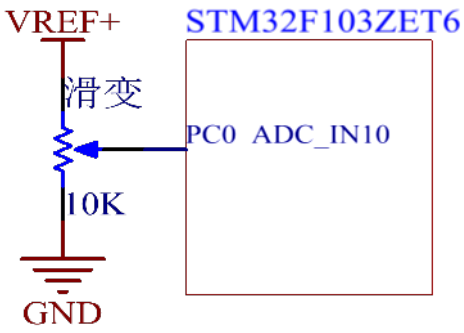

图 4.3 A/D 单通道采集电压实验原理图

将从滑动变阻器取得的分压送到到 STM32 的 PC0 口，也就是内置 ADC1 的通道 10，再利用 STM32 的库函数进行 ADC 实验程序的编写[15]。

该实验的程序流程图如图 4.4 所示。

下面简单的说明一下在编写程序时要使用到的库函数和库定义[16]。

1. ADC_InitTypeDef 为 ADC 初始化结构体，它在库函数里的定义为：

```
typedef struct
{
```

```
    uint32_t ADC_Mode; //采样模式
    FunctionalState ADC_ScanConvMode; //是否开启扫描模式
    FunctionalState ADC_ContinuousConvMode; //是否开启连续转换模式
    uint32_t ADC_ExternalTrigConv; //是否有外界促发转换
    uint32_t ADC_DataAlign; //数据对其方式
    uint8_t ADC_NbrOfChannel; //扫描通道数
}ADC_InitTypeDef;
```

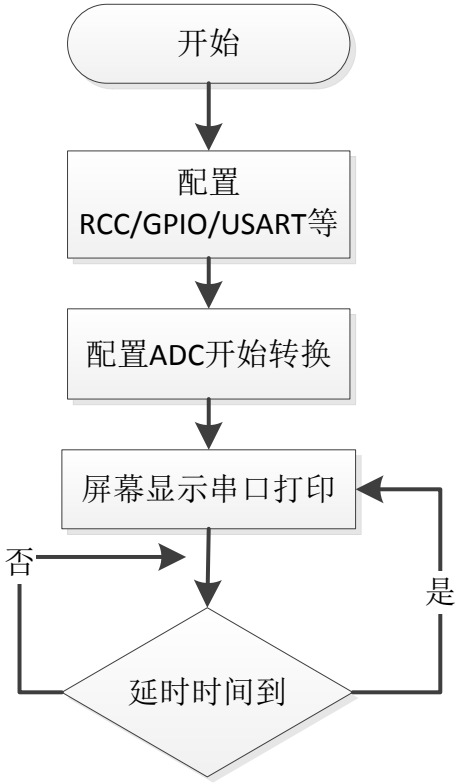

图 4.4 A/D 单通道采集电压实验流程图

2. GPIO_InitTypeDef 为 GPIO 的初始化结构体，它在库函数当中的定义为：

```
typedef struct
{
    uint16_t GPIO_Pin;//要配置的 GPIO 端口引脚号
    GPIOSpeed_TypeDef GPIO_Speed; //配置 GPIO 口速度
    GPIOMode_TypeDef GPIO_Mode;//配置输入输出模式
}GPIO_InitTypeDef;
```

GPIO 的输入输出模式如表 4.2 所列[2]。

表 4.2 GPIO 的输入输出模式

输入输出模式	解释
GPIO_Mode_AIN	模拟输入
GPIO_Mode_IN_FLOATING	浮空输入
GPIO_Mode_IPD	下拉输入
GPIO_Mode_IPU	上拉输入
GPIO_Mode_Out_OD	开漏输出
GPIO_Mode_Out_PP	推挽输出
GPIO_Mode_AF_OD	复用开漏输出
GPIO_Mode_AF_PP	复用推挽输出

3. ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)函数的作用是根据 ADC 初始化结构体对所选用的 ADC 进行初始化工作。
4. void ADC_Cmd(ADC_TypeDef* ADCx, FunctionalState NewState)函数的作用是使能指定的 ADC.
5. void ADC_ResetCalibration(ADC_TypeDef* ADCx) 和 FlagStatus ADC_GetResetCalibrationStatus(ADC_TypeDef* ADCx)函数的作用分别是使能指定 ADC 的复位校准和获取复位校准状态。
6. void ADC_StartCalibration(ADC_TypeDef* ADCx) 和 FlagStatus ADC_GetCalibrationStatus(ADC_TypeDef* ADCx)函数的作用分别是开启 ADC 校准和获取校准状态。
7. void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)函数的作用是设置指定 ADC 的规则组通道的采样时间等。
8. void ADC_SoftwareStartConvCmd(ADC_TypeDef* ADCx, FunctionalState NewState)函数的作用是使能指定 ADC 的软件启动转换。
9. FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG)函数的作用是获取转换标志位状态。
10. uint16_t ADC_GetConversionValue(ADC_TypeDef* ADCx)函数的作用是获取转换结果。
11. void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState)函数的作用是时钟使能。

4.2.3 实验内容和步骤

按建立模版工程的方法创建 STM32 微处理器 A/D 单通道采集电压实验工程，建好工程后开始 ADC 单通道采集电压程序的编写[17]。

这里对 ADC 单通道采集电压程序的编写做重点介绍。

1. 创建 ADC 的初始化程序 void Adc_Init(void)。

在 ADC 的初始化程序 void Adc_Init(void)里要利用 ADC 的配置结构体还有 GPIO 配置结构体对 ADC 还有相应的 GPIO 进行初始化，使能相应的外设时钟，代码段如下：

(1) 使能 ADC1 通道时钟。

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC
|RCC_APB2Periph_ADC1, ENABLE );
//设置 ADC 分频因子 6 72M/6=12,ADC 最大时间不能超过 14M
RCC_ADCCLKConfig(RCC_PCLK2_Div6);
```

(2) GPIO 进行初始化。

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;  //模拟输入引脚
GPIO_Init(GPIOC, &GPIO_InitStructure);
```

(3) ADC1 进行初始化。

```
ADC_DeInit(ADC1);//复位 ADC1,全部寄存器重设为缺省值
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //独立模式
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //单通道模式
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;//连续转换
ADC_InitStructure.ADC_ExternalTrigConv=ADC_ExternalTrigConv_No
ne;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//右对齐
ADC_Init(ADC1, &ADC_InitStructure);//初始化 ADC1
ADC_Cmd(ADC1, ENABLE);//使能指定的 ADC1
ADC1 校准。
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
```

2. 获取采样值程序 u16 Get_Adc(u8 ch)。

(1) 设置 ADC1 的规则组通道，采样时间等：

```
ADC_RegularChannelConfig(ADC1, ch, 1,ADC_SampleTime_239Cycles5);
```

(2) 开始 ADC 转换，取得转换值：

```
ADC_SoftwareStartConvCmd(ADC1, ENABLE);
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));//等待转换结束
return ADC_GetConversionValue(ADC1);//返回最近一次转换结果
```

3. 简单数字滤波程序 u16 Get_Adc_Average(u8 ch,u8 times)。

对连续取得的 times 个值进行平均，可以滤除一些突变的噪声干扰，适用于缓慢变化的信号。

```
for(t=0;t<times;t++)
    temp_val+=Get_Adc(ch);
return temp_val/times;
```

所有程序都编写完成后，将工程进行编译链接，生成 hex 文件。

4.2.4 实验验证与分析

接好 USB 转串口线还有 JLINK,直接使用 JLINK 进行程序的烧写。

将 USB 转串口线接到 STM32 微处理器实验板的串口上，当程序烧写完成后会看到 LCD 屏幕上开始显示采集到的电压的值和电压值变化曲线，打开电脑端匹配的简易串口示波器，设定波特率为 9600，就可以在电脑端看到同样的变化曲线还有采集到的数据。调节滑动变阻器的阻值大小可以看到采集到的电压值的改变和电压曲线的相应变化。

程序的运行效果如图 4.5 和图 4.6 所示。

从图 4.5 和图 4.6 中可以看出，在 LCD 屏和串口示波器上可以显示采集到的电压的实时值和实时的电压曲线。

当调节滑动变阻器时，LCD 屏和串口示波器的显示结果也会跟着变化，实验正确完成。

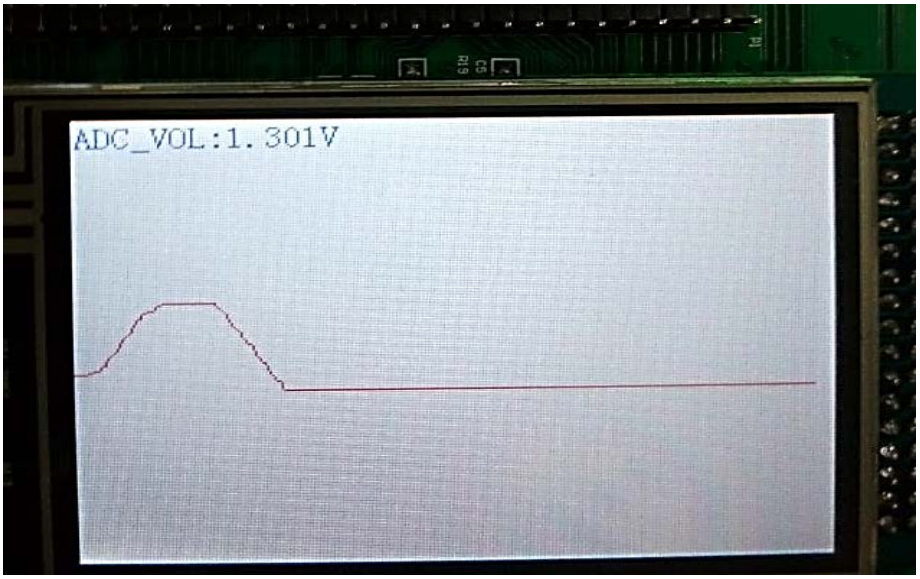

图 4.5　A/D 单通道采集电压实验 LCD 显示结果

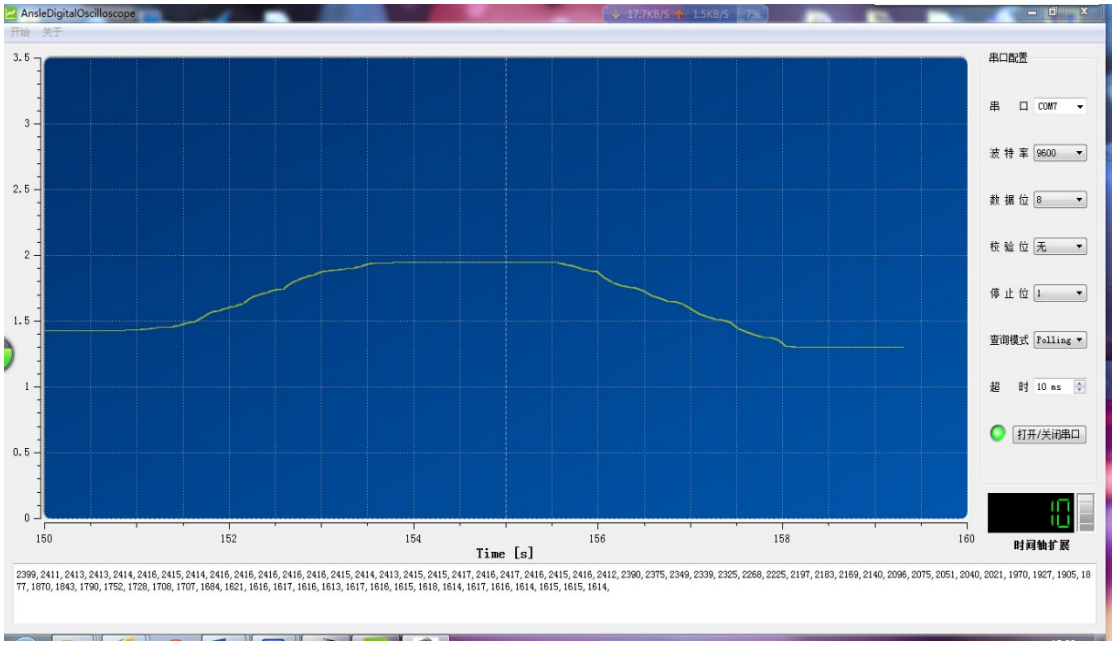

图 4.6　A/D 单通道采集电压实验串口示波器显示结果

4.3　STM32 微处理器 A/D 通道采集正弦波实验

4.3.1　实验目的

掌握利用 DMA 方式进行 STM32 内部 ADC 的操作方法。利用 ADC1 的通道 13 配合 DMA 功能实现小于 3Hz 的正弦波的采集，ADC1 的通道 13 使用连续转换模式进行波形采样将采集到的结果显示到 LED 屏，并通过串口传输到上位机显示波形和采集的数据。

掌握 STM32 内部 ADC 的模拟看门狗的使用，设置模拟看门狗，当被采样的信号大于设定的上下阈值时产生中断，编写中断服务函数。

4.3.2 实验原理

实验配置有电平抬升电路，它的作用是将正弦信号抬升到 0V 以上，以满足 ADC 对于输入电压的要求。这里我们选择用一个滑动变阻器对输入正弦信号的抬升电平进行动态调整，同时要求输入正弦信号的幅度在-1V 到 1V 之间，以防止电压过大烧毁内部 ADC 而造成相应的损失。

实验的原理图如图 4.7 所示。

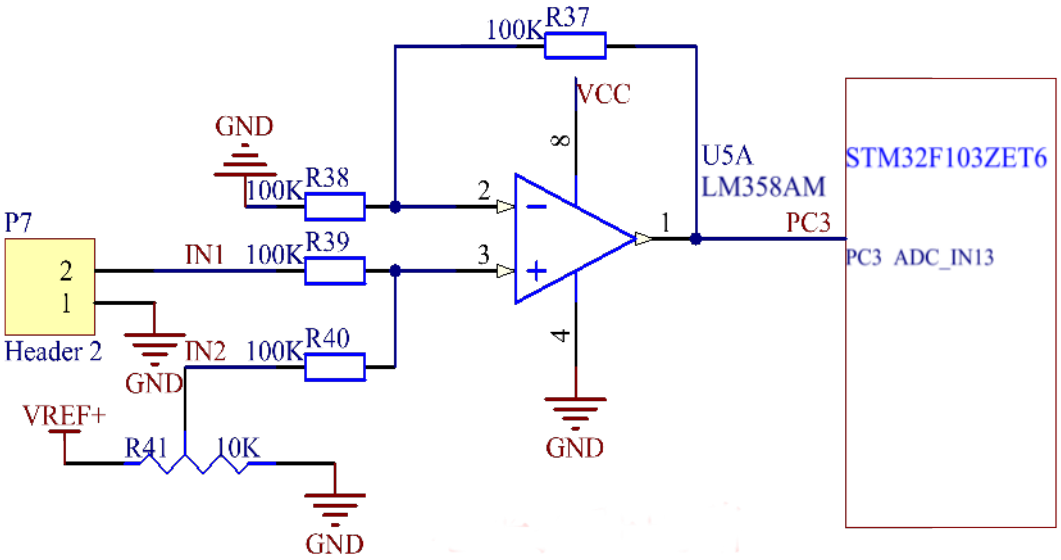

图 4.7 电平抬升电路原理图

本实验设计使用 ADC1 的通道 13 进行 A/D 转化,让其工作在单通道、独立、连续转化模式，设定 ADC1 的时钟为系统时钟的 6 分频即 12MHz，采样时间为 239.5 周期。STM32 内部的 ADC 使用若干个 ADC_CLK 周期对输入电压采样，每个通道可以设置不同的采样时间，ADC 总转换时间 T_{CONV} 按如下公式进行计算[18]：

$$T_{CONV} = \text{采样时间} + 12.5\text{个周期} \quad (4\text{-}1)$$

那么我们设定的总的转换时间应该为 239.5+12.5 = 252 周期 = 21μs，也就是理论上说这个配置可以达到 47kHz 左右的采样率，但是在实际应用中各种处理函数所消耗的时钟周期远大于这个数字，所以实际的采样频率并不会很高。

通过实验验证，在本实验使用的算法不变的前提下，要使 LCD 屏幕和上位机的串口示波器都可以显示稳定清晰的波形，输入正弦信号的频率不应大于 3Hz。

实验的软件流程图如图 4.8 所示。

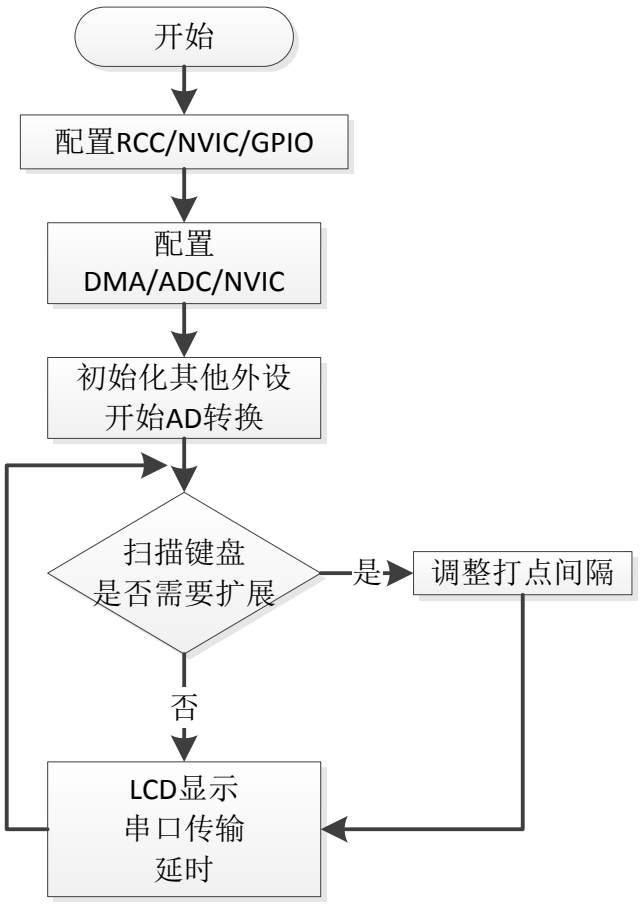

图 4.8 A/D 通道采集正弦波实验流程图

在开始介绍库函数之前，首先对 DMA 做一个简单的介绍[2]：

DMA[2],直接存储器存取，用来提供在外设和存储器之间或者存储器和存储器之间的高速数据传输。无须 CPU 干预，数据可以通过 DMA 快速地移动，这就节省了 CPU 的资源来做其他操作。

STM32 内部两个 DMA 控制器有 12 个通道(DMA1 有 7 个通道，DMA2 有 5 个通道)，每个通道专门用来管理来自于一个或多个外设对存储器访问的请求。还有一个仲裁器来协调各个 DMA 请求的优先权。

接下来我们介绍一下一些使用到的必要的库函数和库定义[2]，对于在上一个实验中介绍到的，这里不再赘述。

1. DMA_InitTypeDef 是 DMA 初始化结构体，它在库函数中的定义为：

```
typedef struct
{
  uint32_t DMA_PeripheralBaseAddr;// 定义 DMA 外设基地址
```

```
    uint32_t DMA_MemoryBaseAddr; //定义 DMA 内存基地址
    uint32_t DMA_DIR;//外设是作为数据传输的目的地还是来源
    uint32_t DMA_BufferSize;//定义指定 DMA 通道的缓存的大小
    uint32_t DMA_PeripheralInc;// 设定外设地址寄存器递增与否
    uint32_t DMA_MemoryInc;// 设定内存地址寄存器递增与否
    uint32_t DMA_PeripheralDataSize;// 设定外设数据宽度
    uint32_t DMA_MemoryDataSize; //设定内存数据宽度
    uint32_t DMA_Mode;//设定工作模式
    uint32_t DMA_Priority;//设置优先级
    uint32_t DMA_M2M;// DMA 通道的内存到内存传输使能与否。
  }DMA_InitTypeDef;
```

相关参数在官方数据手册有详细说明，这里不再赘述。

2. void DMA_DeInit(DMA_Channel_TypeDef* DMAy_Channelx)函数的作用是复位指定的 DMA 通道为默认值，即将所有的寄存器设为默认值。
3. void DMA_Cmd(DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState) 函数的作用是使能指定的 DMA 通道或者关闭指定的 DMA 通道。
4. NVIC_InitTypeDef 是中断控制器的初始化结构体，它的详细定义为：

```
typedef struct
{
  uint8_t NVIC_IRQChannel;//指明中断类型
  uint8_t NVIC_IRQChannelPreemptionPriority; // 指定抢占式优先级别
  uint8_t NVIC_IRQChannelSubPriority;// 指定响应优先级别
  FunctionalState NVIC_IRQChannelCmd; //中断使能
} NVIC_InitTypeDef;
```

5. void NVIC_PriorityGroupConfig(uint32_t NVIC_PriorityGroup)函数的作用是设置中断优先级分组。
6. void NVIC_Init(NVIC_InitTypeDef* NVIC_InitStruct)函数的作用是使用指定的参数初始化中断控制器。
7. void ADC_DeInit(ADC_TypeDef* ADCx)函数的作用是复位指定的 ADC，将

其对应寄存器值设为默认值。

8. void ADC_DMACmd(ADC_TypeDef* ADCx, FunctionalState NewState)函数的作用是使能 ADC 的 DMA 功能。
9. void ADC_AnalogWatchdogThresholdsConfig(ADC_TypeDef* ADCx, uint16_t HighThreshold, uint16_t LowThreshold)函数的作用是指定 ADC 模拟看门狗的阈值。
10. void ADC_AnalogWatchdogSingleChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel)函数的作用是指定模拟看门狗的监控通道。
11. void ADC_AnalogWatchdogCmd(ADC_TypeDef* ADCx, uint32_t ADC_AnalogWatchdog)函数的作用是使能指定 ADC 的模拟看门狗。
12. void ADC_ITConfig(ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState)函数的作用是配置指定 ADC 的模拟看门狗的中断使能。

4.3.3 实验内容和步骤

按建立模版工程的方法创建 STM32 微处理器 A/D 通道采集正弦波实验工程，建好工程后开始 ADC 利用 DMA 单通道连续采集正弦波程序的编写。

这里对 ADC 利用 DMA 单通道连续采集正弦波程序的编写做重点介绍。

1. 创建 ADC 的初始化程序 void Adc_Init(void)。

该函数与上一个实验的 void Adc_Init(void)函数在大部分地方都是相似的，只是这里增加了 ADC 的 DMA 功能还有模拟看门狗功能，下面先对 ADC 的 DMA 配置做详细介绍。

(1) 定义 ADC 寄存器的地址和采样数据的存取变量，代码如下：

```
#define ADC1_DR_Address ((u32)0x4001244C)
__IO uint16_t ADC_ConvertedValue;
```

(2) 配置 DMA，相关代码如下:

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
DMA_InitStructure.DMA_PeripheralBaseAddr=ADC1_DR_Address;
DMA_InitStructure.DMA_MemoryBaseAddr=(u32)&ADC_ConvertedVal
ue;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
```

```
DMA_InitStructure.DMA_BufferSize = 1;
DMA_InitStructure.DMA_PeripheralInc=DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize
_HalfWord;
DMA_InitStructure.DMA_MemoryDataSize=DMA_MemoryDataSize_Ha
lfWord;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;//循环传输
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
```

(3) 根据指定的 DMA 初始化结构体初始化 DMA 通道 1，使能 DMA 通道 1，并开启 ADC1 的 DMA 功能，代码如下：

```
DMA_Init(DMA1_Channel1, &DMA_InitStructure);
DMA_Cmd(DMA1_Channel1, ENABLE); // 使能 DMA channel1
ADC_DMACmd(ADC1, ENABLE);
```

2. 接下来对 ADC 的对模拟看门狗的设置做简单的介绍。

(1) 设定模拟看门狗的上下阈值，并指定其所要监控的 ADC 的通道，因为 ADC 的输入电压要求是在 0V 到 3.3V 之间，所以设定模拟看门狗的上下阈值分别为 4095 和 0，代码如下：

```
ADC_AnalogWatchdogThresholdsConfig(ADC1, 4095, 0);
ADC_AnalogWatchdogSingleChannelConfig(ADC1,ADC_Channel_13);
```

(2) 使能指定 ADC 的模拟看门狗,并开启指定 ADC 的模拟看门狗中断：

```
ADC_AnalogWatchdogCmd(ADC1,ADC_AnalogWatchdog_SingleRegEn
able);
ADC_ITConfig(ADC1, ADC_IT_AWD, ENABLE);
```

3. 中断控制器配置函数 void NVIC_Configuration(void)。

该函数是为了配置和管理 ADC 的模拟看门狗中断，设置中断优先级组别以及 ADC 中断的优先级，部分代码如下：

```
NVIC_InitTypeDef NVIC_InitStructure;
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
```

```
NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

4. ADC1 的中断服务程序完成当输入电压超出模拟看门狗的阈值的时候的处理工作，其部分代码如下：

```
void ADC1_2_IRQHandler()
{
   ADC_ITConfig(ADC1,ADC_IT_AWD,DISABLE);//关闭 ADC 中断
   while(SET == ADC_GetFlagStatus(ADC1,ADC_FLAG_AWD))
   {
      LED1=!LED1;
      ADC_ClearFlag(ADC1,ADC_FLAG_AWD);//清除标志位
      ADC_ClearITPendingBit(ADC1,ADC_IT_AWD);
   }
   ADC_ITConfig(ADC1,ADC_IT_AWD,ENABLE);//开启 ADC 中断
}
```

所有程序都编写完成后，将工程进行编译链接，生成 hex 文件。

4.3.4 实验验证与分析

接好 USB 转串口线还有 JLINK,直接使用 JLINK 进行程序的烧写。

将信号发生器打开，调节输出 2Hz 左右的正弦波，将信号发生器的输出信号接到电平抬升电路的输入端,用 USB 转串口线连接电脑和 STM32 微处理器实验板，当程序烧写完成后会看到 LCD 屏幕上开始显示采集到的正弦信号曲线，打开电脑端匹配的简易串口示波器就可以在电脑端看到同样的正弦信号曲线还有采集到的数据，同时可以观察信号的电压值。

通过实验板上的第一行和第二行按键还可以调节 LCD 屏幕的时间轴扩展，即在时间轴上放大和缩小信号。

本实验的实际的效果图如图 4.9 和 图 4.10 所示。

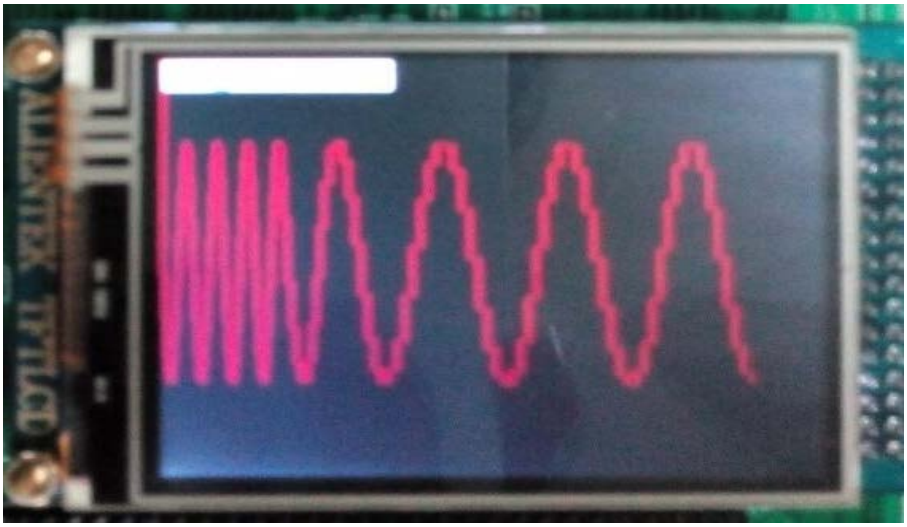

图 4.9 A/D 通道采集正弦波实验 LCD 显示图

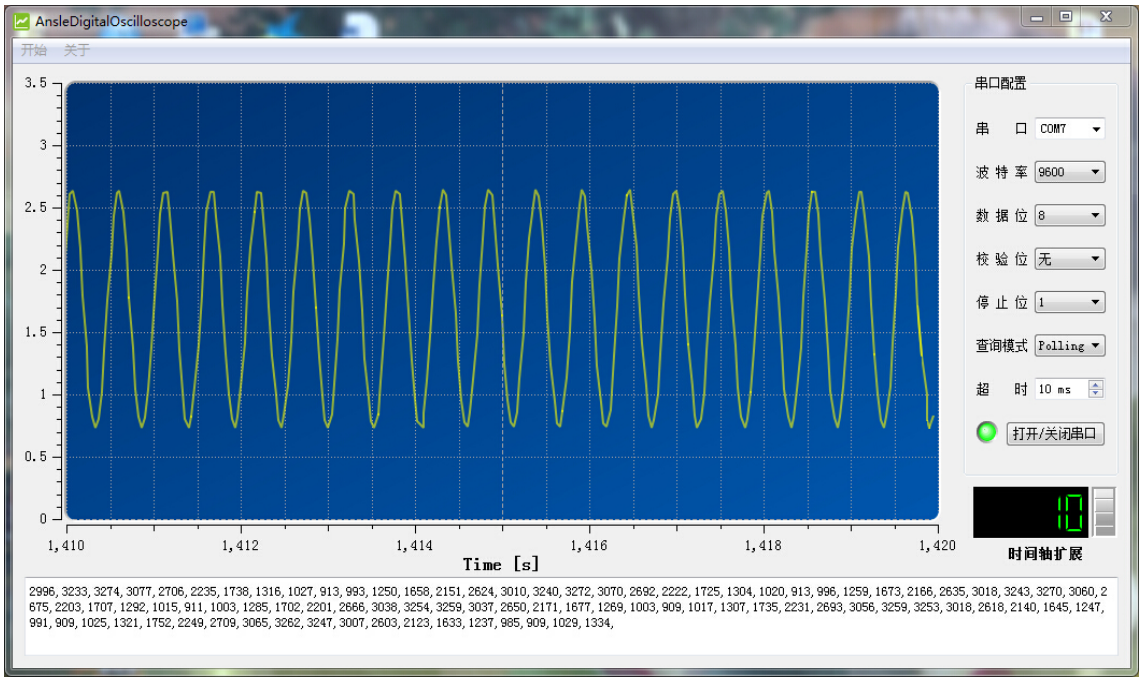

图 4.10 A/D 通道采集正弦波实验串口示波器显示图

通过对实验结果的观察可知，在 LCD 屏和串口示波器上可以显示采集到的正弦波的实时电压值，并可以画出采集到的正弦波的波形曲线，当调节实验板或者串口示波器的时间轴扩展时，波形就会在时间轴进行放大或缩小，实验正确完成。

4.4 STM32 微处理器 D/A 通道产生模拟电压实验

4.4.1 实验目的

熟悉使用 STM32 内部的 DAC 输出简单幅度可调的模拟电压的方法。

利用 STM32 内部的 DAC 的通道 1 输出模拟电压信号，再通过 ADC1 的通道 0 进行采样，来验证输出的电压是否为正确的。

利用按键第一行与第二行可以调节 DAC 电压的输出值，每按一次按键电压输出值的调节步长为(200*3.3/4096)V，电压的最大输出值为(4000*3.3/4096)，最小输出电压值为 0V。

4.4.2 实验原理

实验板上的 DAC 接口电路引出了 STM32 内置 DAC 的两个通道，由第二章相关 STM32 内部 DAC 介绍可知，一旦使能 DAC 的某个通道，相应的 GPIO 引脚（PA4 或者 PA5）就会自动与 DAC 的模拟输出相连(DAC_OUTx)。为了避免寄生的干扰和额外的功耗，引脚 PA4 或者 PA5 在之前应当设置成模拟输入(AIN)。本实验当中我们使用 STM32 内置 DAC 的通道 1 进行实验，所以在进行 DAC 的初始化时，应将 GPIO 口 PA4 设为模拟输入。

指定的数字量经过 DAC 将被线性地转换为模拟电压输出，其范围为 0 到 V_{REF+}。任何一个 DAC 通道引脚上的输出电压与指定输出量 DOR 满足下面的关系[2]：

$$DAC输出 = V_{REF+} * (DOR / 4095) \quad (4\text{-}2)$$

即如果 DOR=2047，那么 DAC 输出的电压值为 3.3*（2047/4095）= 1.65V。

整个实验设计的软件流程图如图 4.11 所示。

下面来介绍一下所要用到的库函数和库定义[11]：

1. DAC_InitTypeDef 是 DAC 初始化结构体，其定义如下所示：

```
typedef struct
{
    uint32_t DAC_Trigger;//DAC 促发方式
    uint32_t DAC_WaveGeneration;//波形发生使用与否
    uint32_t DAC_LFSRUnmask_TriangleAmplitude;// 幅值设置
    uint32_t DAC_OutputBuffer;// DAC1 输出缓存
}DAC_InitTypeDef;
```

2. void DAC_Init(uint32_t DAC_Channel, DAC_InitTypeDef* DAC_InitStruct) 函数的作用是根据指定的初始化类型来初始化 DAC。
3. void DAC_SetChannel1Data(uint32_t DAC_Align, uint16_t Data)函数的作用

是设置 DAC 的输出值。

4. uint16_t DAC_GetDataOutputValue(uint32_t DAC_Channel)函数的作用是获取已经设置的 DAC 的某个通道的输出值。
5. void RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState)函数的作用是使能挂载在 APB1 上的外设的时钟。

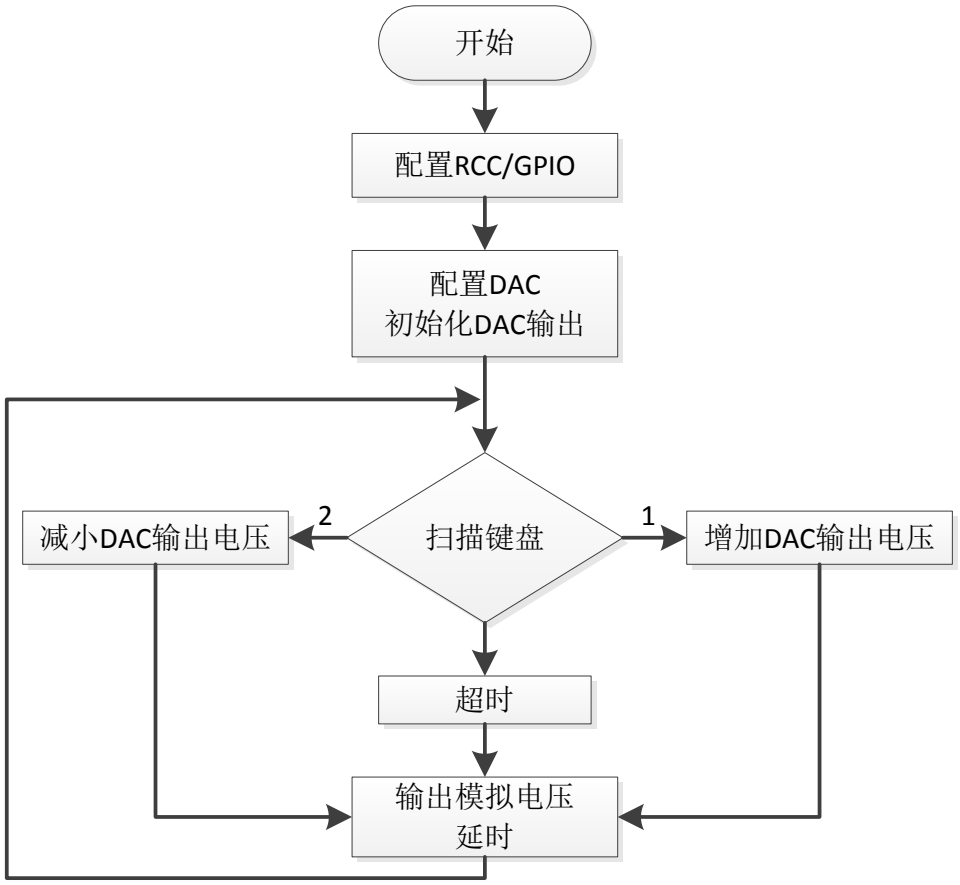

图 4.11 D/A 通道产生模拟电压软件流程图

4.4.3 实验内容和步骤

首先按建立模版工程的方法创建 STM32 微处理器 D/A 通道产生模拟电压实验的工程，建好工程后开始 DAC 产生模拟电压 ADC 采集后显示输出值的程序的编写。

这里对 DAC 产生模拟电压 ADC 采集后显示输出值的程序的编写做重点介绍：

1. DAC 通道 1 输出初始化函数 void Dac1_Init(void)。

   在这个函数里对 DAC 通道 1 进行一系列的初始化操作，包括：

   (1) 使能 PA4 口的时钟并先配置 PA4 口为模拟输入，代码如下：

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE );
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE );
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;// 端口配置
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;//模拟输入
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

(2) 配置 DAC 初始化结构体，部分相关代码如下：

```
DAC_InitType.DAC_Trigger=DAC_Trigger_None;//不使用触发功能
//不使用波形发生
DAC_InitType.DAC_WaveGeneration=DAC_WaveGeneration_None;
//屏蔽幅值设置
DAC_InitType.DAC_LFSRUnmask_TriangleAmplitude=DAC_LFSRUn
mask_Bit0;
DAC_InitType.DAC_OutputBuffer=DAC_OutputBuffer_Disable;
DAC_Init(DAC_Channel_1,&DAC_InitType);
DAC_Cmd(DAC_Channel_1, ENABLE); //使能 DAC 通道 1
DAC_SetChannel1Data(DAC_Align_12b_R, 0);//输出值设置
```

2. 输出模拟电压的调整程序代码段主要完成键盘的扫描工作，根据键盘扫描的结果对 DAC 通道 1 输出的电压值进行更改，代码如下：

```
key=KEY_Scan(0);
if(key==1) //增加按钮按下
{
    if(dacval<4000)dacval+=200;
    DAC_SetChannel1Data(DAC_Align_12b_R, dacval);//设置 DAC 值
}
else if(key==2)//减小按钮按下
{
    if(dacval>200)dacval-=200;
        else dacval=0;
    DAC_SetChannel1Data(DAC_Align_12b_R, dacval);//设置 DAC 值
```

```
}
```

ADC 相关部分的代码与前面 ADC 采集模拟电压的实验的 ADC 相关代码基本相似，这里不再赘述。

所有程序都编写完成后，将工程进行编译链接，生成 hex 文件。

4.4.4 实验验证与分析

首先，接好 USB 转串口线还有 JLINK,直接使用 JLINK 进行程序的烧写。

然后，用杜邦线将 DAC 的通道 1 的输出端口 PA4 与 ADC1 的通道 0 的输入端口 PA0 连接起来，当程序烧写完成后会看到 LCD 屏幕上需要输出的模拟电压的值还有经过 A/D 转换以后的采集到的电压值。

通过实验板上的第一行和第二行按键可以调节 DAC 输出电压的大小，调节的步长为每按第一行的任意按键一次，输出电压增加(200*3.3/4096)V, 每按第二行的任意按键一次，输出电压减少(200*3.3/4096)V。

本实验的实际效果图如图 4.12 所示。

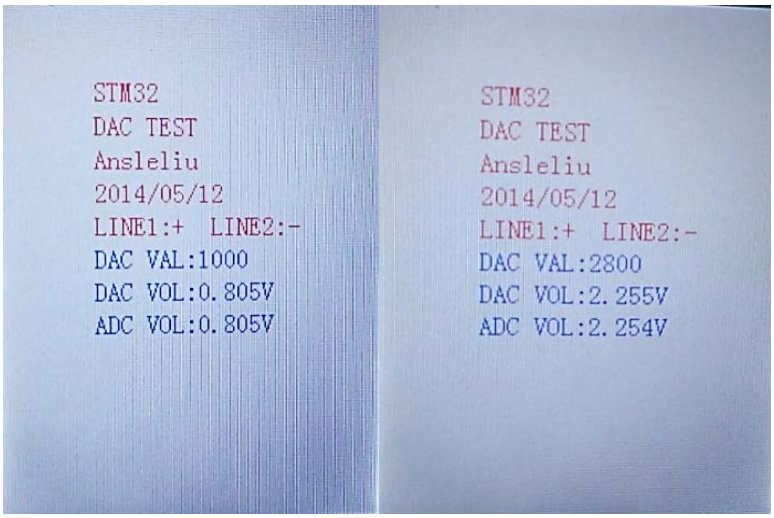

图 4.12 D/A 通道产生模拟电压实验效果图

由实验结果可以看到，在 LCD 屏幕上显示了 DAC 的输出值和 ADC 采样得到的值，当使用键盘调节 DAC 的输出时，ADC 采样得到的电压值也会跟着变化，而且两个值可以很好的保持一致，实验正确完成。

4.5 STM32 微处理器 D/A 通道产生三角波实验

4.5.1 实验目的

掌握使用 STM32 内部 DAC 的波形发生功能配合定时器产生三角波。

掌握 STM32 的定时器配置和定时器中断处理，以及 DMA 的中断处理。

利用 STM32 内部的 DAC 的两个通道产生三角波，一个通道接 ADC1 的通道 0，将输出的三角波采样后显示到 LCD 屏幕上，另一个通道接示波器，通过示波器观察 DAC 输出的三角波波形。

本实验使 STM32 内部 DAC 产生两路最大输出电压都为 2047*3.3/4096V，最小输出电压都为 1023*3.3/4096V 的三角波。设定输出三角波的频率为 1.76kHz。

4.5.2 实验原理

STM32 内置 DAC 可以在 DC 或者缓慢变化的信号上加上一个小幅度的三角波。

可以设置 WAVEx[1:0]位为‘10’选择 DAC 的三角波生成功能，另外设置 DAC_CR 寄存器的 MAMPx[3:0]位可以选择三角波的幅度。内部的三角波计数器每次触发事件之后 3 个 APB1 时钟周期后累加 1。计数器的值与 DAC_DHRx 寄存器的数值相加并丢弃溢出位后写入 DAC_DORx 寄存器。在传入 DAC_DORx 寄存器的数值小于 MAMP[3:0]位定义的最大幅度时，三角波计数器逐步累加。一旦达到设置的最大幅度，则计数器开始递减，达到 0 后再开始累加，周而复始。将 WAVEx[1:0]位置‘0’可以复位三角波的生成[2]。图 4.13 展示了 DAC 三角波的生成的过程与原理，图 4.14 展示了带三角波生成的 DAC 转换的时序图。

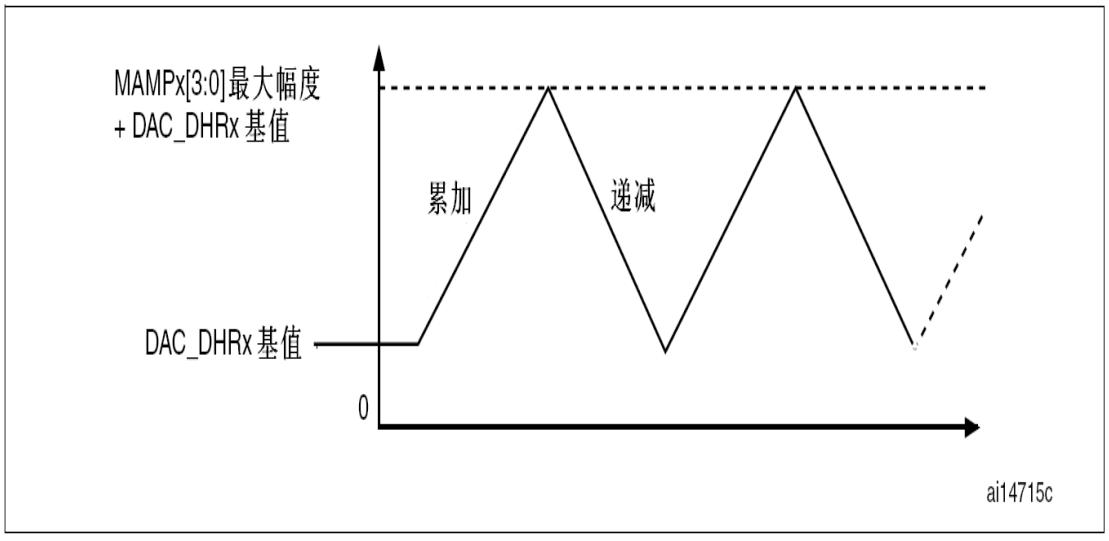

图 4.13 DAC 三角波生成

为了产生三角波，必须使能 DAC 触发，即设 DAC_CR 寄存器的 TENx 位为

‘1’。MAMP[3:0]位必须在使能 DAC 之前设置，否则其值不能修改。

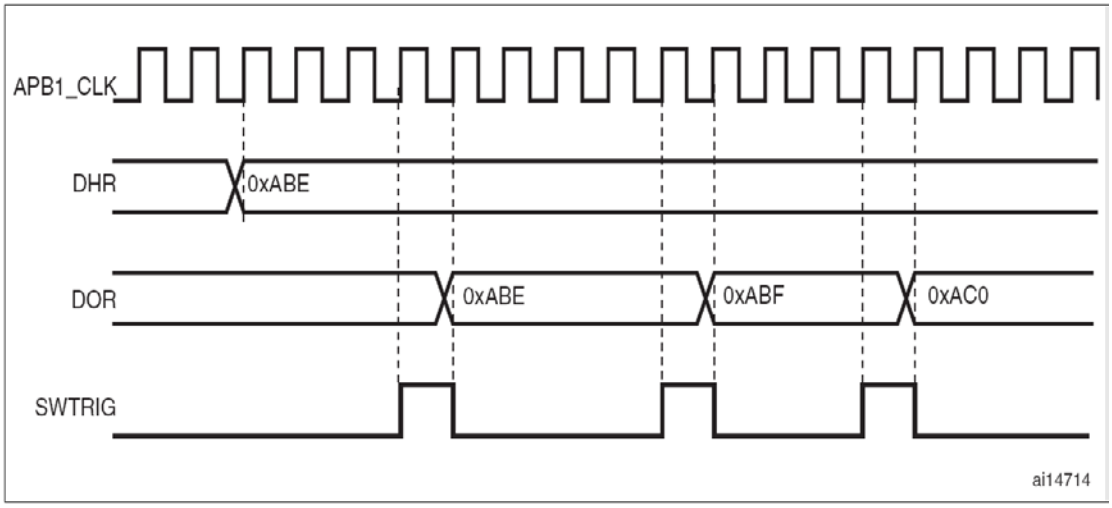

图 4.14 带三角波生成的 DAC 转换(使能软件触发)

如果通过库函数来操作 DAC 产生三角波，所有这些相关的的寄存器都不需要深入的去了解，只需要理解相关库函数的使用即可。

DAC 利用定时器来控制输出三角波的频率，输出三角波的频率 f_{out} 与定时器的计满值 Period、预分频 Prescaler 还有设定的三角波的幅度 Amplitude 有关，他们的关系满足下面公式：

$$f_{out} = 72000000 / ((Period + 1) * (Prescaler + 1)) / (2 * \text{Amplitude}) \quad (4\text{-}3)$$

那么如果设定定时器的计满值 Period 为 9、预分频 Prescaler 为 0 还有设定的三角波的幅度 Amplitude 为 2047，则从 DAC 输出的三角波的频率即为 1.76kHz。

在本实验里使用 ADC 配合 DMA 功能，先采集 4096 个数据，数据采集完成后 DMA 产生中断，在 DMA 中断服务函数里关闭 DMA 和 ADC，等待处理数据，一个 150ms 的定时器会每 150ms 会产生一个定时器中断，在定时器中断里处理采集到的数据，数据处理完成后再打开 ADC 和 DMA 继续采样，如此往复。

ADC 采集到的三角波的幅度利用在采集到的数据数组里找最大值最小值的方法得到，频率通过查找相邻的两个上升沿获得三角波的周期进而得到其频率。在本程序里，ADC 的采样频率设为 461539Hz。

该实验程序流程图如图 4.15 所示。

下面来介绍一下本实验要用到的相关的库函数和库定义。

1. TIM_TimeBaseInitTypeDef 为定时器初始化结构体，其定义如下：

```
typedef struct
```

```
{
   uint16_t TIM_Prescaler;//设置预分频值
   uint16_t TIM_CounterMode;//设置计数方式
   uint16_t TIM_Period; //设置计满值
   uint16_t TIM_ClockDivision;//分频系数
   uint8_t TIM_RepetitionCounter;//高级定时器用
} TIM_TimeBaseInitTypeDef;
```

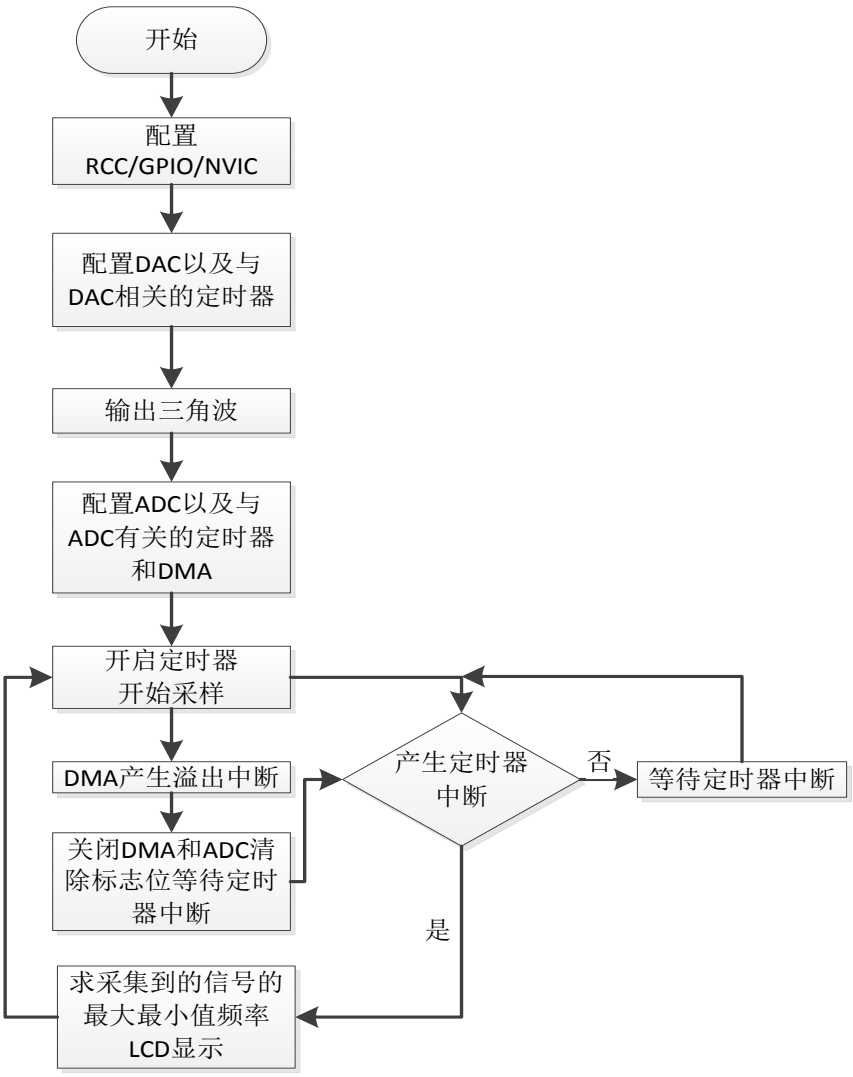

图 4.15 D/A 通道产生三角波软件流程图

2. void TIM_SelectOutputTrigger(TIM_TypeDef* TIMx, uint16_t TIM_TRGOSource)函数的作用是选择促发输出模式。
3. void TIM_TimeBaseInit(TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)函数的作用是根据 TIM_TimeBaseInitStruct 中指定

的参数初始化 TIMx。

4. void TIM_ClearFlag(TIM_TypeDef* TIMx, uint16_t TIM_FLAG)函数的作用是清除定时器标志位。
5. void TIM_ITConfig(TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)函数的作用是使能定时器中断。
6. void TIM_Cmd(TIM_TypeDef* TIMx, FunctionalState NewState)函数的作用是使能定时器。
7. void DAC_SetDualChannelData(uint32_t DAC_Align, uint16_t Data2, uint16_t Data1)函数的作用是设置两个通道的值。
8. void DMA_ITConfig(DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT, FunctionalState NewState)使能 DMA 中断与否。
9. void DMA_ClearFlag(uint32_t DMAy_FLAG)函数的作用是清除 DMA 通道待处理标志位。
10. void DMA_ClearITPendingBit(uint32_t DMAy_IT)函数的作用是清除 DMA 通道中断待处理标志位

其它的使用到的库函数和定义在前面的实验中已经做过介绍，这里不再赘述。

4.5.3 实验内容和步骤

首先按建立模版工程的方法创建 STM32 微处理器 D/A 通道产生三角波实验的工程，建好工程后开始 DAC 产生三角波，ADC 采集后显示输出波形、幅度还有频率的程序的编写。

这里对 DAC 产生三角波 ADC 采集后显示输出波形、幅度还有频率的程序的编写做重点介绍。

1. DAC 产生三角波的设置，需要使能相关外设时钟、配置 GPIO 口、设定定时器、配置 DAC 并将其使能，详细介绍如下：

   (1) 使能时钟:

   ```
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
   ```

   (2) DAC 的两个通道要挂载的 PA4 和 PA5 口的配置：

```
GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_4 | GPIO_Pin_5;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

(3) 设置定时器，定时器的配置关系到输出三角波的频率：

```
TIM_TimeBaseStructure.TIM_Period = 9; //计满值
TIM_TimeBaseStructure.TIM_Prescaler = 0;//预分频
TIM_TimeBaseStructure.TIM_ClockDivision = 0x0;分频系数
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
TIM_SelectOutputTrigger(TIM4, TIM_TRGOSource_Update);
```

(4) DAC 通道 l1 配置，设置其为波形发生模式，并指定其输出三角波的幅度的大小：

```
DAC_InitStructure.DAC_Trigger = DAC_Trigger_T4_TRGO;//定时器促发
//使用波形发生
DAC_InitStructure.DAC_WaveGeneration=DAC_WaveGeneration_Triangle;
DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude=DAC_TriangleAmplitude_2047;//设置输出幅度
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Disable;
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude=DAC_TriangleAmplitude_2047; /设置输出幅度
```

(5) 使能 DAC，使能定时器：

```
DAC_Init(DAC_Channel_2, &DAC_InitStructure);
DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_Cmd(DAC_Channel_2, ENABLE);
DAC_SetDualChannelData(DAC_Align_12b_R, 1024, 1024);
TIM_Cmd(TIM4, ENABLE); //使能定时器 4
```

2. DMA 及其中断处理。

   在上面实验的基础上，这里的 DMA 部分有一些变化：

   (1) 要定义 DMA 通道存储器地址为存储 ADC 采样值的数组 u32 ADC_ConvertedValue[4096]，并指定 DMA 缓冲区的大小为 4096：

   DMA_InitStructure.DMA_MemoryBaseAddr = &ADC_ConvertedValue;

   DMA_InitStructure.DMA_BufferSize = 4096;//定义 DMA 缓冲区大小

   (2) 定义外设数据宽度 32 位，定义存储器数据宽度 32 位：

   DMA_InitStructure.DMA_PeripheralDataSize=DMA_PeripheralDataSize_Word;

   DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;

   (3) 使能 DMA 溢出中断：

   DMA_ITConfig(DMA1_Channel1,DMA_IT_TC, ENABLE);

3. 在 DMA 中断处理函数里将 DMA、DMA 中断和 ADC 关闭，并清理标志位，等待数据处理完毕后再次开启。

   DMA_Cmd(DMA1_Channel1, DISABLE);

   ADC_DMACmd(ADC1, DISABLE);

   DMA_ITConfig(DMA1_Channel1,DMA_IT_TC, DISABLE);

   DMA_ClearFlag(DMA1_FLAG_TC1);

   DMA_ClearITPendingBit(DMA1_IT_TC1);

4. 定时器设置。

   (1) 首先配置定时器初始化结构体：

   TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;

   TIM_TimeBaseStructure.TIM_Period = 299;//每 150ms 产生一次中断

   TIM_TimeBaseStructure.TIM_Prescaler = 35999;

   TIM_TimeBaseStructure.TIM_ClockDivision = 0;

   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

   (2) 根据 TIM_TimeBaseInitStruct 中指定的参数初始化 TIM3：

   TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

   TIM_ClearFlag(TIM3, TIM_FLAG_CC1);

(3) 使能定时器的中断和定时器：

```
TIM_ITConfig(TIM3, TIM_IT_CC1, ENABLE);
TIM_Cmd(TIM3, ENABLE);
```

5. 定时器中断处理。

(1) 求输入三角波的最大值最小值：

```
for(i=0;i<4096;i++)
{
    if(ADC_ConvertedValue[i]>max)
        max=ADC_ConvertedValue[i];
    if(ADC_ConvertedValue[i]<min)
        min=ADC_ConvertedValue[i];
}
```

(2) 查找采集到的信号的上升沿求信号的频率：

```
for(i=4;i<2048;i++)
{ //捕捉上升沿
    if((ADC_ConvertedValue[i-1])<average&&
    (ADC_ConvertedValue[i-4])<average&&
    (ADC_ConvertedValue[i+1])>average&&
    (ADC_ConvertedValue[i+4])>average)
    {//找到上升沿
        if(qsw_n==0)
        {//第一个上升沿
            QSW[0]=i;
            qsw_n++;
        }
        else if(qsw_n==1&&(i-QSW[0]>5))
            PL_flag=461539/(QSW[1]-QSW[0]);//求频率
    }
}
```

所有程序都编写完成后，将工程进行编译链接，生成 hex 文件。

4.5.4 实验验证与分析

首先，接好 USB 转串口线还有 JLINK,直接使用 JLINK 进行程序的烧写。

然后，用杜邦线将 DAC 的通道 1 的输出端口 PA4 与 ADC1 的通道 0 的输入端口 PA0 连接起来，将 DAC 的通道 1 的输出端口 PA5 接到示波器上，打开示波器。当程序烧写完成后会看到 LCD 屏幕上输出 ADC 采集到的 DAC 通道 1 输出的三角波的波形、最大最小值还有频率，通过 DAC 通道 2 连接示波器，观察输出的三角波的波形最大最小值还有频率，这样就可以同经 ADC 采集到的 DAC 通道 1 输出的三角波进行比对验证。

通过实验板上的第一行和第二行按键可以调节 LCD 显示波形的时间轴扩展，即使信号在 X 轴方向上放大和缩小。

实验实际的效果图如下图 4.16 和图 4.17 所示。

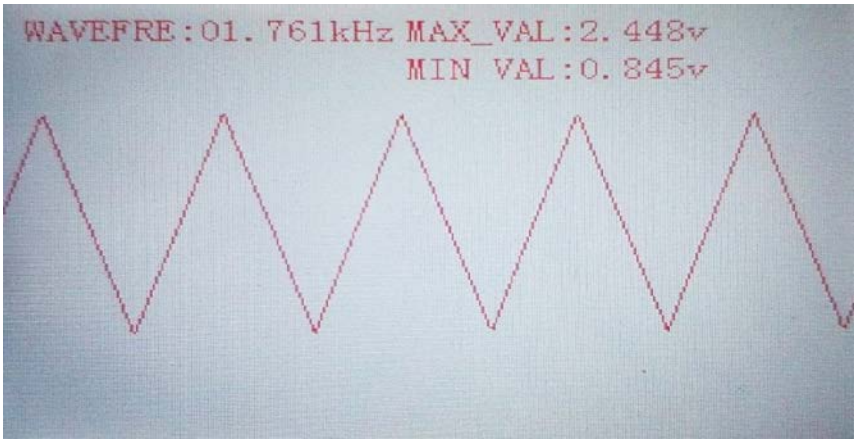

图 4.16 D/A 通道产生三角波实验 LCD 显示结果图

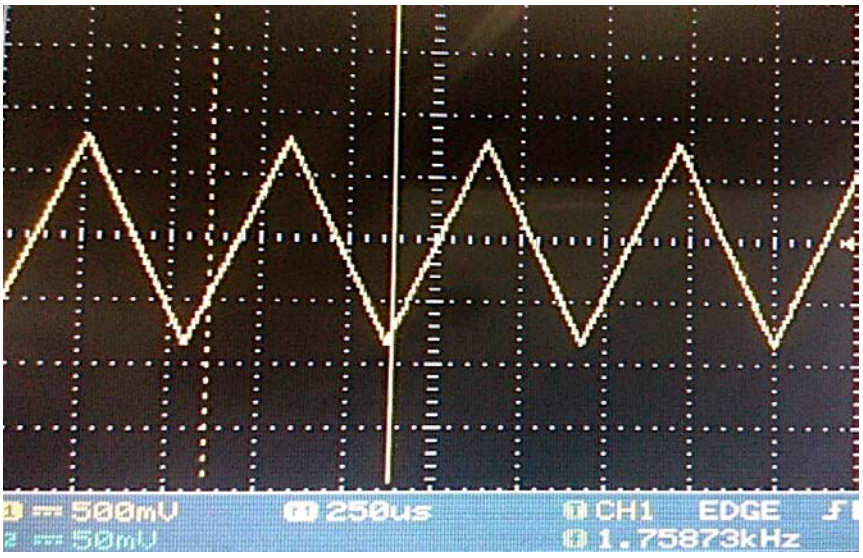

图 4.17 D/A 通道产生三角波实验示波器显示结果图

由实验结果的观察可知，LCD 上显示的 ADC 采集到的 DAC 输出三角波的频率为 1.761kHz 左右，而用示波器观察到的 DAC 输出的三角波的频率为 1.759kHz 左右，两者基本一致，经计算电压幅度也是一致的，实验正确完成。

4.6 STM32 微处理器 A/D 与 D/A 通道的联合实验

4.6.1 实验目的

熟练掌握 STM32 内部 ADC 配合 DMA 功能进行信号采集。

熟练使用 STM32 内部 DAC 配合 DMA 功能和定时器来完成在非波形模式下产生波形。

利用 STM32 内置的 ADC1 通道 0 配合 DMA 采集一路信号发生器产生的方波，然后再通过 DAC 配合 DMA 输出同样频率的方波。

被采集的方波的频率设置为 1.0kHz 到 2.0kHz 之间，峰峰值为 1V。

4.6.2 实验原理

利用 STM32 内部 ADC1 通道 0 采集信号发生器产生的方波的方法跟上一个实验当中 ADC 采样 DAC 输出三角波的方法是一致的，即以 461539Hz 的采样频率去采集从信号发生器产生的方波，每次采集 4096 个点，采集完成后 DMA 产生中断，在 DMA 中断服务函数中 ADC 和 DMA 都关闭并且清除标志位，等待 150ms 的定时器中断服务程序来处理数据。

在定时器中断服务函数当中求采集到的方波的最大值最小值，找到两个上升沿来确定一个完整的方波周期，这样就可以求得采集到的方波的频率，再从这个方波周期里等间隔取 32 个值存放到一个大小为 32 的数组中，该数组是 DAC 产生方波的数据源，DAC 在非波形发生模式下产生的信号的频率 f_{out} 与定时器的计满值 Period、预分频 Prescaler 还有设定的 DAC 输出数据源数据个数 num 有关，他们的关系如下面的计算公式所示：

$$f_{out} = 72000000 / ((Period + 1) * (Prescaler + 1)) / num \quad (4\text{-}4)$$

假设输入的方波的频率为 f_{in}，DAC 定时器的预分频为 0，计满值为 Peri，而 $num = 32$，那么可以根据求得的输入方波的频率来计算 DAC 的定时器的计满值 Peri，他们的关系为：

$$Peri = 2250000 / f_{in} \quad (4\text{-}5)$$

整个 A/D 与 D/A 通道的联合实验的软件流程图如图 4.18 所示。

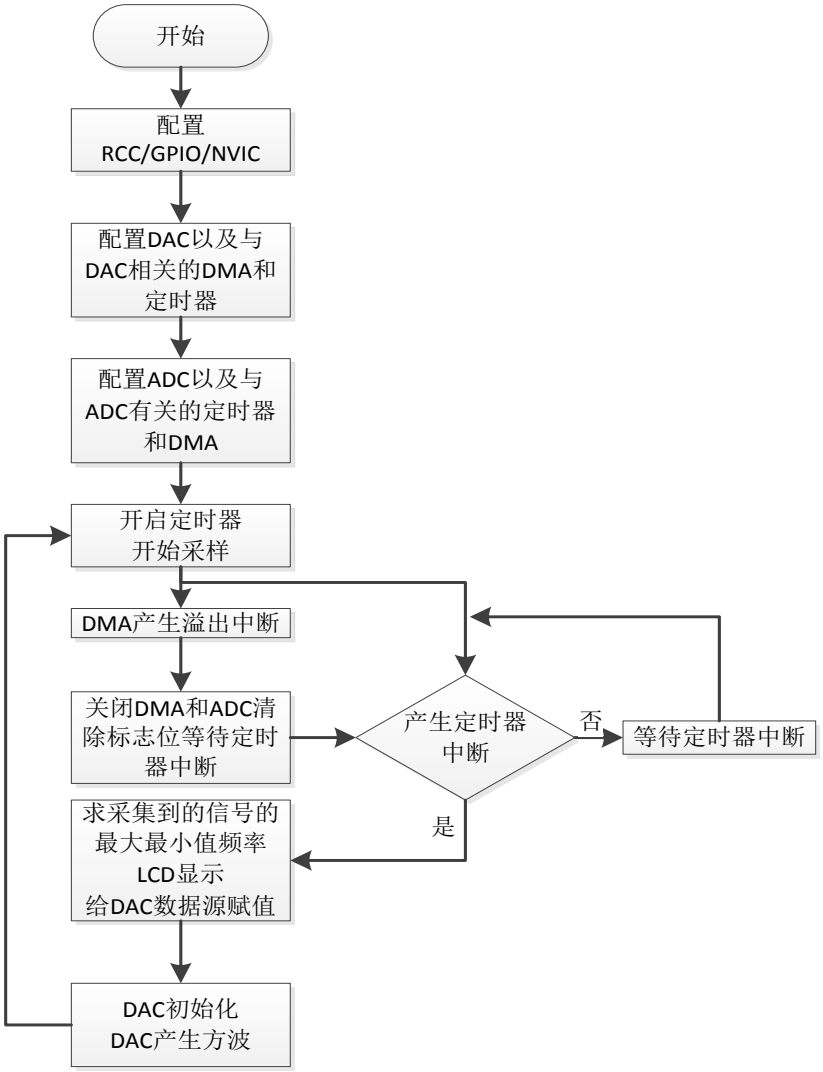

图 4.18 A/D 与 D/A 通道的联合实验软件流程图

本实验用到的库函数和库定义在线面的实验中大多已经说明，这里就不再赘述。

4.6.3 实验内容和步骤

首先按建立模版工程的方法创建 STM32 微处理器 A/D 与 D/A 通道的联合实验的工程，建好工程后开始 ADC1 通道 0 配合 DMA 采集一路方波，然后再通过 DAC 配合 DMA 输出同样频率的方波的程序的编写。

这里对 ADC1 通道 0 配合 DMA 采集一路方波，然后再通过 DAC 配合 DMA 输出同样频率的方波的程序做详细的讲解。

1. 程序中有关于 ADC 采样的程序与上一个实验的基本相同，在其中的定时器中断里增加了 DAC 的定时器计满值计算的代码，给 DAC 数据源赋值代码

以及启动 DAC 输出波形的代码。

(1) DAC 定时器计满值的代码如下:

```
peri=2250000/(461539/(QSW[1]-QSW[0]));
```

(2) 给 DAC 数据源赋值代码如下：

```
num=(QSW[1]-QSW[0])/32;//取值间距
for(lab=0; lab<32;lab++)
{//取值
    Sine12bit[lab]=ADC_ConvertedValue[QSW[0]+lab*num];
}
```

(3) 启动 DAC 输出波形的代码如下：

```
Dac_close();//关闭 DAC 以及与 DAC 有关的 DMA 和定时器
DAC_Mode_Init(peri);//输出波形，内部初始化
```

2. DAC 输出方波结合了 DMA，需要配置 DMA 的外设数据地址为 DAC 寄存器地址，设置 DMA 的内存数据地址为要输出的方波的数据数组的地址，指定数据的传输方向内存至外设，指定缓存大小为 32 字节，相关代码如下：

```
DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR12RD_Address;
DMA_InitStructure.DMA_MemoryBaseAddr= (uint32_t)&DualWave32bit;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
DMA_InitStructure.DMA_BufferSize = 32;
```

3. 下面在配置 DMA 输出的程序里填充正弦波形数据，代码如下所示：

```
for (Idx = 0; Idx < 32; Idx++)
{
    DualWave12bit[Idx]=(Wave32bit[Idx]<<16)+(Wave32bit[Idx]);
}
```

4. 在 DAC 再次输出波形之前先关掉之前的 DAC，并清除所有配置，代码如下所示：

```
TIM_Cmd(TIM2, DISABLE);//关闭定时器
DAC_Cmd(DAC_Channel_1, DISABLE);//关闭通道 1
DAC_Cmd(DAC_Channel_2, DISABLE);//关闭通道 2
DAC_DMACmd(DAC_Channel_2, DISABLE);//关闭 DAC 的 DMA 请求
```

```
DAC_DMACmd(DAC_Channel_1, DISABLE);
DMA_Cmd(DMA2_Channel4, DISABLE);//关闭 DMA
```

4.6.4 实验验证与分析

首先，接好 USB 转串口线还有 JLINK,直接使用 JLINK 进行程序的烧写。

然后，将信号发生器调节到 1kHz 到 2kHz 之间，峰峰值调节为 1V，接入方波信号到电平抬升电路的输入端，同时将示波器也接到电平抬升电路的输入端，以便对比 DAC 输出方波与被 ADC 采集的信号发生器产生的方波。

当程序烧写完成后会看到 LCD 屏幕上输出 ADC 采集到方波的波形、最大值最小值还有频率，通过 DAC 通道 1 连接示波器，观察输出的三角波的波形最大最小值还有频率，这样就可以同经 ADC 采集到的方波进行比对验证。

通过实验板上的第一行和第二行按键可以调节 LCD 显示波形的时间轴扩展，即使信号在 X 轴方向上放大和缩小。

实验实际的效果图如下图 4.19 和图 4.20 所示。

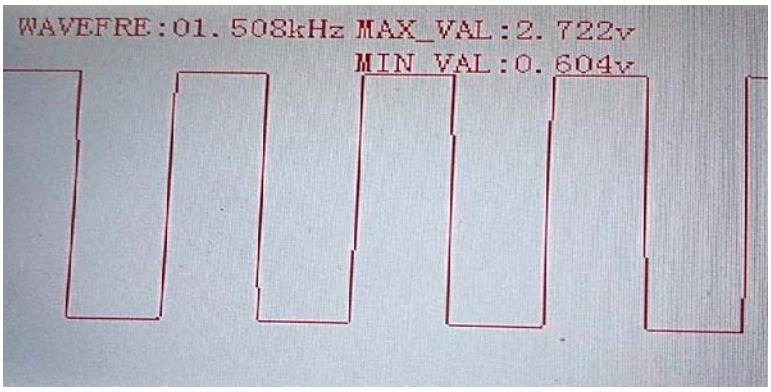

图 4.19 A/D 与 D/A 通道的联合实验 LCD 显示结果图

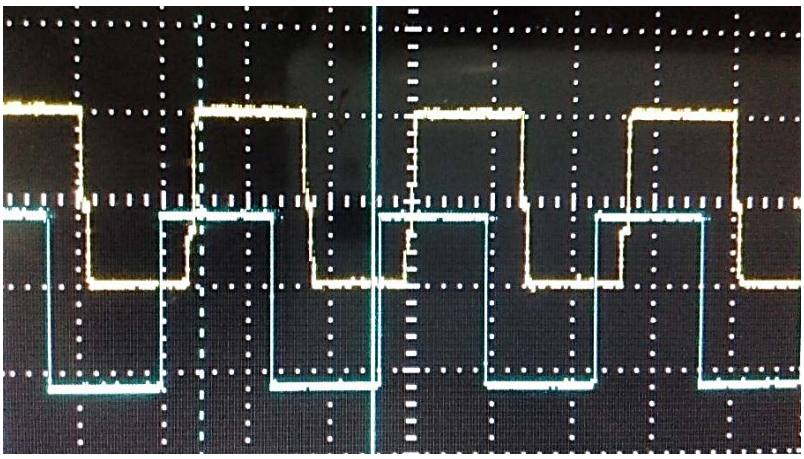

图 4.20 A/D 与 D/A 通道的联合实验示波器显示结果图

由实验结果的观察可知，LCD 现实的 ADC 采集到的方波的频率为 1.508kHz，最大值为 2.722V，最小值为 0.604V，在通过示波器观察得知由 DAC 输出的方波的频率还有幅度大小都与信号发生器输出的无误，所以实验正确完成。

第五章 总结与展望

5.1 论文总结

本文完成了基于 STM32F103ZET6 微处理器的实验平台设计和和 PCB 绘制，以及 STM32 微处理器实验平台当中，有关于 STM32 内部的 ADC 和 DAC 外围电路设计，并为 STM32 微处理器实验平台开发和配置了 STM32 内部的 ADC 和 DAC 相关的应用实验，较好的完成了任务书的要求。

在整个毕业设计的过程中主要做了以下几个方面的工作：

1. 完成了 STM32F103ZET6 微处理器实验平台的 PCB 设计、焊接和调试工作。
2. 完成了基于 STM32F103ZET6 微处理器实验板的 STM32 内置 ADC 和 DAC 的相关实验的设计和实现工作。
3. 编写了大量有对 STM32F103ZET6 的内部 ADC 和 DAC 进行操作的实验代码。
4. 编写了在 PC 上运行的窗口数字示波器，并配合完成 STM32 内置 ADC 和 DAC 的相关实验的设计和验证工作。
5. 阅读了大量关于 STM32 的技术文档，和网络博客，熟练掌握了 STM32 内部 ADC、DAC、DMA、定时器、GPIO 的操作和使用。
6. 熟练掌握了 PCB 的设计和制作，熟练掌握了 Altium Designer 的使用。

经过实验验证，本文相关的各各部分工作都很正常，实验结果直观且准确，基本完成了预定的目标。

通过毕业设计，我学到了关于 PCB 板的设计的知识，同时积累了大量 PCB 设计的经验和教训。加深了对 STM32 微处理器的认识和了解，尤其是对它内部的 ADC 和 DAC 认识的更全面了，同时对于使用 C 语言操作 STM32 微处理器的外设也变得得心应手。

当然整个过程中也存在许多的不足和失误，在进行 PCB 板的设计和制作的过程中，忽略了一些元件的封装与引脚的关系，当把 STM32 微处理器实验板做了出来后，调试不通过，供电部分存在问题，后来经过检测发现是 AMS1117 的封装选错了，指定封装的引脚排列和实际的 AMS1117 是不同的，经过把 AMS1117 重新按正确的方式焊接以后就可以正常工作了。另外在 USB 供电部分，地线隔空了，

没有连到电路里去，后用导线把它接到板子的地线以后问题就解决了。当把一个一个问题都解决以后板子完全可以正常工作，且工作稳定。

5.2 工作展望

经过这一段时间对 STM32 及其内置 ADC 和 DAC 的了解和实际应用，我感觉 STM32 是非常强大的一个微处理器，而且在模数和数模转换方面可以做的非常好，而且我对模拟信号的采集与处理显示产生了浓厚的兴趣，我希望以后可以在这次毕业设计论文的基础上完成一个基于 STM32 微处理器的手持示波器的设计和实现工作，然后以后就可以随身携带并方便的在任何地方使用它进行信号的检测了。

致 谢

历时将近半年的时间终于将这篇论文写完，在论文的写作过程中遇到了无数的困难和障碍，但都在同学和老师的帮助下度过了。尤其要强烈感谢我的论文指导老师—冯育长老师，他对我进行了无私的指导和帮助，不厌其烦的帮助进行论文的修改和改进。另外，在实验室进行 PCB 板绘制、焊接和调试的时候，实验室的何先灯老师还有在实验室一起做毕业设计的同学们也给我提供了很多方面的支持与帮助。在此向帮助和指导过我的各位老师和同学表示最衷心的感谢！同时感谢这篇论文所涉及到的各位专家学者。

本文引用了一些学者的研究文献和著作，如果没有各位学者的研究成果的帮助和启发，我将很难完成本篇论文的写作。

感谢我的同学和朋友，在我写论文的过程中给予我了很多理论素材，还在论文的撰写和排版灯过程中提供热情的帮助。

由于我的学术水平有限，所写论文难免有不足之处，恳请各位老师和学友批评和指正！

参考文献

[1] 意法半导体. STM32F103xCDE 数据手册中文版. 第五版. 2009，3. 1-75

[2] 意法半导体. STM32F103xxx 参考手册. 第十版. 2010. 105-198

[3] ALIENTEK. STM32 开发指南库函数版. 版本 1.2. 2012. 305-352

[4] STMicroelectronics. How to get the best ADC accuracy in STM32F10xxx devices. Rev1. 2008. 1-26

[5] Analog Devices. REF196 Datasheet. Rev.E. Jan 2003. 1-24

[6] 王逸彬.冰凌科技 STM32F103ZET6 开发板用户手册. 版本 4.6. 2010. 65-68

[7] 孙肖子. 陈南. 易运晖等. 电子设计指南. 第一版. 北京：高等教育出版社，2006. 75-273

[8] NXP Semiconductors. LM358 Datasheet. 2002. 1-12

[9] 李东生. 张勇. 许四毛等. ProteL 99SE 电路设计技术入门与应用. 第一版. 北京：电子工业出版社，2002. 29-78

[10] 王建农. 王伟. Altium Designer10 入门与 PCB 设计实例. 第一版. 北京 ：国防工业出版社, 2013. 53-276

[11] Advanced Monolithic Systems. AMS1117 Datasheet. 1-7

[12] 郭天祥. Altium Design6.9 PCB 设计教程. 2008

[13] 刘波文. ARM Cortex-M3 应用开发实例详解. 第一版. 北京：电子工业出版社，2011. 46-112

[14] ALIENTEK. 基于 STM32 固件库 V3.5 建立 keil 工程详细步骤. 2012. 1-19

[15] 蒙博宇. STM32 自学笔记. 第一版. 北京：北京航空航天大学出版社，2012. 101-265

[16] STM32F10x_StdPeriph_Driver_3.5.0(中文版). 版本 1.7.3 2011

[17] 李宁. 基于 MDK 的 STM32 处理器开发应用. 第一版. 北京：北京航空航天大学出版社，2008. 184-277

[18] http://heroxx.blog.163.com/blog/static/542358020105220354626 6/