存档编号

赣 南 师 范 大 学 学 士 学 位 论 文

**基于STM32的数字示波器设计**

教学学院 ： **物理与电子信息学院**

届 别 ： 2019 届

专 业 ： 通信工程

学 号 ： 150807027

姓 名 ： 施慧玲

指导教师 ： 刘小燕

完成日期 ： 2019年5月

**作者声明**

本毕业论文（设计）是在导师的指导下由本人独立撰写完成的，没有剽窃、抄袭、造假等违反道德、学术规范和其他侵权行为。对本论文（设计）的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。因本毕业论文（设计）引起的法律结果完全由本人承担。

毕业论文（设计）成果归赣南师范大学所有。

特此声明。

作者专业 ： 通信工程

作者学号 ： 150807027

作者签名 ：

年 月 日

**基于单片机的数字示波器设计**

**施慧玲**

*Design of Digital Oscilloscope Based on STM32*

***Shi Huiling***

空1格

**年 月 日**

**摘 要**

本设计是基于ARM (Advanced RISC Machine) 最新的Cortex-M3 内核所设计的基于STM32控制器的数字示波器，主要是设计前端硬件电路和算法程序的设计并完成预期目的。首先是设计完整的系统框图，井对各个组成部分的模块进行分析和设计，对方案的选定和相关参数进行计算，井给出各个模块所使用的元器件。所需要设计的模块包括前端信号调理、电源部分、控制器接口外国电路、TFT显示部分。软件设计部分包括有A/D采样部分、TFT显示部分、数值处理部分、UC/OS移植和GUI的设计，以此来设证一个完整的系统，达到预期的目标。

什么？

本设计实现了电信号包括周期信号和非周期信号波形的显示，实时采样速率最高可达力IM。采用品示器件是彩色的TFT (240\*320) 动态显示。另外，还实现了对模拟信号的调理，让其电压值符合AD的采样范园，通过数值处理计算出波形的相关参数并显示在TFT中等功能。在附加功能中还设计了信号发生器。因此，整个系统更加完整、完善，从而达到预期目的。

**关键字:** STM32**;数字示波器;**GUI;TFT

重复

删除空行

**Abstract**

This design is based on ARM (Advanced RISC Machine) latest cortex-m3 kernel design based on STM32 controller digital oscilloscope, mainly designed front-end hardware circuit and algorithm program design and achieve the desired purpose. Firstly, a complete system block diagram is designed. The well analyzes and designs the modules of each component, selects the scheme and calculates relevant parameters, and provides the components used by each module. The modules to be designed include front-end signal conditioning, power supply, foreign circuit of controller interface and TFT display. The software design part includes A/D sampling part, TFT display part, numerical processing part, UC/OS port and GUI design, so as to set up A complete system and achieve the expected goal.

This design realizes the display of electric signal including periodic signal and aperiodic signal waveform. The color TFT (240\*320) dynamic display is adopted. In addition, the adjustment of analog signal is realized to make its voltage value conform to AD sampling template. Relevant parameters of waveform are calculated through numerical processing and displayed in TFT. In addition, the signal generator is designed. Therefore, the whole system is more complete and perfect, so as to achieve the desired purpose.

空1格

删除空行

**Key words:** STM32; Digital oscilloscope; GUI; TFT

**目 录**

[1. 选题背景与意义](#_Toc13920_WPSOffice_Level1) [1](#_Toc13920_WPSOffice_Level1)

使用理科建议格式：左右对齐

[1.1 选题依据](#_Toc28279_WPSOffice_Level2) [1](#_Toc28279_WPSOffice_Level2)

[1.2 设计目的](#_Toc24607_WPSOffice_Level2) [1](#_Toc24607_WPSOffice_Level2)

[1.3 设计意义](#_Toc16383_WPSOffice_Level2) [1](#_Toc16383_WPSOffice_Level2)

[1.4 应用分析](#_Toc23689_WPSOffice_Level2) [1](#_Toc23689_WPSOffice_Level2)

[1.5本章小结](#_Toc600_WPSOffice_Level2) [2](#_Toc600_WPSOffice_Level2)

[2. 基于STM32数字示波器系统总体设计](#_Toc28279_WPSOffice_Level1) [3](#_Toc28279_WPSOffice_Level1)

[2.1系统总体设计框图](#_Toc20177_WPSOffice_Level2) [3](#_Toc20177_WPSOffice_Level2)

[2.2 总体设计框图](#_Toc24358_WPSOffice_Level2) [3](#_Toc24358_WPSOffice_Level2)

[2.3 硬件系统方案设计 5](#_Toc32405_WPSOffice_Level2)

[2.4 本章小结](#_Toc23022_WPSOffice_Level2) [6](#_Toc23022_WPSOffice_Level2)

[3. 基于STM32数字示波器硬件电路设计 8](#_Toc24607_WPSOffice_Level1)

[3.1 电源部分 8](#_Toc23075_WPSOffice_Level2)

[3.2 信号调理部分 8](#_Toc9531_WPSOffice_Level2)

[3.3 DIA输出滤波部分 1](#_Toc6527_WPSOffice_Level2)0

[3.4系统控制部分](#_Toc20988_WPSOffice_Level2) [11](#_Toc20988_WPSOffice_Level2)

[3.5本章小结](#_Toc26328_WPSOffice_Level2) [13](#_Toc26328_WPSOffice_Level2)

[4. 基于STM32数字示波器系统软件设计 1](#_Toc16383_WPSOffice_Level1)4

[4.1信号采集和显示部分 1](#_Toc12988_WPSOffice_Level2)4

[4.2 A/D采样设置](#_Toc8023_WPSOffice_Level2) [15](#_Toc8023_WPSOffice_Level2)

[4.3 DMA配置](#_Toc22943_WPSOffice_Level2) [18](#_Toc22943_WPSOffice_Level2)

[4.4 显示部分](#_Toc6059_WPSOffice_Level2) [19](#_Toc6059_WPSOffice_Level2)

[4.5 DAC信号输出部分 2](#_Toc3917_WPSOffice_Level2)2

[4.6 系统图形化设计 2](#_Toc27077_WPSOffice_Level2)5

[4.7本章小结 2](#_Toc21041_WPSOffice_Level2)9

[5. 系统调试与结果 3](#_Toc23689_WPSOffice_Level1)1

[5.1系统测试方案 3](#_Toc15774_WPSOffice_Level2)1

[5.2系统测试及分析 3](#_Toc10424_WPSOffice_Level2)5

结束语

[5.3 测试结果分析 3](#_Toc19707_WPSOffice_Level2)5

[6. 总结与展望 4](#_Toc600_WPSOffice_Level1)8

主要参考文献

[文献参考](#_Toc20177_WPSOffice_Level1) [49](#_Toc20177_WPSOffice_Level1)

[致谢 5](#_Toc24358_WPSOffice_Level1)0

[附录](#_Toc32405_WPSOffice_Level1)[一..........................................................................................................................5](#_Toc32405_WPSOffice_Level1)1

[附录二..........................................................................................................................5](#_Toc32405_WPSOffice_Level1)1

[附录三..........................................................................................................................5](#_Toc32405_WPSOffice_Level1)2

[附录四..........................................................................................................................5](#_Toc32405_WPSOffice_Level1)2

[附录五..........................................................................................................................5](#_Toc32405_WPSOffice_Level1)2

页码不对

1. **选题背景与意义**

最早的模拟示波器使用成像管技术将电极产生的电场应用到荧光屏上的荧光粒子上，产生荧光来显示波形。传统的模拟示波器在测量某些周期性信号（如正弦波、方波、三角波等）时，可以更稳定地显示被测信号波形，但如果不能正常显示非周期信号或瞬时信号，则波形会丢失。尤其是低频信号测量时，失真率较大。当输入频率小于100Hz时，波形闪烁非常严重。当测量小于30Hz的信号时，信号的波形和周期需要通过光点的移动提前确定。产生的错误太大了。而且，由于模拟示波器比较重，不利于携带。随着数字示波器的出现，数字示波器具有强大的波形处理能力。数字示波器，如Ticks和Froc，可以直接执行FFT和其他开关，特别是对于小信号。测量的畸变率相对较小。此外，数字示波器体积小，易于通过USB或RS232进行通讯，节省了实时波形，便于信号采集和存储。因此，它们得到了广泛的应用。

**1.1 选题依据**

本设计是基于ARM Cortex-M3数字示波器的设计。它可以通过AD采样和数字处理显示周期和非周期信号，并在TFT上显示信号的峰值、最大位、最小字母、频率和动态实时波形。系统设计包括前端信号调理、AD采样控制、J-Tag仿真接口、TFT实时波形显示等，具有便携式、实时数据采集等优点。

**1.2 设计目的**

段首统一缩进2字符

本设计的目的是接收任意一个中心信号，实时显示接收到的波形，测量波形的峰值、最大值、最小值和频率。完成动态波形显示，显示波形的频率和振幅特性。设计中增加了信号发生器的功能，使系统更加完整。

**1.3 设计意义**

本设计实现了周期信号和瞬时信号的采集与显示，克服了传统模拟示波器只能显示周期信号的缺点，并且由于采用马赛克设计，设计体积小。本设计采用STM 32系列ARM芯片作为主控制器。使用彩色TFT（分辨率240\*320）液晶显示屏，最大实时采样率可达1米。前端模拟信号调节包括阻抗匹配、信号衰减和放大。最后给出了波形的频率和幅度特性，得到了波形的峰值和频率。此外，该设计基于嵌入式数字示波器，控制器内部集成了A/D和D/A。它具有体积小、可靠性高的优点。便携式信号采集具有重要意义，能够满足本设计的要求。

**1.4 应用分析**

本设计采用STM32芯片完成数据采集与量化、A/D转换、波形分析、波形显示、触摸屏操作等部分。STM32处理器是基于ST低成本市场设计的ARM-M3核心。由于微处理器成本高、性能强，非常适合嵌入式开发的设计要求。本设计采用STM32芯片，尤其是基于最新Cortex-M3内核的嵌入式数字示波器。单片机为STM32F103ZET6，软件包为LOF-144。

ARM作为嵌入式时代的主流嵌入式控制器产品，随着嵌入式技术的发展，对嵌入式示波器的功能也越来越强大。增加USB、RS 232等通信协议有利于实时数据采集。实时采样更有效。本设计是基于ARM示波器的设计。具有数据采集、信号调理、显示和输出功能。系统运行稳定，采集的数据实时准确。

RS-232

软件包？

**1.5 本章小结**

本章主要介绍了本设计的背景和预期目标。根据ARM处理器在示波器设计中的优势，设计了符合要求的示波器方案。下一步是确定各模块的设计和参数的设计。

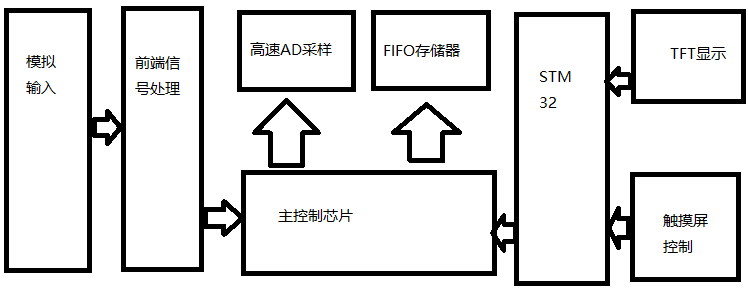
理科论文一般不需要小结

本设计的详细设计要求在哪？

1. **基于STM32数字示波器系统总体设计**

**2.1 系统总体设计框图**

不能一开始就是图，先文后图，文为主，图为辅，下同



主控制芯片和右边的STM32什么关系，系统是用了2块控制器吗？

细线条

2.1 总体设计框图

**2.2 硬件系统方案设计**

图序和图题小四宋体加粗，下同

本设计模拟硬件电路部分主要包括:输入阳抗部分、信号衰减和放大部分、主控制器芯片、以及显示部分。

**2.2.1 输入阻抗**

输入阻抗是输入电路的等效阻抗。它是指在输入端加一个电压源来测量输入端电流，然后输入阻抗Rin为U/I。从相关数据可以看出，在测量中输入可以看作电阻的两端。作为电阻的电阻，它是设计的输入阻抗。通常的输入阻抗是由电路中的电容和电阻来连接的。对于低频信号，可视为开路。输入阻抗只是电阻的一部分。设计数字示波器时，输入电阻设为1M。为了减小数字示波器内部阻抗对外部环境的影响，当输入频率逐渐增大时，电容效应不那么重要。参考相关示波器数据后，将数字示波器的输入阻抗设置为1M。将示波器的输入电容设置为20p，以便在采集信号时尽量减小示波器对外界信号的影响。在接下来的硬件设计中，将对相关参数进行仿真。

**2.2.2 信号的衰减和放大**

格式不规范

对于A/D采样，则必须需要注意：

ADC对于输入电压的采样范围；

A/D对于电压的采样精度；

输入信号前端滤波处理(减少干扰)。

由于设计输入为交流信号(正弦波、方波等)在参考STM32的相关ADC数据后，必须对ADC前端的信号进行调整，使输入信号在采样前达到A/D值，即设计相应的信号调理电路。对于通常包括电阻衰减、部分电压衰减和放电衰减在内的衰减电路，各有优缺点。你可以考虑使用放大器衰减。当输入阻抗增大时，可以使用放大器连接的电压跟随器进一步增大输入阻抗。在接下来的硬件电路设计中，给出了相应的电路和参数设计。

**2.2.3 主控制芯片**

本设计通过调整输入电压信号来满足ADC的输入条件，将信号输入到ADC引脚进行采样和量化。相关的微处理器包括C51、DSP和ARM处理器。首先，由于C51内部资源有限，包括寄存器和Flash，所以放弃了使用C51微控制器。如果在设计中选择DSP，由于DSP对外部信号的要求较高，设计中使用的芯片数量会增加。由于DSP的成本高于ARM，如果将DSP应用于信号采集，信号放大芯片将受到限制，难以选择。最新的STM32是由意大利半导体公司基于ARMCortex M3 core设计的处理器。这是首选的目标，成本和易于操作。由于STM32F103处理器内置了自己的ADC，因此它可以直接调用内部硬件DMA，从而最大限度地减少系统资源的使用。因此，如果采用ARM处理器进行AD采样，可以选择内部ADC来降低电路设计的复杂度。此外，stm32系统资源丰富，功能强大。这不是嵌入式开发的选项。

不规范的写法

因此，本设计采样STM32 处理器作为控制器。

本设计采用增强型STM32系列，包括ADC、DAC和硬件DMA。这简化了外围电路的设计，干扰相对较小，系统时钟频率为72MHz。该设备具有良好的数据处理能力，能够满足内部闪存和数据存储的要求。由于GPIO配置更加灵活，控制显示设备更加灵活方便。

总而言之，本设计采用ZE系列STM32，性能强大，满足数据处理能力的要求。

**2.2.4 显示部分**

在本设计中，由于实时采样数据以动态波形曲线的形式显示，因此对显示设备的像素要求较高。可以在联系人中考虑的选项包括:

选项1:使用LCD12864;

选项2:使用TFT(Thin Film Tansistor))彩色屏幕。考虑到本设计需要更多的像素和各种窗口菜单，原始的LCD12864像素非常罕见。因此，实时波形显示首选TFT(240 \* 320)触摸屏，使用触摸屏控制系统方便。

液晶显示器的相关参数如下:

(1)色度:采用RGB三基色。色度的数量可以达到1600万色彩度。(2)比较值。(3)亮度值。(4)响应时间:控制LCD屏幕上的图像显示和下一次刷新所需的时间。如果时间太长，图像和图形会重叠。

**2.2.5 附加功能**

为了充分利用STM32的内部资源，内部DAC可以用来模拟正弦、方波和锯齿波的输出，这些波形经过输出滤波器后作为信号发生器。参照STM32，预置频率力为0- 20kHz。范围控制在0-3.3V之间。当输出模拟信号时，我们需要设计输出电路的外部电路，主要是滤波电路。预设计滤波器的截止频率为Fs = 25kHz波形，无失真。在详细介绍了硬件结构后，给出了计算内容和仿真结果。完成该部分后，整个设计系统更加完善，功能更加完善。

**2.3 系统软件方案设计**

设计软件分析主要分为：示波器件分析和信号发生器软件分析。

**2.3.1 示波器软件分析**

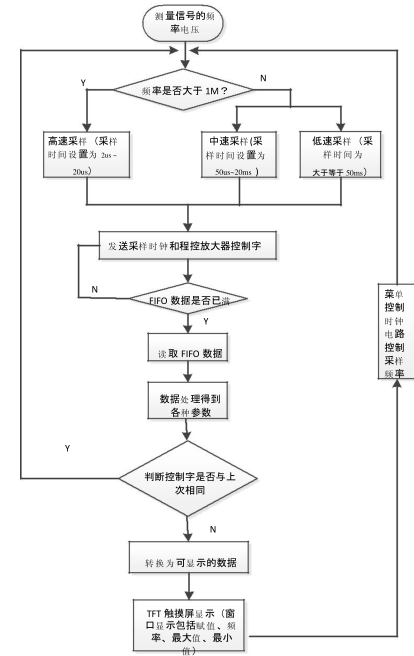


图2.2 示波器软件需求分析流程图

最好重画

模拟信号（任何波形的信号）由示波器探头输入。由于不能通过输入信号的分配来确定预峰值（-5-5V），所以必须首先在输入信号之后调整和转换信号。在输出信号满足范围（0-2V）之后执行A / D采样转换。AD转换的数据存储在内部存储器中。同时，ARM处理器存储广告转换的数据。读取前存储在RAM中的数据通过数学变换处理，包括峰值，最大值和频率的计算，并将处理后的数据发送到TFT进行动态显示。当下一个AD采样数据到达时，刷新原始数据以动态更改波形。同时，在触摸屏上设置菜单以操作系统和操作包。它包括通道选择（Ch1，Ch2），触发线（Tigger），垂直刻度，水平刻度值（时间），频率和波形显示。每个菜单的选择由与触摸相关联的触摸屏区域控制。软件和图形界面设计将逐步满足这一要求。

**2.3.2 信号发生器软件分析**

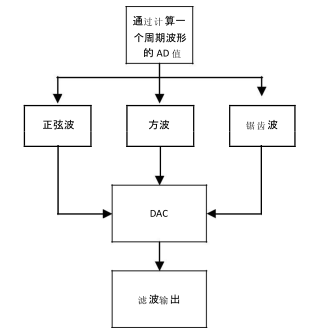


图2.3 信号发生器的软件需求分析

信号发生器设计用于计算波形周期中的100个AD值（例如正弦波，三角波和其他标准波形），并将这些AD值存储在一个数组中。数据集的模拟输出通过内部DMA循环触发更新。当下一个周期到来时，更新数组中的原始值以实现循环输出公式。当然，波形模拟信号的目的是通过在输出的前端添加滤波器来滤除不需要的干扰。

**2.4 本章小结**

本章主要分析了整个系统的设计，包括软硬件设计，确定了初步设计方案和需求分析，并分析了ARM在设计中与其他控制器不同的独特优势。本文介绍了示波器硬件电路的设计思想和软件要求，包括硬件部分的设计思想和软件部分的设计。在完成数字示波器系统的程序选择之后，在附加功能的基础上增加信号发生器部分以改进整个系统。下一步是设计相关电路实现一些特定的参数指标。

**3. 基于STM32数字示波器硬件电路设计**

**3.1 电源部分**

稳定的电源对整个系统稳定的运行是至关重要，参看STM32数据手册，由于STM32内部供电源为3V3的线性电源，因此采用了LM1117\_3.3稳压芯片产生3.3V电压，以实现整个系统对电源的需求，本设计采用的封装是SOT-32，LM1117\_3.3。

说明清楚点

句号

本设计的电路电源的应用如图3.1所示：

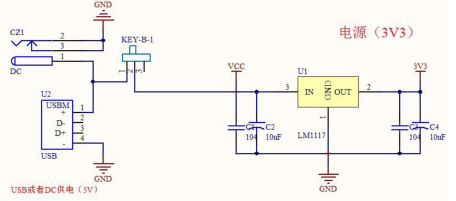


图3.1 系统电源电路

所需要说明的是在电源的布线时一些滤波电容应当尽量靠近IC引脚，这样泷波效果会更好。

**3.2 信号调理部分**

图未居中，先删除段首缩进再居中。其他地方也检查一下

由于STM 32的采样范围为0~3.3V，设计的采样范围为-5~5V。 因此，当原始输入信号衰减到（-1-1V）的范围时，由于不能对STM32的AD负压进行采样，所以必须增加负压。参考方案包括将电压（1V）除以参考源TL431，并根据AD集合的允许范围将原始模拟信号电位增加到（0-2V），以便可以对AD进行采样和转换。这个想法如图3.2所示。



图3.2 信号调理的设计思路

整体信号调理电路见附录1所示。

鉴于设计的精确度和所选电阻的值，一些电阻器被电位器取代以实现这一目标将信号调理电路模型引入Tina进行仿真。分析参数包括模拟电路的幅频特性。

以及观察到的波形是否失真，如图3.3所示。

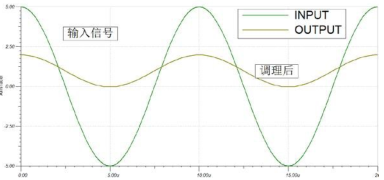


图3.3 信号调理仿真结果

仿真结果表明，信号调整后的输入信号无失真，输出幅度控制在0-2V，波形相位保持不变。

根据项目要求，由于输入模拟电压信号的峰值为10V，电路中的电流相对较大，因此选择TI电流反馈宽带放大器THS3091，采用的封装是SOP-8。由于THS3091及热量会较大，因此必须正确处理散热垫。芯片的引脚分布如图3.4所示。

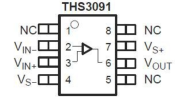


图3.4 THS3091芯片引脚分布

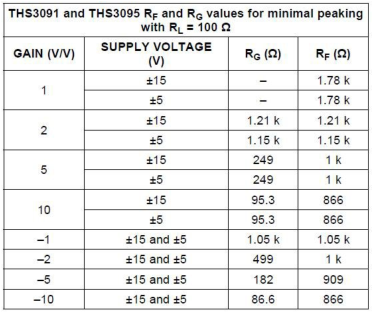
参照THS3091数据手册得到以下相关参数：

THS3091为电流反馈型运算放大器，所需电源为双电源:-5～15V,最大带宽:210MHz，高转换速率: 7300V/ms。

删掉下字，下同

　　在引脚Vin+(3引脚)输入电压范围为-5～5V，分别在第一级设置为电压跟随器、和反向比例放大器。

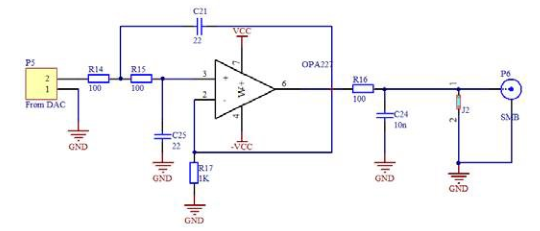
根据数据手册内部的Recommended Resistor Table（推荐电阻取值）如下表3.1所示。根据表格在设计运放的反馈电阻时，R1=1.8K，经过显示的电路测试满足要求。

表3电阻推荐取值表

**3.3 DIA输出滤波部分**

表序表题在表格上方，小四宋体加粗居中，表序和表题间空2格

在此基础上，STM 32的DA用于模拟正弦波，方波和锯齿波。预设频率范围为0~20 kHz。对于模拟信号输出，需要对模拟输出进行滤波。截止率设置为fs = 25 kHz。原理图如下图3.5所示。



重点在于设计方法，设计过程

图3.5 DA输出滤波器部分

将原理图模型进行仿真，仿真其传输特性，仿真结果为如下图3.6 所示。

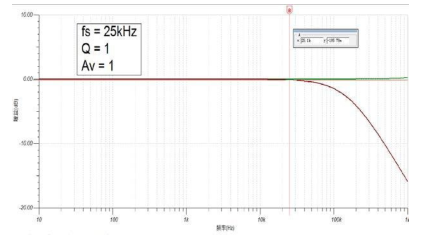


图3.6 滤波仿真结果图

当频率fs = 25KHZ且av = 1时，模拟结果中涉及的电路开始衰减。它不会在0-25khz之前衰减，符合设计要求。

在芯片选择方面，选择了TI公司的OPA227。芯片封装采用DIP-8。芯片的引脚图如图3.7所示。

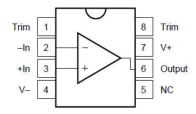


图3.7 OPA227 芯片引脚图

在绘制PCB时，一些滤波电容应尽可能靠近芯片引脚，以减少外部干扰。 由于前端电流运算放大器通常会产生大量热量，因此还必须妥善处理散热问题。 应注意集成电路底部的散热器布局。为了提高精度，一些电阻器将被精密电位器取代，以达到所需的目标。有关PCB布局和接线图，请参见附录2。

**3.4 系统控制部分**

在系统的控制部分，触摸屏的原理主要用于控制相应的菜单切换各种选项，从而控制整个系统。

在软件设计过程中，预设了几个虚拟菜单键和相应的中断响应。当点击或压缩触摸屏的某些区域时，激活中断响应并处理相应的时间，这将在随后的软件和GUI设计中详述。触摸屏的结构如图3.8所示。

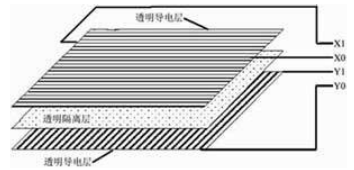


图3.8 触摸屏结构

当连接某个点区域的导电层时，将产生电压值，即当触摸屏用于转换每个点产生的电压值以确定触点的坐标点时，需要AD转换芯片，也就是控制器。该设计采用四线触摸屏，控制芯片AD为TI ADS7843的12位数模转换芯片，最大转换速率为125 kHz。如图 3.9所示。

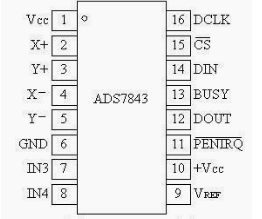


图3.9 ADS7843引脚图

当物理挤压或点击触摸屏时，ADS 7843通过中断请求通知控制器触摸，该中断请求触发与当前区域中的中断请求处理相关的动作以控制相应的菜单选项。 当软件部分在ADS 7843外部设置相应的菜单目录时，电路如图3.10所示。



图3.10 触摸芯片外围电路

所采用的是TFT（3.2＇）240\*320像素点分布如图3.11所示。

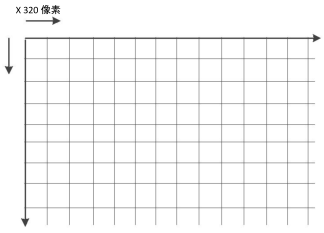


图3.11 TFT像素点分布

所要说明的是，在软件设计中，将每一个像素点设置对应的电压值以此来描绘对应的电压曲线，使得波形能够动态的变化。

**3.5 本章小结**

本章是对硬件部分进行了细致的解释，包括了电源部分，放大部分，控制部分，芯片采集及显示部分。硬件部分芯片采集部分是重点，包括采样部分，数据计算和显示部分的使能。当然也对运放芯片的选型和相关参数的设计取值给予说明。下一步就是STM32对于软件的设计。

**4. 基于STM32数字示波器系统软件设计**

系统软件设计主要包括，AD的采样设计、DAC设计以及显示的图形化设置，下面详细介绍各个部分内容。

**4.1 信号采集和显示部分**

该设计采用运算放大器对输入模拟信号进行衰减和放大，使电压值达到0~3.3V的要求，实现采集，数字处理和波形显示。本设计采用STM32F103ZET6完成模数转换芯片功能和数值计算，计算和显示功能。这比DSP使用多个芯片实现其目标所需的电路要简单得多。如果使用STM 32处理器，则信息获取部分的主要任务是将模拟电路信号转换为数字信号，并且数据计算和显示部分将遵循A / D转换部分。STM32F103ZET6增强型系列是是ARM-M3系列中最具成本效益和功能的。其内部资源足以满足硬件条件和操作方面的设计要求。

**4.1.1 显示部分**

显示部分中使用的显示装置是320×240 TFT屏幕。由于STM 32具有多引脚和多引脚结构，因此可以方便地控制TFT的处理操作。该应用程序使TFT能够打开开始屏幕。TFT显示装置具有动态波形显示稳定，运行稳定的优点。与传统的CRT相比，TFT具有体积小，厚度薄，重量轻，能耗低，工作电压低，无辐射，无闪烁等特点。而且，外部驱动电路设计简单方便，波形动态显示功能可通过简单的连接和驱动实现。

薄膜晶体管（TFT）是一种相对较新的显示器件。每个像素的步骤根据分辨率而不同。此设计采用240×320 TFT屏幕。

导电膜通常连接到TFT表面，我们通常将其称为触摸屏。

有许多类型的触摸屏，例如电容式触摸屏和电阻式触摸屏。由于STM 32没有太多处理能力，因此无法驱动电容式触摸屏（通常是Cortex-A系列），这些触摸屏通常用于高成本设计（如智能手机）。由于其高价格，这种设计采用TFT电阻式触摸屏。我们最常用的触摸屏是电阻式触摸屏。

当电阻式触摸屏的导电膜被物理挤压时，两侧的导电膜将彼此接触，从而反馈电压值。我们需要做的是收集该点的电压值，并通过AD的转换值表示该点的坐标（X,Y），以确定接触区域的触发事件选项。因此，我们需要一个触摸屏控制器，尤其是AD转换器芯片。本设计中使用的四线控制器AD芯片是ADS7843，它是一个12位分辨率和125 kHz转换速率AD转换器。它包括由多个模拟开关和12位AD转换器组成的测量电路网络。AD芯片可以通过处理器STM32F103ZET6连接到电路。SPI总线用于连接该电路中的四线控制器。根据控制器发送的不同测试指令，打开或关闭不同的模拟开关，提供当前位置的电压值，此时对电压值进行AD采样。从而确定该点的坐标值。图4.1是触摸屏测量关系的示意图。

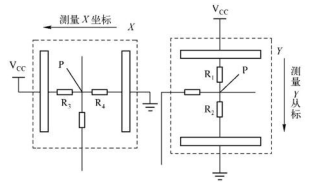


图4.1 触摸屏测量关系

每当导电膜接触，ADS7843就会通过中断的方式通知处理器STM32F103ZET6，有触摸的事件发生时，会产生中断请求。

**4.1.2 数值计算部分**

数据采集后，处理波形显示参数，串口设计用于计算峰峰值，最大值，最小值，频率和占空比（方波），以控制传输使能。寄存器计数功能由通用多路复用硬件完成。注册计数适用于一般和多功能组件。在STM 32中，每个GPIO端口都有两个32位寄存器。可以根据需要灵活配置每个端口位。

具体的数值计算部分详细列出了软件中的相关功能和算法。

**4.2 A/D采样设置**

**4.2.1 信号采集部分**

信号采集的首先要做的是AD转换。如果选择ADS7812芯片并满足预期要求，但其工作温度相对较小，则芯片所需的温度补偿将完成。然而，由于STM32中的AD采样和数据处理期间的环境温度相同，因此消除了需要温度补偿的措施并简化了电路。内部ADC可以直接使用DMA操作，从而减少对系统资源的占用。

使用STM 32内的AD可以测量两个内部信号和16个外部信号。每个通道的AD转换模式还包括执行单个，连续和不连续的多个工作模式。转换结果保存到16位数据寄存器中。

信号调理后，-5~SV的输入电压为0~2V。采样在ADC范围内进行。根据不同的频率范围定时触发采样。使用DMA，最大采样频率可达1米。

ADC\_0和ADC\_1分别从原理图中导出。见图4.2。

硬件的归硬件，软件的归软件

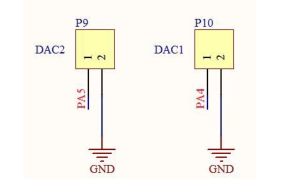


图4.2 ADC引脚

需要说明的是，本设计中使用的处理器部分是采用一块STM32F103ZET6的最小系统板来实现的，因此AD的引脚和后面的DA引脚都是通过排线进行连接的， 可能在接触方面不是特别好。

**4.2.2 设置采样时间**

模拟信号经调理后，根据频率范围分为高速采样、中速采样、低速采样。具体流程如图4.5所示。

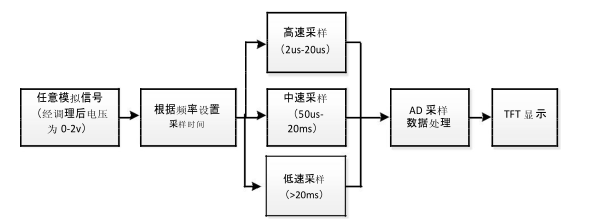


图4.3 采样流程设置

不规范，不能用1.

STM32内部的AD是逐次逼近型的AD转换器。最多包含有18个通道。STM32中的AD可以进行多种模式的转换。所需要强调的是STM32 的ADC输入时钟不得超过14M，并且只由PCLK2分频产生。

在使用ADC的时候，需要配置相关参数。

1. A/D的模式，本设计中我们设置为独立模式
2. A/D的通道设置，由于是通道的AD采样，所以我们设置成DISABLE
3. A/D的转换设置，由于我们设置了定时器定时触发采集，所以我们设置为ENABLE
4. AD的触发选择，我们设置为定时器触发，

ADC\_ExternalTrigConv\_T1\_CC2

ADC\_ExternalTrigConv\_T2\_CC2

ADC\_ExternalTrigConv\_T3\_T

ADC\_ExternalTrigConv\_T4\_CC4

最后使能ADC。

具体是采用通过预设的采样时间设置定时器的预分频值和重载值。AD采样中需要将AD的采样优转化为电压值。

对于ADC的采样值所对应的电压值信号的计算在STNI32 手册中给出，当前的电压值和ADC值.他们存在以下关系，式4-1

自己输入，不能用图



反过程得到当前电压值，再进一步，效率会更高。可以设置为: Volue = (ADC Conv\*825)>>10;//移位计算

在程序设计时，为了方便在后面将每个点的电压值与TFT中的像素点对应，因此定义设置了相关变量，以达到要求。

以下是什么

static volatile int\_vSampleCount=0; //总采样数为可变变量

static const int\_cSampleCmpStart=MAX\_VALUE; //adc采样中的buff与电平比较的起点

static const int\_cSampleCmpEnd=MAX\_VALUE \* 2; //adc采样中的buff与电平比较的终点

static u16\_aMemSize[3][MAX\_VALUE \*3]= {0,};

static const float\_fCoef = (FULL\_VOLT\_VOLUE/0xFF); //电压与AD值的系数(AD\*\_fCoef= V)-->AD/4096 //显示的y坐标值与AD采样的AD值，对应系数(AD-2047= \_ fDisToADCoef\*scale\*Y);

static const float\_fDisToADCoef=0xFFF\*1000/GRAPH\_SCALE\_V/FULL\_VOLT\_VOLUE;

//AD采样值与显示的y坐标值对应系统(Y=(AD-2047)\*\_fADToDisCoef/scale)

Static const float\_fADToDisCoef=FULL\_VOLT\_VOLUE\*EGAPH\_SCALE\_V/0xFFF/1000; //将采样的AD值赋给Y坐标表示其赋值的大小。

当然在示波器中，触发线的引用也是非常重要的，在测量一些波形时需要调节触发线来满足测量条件，在后续参数计算是会运用此触发线判断波形是上升还是下降，由此来算周期、频率、占空比。

触发沿的电压的相关函数如下:

static int AppOSCGetLevel(ParamStruct\*pParam)

{

float fTmp;

int tmp:

tmp=pParam->trigger.source?pParam->ch2.vertical.scale:pParam->ch1.vertical scale;

//通过触发源选择出幅度对应电压值

tmp=pParam->trigger.source?pParam->ch2.vertical.pos:pParam->ch1.vertical.pos;

//计算出触发线对应的AD值

return (int)((fTmp\*(pParam->trigger.level-tmp)\*\_fDisToADCoef)+HALF\_VOLT\_VOLUE);

}

对于触发线的引用使得电路的整体结构变得简单，特别是在测频率时，大部分的测量方案是将波形单独引出进行比较输出成方波信号，测量单位时间内计数通过的上升沿或下降沿的数目，以此来测量周期或者频率。而触发线的引用我们只需判断波形在某个范围内，通过和触发线电压值的比较，判断是上升或是下降(通过AD的采样间隔点)，在某个范围内，同时经过上升下降和上升，那么，则满足于一个周期的定义，方便电路的设计。

**4.2.3 设置转换模式**

STM32中AD采样主要分单次(single conversion mode)、连续(continous conversion mode)、间断模式(discontinuous conversion mode)。

1. 单通道单次、连续转换

单通道单次采样意味着ADC在采样一次后停止转换。 但是，连续采样意味着下一次转换将在ADC采样转换后立即发生。 因此，单通道单通道连续转换模式主要针对单输入通道。

1. 单独连续转换多个通道后，当同一组中的最后一个通道已转换时，ADC将停止，直到下一次触发。 在连续变换模式中，ADC从组的第一个变换序列循环。 换句话说，通常建议在此模式下使用DMA来释放CPU资源并降低系统开销。

3.不连续模式通常将转换划分为规则组，并通过外部触发事件触发转换。

4.双ADC模式类似于单个ADC。 双ADC具有独立模式，触发模式，交替模式等。 将两者结合起来。

**4.3 DMA配置**

DMA(Direct Memory Access)、直接存储器存取，形象点就好比是一个搬运工，将外设的数据和内部存储数据直接进行高速传输，尽可能的减少CPU资源的占用，STM32中含有两路DMA，根据不同优先级的DMA请求来安排相关操作。参考《STM32F系列ARM内核32位高性能微控制器参考手册》将DMA配置为以下模式：

/\*----DMA1 channell configuration------------\*/

DMA\_Delnit(DMA\_Channell); //开启DMA1的第一通道

DMA\_InitStructure.DMA\_PeripheralBaseAddr=ADC1\_DR\_Address; //DMA对应的外设地址

DMA\_InitStructure.DMA\_MemoryBaseAddr=(u32)ADCBuffer; //内部存储基地址

DMA\_InitStructure.DMA\_DIR=DMA\_DIR\_PeripheralSRC; //DMA转换模式为SRC由外设搬移到内存

DMA\_InitStructure.DMA\_BufferSize=(u32)(adcBufCountADC\_BUFF\_COUNT\*2); //设置DMA缓存大小

DMA\_InitStructure.DMA\_PeripheralInc=DMA\_PeripheralInc\_Disable; //接收一次数据后设备地址禁止后移

DMA\_InitStructure.DMA\_MemoryInc=DMA MemoryInc Enable; //关闭接收一次数据后目标内存地址后移

DMA\_InitStructure.DMA\_PeripheralDataSize

DMA\_PeripheralDataSize\_HalfWord; //定义外设数据宽度为16位

DMA\_InitStructure.DMA\_MemoryDataSize

DMA\_MemoryDataSize\_HalfWord; //定义DMA搬移数据尺寸，HalfWord就为16

DMA\_InitStructure.DMA\_Mode=DMA\_Mode\_Normal; //设置为正常模式

DMA\_InitStructure.DMA\_Priority=DMA\_Priority\_Medium; //设置DMA优先级为中

DMA\_InitStructure.DMA\_M2M=DMA\_M2M\_Disable; //禁用M2M

DMA\_Init(DMA1\_Channell,&DMA\_InitStructure);

/\* Enable DMAI channell \*/

DMA\_Cmd(DMA1\_Channell,ENABLE);

DMA\_ITConfig(DMA1\_Channell,DMA\_ IT\_ TC,ENABLE);

**4.4 显示部分**

LCD驱动部分采用的是ili9320驱动芯片，相关引脚定义如下:

#define nCsPin GPIO\_Pin\_8

#define RsPin GPIO\_Pin\_9

#define nWrPin GPIO\_Pin\_10

#define nRdPin GPIO\_Pin\_11

#define nRstPin GPIO\_Pin\_12

#define Lcd\_LightPin GPIO\_Pin\_13

#define CtrlPin\_NCS GPIO\_Pin\_2 /\* PB.02 \*/

#de2fine CtrlPin\_RS GPIO\_Pin\_7 /\* PD.07 \*/

#define CtrlPin\_NWR GPIO\_Pin\_15 /\* PD.15 \*/

以下函数定义LCD驱动器的访问地址，32位寻址方式。

#define LCD\_BASE ((uint32\_t)(0x60000000 | 0x0C000000))

#define LCD ((LCD\_TypeDef \*) LCD\_BASE)

设置LCD的物理坐标

void LCD\_SetCursor(uint16\_t Xpos, uint16\_t Ypos)

{

LCD\_WriteReg(0x0200, Ypos); /\* px \*/

LCD\_WriteReg(0x0201, 399 - Xpos); /\* py \*/

}

设置LCD水平和垂直方向的起始地址和结束地址。共12\*8个单元格。其电路图如图4.4所示。

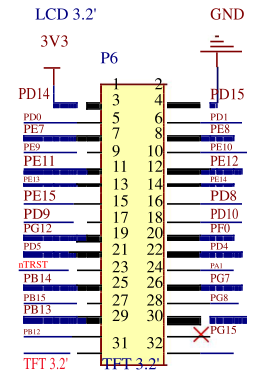


图4.4 TFT接口电路

其中TFT驱动芯片为ili9320驱动芯片，本设计采用FSMC总线方式来驱动TFT触摸屏，在程序设计是需要通过初始化静态存储控制器 FSMC（Flexible Static

Memory Controller总线）。具体接口如下，TFT和插座的接口电路如图4.5所示。

硬件端口说明：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* PE0~15 <----> DB0~15 \*\*

\*\* PD15 <----> nRD \*\*

\*\* PD14 <----> RS \*\*

\*\* PD13 <----> nWR \*\*

\*\* PD12 <----> nCS \*\*

\*\* PD11 <----> nReset \*\*

\*\* PC0 <----> BK\_LED \*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

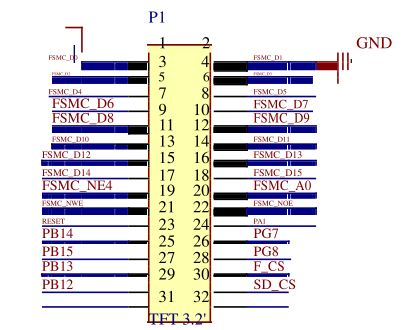


图4.5 插座接口定义

STM32中可变静态存储控制器（FSMC）模块能够与同步或异步存储器进行操

作。每一次新的FSMC操作前，原先的FIFO要先被清空。

FSMC结构图参照STM32数据手册如下：

配置的函数如下：

/\* FSMC Configuration \*/

/\* SRAM Bank 4 \*/

/\* FSMC\_Bank1\_NORSRAM4 configuration \*/

Timing\_read.FSMC\_AddressSetupTime = 0;

Timing\_read.FSMC\_DataSetupTime = 2;

Timing\_read.FSMC\_AccessMode = FSMC\_AccessMode\_A;

Timing\_write.FSMC\_AddressSetupTime = 0;

Timing\_write.FSMC\_DataSetupTime = 1;

Timing\_write.FSMC\_AccessMode = FSMC\_AccessMode\_A;

/\* Color LCD configuration

LCD configured as follow:

- Data/Address MUX = Disable //数据线和地址线不复用

- Memory Type = SRAM //存储器类型为SARM

- Data Width = 16bit //数据馈宽度为16位

- Write Operation = Enable //写使能

- Extended Mode = Enable //禁止拓展模式

- Asynchronous Wait = Disable \*/ //

**4.5 DAC信号输出部分**

为了使系统更加完整，模拟信号输出使用STM32的内部DA。输出波形包括正弦波，方波和齿波。DAC具有STM32的以下特性：两个DAC转换器对应一个输出通道，8或12位输出，12位数据模式左或右对齐同步更新功能，噪声波形，三角波形生成，每个通道的双通道同时或单独转换，DMA功能和外部触发转换。

设计是将计算周期信号存储到阵列中，并通过DMA周期更新DAC输出，以达到预定目的。 下次分别给出波形设置和频率参数设计。

**4.5.1 波形设置**

STM32内部的DA功能比较强大井且能够配合DMA控制器使用，在产生波形时，我们预先在STM32中计算一个周期的波形的AD位共100个，并将这些点存入数组。并通过DMA循环更新DA的值，在下一个周期再来的时候，原先的数组被更新。

产生正弦波的DA函数:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void\_SinWave(u16 \*buf, u16 len, u16 max, u16 min) //产生正弦波的函数

{

u16 rang = (max - min) >> 1;

u16 level = (max + min) >> 1;

u16 i = 0;

for (i = 0; i < len; i++)

{

\*buf++ = (u16)(sin(i \* \_cfScal) \* rang + level); //\_cfScal = 2\*π/100 ;

}

}

其中的sin()为math.h内部中的数学函数。

以下为方波的DA函数：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Static void\_SquareWave(u16 \*buf, u16 len, u16 max, u16 min, u16 duty)

{

u16 split = 0;

u16 i = 0;

if (duty == 0)

{

for (i = 0; i < len; i++)

\*buf++ = min;

return;

}

if (duty == 100)

{

for (i = 0; i < len; i++)

\*buf++ = max;

return;

}

split = len \* duty / 100;

for (i = 0; i < split; i++)

{

\*buf++ = max;

}

for (i = split; i < len; i++)

{

\*buf++ = min;

}

}

以下为锯齿波的DA函数：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void \_RampWave(u16 \*buf, u16 len, u16 max, u16 min)

{

float oceff;

int i；

if (len == 0)

return;

oceff = (max - min) / len;

for (i = len; i > 0; i--)

{

\*buf++ = (u16)(i \* oceff) + min;

}

}

**4.5.2 频率和参数设置**

在程序设计中预设了DA幅值为0~3300mV，频率最大设置为20kHz， 输出时只要控制其输出滤波就能达到预期的目的，因此在程序设计时通过修改分频值、

重载值、计数值来改变DAC的输出频率。

void BSP\_DacTimUpdata(float tim)

{

u16 prescaler; //分频值

u16 period; //重载值

u32 count; //计数值

//tim:一个周期的时间 ,count:一个周期每更新一个点所用的计数值(一个周期DAC\_BUFF\_LEN 个点)

count = tim \* SystemCoreClock / DAC\_BUFF\_LEN;

\_DACTimUpdateCal(count, &prescaler, &period);

TIM\_PrescalerConfig(TIM6, prescaler - 1, TIM\_PSCReloadMode\_Update);

TIM\_SetAutoreload(TIM6, period - 1);

}

在DAC中我们设计了通过循环更新DA值达到输出连续的波形部分函数如

下：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name :u16 \*BSP\_DacGetBuff100Bit(void)

\* Description : 获取DAC更新BUFF首地址，长度为100个u16 数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

u16 \*BSP\_DacGetBuff100Bit(void)

{

return &Escalator100Bit[0];

}

**4.6 系统图形化设计**

这一部分设计主要采用了一些简单的用户界面GUI，同时也设计了相关用于构建GUI的函数。

u16 GUI\_Color565(u32 RGB); // RGB颜色转为16位(565)

void GUI\_Text(u16 x, u16 y, u8 \*str, u16 len,u16 Color, u16 bkColor); //插入图片，

其中str为图片转换的数组指针

void GUI\_Line(u16 x0, u16 y0, u16 x1, u16 y1,u16 color); //画线

void GUI\_Circle(u16 cx,u16 cy,u16 r,u16 color,u8 fill); //画圆

void GUI\_Rectangle(u16 x0, u16 y0, u16 x1, u16 y1,u16 color,u8 fill); //画矩形

void GUI\_Square(u16 x0, u16 y0, u16 with, u16 color,u8 fill); //画正方形

void Touch\_Initializtion(void);

u16 Touch\_GetPhyX(void);

u16 Touch\_GetPhyY(void);

u16 Touch\_MeasurementX(void);

u16 Touch\_MeasurementY(void);

通过各个函数间的组合构成些简单的GUI

**4.6.1 操作系统的应用**

在设计示波器时，任务的切换和图形界面菜单的选择参照《嵌入式实时操作系统》进行UC/OS在STM32 的移植。

UC/OS 可以管理多达64个任务，参看一些示波器设计在本设计中预设的任务有：信号发生器任务(Generator Task)、示波器任务(OSC Task)、触摸屏控制任务 (Touch Task)、GUI显示任务(Display Task)。

UC/OS系统主要的特点如下:

UC/OS体积小，便于裁剪，可裁剪至2kb

UC/OS还包括全部实时系统所具有的功能，全部编译后的UC/OS的内核仅

有6-10KB。

UC/OS虽然占用空间比较小但是他的拓展能力很强，可以根据自己需要添加文件系统等操作。

在STM32中设计了如下任务并创建包括信号发生器任务(Generator Task)、 示波器任(OSC Task)、触摸屏控制任务(Touch Task)、GUI显示任务(Display Task)。如下函数：

static void TaskStart(void)

{

OS\_ERR err;

led\_init();

SysTickInit();

OSStatTaskCPUUsageInit((OS\_ERR \*)&err);

OSTaskCreate( (OS\_TCB \*)&task1TCB,

(CPU\_CHAR \*)"Generator Task",

(OS\_TASK\_PTR)AppGeneratorSerive, //信号发生器任务

(void \*)0,

(OS\_PRIO )GeneratorTASK\_PRIO,

(CPU\_STK \*)&GeneratorStk[0],

(CPU\_STK\_SIZE)TASK1\_STK\_SIZE / 10,

(CPU\_STK\_SIZE)TASK1\_STK\_SIZE,

(OS\_MSG\_QTY )0,

(OS\_TICK )0,

(void \*)0,

(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR),

(OS\_ERR \*)&err);

OSTaskCreate( (OS\_TCB \*)&task2TCB,

(CPU\_CHAR \*)"OSC Task", //示波器任务

(OS\_TASK\_PTR)AppOSCSerive,

(void \*)0,

(OS\_PRIO ) OSCTASK\_PRIO,

(CPU\_STK \*)&OSCStk[0],

(CPU\_STK\_SIZE)TASK2\_STK\_SIZE / 10,

(CPU\_STK\_SIZE)TASK2\_STK\_SIZE,

(OS\_MSG\_QTY )0,

(OS\_TICK )0,

(void \*)0,

(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR),

(OS\_ERR \*)&err);

OSTaskCreate( (OS\_TCB \*)&task3TCB,

(CPU\_CHAR \*)"Touch Task",

(OS\_TASK\_PTR)TouchSerive, //触摸屏任务

(void \*)0,

(OS\_PRIO )TouchTask\_PRIO,

(CPU\_STK \*)&TouchStk[0],

(CPU\_STK\_SIZE)TASK3\_STK\_SIZE / 10,

(CPU\_STK\_SIZE)TASK3\_STK\_SIZE,

(OS\_MSG\_QTY )0,

(OS\_TICK )0,

(void \*)0,

(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR),

(OS\_ERR \*)&err);

OSTaskCreate( (OS\_TCB \*)&dispTCB,

(CPU\_CHAR \*)"LCD display",

(OS\_TASK\_PTR)GUIAPP\_Task, //显示任务

(void \*)0,

(OS\_PRIO )Disp\_PRIO,

(CPU\_STK \*)&dispStk[0],

(CPU\_STK\_SIZE)TASK4\_STK\_SIZE / 10,

(CPU\_STK\_SIZE)TASK4\_STK\_SIZE,

(OS\_MSG\_QTY )3,

(OS\_TICK )0,

(void \*)0,

(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR),

(OS\_ERR \*)&err);

OSTaskCreate( (OS\_TCB \*)&guiTCB,

(CPU\_CHAR \*)"gui task",

(OS\_TASK\_PTR)GUISerive, //GUI控制任务

(void \*)0,

(OS\_PRIO )GUI\_PRIO,

(CPU\_STK \*)&guiStk[0],

(CPU\_STK\_SIZE)TASK4\_STK\_SIZE / 10,

(CPU\_STK\_SIZE)TASK4\_STK\_SIZE,

(OS\_MSG\_QTY )3,

(OS\_TICK )0,

(void \*)0,

(OS\_OPT)(OS\_OPT\_TASK\_STK\_CHK|OS\_OPT\_TASK\_STK\_CLR),

(OS\_ERR \*)&err);

OSSemCreate( (OS\_SEM \*)&taskSem,

(CPU\_CHAR \*)"taskSem",

(OS\_SEM\_CTR )0,

(OS\_ERR \*)err);

OSTaskDel( (OS\_TCB \*)&taskStartTCB,

(OS\_ERR \*)&err);

}

**4.6.2 图形化界面设计**

UCOS作为一种开源的部分代码免费的操作系统，具有代码小移植性强、易裁剪性，被得到广泛的应用。由于时间比较紧，在设计这部分时，本设计中直接移植了UCOS操作系统，并简要的查找了相关代码函数功能。

图形用户界面简称GUI，在本设计中设计了简单的、友好的操作界面，包含有开机画面、菜单栏以及信号输入所需的Numpad，如下图4.6和4.7所示。

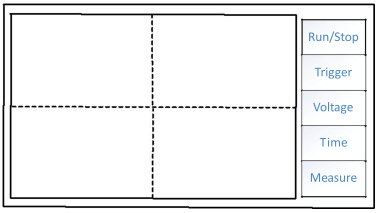


图4.6 初始化界面

相关的操作:

点击Run/Stop. 运行或者停止示波器；

点击Trigger,可以调节触发线；

点击Voltage,更改电压赋值的格点大小；

点击Time，调节水平缩放；

点击Measure,则可测量相关参数；

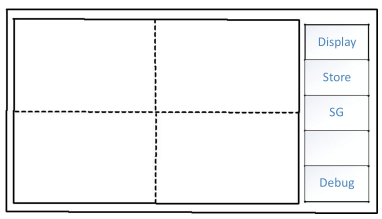


图4.7另一页菜单

相关操作

点出Displays示或隐藏当前信号波形；

点击Store存储功能(在本设计中未设计)；

点击SG信号发生器功能在内部设置了发生波形、频率；

点击Debug5在看当前系统内存的占用率；

当点击某些菜单时需要输入某些参数，比如在信号发生器中需要键人相关参数:电压的最大值、最小值以及频率大小。因此在设计这些菜单时还需要创建虚拟键盘，通过触摸屏的控制输入相关数值键位。包括有(O、1、2、3、4、5、6、7、8、9、Clr清除、Ent确定)。

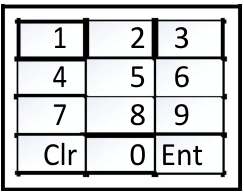


图4.8 Numpad键盘输入

每一个按键对应一个中断处理函数，譬如在触摸屏点击0，则会对应的在系统内部输人“0”。

**4.7 本章小结**

本章主要是对软件部分进行了细致的分析。包括A/D转换调用定时器的定时采样，和AID的交替采样，TFT分析设置等。本章函数多是根据STM32系列的函数库V3.5, 和UCGUI的相关函数并且设计了简单的GUI菜单操作。引用库函数的目的是能够在短时向内了解内部结构，便于快速掌握。

**5. 系统调试与结果**

**5.1 系统测试方案**

测试时，需要验证电路和方案是否正确，在测试中设计了前端信号测量、采样方案的选择、参数的测量。

**5.1.1 前端信号测量**

该设计的输入信号显示在示波器上。由于电压不能为负，因此需要设计输入电压。电压波动应在A / D转换电压范围内。为了确保输入电压为正，对电压进行预处理以改善电压位置。

方案1：使用集成运算放大器和外部直流电压偏移（使用参考电压源）来增加和减少信号电压。使用运算放大器添加。每个点的电压值发生变化，A / D转换范围保持不变，波形不易失真。以下模拟可用于比较分析。如果输入模拟信号是幅度正弦波（-5~5v），如图5.1所示。

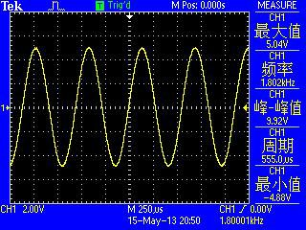
。

图5.1 输入模拟信号

通过信号调理后信号输出为图5.2所示。

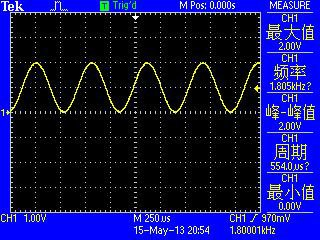


图5.2 输出信号

由此可以看出，经过信号变换后幅值满足（0～2V）的要求。

将输入和输出信号进行对比，可得到图5.3所示。

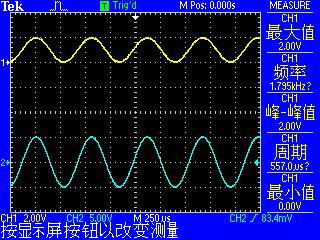


图5.3输入和输出信号对比

可以看出，在输入信号和输出信号之间的比较中没有失真或相移。

方案2：串联电阻以增加电压。为了提高精度，有必要精确计算电阻，这会影响结果的输出。简而言之，选择集成运算放大器来增加电压。

**5.1.2 信号采样方案**

方案1：外部模拟数字转换电路.ADS7812转换速度可达2M/ S，但精度不高，因此被丢弃。

方案2：STM 32与其自己的12位ADC一起使用。当时钟设置为56M时，ADC转换时间为1us，电压转换范围为0~3.6v，具有双采样保持功能，外围电路相对简单。

简而言之，选择ARM自己的AD转换模块。

**5.1.3 参数测量方案**

本设计中通过AD测量分别算出所测的实时信号的峰峰值(Peak-peak) ,最大值(Vmax)，最小值(Vmin)，实时信号频率(Freq)，占空比(Duty)。

**5.1.4 最大值、最小值、峰峰值测量**

最大值、最小值测量是采用在for循环中，逐次比较各个数值的大小。

for(i=1;i< (len>> 1); ++) //逐次比较

{

tmp.volt.arg += arr[i];

if (arr[i ] < tmp.volt.min)

tmp.volt.min = arr[i]; if

(arr[i] > tmp.volt.max)

tmp.volt.max = arr[i]

｝ //最大值、最小值

tmp.volt.vpp = tmp.volt.max -ump.volt.min； //峰峰值

**5.1.5 频率测量**

方案1：比较器比较并输出波形以形成方波，并且通过计算从方波的上升沿到数字时间的周期来测量频率一次。缺点：电路复杂，信号比较必须单独提取，计算误差大。如图5.4所示，

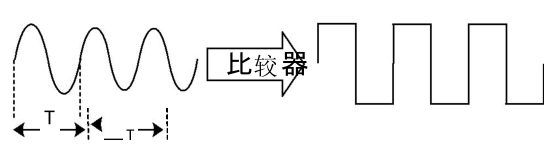
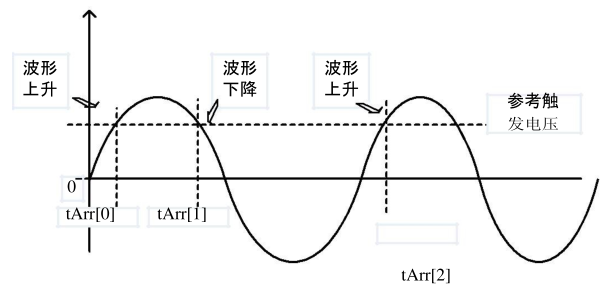


图5.4比较器方案

在测量过程中，通过比较将信号输出为方波，并通过计算上升沿（或下降沿）来记录每单位时间的周期数，以便测量周期和频率。该方案的缺点是：电路复杂，信号比较和输出需要单独处理，I / O占用，计算误差大，尤其是非周期信号。

方案2：采用三个相邻点，ar [0]，ar [1]和ar [2]，如图5.5所示。



图S.5频率测量示意图

通过判断ar[0]、ar[1]、ar[2]和触发线电压level的大小来判断波形处于上升或是下降。具体如下代码:

static int AppOSCCheckTrigger(s16 \*arr, int level)

{

if ((arr[0] <= level) && (arr[1] >= level) && (arr[2] >=level))

return UP; //上升

if ((arr[0] >= level) && (arr[1] <= level) && (arr[2] <=level))

return DOWN; //下降

return 0;

}

若上升则把此时的计数值赋给tArr[0],此后若满足下降条件则把那时刻计数值赋给tArr[1],若在下降后又遇到上升条件则把此时的计数值赋给tArr[2]，具体是通过switch语句切换执行。

switch (index)

{

case 0:

if (AppOSCCheckTrigger(&arr[i - 1], level) == slope) //上升

{ tArr[0] = 0;

start = i;

index++;}break;

case 1:

if (AppOSCCheckTrigger(&arr[i - 1], level) == \_slope) //下降

{

end = i;

index++;

}

break;

case 2:

if (AppOSCCheckTrigger(&arr[i - 1], level) == slope) //再次遇到上升波形 {

tArr[1] += end - start;

tArr[2] += i - start;

start = i;

index = 1; //返回case1循环进行筛选判断

argCont++;

}

break;

}

周期：fTmp = tArr[2]–tArr[0]

频率：Freq = 1/fTmp;

通过此方法计算出的频率精度比较高，并且不需要外引出电路进行信号变换。故选方案2。

**5.1.6 占空比**

通过以上的测量，根据占空比(方波)的定义可知式5-1:

Duty=（tArr[1]-tArr[0]/tArr[2]-tArr[0]）\* 100% (式5-1)

**5.2 系统测试及分析**

**5.2.1 系统设计平台**

设计中包括有硬件设计平台和软件设计系统。

硬件设计平台: Alium Designer、Tina 仿真、Protues、 Keil4、 FilterPro 滤波器设计、J-link仿真工具。

软件设计系统: DataFit数据拟合软件辅助数据校准、Officc2013文档编辑、Visio2013流程图绘制。

**5.2.2 系统测试仪器**

直流稳压电源(±12V)，Rigol信号发生器，万能表，泰克TDS1012C示波器（100MHz）。

**5.2.3 测量步骤**

(1)信号发生器输入-5～5V正弦信号，送入电路经过信号调理后测的输出电压是否在0～2V之间。

(2)测量衰减电路的输出电压，与起始输入相比波形是否失真。

(3)调节输入信号的频率，观察在0～200KHz范围内是否失真。

(4)以上测试通过后， 开始具体某个信号的相关参数的测量井与泰克示波器的示值进行对比，校正以提高准确度。

(5)通过软件修正相关系数提高精度。

**5.3 测试结果分析**

本设计的测试分为了基本功能(示波器)的测试和发挥部分(信号发生器)的测试。在测试时需要验证相关参数与预设目标是否相同。

相关参数设计：

输入阻抗:在前面硬件设计时，我们设计了输入电阻Ri=1MΩ、满足所设计要求Ri=100k条件。

垂直灵敏度: 0.1V/div，1V/div。

**5.3.1 基本功能测试**

刚试时采用输入标准的信号发生器，包括有峰峰值、频率、最小值、最大值、占空比(对于方波)与所设计的系统的测量值进行对比，并测量其误差：

在测试时分为频率为1KHz，50Hz和100kHz，而峰峰值分别设置为2.0、4.0、6.0、8.0、10.0V，分别测试方波、正弦波和三角波。

1.方波测试

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） | 占空比（Duty） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 1.0k | 50% |
| 4.0V | 2.0V | -2.0V | 1.0k | 50% |
| 6.0V | 3.0V | -3.0V | 1.0k | 50% |
| 8.0V | 4.0V | -4.0V | 1.0k | 50% |
| 10.0V | 5.0V | -5.0V 五号 | 1.0k | 50% |
| 测试结果 | 2.07V | 1.09V | -0.997V | 996 | 49.8% |
| 4.01V | 2.06V | -1.95V | 997 | 49.8% |
| 5.98V | 3.07V | -2.96V | 997 | 49.8% |
| 7.90V | 4.12V | -3.87V | 1.0k | 49.7% |
| 10.1V | 5.12V 以下红色部分都要改正 | -4.86V | 997 | 49.8% |
| 误差% | 1.267% | 4.014% | 3.983% | 1.01% | 0.01% |

表5.1 输入标准信号为（-5～5V）频率为1kHz方波

误差分析：在输入频率较小的时候，采样的速度能够满足要求，波形比较标准



图5.1 输入1kHz方波 图 5.2 TFT波形

当输入方波为10kHz时，结果如表5.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） | 占空比（Duty） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 1k | 50% |
| 4.0V | 2.0V | -2.0V | 50k | 50% |
| 6.0V | 3.0V | -3.0V | 50k | 50% |
| 8.0V | 4.0V | -4.0V | 50k | 50% |
| 10.0V | 5.0V | -5.0V | 50k | 50% |
| 测试结果 | 2.10V | 1.11V | -0.951V | 49.94k | 49.8% |
| 4.09V | 2.0V | -1.91V | 49.98k | 49.8% |
| 5.98V | 3.07V | -2.96V | 49.97k | 49.8% |
| 7.90V | 4.12V | -3.87V | 49.90k | 49.7% |
| 10.1V | 5.08V | -4.81V | 49.91k | 48.8% |
| 误差% | 4.267% | 5.038% | 6.983% | 4.01% | 0.09% |

表5.2输入标准信号为（-5~5V）频率为50kHz方波

误差分析：在输入频率为50kHz时候，由于频率的增加在测量时波形的波动性也比较大，因此误差相对比1kHz大一点，为了测试该系统的最大测量频率，如表3在测试时调节输入的标准频率为100kHz，观察测试结果和波形是否失真

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 100k |
| 4.0V | 2.0V | -2.0V | 100k |
| 6.0V | 3.0V | -3.0V | 100k |
| 8.0V | 4.0V | -4.0V | 100k |
| 10.0V | 5.0V | -5.0V | 100k |
| 测试结果 | 2.01V | 1.02V | -0.998V | 98.7k |
| 4.09V | 2.1V | -1.90V | 99.7k |
| 6.1V | 3.1V | -2.7V | 99.7k |
| 8.06V | 3.9V | -4.0V | 99.7k |
| 10.2V | 5.1V | -4.9V | 99.7k |
| 误差% | 1.433% | 7.038% | 7.444% | 6.01% |

表5.3输入标准信号为（-5~5V）频率为100kHz方波

误差分析：在输入频率为100kHz时候，由于频率较大，波形的波动性也比较大，因此误差相对比1kHz和50kHz大很多，其原因主要是处理器的性能不能满足。

2.正弦波测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 正弦波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 1k |
| 4.0V | 2.0V | -2.0V | 1k |
| 6.0V | 3.0V | -3.0V | 1k |
| 8.0V | 4.0V | -4.0V | 1k |
| 10.0V | 5.0V | -5.0V | 1k |
| 测试结果 | 1.95V | 1.1V | -0.890V | 1k |
| 3.92V | 2.07V | -1.97V | 985 |
| 5.85V | 3.04V | -2.87V | 1k |
| 7.88V | 4.04V | -3.97V | 997 |
| 10V | 5.05V | -4.9V | 996 |
| 误差% | 3.4% | 5.038% | 7.63% | 2.01% |

表5.4输入标准信号为（-5～5V）频率为1kHz正弦波

误差分析：在较小的频率范围内，误差会比较小

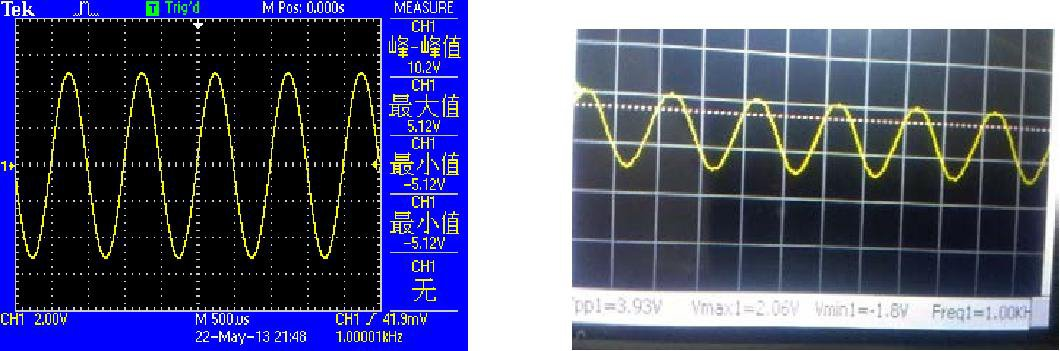


图5.3 输入标准1k正弦波 图5.4 TFT 显示1k正弦波

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 正弦波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 50k |
| 4.0V | 2.0V | -2.0V | 50k |
| 6.0V | 3.0V | -3.0V | 50k |
| 8.0V | 4.0V | -4.0V | 50k |
| 10.0V | 5.0V | -5.0V | 50k |
| 测试结果 | 1.95V | 1.1V | -0.901V | 49.7k |
| 3.92V | 2.04V | -1.8V | 49.3k |
| 6.0V | 3.04V | -2.87V | 49.3k |
| 7.88V | 4.04V | -3.8V | 49.7k |
| 9.89V | 4.95V | -4.87V | 49.7k |
| 误差% | 4.4% | 6.038% | 7.63% | 3.01% |

表5.5输入标准信号为（-5~5V）频率为50kHz正弦波

误差分析：在输入频率为50k的时候，误差会相应的增大一些。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 三角波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 100k |
| 4.0V | 2.0V | -2.0V | 100k |
| 6.0V | 3.0V | -3.0V | 100k |
| 8.0V | 4.0V | -4.0V | 100k |
| 10.0V | 5.0V | -5.0V | 100k |
| 测试结果 | 1.95V | 1.1V | -0.901V | 99.7k |
| 3.92V | 2.04V | -1.8V | 99.3k |
| 6.0V | 3.04V | -2.87V | 99.3k |
| 7.88V | 4.04V | -3.8V | 99.7k |
| 9.89V | 4.95V | -4.87V | 99.7k |
| 误差% | 4.4% | 6.038% | 7.63% | 3.01% |

表5.6输入标准信号为（-5～5V）频率为100kHz正弦波

误差分析：当输入信号比较大的时候，由于采样能力的限制，误差会相对增大。不过在测试中所测量的频率满足所设计的要求。

3.三角波测试

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 三角波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 1k |
| 4.0V | 2.0V | -2.0V | 1k |
| 6.0V | 3.0V | -3.0V | 1k |
| 8.0V | 4.0V | -4.0V | 1k |
| 10.0V | 5.0V | -5.0V | 1k |
| 测试结果 | 2.01V | 1.11V | -0.996V | 1k |
| 4.04V | 2.05V | -1.89V | 910 |
| 6.09V | 2.95V | -3.1V | 994 |
| 8.05V | 4.06V | -3.8V | 1k |
| 9.91V | 5.05V | -4.7V | 1k |
| 误差% | 5.4% | 7.038% | 6.63% | 5.01% |

表5.7输入标准信号为（-5～5V）频率为1kHz三角波

误差分析：在频率较小的时候，所测量的相关参数比较准确。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 三角波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 50k |
| 4.0V | 2.0V | -2.0V | 50k |
| 6.0V | 3.0V | -3.0V | 50k |
| 8.0V | 4.0V | -4.0V | 50k |
| 10.0V | 5.0V | -5.0V | 50k |
| 测试结果 | 1.95V | 1.1V | -0.901V | 49.7k |
| 3.92V | 2.04V | -1.8V | 49.3k |
| 6.0V | 3.04V | -2.87V | 49.3k |
| 7.88V | 4.04V | -3.8V | 49.7k |
| 9.89V | 4.95V | -4.87V | 49.7k |
| 误差% | 4.4% | 6.038% | 7.63% | 3.01% |

表5.8输入标准50k（-5～5V）三角波

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 三角波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 信号发生器标准信号 | 2.0V | 1.0V | -1.0V | 100k |
| 4.0V | 2.0V | -2.0V | 100k |
| 6.0V | 3.0V | -3.0V | 100k |
| 8.0V | 4.0V | -4.0V | 100k |
| 10.0V | 5.0V | -5.0V | 100k |
| 测试结果 | 1.95V | 1.1V | -0.901V | 99.7k |
| 3.62V | 2.04V | -1.8V | 99.1k |
| 6.1V | 3.04V | -2.87V | 99.3k |
| 7.89V | 4.04V | -3.8V | 99.7k |
| 9.85V | 4.90V | -4.85V | 99.6k |
| 误差% | 5.4% | 6.038% | 7.63% | 4.01% |

表5.9输入标准信号为（-5～5V）频率为100kHz 三角波

误差分析:随着测量频率变大时，所测量的精度也就会变小，误差会增大。

**5.3.2 发挥部分测试**

在发挥部分设计利用了STM32内部的DA进行循环输出模拟信号，预设值为输出正弦波、错齿波、方波。并且最大值(Vmax)、最小值(Vmin)、 峰峰值(Vpp)、频率(Fre)。

在测试的时候通过D/A输出模拟信号，经滤波后使用泰克示波器进行测量观察相关参数，即:最大值、最小值、峰峰值、频率和占空比(方波)，井且观察波形是否失真，在这里我们只是简单的测了点数据。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） | 占空比（Duty） |
| 系统输出 | 2.30 | 2.50 | 0.2 | 1k | 50% |
| 2.30 | 2.5 | 0.2 | 10k | 50% |
| 2.30 | 2.5 | 0.2 | 20k | 80% |
| 标准示波器测试 | 2.44 | 2.60 | 0.16 | 997 | 49.7% |
| 2.37 | 2.54 | 0.18 | 9.89k | 50.1% |
| 2.37 | 2.54 | 0.16 | 19.95 | 79.9% |

表5.10 D/A 输出方波测试

以下是相关测试波形

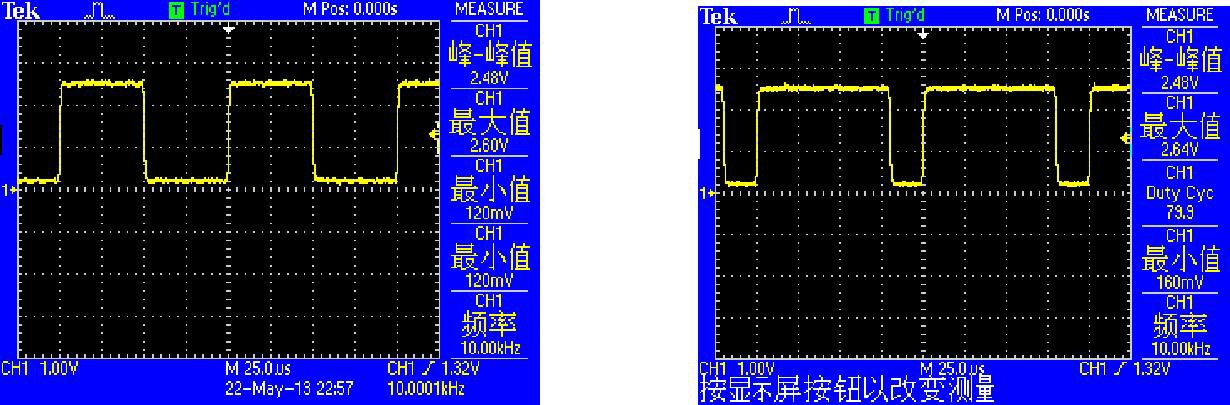


图5.5 50%的10k方波 图5.6 80%的10k方波

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 正弦波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 系统输出 | 2.30 | 2.50 | 0.2 | 100 |
| 1.3 | 2.50 | 1.2 | 20k |
| 标准示波器测试 | 2.39 | 2.52 | 0.16 | 100 |
| 1.40 | 2.52 | 1.12 | 20.1k |

表5.11 D/A 输出为正弦波测试

以下是测试波形

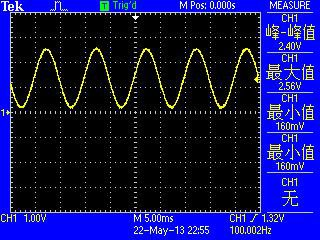


图5.7 输出100hz正弦波

以下是部分测试波形

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 锯齿波信号 | 峰峰值（Vpp） | 最大值（Vmax） | 最小值（Vmin） | 频率（Fre） |
| 系统输出 | 2.30 | 2.50 | 0.2 | 1k |
| 2.3 | 2.50 | 0.2 | 10k |
| 标准示波器测试 | 2.39 | 2.52 | 0.16 | 997 |
| 2.24 | 2.48 | 0.2 | 9.99k |

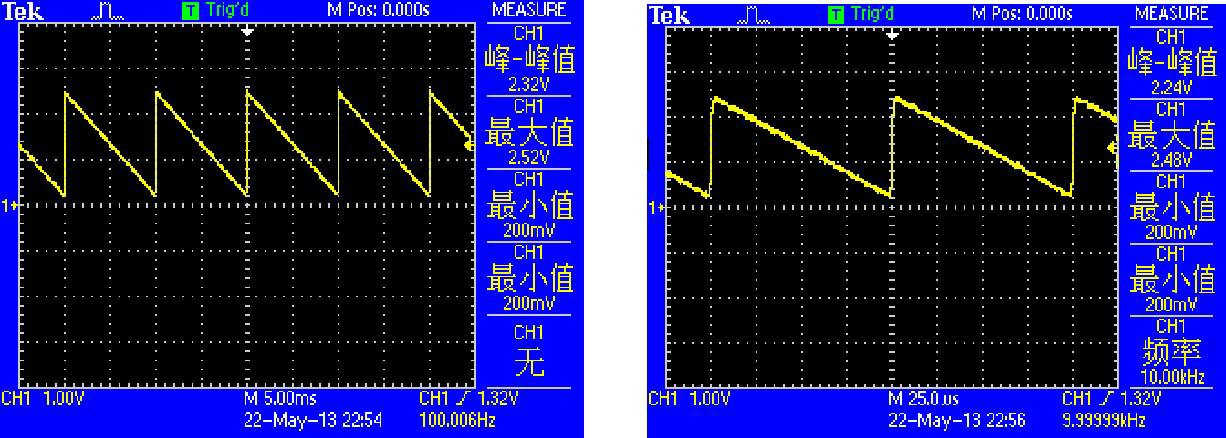
 表5.12 D/A锯齿波测试

图5.8输入100MHz锯齿波 图5.9 输出10k的锯齿波

信号发生器的基本功能和测试结果基本满足预定所设计的要求

**6. 总结与展望**

本设计系统能够完成对周期信号和非周期信号的实时显示，由于采用TFT(240\*320)的3.2寸的屏幕，实现周期波形的左右平移，以及幅值的缩放，在一定的频率范围内，所显示的波形无失真，并对相关参数进行测量。

此外，利用STM32内部的DA使用DMA循环触发，模拟输出正弦波、方波和锯齿波，相关参数满足预定设计的要求。

本设计所采用STM32优势:

(1)应用STM32自带的AD转换模块，节省了通道的时间和空间设置，以及外部的温度补偿。

(2)应用STM32的内部资源DMA等，使得系统资源得到优化。当然在本设计中还有很多不足之处有待改进，还有许多功能有待拓展。在测量一些较高的频率时波形会发生失真。追究其原因，主要是STM32系统时钟限制了所采样的频率，若使用较高端的处理器(如ARM9、ARMII等)，那么测量的精度会提高很多。

**主要参考文献**

1. 黄智伟、朱卫华．STM32F32位ARM微控制器应用设计与实践[M]. 北京航空航天大学出版社．2012：19-38

段前段后各空一行，小二号宋体，加粗

1. 蒙博宇．STM32自学笔记[M]. 北京航空航天大学出版社．2012：34-56
2. （美）拉伯罗斯著，宫辉等译．嵌入式实时操作系统μC/OS-III[M]．(第2 版)．2011：67-79

此处没有点。

结尾处都有全角黑点。

1. 薛圆圆．21 天学通 ARM开发嵌入式开发[M]．(第 2 版). 电子工业出版社．2011：58-61
2. 刘军．例说STM32 [M]. 北京航空航天大学出版社. 2011：15-24

行首空2格，括号后空一格，再接相应的文献信息。

1. 黄智伟．全国大学生电子设计竞赛训练教程[M]. 电子工业出版社．2010：14-21
2. ST技术支持．*STM32 Reference manua* [J]．2009：13-22
3. ST技术支持．STM32F系列 ARM 内核32位高性能微控制器参考手册 [J]．2009：11-26
4. ST User Manual．*Using the STM32F101xx and STM32F103xx DMA controller* [J]．2009：6-16
5. TFT 显示驱动．*TFT LCD Single Chip Driver 240RGBx320 Resolution and 16.7M color*[M]．2009： 5-12
6. ST．*ARM®-based 32-bit MCU STM32F101xx and STM32F103xxfirmware library* [M]．2008：7-17
7. ST Application note．*TFT LCD interfacing with the high-density STM32F10xxx FSMC*[J]．2008： 5-12
8. 邵贝贝著．嵌入式实时操作系统μC/OS-II．(第 2 版)[M]. 北京航空航天大学出版社．2007：
9. 刘全,韩大强,李柯霖．便携式20M数字存储示波器[J]. 电子制作. 2005：70-75.
10. ARM 公司.ARM Architecture Reference Manual [M].2000：
11. THS3091．*HIGH-VOLTAGE,LOW-DISTORTION,CURRENT-FEEDBACK OPERATIONALAMPLIFIERS*[J].TI Datasheet，2009： 5-12
12. OPA227《*High Precision,Low Noise OPERATIONALAMPLIFIERS*》 [J].Datasheet.Page7~9
13. ADS7843 Touch Screen Controller Datasheet. [J]，2008：15-18
14. ST Application Note《*TFT LCD interfacing with the high-density STM32F10xxx FSMC*》[J]，2006：Page16-20
15. ST Application Note 《*How to achieve 32-bit timer resolution using the link system in STM32F10x microcontrollers*》[J]，2005：7-17
16. ST PPT《*STM32 – 32-bit Cortex™-M MCUs*》[J]，2004：1-13

**致 谢**

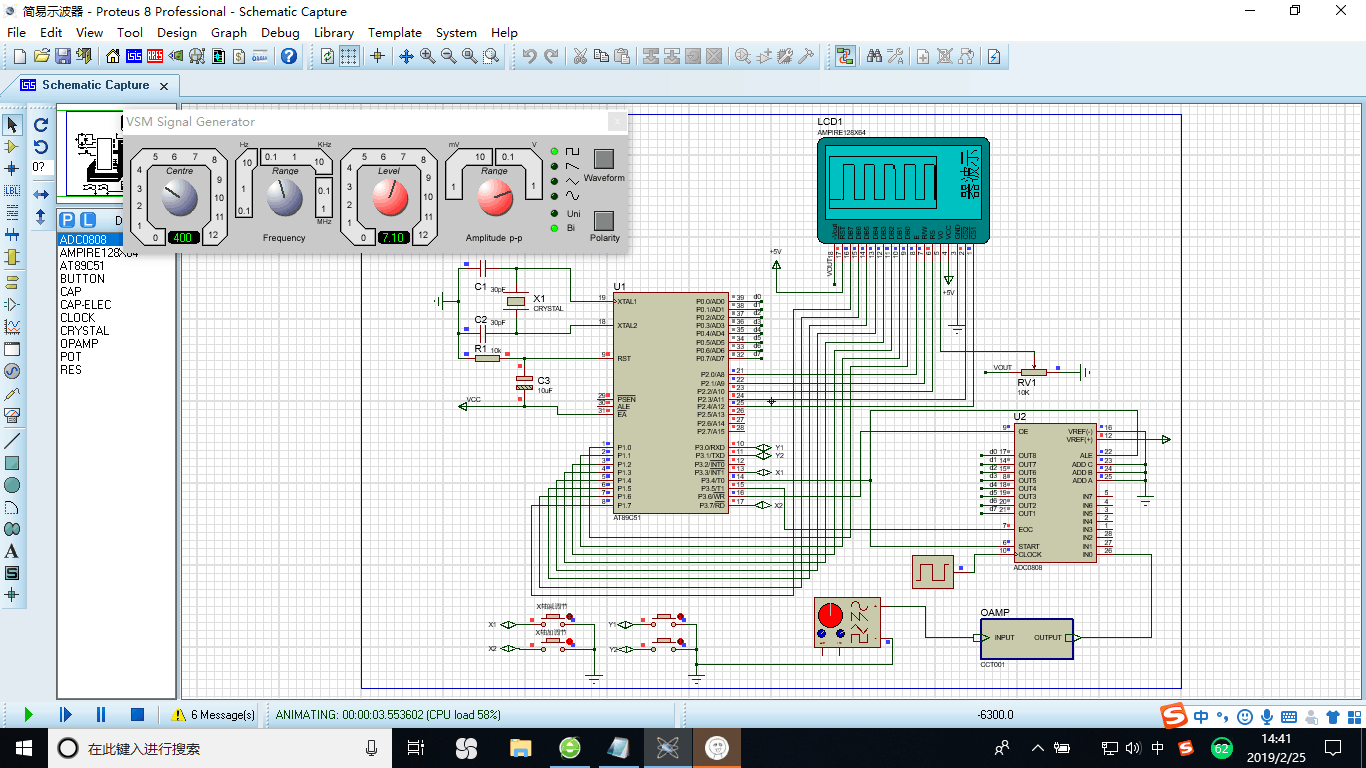
通过这次毕业设计很好的检验了我在大学四年中所学理论知识的掌握程度，锻炼了我的实际动手能力，为以后的学习、工作奠定了基础。回顾这几个月的设计制作，感受良多。

首先感谢刘小燕老师，感谢您在我的毕业设计期间对我的亲切关怀和悉心指导。在您的指导下，我学到了许多知识，还锻炼了对知识的应用能力，这些将使我终生受益匪浅。在此向您致以最诚挚的敬意。

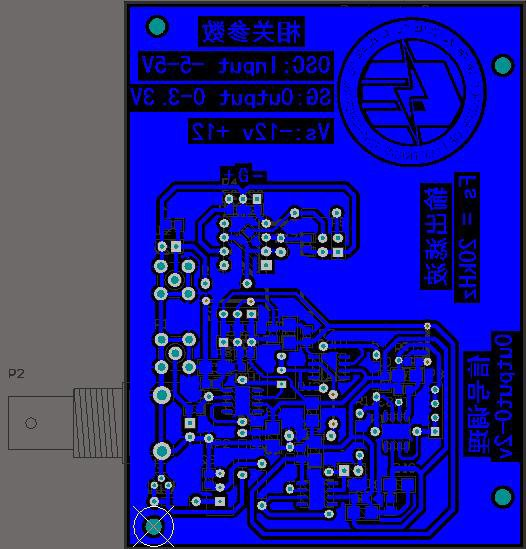
在试验调试的过程中，感谢学校在试验室及网络资料查阅方面提供的帮助。感谢赣南师范大学的所有老师，你们四年的教诲，让我成熟进步许多:感谢通信工程班的所有同学,你们如兄弟姐妹般的帮助，让我感受到友谊的珍贵。

感谢所有对我论文进行了评审和答辩的老师。

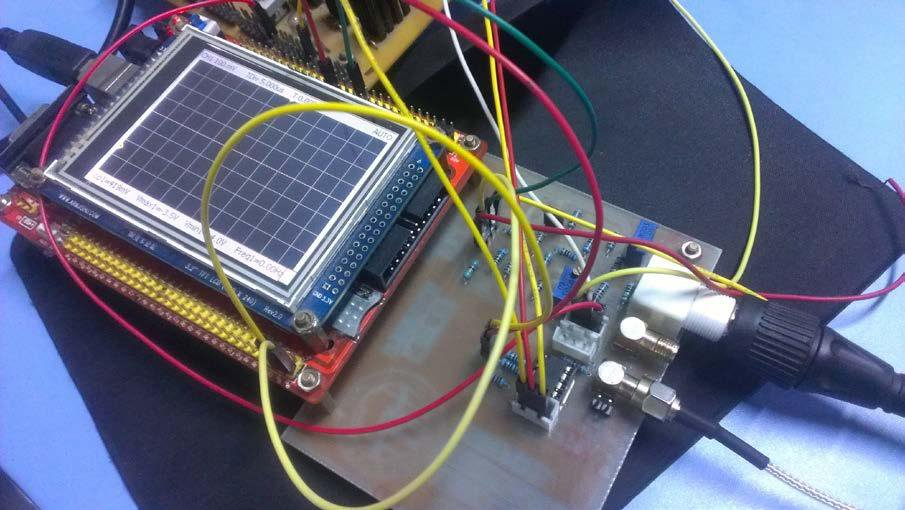
**附录1 原理图**



**附录2 PCB布局布线**



**附录3 实物图**



**附录4 元器件清单**

|  |  |  |
| --- | --- | --- |
| 名称 | 说 明 | 数量 |
| 电 阻 | 1k、1.21k、1M 等 | 若干 |
| 电 容 | 104、103、22、10uF( 钽电容) | 若干 |
| BNC | 示波器接头 | 1 |
| SMA | 信号发生器输出 | 2 |
| OPA227 | 运放 | 1 |
| THS3091 | 运放 | 4 |
| STM32F103ZET6 | 主控制器 | 2 |
| TFT 模块 | 触摸屏与控制 | 1 |
| 接插件 | 接插件 | 若干 |

**附录5 程序代码（部分）**

示波器数据采样及分析

1. 通过选定的时间轴确定采样周期，

2us～20us 高速1MHz采样

50us～50ms中速采样，采样周期等于选定的时间轴

100ms～1s低速采样，采样周期等于选定的时间轴，循环采样实时显示

1. 采样数据转为显示的y坐标数据。
2. 通过触发线，确定波形数据范围。
3. 通过设定的时间，幅值中心点，放大倍数，调整显示波形。
4. 计算波形参数，Vpp，Vmax，Vmin，Freq，Duty等。
5. 发送显示数据。

#include "AppOSC.h"

#include "GUIAPP.h"

#include "os.h"

#include "string.h"

#include "App.h"

#include "Math.h"

#include "stdlib.h"

#define FULL\_VOLT\_VOLUE 3300.0 // 3300mV

#define HALF\_VOLT\_VOLUE (FULL\_VOLT\_VOLUE / 2)

typedef void FunCal(u16 \*, u16 \*, u16 \*, u16);

OS\_MEM pMem;

static ParamStruct \*\_apParam = &OSCParam;

static ParamStruct \_aParamLast = {0};

static volatile int \_vSampleCount = 0; // 总采样数为可变变量

static const int \_cSampleCmpStart= MAX\_VALUE ; //adc采样中的buff于电平比较的起点

static const int \_cSampleCmpEnd= MAX\_VALUE \* 2; //adc采样中的buff于电平比较的终点

static u16 \_aMemSize[3][MAX\_VALUE \* 3] = {0,};

static const float \_fCoef = (FULL\_VOLT\_VOLUE / 0xFFF); //电压与 AD值的系数 (AD \* \_fCoef = V) --> AD/4096

//显示的 y 坐标值与 AD采样的 AD 值，对应系数(AD - 2047 = \_fDisToADCoef\* scale \* Y)

static const float \_fDisToADCoef = 0xFFF \* 1000 / GRAPH\_SCALE\_V /FULL\_VOLT\_VOLUE; //AD采样值与显示的y坐标值对应系统(Y=(AD-2047)\* \_fADToDisCoef/scale)

static const float \_fADToDisCoef =FULL\_VOLT\_VOLUE \* GRAPH\_SCALE\_V/0xFFF /1000;

static FunCal \* volatile \_pFunCal = NULL;

static int AppOSCGetLevel(ParamStruct \*pParam)

{

float fTmp;

int tmp;

tmp = pParam->trigger.source? pParam->ch2.vertical.scale :

pParam->ch1.vertical.scale; //通过触发源选择出幅度对应电压值

fTmp = MemItoF(tmp);

tmp=pParam->trigger.source?pParam->ch2.vertical.pos:pParam->ch1.vertical.pos; //计算出触发线对应的AD值

return (int)((fTmp \* (pParam->trigger.level - tmp) \* \_fDisToADCoef) +HALF\_VOLT\_VOLUE);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name :static int AppOSCCheckTrigger(s16 last, s16 cur, u16 level)

\* Description :arr 前后检查3个数据 判断上升还是下降

\* Input :arr 为向前三个数据

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static int AppOSCCheckTrigger(s16 \*arr, int level)

{

if ((arr[0] <= level) && (arr[1] >= level) && (arr[2] >=level))

return UP; //上升

if ((arr[0] >= level) && (arr[1] <= level) && (arr[2] <=level))

return DOWN; //下降

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name :static void AppOSCSlowModeGetValue(u16 \*ch1, u16 \*ch2, u16

\*source, u16 len)

\* Description : 获取采样数据 慢速度采样模式50ms<t以上的时间轴

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCSlowModeGetValue(u16 \*ch1, u16 \*ch2, u16 \*source, u16 len)

{

while (len--)

{

\*ch1++ = \*source++;

\*ch2++ = \*source++;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name :static void AppOSCNormalModeGetValue(u16 \*ch1, u16 \*ch2,

u16 \*source, u16 len)

\* Description : 获取采样数据 中等速度采样模式 获取缓冲区中的数据 20us < t<= 50ms

\*只有在采样数据被填充满后，才进行复制

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCNormalModeGetValue(u16 \*ch1, u16 \*ch2, u16 \*source, u16len)

{

if (BSPAdcConvertFlag()) //

{

while (len--)

{

\*ch1++ = \*source++;

\*ch2++ = \*source++;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name :static void AppOSCFastModeGetValue(u16 \*ch1, u16 \*ch2, u16\*source, u16 len)

\* Description : 获取采样数据，快速模式获取缓冲区中的数据

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCFastModeGetValue(u16 \*ch1, u16 \*ch2, u16 \*source, u16 len)

{

int i = 0;

int j = 0;

// 显示屏一网格时间放大10的9次方倍(ns)，再除以每网格的像素点数，1个像素点显示一个数

Int sampleTime = (int)(MemItoF(\_aParamLast.hor.scale)\* 1e9 /GRAPH\_SCALE\_H);

for (i = 1; i < ADC\_BUFF\_COUNT - 1; i++) // 初 始 化ADC\_BUFF\_COUNT = 1500

{

while (j < len)

{

if ((j \* sampleTime) >= (i \*1000))

break;

else

{

\*ch1++ = source[i << 1];

\*ch2++ = source[(i << 1) + 1];

j++;

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \* Function Name :void AppOSCSampleInit(void)

\* Description : 采样初始化

\* Input : fScale ,count ,fun

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCSampleInit(float fScale, int \*count, void \*\*fun)

{

OS\_ERR err;

int tmp = 0;

void \*pFunTmp = NULL;

BSPAdcClose();

OSTimeDly( (OS\_TICK )2,(OS\_OPT )OS\_OPT\_TIME\_DLY,(OS\_ERR \*)&err);

if (fScale > 2.001e-2) //50ms及以上缓冲区大小等于屏幕大小

{

int sampleTime = (int)(fScale \* 1e9 /GRAPH\_SCALE\_H); tmp = MAX\_VALUE;

BSPAdcRegSimultMode(sampleTime, tmp, 0); // 设置为循环采样

pFunTmp = AppOSCSlowModeGetValue;

}

else if (fScale > 2.001e-5) //20us 以上用定时器触发AD采样

{

// 显示屏一网格时间放大10的9次方倍(ns)，再除以每网格的像素点数，两个像素点显示一个数，所以再除2

int sampleTime = (int)(fScale \* 1e9 /GRAPH\_SCALE\_H); tmp = MAX\_VALUE \* 3;

BSPAdcRegSimultMode(sampleTime, tmp, 1); // 设置为正常采样

pFunTmp = AppOSCNormalModeGetValue;

}

else

{

//20us 及以下用高速采样模式

BSPAdcFastRegSimultMode();

pFunTmp= AppOSCFastModeGetValue;

tmp =MAX\_VALUE \* 3;

}

\*fun = pFunTmp;

\*count = tmp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name :static void AppOSCSmpTurnToDisY(float scale, s16 \*dis, s16\*smp, u16 len)

\* Description : 采样点转为显示屏上y坐标

\* Input :smp:采样值dis:坐标植 scale: 每网格对应的电压值

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCSmpTurnToDisY(float scale, s16 \*dis, s16 \*smp, u16 len)

{

if (scale == 0)

return;

while (len--)

{ //采样值先转为电压值再把电压值转为像素坐标

\*dis++ = (\*smp++ - HALF\_VOLT\_VOLUE) \* \_fADToDisCoef / scale;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function Name :static void AppOSCGetDisInfo(ParamStruct \*pParam, s16 \*arr,

u16 len)

\* Description :根据采样的数据，计算出各个参数:Vmax, Vmin, Varg, Freq, 等

\* Input :pParam:参数表， arr: 通过此数组数据进行计算，len:arr 数组数据各数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCGetDisInfo(ParamStruct \*pParam, s16 \*arr, u16 len)

{

MeasureStruct tmp = {0,};

volatile float fTmp ,volue\_max,volue\_vpp,volue\_min ,volue\_temp;

int i = 0; //

int start = 0, end = 0, argCont = 0;

int index = 0;

int tArr[3] = {0,};

int slope = pParam->trigger.slope; //边沿触发状态

int \_slope = !(slope - 1) + 1; //如果 slope == UP 时， \_slope ==DOWN;

float tScale = MemItoF(pParam->hor.scale) / GRAPH\_SCALE\_H; //水平刻度坐标

int level = AppOSCGetLevel(pParam);

if (pParam->trigger.source && !pParam->ch2.enable) //如果用触发源的通道不被显示，不进行计算

goto funEnd;

else if (!pParam->trigger.source && !pParam->ch1.enable)

goto funEnd;

tmp.volt.arg = tmp.volt.min = tmp.volt.max = tmp.volt.vpp = arr[0];

for (i = 1; i < (len >> 1); i++) //逐次比较

{

tmp.volt.arg += arr[i];

if (arr[i ] < tmp.volt.min)

tmp.volt.min = arr[i];

if (arr[i] > tmp.volt.max)

tmp.volt.max = arr[i];

switch (index)

{

case 0:

if (AppOSCCheckTrigger(&arr[i - 1], level) == slope) //上升

{

tArr[0] = 0;

start = i;

index++;

}

break;

case 1:

if (AppOSCCheckTrigger(&arr[i - 1], level) == \_slope) //下降

{

end = i;

index++;

}

break;

case 2:

if (AppOSCCheckTrigger(&arr[i - 1], level) == slope) //再次遇到上升波形

{

tArr[1] += end - start;

tArr[2] += i - start;

start = i;

index = 1; //返回case 1循环进行筛选判断

argCont++;

}

break;

}

}

if ((tArr[0] < tArr[1]) && (tArr[1] < tArr[2]))

{

fTmp = ((float)(tArr[2] - tArr[0]) / argCont) \* tScale;

tmp.time.period = MemFtoI(fTmp); //浮点数保存到整形变量中

fTmp = 1 / fTmp;

tmp.time.freq = MemFtoI(fTmp);

fTmp = ((slope == DOWN)? ((float)(tArr[2] - tArr[1]) / (tArr[2] - tArr[0])) :

((float)(tArr[1] - tArr[0]) / (tArr[2] - tArr[0]))) \* 100.0; //放大100 倍百分比值

tmp.time.duty = MemFtoI(fTmp);

fTmp = 100.0 - fTmp;

tmp.time.\_duty = MemFtoI(fTmp);

}

else

{

memset(&tmp.time, 0, sizeof(tmp.time));

i = pParam->trigger.source? pParam->ch2.vertical.probe :

pParam->ch1.vertical.probe;

tmp.volt.arg /= len;

fTmp = (tmp.volt.arg - HALF\_VOLT\_VOLUE) \* \_fCoef / 1000 \* i;

tmp.volt.arg = MemFtoI(fTmp); //浮点数保存到整形变量中

fTmp = (tmp.volt.max - HALF\_VOLT\_VOLUE) \* \_fCoef / 1000 \* i;

volue\_max = fTmp ;

volue\_temp = fTmp ;

volue\_max = volue\_max \*10 ; // 先扩大10倍

volue\_max = 0.5182152897\*volue\_max + 1.922777223 ;

tmp.volt.max = MemFtoI(volue\_max); //最大值

fTmp = (tmp.volt.min - HALF\_VOLT\_VOLUE) \* \_fCoef / 1000 \* i;

volue\_min = fTmp;

volue\_min = volue\_min /10;

volue\_min = 53.00667634\*volue\_min + 2.0265965436;

tmp.volt.min = MemFtoI(volue\_min); //最小值

fTmp = MemItoF(MemFtoI(volue\_temp) ) - fTmp;

volue\_vpp = 5.246782818\*fTmp-0.0756364427;

volue\_vpp = volue\_vpp ; // 用统计学软件辅助校准

tmp.volt.vpp = MemFtoI(volue\_vpp); //峰峰值

}

funEnd:

if (pParam->trigger.source)

{

pParam->ch2.measureVal = tmp;

memset(&pParam->ch1.measureVal, 0, sizeof(MeasureStruct));

}

else

{

pParam->ch1.measureVal = tmp;

memset(&pParam->ch2.measureVal, 0, sizeof(MeasureStruct));

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static void AppOSCGetDisHOffset(ParamStruct \*pParam, u32\*arr, u16 len)

0 Description :整体数组通过时间参数的偏移量进行偏移

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCGetDisHOffset(ParamStruct \*pParam, s16 \*arr, u16 len)

{

int pos = pParam->hor.pos;

if (MemItoF(pParam->hor.scale) > 2.001e-2) //50ms 及以上用循环模式不进行寻找起始点

return;

if (abs(pos) >= (len /6))

return;

if (pos == 0)

return;

if (pos > 0)

{

memmove(arr + pos, arr, sizeof(\*arr) \* (len - pos));

0/ for (i = 0; i < pos; i++)

// arr[i] = 0x7FFF; //无效值

}

else

{

memmove(arr, arr + abs(pos), sizeof(\*arr) \* (len - abs(pos)));

0/ for (i = (len - abs(pos)); i < len; i++)

// arr[i] = 0x7FFF; //无效值

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static void AppOSCGetDisVOffset(ParamStruct pParam, s16 \*arr,u16 len)

0 Description :显示的数据数组根据幅值参数的偏移量，进行垂直偏移。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCGetDisVOffset(int pos, s16 \*arr, u16 len)

{

while (len--)

{

\*arr++ += pos;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static void AppOSCGetDisStartPos(ParamStruct \*pParam, s16\*smp, int \*start)

\* Description : 根据触发线寻找数组中的数据的起始点

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void AppOSCGetDisStartPos(ParamStruct \*pParam, s16 \*smp, int \*start)

{

int i = 0;

int slope = 0;

int level = 0;

if (MemItoF(pParam->hor.scale) > 2.001e-2) //50ms 及以上用循环模式不进行寻找起始点

{

\*start = 0;

return;

}

if (pParam->trigger.source && !pParam->ch2.enable) //如果用触发源的通道不被显示， 不进行查找起始点

{

\*start = (\_cSampleCmpStart >>

1); return;

}

else if (!pParam->trigger.source && !pParam->ch1.enable)

{

\*start = (\_cSampleCmpStart >>

1); return;

}

//计算出触发线对应的AD 值

level= AppOSCGetLevel(pParam);

slope= pParam->trigger.slope;

for (i = \_cSampleCmpStart; i < \_cSampleCmpEnd; i++)

{

if (AppOSCCheckTrigger(&smp[i - 1], level) == slope)

break;

}

\*start = i - (\_cSampleCmpStart >> 1); //起点在MAX\_VALUE / 2 + i 的位置

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

0 Function Name :void AppOSCSerive(void)

0 Description :数据采集服务程序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void AppOSCSerive(void)

{

OS\_ERR err;

s16 \*pSmpCh1;

s16 \*pSmpCh2;

s16 \*pDis;

vs32 scaleLast = 0;

OSMemCreate( (OS\_MEM \*)&pMem, // 创建一块内存

(CPU\_CHAR \*)"pMem",

(void \*)\_aMemSize,

(OS\_MEM\_QTY )GUI\_COUNTOF(\_aMemSize),

(OS\_MEM\_SIZE )sizeof(\_aMemSize[0]),

(OS\_ERR \*)&err);

pSmpCh1 = OSMemGet( (OS\_MEM \*)&pMem, // 获取一个内存块样用来保存采样的全部数据

(OS\_ERR \*)&err);

pSmpCh2 = OSMemGet( (OS\_MEM \*)&pMem, //获取一个内存块样用来保存采样的全部数据

(OS\_ERR \*)&err);

while(1)

{

if (scaleLast != \_apParam->hor.scale) //时间轴被改动

{

scaleLast = \_apParam->hor.scale; //时间轴被修改后，根据时间片进行采样初始化

AppOSCSampleInit(MemItoF(scaleLast),(int\*)&\_vSampleCount,(void \*)&\_pFunCal); // 等待初始化稳定

OSTimeDly( (OS\_TICK )5,

(OS\_OPT )OS\_OPT\_TIME\_DLY,

(OS\_ERR \*)&err);

}

if (\_pFunCal != NULL) // 获取有效数据

{

LockSharedVariable();

\_aParamLast = \*\_apParam;

UnlockSharedVariable();

if (!(\_aParamLast.stat )) //非"停止状态 "下， 提取采样数据

(\*\_pFunCal)((u16\*)pSmpCh1,(u16\*)pSmpCh2,BSPAdcGetBuff(),\_vSampleCount);

pDis = OSMemGet( (OS\_MEM \*)&pMem, // 获取一个内存块用来保存显示的部分数据

(OS\_ERR \*)&err);

if (err == OS\_ERR\_NONE) //获取内存块成功。在显示部分没有处理完成时，获取内存会失败

{

int start;

s16 \*pTmp;

pTmp = \_aParamLast.trigger.source? pSmpCh2 : pSmpCh1; // 根据波形，计算相关参数 Vmax,Vmin,Freq,Duty 等等

AppOSCGetDisInfo(&\_aParamLast, pTmp,\_vSampleCount);

LockSharedVariable();

\_apParam->ch1.measureVal =\_aParamLast.ch1.measureVal;

\_apParam->ch2.measureVal= \_aParamLast.ch2.measureVal;

UnlockSharedVariable(); // 根据触发线，计算起始点

AppOSCGetDisStartPos(&\_aParamLast, pTmp, &start);

AppOSCGetDisHOffset(&\_aParamLast, pSmpCh1,\_vSampleCount);

AppOSCGetDisHOffset(&\_aParamLast, pSmpCh2,\_vSampleCount);

pTmp = pDis + MAX\_VALUE; //后半部分首地址， 存放通道 2 数据

//数据转换为显示的坐标数据

AppOSCSmpTurnToDisY(MemItoF(\_aParamLast.ch1.vertical.scale),pDis,pSmpCh1+start,MAX\_VALUE);

AppOSCSmpTurnToDisY(MemItoF(\_aParamLast.ch2.vertical.scale),pTmp,pSmpCh2+start,MAX\_VALUE);

AppOSCGetDisVOffset(\_aParamLast.ch1.vertical.pos, pDis,MAX\_VALUE);

AppOSCGetDisVOffset(\_aParamLast.ch2.vertical.pos, pTmp,MAX\_VALUE);

OSTaskQPost( (OS\_TCB \*)&dispTCB, // 发送数据到显视屏控制任务

(void \*)pDis,

(OS\_MSG\_SIZE )(sizeof(u16) \* MAX\_VALUE \* 2),

(OS\_OPT )OS\_OPT\_POST\_FIFO,

(OS\_ERR \*)&err);

}

BSPAdcResetOpen(); // 启动下一次采样

}

OSTimeDly( (OS\_TICK (OS\_OPT (OS\_ERR)250, )OS\_OPT\_TIME\_DLY, \*)&err);

}

}

/\*信号发生器主程序 \*/

/\*

1 信号发生器参数被修改时， 重新计算1周期100个点的AD值

2 通过计算出的AD值，实时刷新DA端口

\*/

#include <string.h>

#include <math.h>

#include "AppGenerator.h"

#include "os.h"

#include "GUIApp.h"

static const u32 \_cBuffLen = 100;

static const float \_cPI = 3.14159265358979323846;

static const float \_cfScal = \_cPI \* 2 / \_cBuffLen;

static float \_Period = 0.001;

static SGParamStruct SGParamLast = SGParamDefault();

static SGParamStruct \*SGParamCur = &OSCParam.sg;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static void \_SinWave(u16 \*buf, u16 len, u16 max, u16 min)

0 Description : 产生正弦波

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void \_SinWave(u16 \*buf, u16 len, u16 max, u16 min)

{

u16 rang = (max - min) >> 1;

u16 level = (max + min) >> 1;

u16 i = 0;

for (i = 0; i < len; i++)

{

\*buf++ = (u16)(sin(i \* \_cfScal) \* rang + level);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static void \_SquareWave(u16 \*buf, u16 len, u16 max, u16 min,u16 duty)

0 Description : 产生方波

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void \_SquareWave(u16 \*buf, u16 len, u16 max, u16 min, u16 duty)

{

u16 split = 0;

u16 i = 0;

if (duty == 0)

{

for (i = 0; i < len; i++)

\*buf++ = min;

return;

}

if (duty == 100)

{

for (i = 0; i < len; i++)

\*buf++ = max;

return;

}

split = len \* duty / 100;

for (i = 0; i < split; i++)

{

\*buf++ = max;

}

for (i = split; i < len; i++)

{

\*buf++ = min;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :static \_RampWave(u16 \*buf, u16 len, u16 max, u16 min)

0 Description : 产生锯齿波

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static void \_RampWave(u16 \*buf, u16 len, u16 max, u16 min)

{

float oceff;

int i;

if (len == 0)

return;

oceff = (max - min) / len;

for (i = len; i > 0; i--)

{

\*buf++ = (u16)(i \* oceff) + min;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0 Function Name :void AppGeneratorSerive(void)

0 Description : 信号发生器主程序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void AppGeneratorSerive(void)

{

OS\_ERR err;

u16 \*buf = NULL;

BSP\_DacOpen();

BSP\_DacTimUpdata(\_Period);

buf = BSP\_DacGetBuff100Bit();

\_SquareWave(buf, \_cBuffLen, 0, 0, 0);

while (1)

{

OSTimeDly( (OS\_TICK )20,

(OS\_OPT )OS\_OPT\_TIME\_DLY,

(OS\_ERR \*)&err);

if (memcmp(SGParamCur, &SGParamLast, sizeof(SGParamLast)) != 0)

{

LockSharedVariable();

SGParamLast = \*SGParamCur;

UnlockSharedVariable();

BSP\_DacTimUpdata(SGParamLast.period);

buf = BSP\_DacGetBuff100Bit();

switch (SGParamLast.wave)

{

case sine :

\_SinWave(buf,\_cBuffLen,SGParamLast.max,SGParamLast.min);

break;

case square:

\_SquareWave(buf,\_cBuffLen,SGParamLast.max,SGParamLast.min,SGParamLast.duty);

break;

case ramp:

\_RampWave(buf,\_cBuffLen,SGParamLast.max,SGParamLast.min);

break;

default:

\_SquareWave(buf, \_cBuffLen, 0, 0,0);

break;

}

}

}

}