

Cortex-M4原理与实践实验报告

题目： TFT LCD 触摸控制实验

**作者姓名 林宇航**

**指导教师**  **郭方洪**

**专业班级 自动化1901**

**学 院 信息工程学院**

**提交日期** 2021年12月23日

**一、实验目的**

1. 了解四线电阻触摸屏的工作原理。

2. 学习触点坐标计算方法。

3. 了解触屏控制芯片 TSC2046 的控制方式。

**二、实验设备**

1.计算机一台，操作系统为Windos11，装有CCSv11.0软件。

2.EK-TM4C1294XL实验开发板一块。

3.USB连接线一条。

**三、实验原理和流程**

**1. 四线电阻触摸屏的工作原理**

四线电阻屏的结构如下图 1 所示，在玻璃或丙烯酸基板上覆盖有两层透平，均匀导 电的 ITO 层，分别做为 X 电极和 Y 电极，它们之间由均匀排列的透明格点分开绝缘。其中 下层的 ITO 与玻璃基板附着，上层的 ITO 附着在 PET 薄膜上。X 电极和 Y 电极的正负端由 “导电条”（图中黑色条形部分）分别从两端引出，且 X 电极和 Y 电极导电条的位置相互垂 直。引出端 X-，X+，Y-，Y+共四条线。当有物体接触触摸屏并施以一定的压力时，上层的 ITO 导电层发生形变与下层 ITO 发生接触，该结构可以等效为相应的电路，如下图 2 所 示：

图示, 工程绘图

描述已自动生成

图 1 四线电阻屏结构

图示, 工程绘图

描述已自动生成

图 2 触摸的等效电路

**2. 触点坐标计算方法**

计算 Y 坐标，在 Y+电极施加驱动电压 Vdrive，Y-电极接地，X+做为引出端测量得到接 触点的电压，由于 ITO 层均匀导电，触点电压与 Vdrive 电压之比等于触点 Y 坐标与屏高度之比。

计算 X 坐标，在 X+电极施加驱动电压 Vdrive，X-电极接地，Y+做为引出端测量得到接 触点的电压，由于 ITO 层均匀导电，触点电压与 Vdrive 电压之比等于触点 X 坐标与屏宽度之比。

测得的电压通常由 ADC 转化为数字信号，再进行简单处理就可以做为坐标判断触点的 实际位置。上面的计算有一个缺陷，就是没有考虑电极抽头引线和驱动电极的电路的寄生电 阻，这部分电阻并不包含在 ITO 电阻之内，而且受环境温度影响阻值波动，很可能影响计算的正确性。

**3. TSC2046 控制芯片简介**

TSC2046 为 Ti 的一款低电压 I/O触摸屏控制器，是 ADS7846 触屏控制器的下一代产品，支持4线SPI。支持IO口操作电压 1.5~5.25V。与ADS7846 芯片 100%兼容引脚。拥有能用于辅助输入 2.5V 参考电压，电源监控以及温度测量模式。当不需要内部参考电压时，也能让参考电压掉电。TSC2046 小于 0.75mW 的电源消耗（2.7V 工作电压，参考电压掉电 的情况下），高达 125kHz 的采样速率，是一款用于电池供电系统的电阻屏，例如 PDA 等设备的理想选择。

TSC204支持单端工作模式与差分工作模式，精度支持 8位与12 位，本实验配置TSC2046工作于 12 位差分工作模式。内部控制寄存器以及各位作用分别如下表 1，表 2 所示：

表 1 TSC2046内部寄存器

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **BIT 7 (MSB)** | **BIT 6** | **BIT 5** | **BIT 4** | **BIT 3** | **BIT 2** | **BIT 1** | **BIT 0 (LSB)** |
| S | A2 | A1 | A0 | MOD E | SER/DF R | PD1 | PD0 |

|  |  |  |
| --- | --- | --- |
| **BIT** | **NAME** | **DESCRIPTION** |
| 7 | S | Start bit. Control byte starts with first high bit on DIN. A new control byte can start every 15th clock cycle in 12-bit conversion mode or every 11th clock cycle in 8-bit conversion mode. |
| 6-4 | A2-A0 | Channel Select bits. Along with the SER/DFR bit, these bits control the setting of the multiplexer input, touch driver switches, and reference inputs. |
| 3 | MODE | 12-Bit/8-Bit Conversion Select bit. This bit controls the number of bits for the next conversion: 12-bits (low) or 8-bits (high). |
| 2 | SER/D¯ ¯F¯¯R¯ | Single-Ended/Differential Reference Select bit. Along with bits A2-A0, this bit controls the setting of the multiplexer input, touch driver switches, and reference inputs. |
| 1-0 | PD1-PD0 | Power-Down Mode Select bits. |

表 2 TSC2046内部寄存器各位说明

PD1，PD0 省电模式选择如下表 3 所示：

表 3 TSC2046PD1，PD0 省电模式选择

|  |  |  |  |
| --- | --- | --- | --- |
| **PD1** | **PD0** | P¯¯¯E¯¯N¯¯I¯R¯¯¯Q¯ | **DESCRIPTION** |
| 0 | 0 | Enabled | Power-Down Between Conversions. When each conversion is finished, the converter enters a low-power mode. At the start of the next conversion, the device instantly powers up to full power. No need for additional delays to ensure full operation, and the very first conversion is valid. The Y− switch is on when in power-down. |
| 0 | 1 | Disabled | Reference is off and ADC is on. |
| 1 | 0 | Enabled | Reference is on and ADC is off. |
| 1 | 1 | Disabled | Device is always powered. Reference is on and ADC is on. |

对采样通道 A0~A2 的选择如下表 11-4 所示：

表 4 TSC2046 采样通道选择

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A 2** | **A 1** | **A 0** | **+R EF** | **−R EF** | **Y**  **−** | **X**  **+** | **Y**  **+** | **Y- POSITI ON** | **X- POSITI ON** | **Z1- POSITI ON** | **Z2- POSITI ON** | **DRIVE RS** |
| 0 | 0 | 1 | Y+ | Y− |  | +I N |  | Measure |  |  |  | Y+, Y− |
| 0 | 1 | 1 | Y+ | X− |  | +I N |  |  |  | Measure |  | Y+, X− |
| 1 | 0 | 0 | Y+ | X− | +I N |  |  |  |  |  | Measure | Y+, X− |
| 1 | 0 | 1 | X+ | X− |  |  | +I N |  | Measure |  |  | X+, X− |

通过上述介绍，了解控制寄存器的各位作用后，我们需要配置 TSC2046 工作于 12 位差分模式读取 x 轴测量值和 y 轴测量值，需要通过 SSI写入 0xD0，读取 x 方向转换数据，写入 0x90，读取 y 方向转换数据。对 TSC2046 的 SSI 操作时序特性和时序图分别如下表 5 和图 3 所示：

表 5 TSC2046的时序特性

表格

描述已自动生成

图示

描述已自动生成

图 3 TSC2046的时序图

**流程图：**



**四、实验代码、注释**

**实验代码：**

#include <stdbool.h>

#include <stdint.h>

#include <string.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_epi.h"

#include "inc/hw\_ints.h"

#include "driverlib/epi.h"

#include "driverlib/gpio.h"

#include "driverlib/sysctl.h"

#include "driverlib/rom.h"

#include "driverlib/rom\_map.h"

#include "driverlib/pin\_map.h"

#include "driverlib/systick.h"

#include "driverlib/interrupt.h"

#include "driverlib/ssi.h"

#include "driverlib/fpu.h"

#include "utils/uartstdio.h"

#include "TFTinit/TFT\_400x240\_OTM4001A\_16bit.h"

#include "EPIinit/EPIinit.h"

#include "driverlib/debug.h"

#include "grlib/grlib.h"

#include "driverlib/gpio.h"

#include "driverlib/adc.h"

#include "inc/hw\_gpio.h"

#include "inc/tm4c1294ncpdt.h"

#include "driverlib/uart.h"

#include "IQmath/IQmathLib.h"

#include "TOUCHinit/TOUCH\_TSC2046.h"

volatile uint32\_t ui32SysClock;

#ifndef M\_PI

#define M\_PI 3.14159265358979323846F

#endif

//系统时钟频率（Hz）

uint32\_t g\_ui32SysClock;

extern uint32\_t GetData[6];

uint32\_t TouchXData[6];

uint32\_t TouchYData[6];

uint32\_t TouchZData[6];

//用于跟踪时间的系统时钟计数器

static volatile unsigned long g\_ulTickCount;

#define SERIES\_LENGTH 240

//创建用于保存数据系列值的缓冲区。它必须足够大，以保持将在条形图上显示的系列中的数据点的最大数量

typedef struct \_Series

{

uint32\_t xAxis;

int32\_t data;

}tSeries;

static tSeries g\_cSeries[240];

//每秒SysTick滴答数

#define TICKS\_PER\_SECOND 40

#define FSECONDS\_PER\_TICK (1.0F / (float)TICKS\_PER\_SECOND)

//这是此SysTick中断的处理程序。它只是增加一个用于计时的计数器。

void

SysTickIntHandler(void)

{ //更新我们的计时计数器。

g\_ulTickCount++;

if(g\_ulTickCount>=320)

g\_ulTickCount = 0;

}

//配置UART及其管脚。这必须在UARTprintf()之前调用

void

ConfigureUART(void)

{//使能UART使用的GPIO外设。

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//使能UART0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

//为UART模式配置GPIO管脚

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//初始化控制台I/O的UART

UARTStdioConfig(0, 115200, g\_ui32SysClock);

}

void DrawTriWave(void)

{

uint16\_t ui32Loop = 0,ulItemCount=0,FinishCalculateFlag=0,ulLastTickCount=0;

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",CYAN,LIGHTBLUE);//显示GC211 TFTLCD test!

TFTLCD\_ShowString(60,40,"TriWave",GREEN,BLACK);//显示TriWave

TFTLCD\_DrawHorizontalLine(0,240,63,GREEN);//显示横边界线

TFTLCD\_DrawHorizontalLine(0,240,128,GREEN);

TFTLCD\_DrawHorizontalLine(0,240,193,GREEN);

TFTLCD\_DrawVerticalLine(64,193,0,GREEN);//显示竖边界线

TFTLCD\_DrawVerticalLine(64,193,239,GREEN);

TFTLCD\_ShowString(12,46,"1",WHITE,BLACK);

TFTLCD\_ShowString(2,195,"-1",WHITE,BLACK);//显示1和-1

while(1)

{ int count;

int Tlenth=80;

while(ulLastTickCount == g\_ulTickCount)

{}//等待下一个计时器

ulLastTickCount = g\_ulTickCount;

// 准备将新数据点添加到条形图。如果条形图中的项目数达到最大值，

//则数据点需要在缓冲器中“滑下”，从而可以在末尾添加新的数据

if(ulItemCount == SERIES\_LENGTH)

{

FinishCalculateFlag = 1;

}

// 否则，序列数据缓冲区不足，只需增加数据点的计数

else

{

g\_cSeries[ulItemCount].xAxis = ++ulItemCount;

}

count=g\_ulTickCount;

while(count>=Tlenth)

{

count-=Tlenth;

}

if(count<Tlenth/2)

g\_cSeries[ulItemCount - 1].data =count;

else if(count<Tlenth)

g\_cSeries[ulItemCount - 1].data =(-1)\*count+Tlenth;

if(!FinishCalculateFlag)

{//如果还没有计算三角波，请继续计算。

TFTLCD\_DrawPoint(g\_cSeries[ulItemCount - 1].xAxis,g\_cSeries[ulItemCount - 1].data+110,BLUE);

}

else{break;}//若画完，则退出

}

}

void TFTLCD\_DrawCircle(uint32\_t x,uint32\_t y,uint32\_t r,uint32\_t color)

{

int num=800;//画圆点数

int i;

float fr;

\_iq24 angle,xi,yi,sin\_angle,cos\_angle;

fr=\_IQ(r);

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

{

angle=\_IQ((float)M\_PI\*(float)i/((float)(num/2)));//计算角度

sin\_angle=\_IQ24sin(angle);//sin值

cos\_angle=\_IQ24cos(angle);//cos值

xi=\_IQ24mpy(cos\_angle,fr);//计算x值

yi=\_IQ24mpy(sin\_angle,fr);//计算y值

TFTLCD\_DrawPoint(x+\_IQint(xi),y+\_IQint(yi),color);//画圆

}

}

void DrawSinWave(void)

{

uint16\_t ulItemCount=0,FinishCalculateFlag=0,ulLastTickCount=0;

float fElapsedTime;

\_iq24 fRadians,fSine;

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",CYAN,LIGHTBLUE);//显示GC211 TFTLCD test!

TFTLCD\_ShowString(60,40,"y=sin(pi\*t/2)",GREEN,BLACK);//显示y=sin(pi\*t/2)

TFTLCD\_DrawHorizontalLine(0,240,63,GREEN);//显示横边界线

TFTLCD\_DrawHorizontalLine(0,240,128,GREEN);

TFTLCD\_DrawHorizontalLine(0,240,193,GREEN);

TFTLCD\_DrawVerticalLine(64,193,0,GREEN);//显示竖边界线

TFTLCD\_DrawVerticalLine(64,193,239,GREEN);

TFTLCD\_ShowString(12,46,"1",WHITE,BLACK);//显示1和-1

TFTLCD\_ShowString(2,195,"-1",WHITE,BLACK);

while(1)

{//等待下一个计时器

while(ulLastTickCount == g\_ulTickCount)

{}

ulLastTickCount = g\_ulTickCount;

// 准备将新数据点添加到条形图。如果条形图中的项目数达到最大值，

//则数据点需要在缓冲器中“滑下”，从而可以在末尾添加新的数据

if(ulItemCount == SERIES\_LENGTH)

{

FinishCalculateFlag = 1;

}

// 否则，序列数据缓冲区不足，只需增加数据点的计数。

else

{// 增加已添加到条形图系列数据缓冲区的项目数

g\_cSeries[ulItemCount].xAxis = ++ulItemCount;

}

// 以十进制秒为单位，以浮点格式计算经过的时间

fElapsedTime = (float)g\_ulTickCount \* FSECONDS\_PER\_TICK;

//将时间转换为弧度 //

fRadians = \_IQ24(fElapsedTime \* (float)M\_PI);//\_IQ24将浮点常量或变量转换为IQ数

// 调整波浪周期。这将给我们一个4秒或0.25赫兹的波周期。

//这个数字是任意选择，以提供一个好看的波形显示。

fRadians = \_IQdiv2(fRadians);//IQdiv2除以弧度2

// 计算正弦。乘以0.5以减小振幅

fSine = \_IQ24sin(fRadians);//计算输入值的正弦值

// 最后，将正弦值保存到序列数据点缓冲区的最后一个位置

g\_cSeries[ulItemCount - 1].data = \_IQ24mpyI32int(fSine,64); //\_IQ24mpyI32int multiplies an IQ number fSine by an integer 32, returning the integer portion of the result.

if(!FinishCalculateFlag)

{// 如果我们还没有计算正弦值，继续计算

TFTLCD\_DrawPoint(g\_cSeries[ulItemCount - 1].xAxis,g\_cSeries[ulItemCount - 1].data+128,BLUE);

}

else{break;}//若画完则退出

}

}

void main()

{

uint16\_t ui32Loop = 0;

FPUEnable();

FPULazyStackingEnable();

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_480), 120000000);

SysTickPeriodSet(g\_ui32SysClock / TICKS\_PER\_SECOND);

IntMasterEnable();//启用对处理器的中断

SysTickIntEnable();//启用SysTick中断

SysTickEnable();//使能SysTick

ConfigureUART();

EPIGPIOinit();

TOUCH\_TSC2046init(g\_ui32SysClock);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

GPIOIntEnable(GPIO\_PORTB\_BASE,GPIO\_INT\_PIN\_0);

GPIOIntTypeSet(GPIO\_PORTB\_BASE,GPIO\_PIN\_0,GPIO\_FALLING\_EDGE);

UARTprintf("TFTLCD touch screen test\n");

UARTprintf("EPI Type: host-bus 16-bit interface\n");

UARTprintf("SSI Type: SSI0 8bit\n");

TFT\_400x240\_OTM4001Ainit(g\_ui32SysClock);//打开背光

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",YELLOW,LIGHTBLUE);//显示文字

SSIDataPut(SSI0\_BASE,0xd0);

int i,j,t;

while(1)

{

if((TouchXData[5]<=240)&&(TouchYData[5]<=400))

{

TFTLCD\_DrawPoint(TouchXData[5],TouchYData[5]+50,BLUE);

}

for(ui32Loop=0;ui32Loop<=5;ui32Loop++)

{

SSIDataPut(SSI0\_BASE,0x90);

SysCtlDelay(3);

SSIDataGet(SSI0\_BASE,&TouchXData[ui32Loop]);

SysCtlDelay(3);

SSIDataPut(SSI0\_BASE,0xd0);

SysCtlDelay(3);

SSIDataGet(SSI0\_BASE,&TouchYData[ui32Loop]);

SysCtlDelay(3);

}

for(i=0;i<4;i++)

{//将取得的点进行冒泡排序，用于中值滤波法

for(j=0;j<4;j++)

{

if(TouchXData[j]<TouchXData[j+1])

{

t=TouchXData[j];

TouchXData[j]=TouchXData[j+1];

TouchXData[j+1]=t;

}

}

}

for(i=0;i<4;i++)

{//将取得的点进行冒泡排序，用于中值滤波法

for(j=0;j<4;j++)

{

if(TouchYData[j]<TouchYData[j+1])

{

t=TouchYData[j];

TouchYData[j]=TouchYData[j+1];

TouchYData[j+1]=t;

}

}

}

TouchXData[5]=TouchXData[2];//取数据中值

TouchYData[5]=TouchYData[2];//达到滤波效果

TFTLCD\_ShowData(100,300,TouchXData[5],BLACK,WHITE);

TFTLCD\_ShowData(100,320,TouchYData[5],BLACK,WHITE);

TOUCH\_PointAdjust(&TouchXData[5], &TouchYData[5]);

}

}

**实验现象：**

图片包含 游戏机, 电子, 电路

描述已自动生成

**五、思考题**

思考为什么触摸屏液晶上会产生噪点？结合TFT LCD 液晶显示实验中的函数，尝试编写一个触控按键函数，在液晶上绘制一个按键（大小，形状自定），当触摸到按键区域时，完成特定功能。

**答：**

**实验代码：**

#include <stdbool.h>

#include <stdint.h>

#include <string.h>

#include "inc/hw\_memmap.h"

#include "inc/hw\_types.h"

#include "inc/hw\_epi.h"

#include "inc/hw\_ints.h"

#include "driverlib/epi.h"

#include "driverlib/gpio.h"

#include "driverlib/sysctl.h"

#include "driverlib/rom.h"

#include "driverlib/rom\_map.h"

#include "driverlib/pin\_map.h"

#include "driverlib/systick.h"

#include "driverlib/interrupt.h"

#include "driverlib/ssi.h"

#include "driverlib/fpu.h"

#include "utils/uartstdio.h"

#include "TFTinit/TFT\_400x240\_OTM4001A\_16bit.h"

#include "EPIinit/EPIinit.h"

#include "driverlib/debug.h"

#include "grlib/grlib.h"

#include "driverlib/gpio.h"

#include "driverlib/adc.h"

#include "inc/hw\_gpio.h"

#include "inc/tm4c1294ncpdt.h"

#include "driverlib/uart.h"

#include "IQmath/IQmathLib.h"

#include "TOUCHinit/TOUCH\_TSC2046.h"

volatile uint32\_t ui32SysClock;

#ifndef M\_PI

#define M\_PI 3.14159265358979323846F

#endif

//系统时钟频率（Hz）

uint32\_t g\_ui32SysClock;

extern uint32\_t GetData[6];

uint32\_t TouchXData[6];

uint32\_t TouchYData[6];

uint32\_t TouchZData[6];

//用于跟踪时间的系统时钟计数器

static volatile unsigned long g\_ulTickCount;

#define SERIES\_LENGTH 240

//创建用于保存数据系列值的缓冲区。它必须足够大，以保持将在条形图上显示的系列中的数据点的最大数量

typedef struct \_Series

{

uint32\_t xAxis;

int32\_t data;

}tSeries;

static tSeries g\_cSeries[240];

//每秒SysTick滴答数

#define TICKS\_PER\_SECOND 40

#define FSECONDS\_PER\_TICK (1.0F / (float)TICKS\_PER\_SECOND)

//这是此SysTick中断的处理程序。它只是增加一个用于计时的计数器。

void

SysTickIntHandler(void)

{ //更新我们的计时计数器。

g\_ulTickCount++;

if(g\_ulTickCount>=320)

g\_ulTickCount = 0;

}

//配置UART及其管脚。这必须在UARTprintf()之前调用

void

ConfigureUART(void)

{//使能UART使用的GPIO外设。

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

//使能UART0

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

//为UART模式配置GPIO管脚

GPIOPinConfigure(GPIO\_PA0\_U0RX);

GPIOPinConfigure(GPIO\_PA1\_U0TX);

GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

//初始化控制台I/O的UART

UARTStdioConfig(0, 115200, g\_ui32SysClock);

}

void DrawTriWave(void)

{

uint16\_t ui32Loop = 0,ulItemCount=0,FinishCalculateFlag=0,ulLastTickCount=0;

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",CYAN,LIGHTBLUE);//显示GC211 TFTLCD test!

TFTLCD\_ShowString(60,40,"TriWave",GREEN,BLACK);//显示TriWave

TFTLCD\_DrawHorizontalLine(0,240,63,GREEN);//显示横边界线

TFTLCD\_DrawHorizontalLine(0,240,128,GREEN);

TFTLCD\_DrawHorizontalLine(0,240,193,GREEN);

TFTLCD\_DrawVerticalLine(64,193,0,GREEN);//显示竖边界线

TFTLCD\_DrawVerticalLine(64,193,239,GREEN);

TFTLCD\_ShowString(12,46,"1",WHITE,BLACK);

TFTLCD\_ShowString(2,195,"-1",WHITE,BLACK);//显示1和-1

while(1)

{ int count;

int Tlenth=80;

while(ulLastTickCount == g\_ulTickCount)

{}//等待下一个计时器

ulLastTickCount = g\_ulTickCount;

// 准备将新数据点添加到条形图。如果条形图中的项目数达到最大值，

//则数据点需要在缓冲器中“滑下”，从而可以在末尾添加新的数据

if(ulItemCount == SERIES\_LENGTH)

{

FinishCalculateFlag = 1;

}

// 否则，序列数据缓冲区不足，只需增加数据点的计数

else

{

g\_cSeries[ulItemCount].xAxis = ++ulItemCount;

}

count=g\_ulTickCount;

while(count>=Tlenth)

{

count-=Tlenth;

}

if(count<Tlenth/2)

g\_cSeries[ulItemCount - 1].data =count;

else if(count<Tlenth)

g\_cSeries[ulItemCount - 1].data =(-1)\*count+Tlenth;

if(!FinishCalculateFlag)

{//如果还没有计算三角波，请继续计算。

TFTLCD\_DrawPoint(g\_cSeries[ulItemCount - 1].xAxis,g\_cSeries[ulItemCount - 1].data+110,BLUE);

}

else{break;}//若画完，则退出

}

}

void TFTLCD\_DrawCircle(uint32\_t x,uint32\_t y,uint32\_t r,uint32\_t color)

{

int num=800;//画圆点数

int i;

float fr;

\_iq24 angle,xi,yi,sin\_angle,cos\_angle;

fr=\_IQ(r);

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

{

angle=\_IQ((float)M\_PI\*(float)i/((float)(num/2)));//计算角度

sin\_angle=\_IQ24sin(angle);//sin值

cos\_angle=\_IQ24cos(angle);//cos值

xi=\_IQ24mpy(cos\_angle,fr);//计算x值

yi=\_IQ24mpy(sin\_angle,fr);//计算y值

TFTLCD\_DrawPoint(x+\_IQint(xi),y+\_IQint(yi),color);//画圆

}

}

void DrawSinWave(void)

{

uint16\_t ui32Loop = 0,ulItemCount=0,FinishCalculateFlag=0,ulLastTickCount=0;

float fElapsedTime;

\_iq24 fRadians,fSine;

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",CYAN,LIGHTBLUE);//显示GC211 TFTLCD test!

TFTLCD\_ShowString(60,40,"y=sin(pi\*t/2)",GREEN,BLACK);//显示y=sin(pi\*t/2)

TFTLCD\_DrawHorizontalLine(0,240,63,GREEN);//显示横边界线

TFTLCD\_DrawHorizontalLine(0,240,128,GREEN);

TFTLCD\_DrawHorizontalLine(0,240,193,GREEN);

TFTLCD\_DrawVerticalLine(64,193,0,GREEN);//显示竖边界线

TFTLCD\_DrawVerticalLine(64,193,239,GREEN);

TFTLCD\_ShowString(12,46,"1",WHITE,BLACK);//显示1和-1

TFTLCD\_ShowString(2,195,"-1",WHITE,BLACK);

while(1)

{//等待下一个计时器

while(ulLastTickCount == g\_ulTickCount)

{}

ulLastTickCount = g\_ulTickCount;

// 准备将新数据点添加到条形图。如果条形图中的项目数达到最大值，

//则数据点需要在缓冲器中“滑下”，从而可以在末尾添加新的数据

if(ulItemCount == SERIES\_LENGTH)

{

FinishCalculateFlag = 1;

}

// 否则，序列数据缓冲区不足，只需增加数据点的计数。

else

{// 增加已添加到条形图系列数据缓冲区的项目数

g\_cSeries[ulItemCount].xAxis = ++ulItemCount;

}

// 以十进制秒为单位，以浮点格式计算经过的时间

fElapsedTime = (float)g\_ulTickCount \* FSECONDS\_PER\_TICK;

//将时间转换为弧度 //

fRadians = \_IQ24(fElapsedTime \* (float)M\_PI);//\_IQ24将浮点常量或变量转换为IQ数

// 调整波浪周期。这将给我们一个4秒或0.25赫兹的波周期。

//这个数字是任意选择，以提供一个好看的波形显示。

fRadians = \_IQdiv2(fRadians);//IQdiv2除以弧度2

// 计算正弦。乘以0.5以减小振幅

fSine = \_IQ24sin(fRadians);//计算输入值的正弦值

// 最后，将正弦值保存到序列数据点缓冲区的最后一个位置

g\_cSeries[ulItemCount - 1].data = \_IQ24mpyI32int(fSine,64); //\_IQ24mpyI32int multiplies an IQ number fSine by an integer 32, returning the integer portion of the result.

if(!FinishCalculateFlag)

{// 如果我们还没有计算正弦值，继续计算

TFTLCD\_DrawPoint(g\_cSeries[ulItemCount - 1].xAxis,g\_cSeries[ulItemCount - 1].data+128,BLUE);

}

else{break;}//若画完则退出

}

}

void main()

{

uint16\_t ui32Loop = 0,temp=0;

FPUEnable();

FPULazyStackingEnable();

g\_ui32SysClock = SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |

SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL |

SYSCTL\_CFG\_VCO\_480), 120000000);

SysTickPeriodSet(g\_ui32SysClock / TICKS\_PER\_SECOND);

IntMasterEnable();//启用对处理器的中断

SysTickIntEnable();//启用SysTick中断

SysTickEnable();//使能SysTick

ConfigureUART();

EPIGPIOinit();

TOUCH\_TSC2046init(g\_ui32SysClock);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

SysCtlGPIOAHBEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0);

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

GPIOIntEnable(GPIO\_PORTB\_BASE,GPIO\_INT\_PIN\_0);

GPIOIntTypeSet(GPIO\_PORTB\_BASE,GPIO\_PIN\_0,GPIO\_FALLING\_EDGE);

UARTprintf("TFTLCD touch screen test\n");

UARTprintf("EPI Type: host-bus 16-bit interface\n");

UARTprintf("SSI Type: SSI0 8bit\n");

TFT\_400x240\_OTM4001Ainit(g\_ui32SysClock);//打开背光

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",YELLOW,LIGHTBLUE);//显示文字

TFTLCD\_DrawCircle(25,300,25,BLUE);//画正弦按钮圆

TFTLCD\_DrawCircle(210,300,25,BLUE);//画三角波按钮圆

SSIDataPut(SSI0\_BASE,0xd0);

int xmid,ymid,i,j,t;

while(1)

{

if((TouchXData[5]<=240)&&(TouchYData[5]<=400))

{

TFTLCD\_DrawPoint(TouchXData[5],TouchYData[5]+50,BLUE);

}

for(ui32Loop=0;ui32Loop<=5;ui32Loop++)

{

SSIDataPut(SSI0\_BASE,0x90);

SysCtlDelay(3);

SSIDataGet(SSI0\_BASE,&TouchXData[ui32Loop]);

SysCtlDelay(3);

SSIDataPut(SSI0\_BASE,0xd0);

SysCtlDelay(3);

SSIDataGet(SSI0\_BASE,&TouchYData[ui32Loop]);

SysCtlDelay(3);

}

for(i=0;i<4;i++)

{//将取得的点进行冒泡排序，用于中值滤波法

for(j=0;j<4;j++)

{

if(TouchXData[j]<TouchXData[j+1])

{

t=TouchXData[j];

TouchXData[j]=TouchXData[j+1];

TouchXData[j+1]=t;

}

}

}

for(i=0;i<4;i++)

{//将取得的点进行冒泡排序，用于中值滤波法

for(j=0;j<4;j++)

{

if(TouchYData[j]<TouchYData[j+1])

{

t=TouchYData[j];

TouchYData[j]=TouchYData[j+1];

TouchYData[j+1]=t;

}

}

}

TouchXData[5]=TouchXData[2];//取数据中值

TouchYData[5]=TouchYData[2];//达到滤波效果

TFTLCD\_ShowData(100,300,TouchXData[5],BLACK,WHITE);

TFTLCD\_ShowData(100,320,TouchYData[5],BLACK,WHITE);

TOUCH\_PointAdjust(&TouchXData[5], &TouchYData[5]);

if((TouchXData[5]<=50)&&(TouchXData[5]>=1)&&(TouchYData[5]<=300)&&(TouchYData[5]>=250))

{ //设计显示正弦函数按钮

TFT\_400x240\_OTM4001Ainit(g\_ui32SysClock);

DrawSinWave();

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",YELLOW,LIGHTBLUE);

TFTLCD\_DrawCircle(25,300,25,BLUE);

TFTLCD\_DrawCircle(210,300,25,BLUE);

}

if((TouchXData[5]<=235)&&(TouchXData[5]>=185)&&(TouchYData[5]<=300)&&(TouchYData[5]>=250))

{ //设计显示三角波函数按钮

TFT\_400x240\_OTM4001Ainit(g\_ui32SysClock);

DrawTriWave();

TFTLCD\_ShowString(50,10,"GC211 TFTLCD test!",YELLOW,LIGHTBLUE);

TFTLCD\_DrawCircle(25,300,25,BLUE);

TFTLCD\_DrawCircle(210,300,25,BLUE);

}

}

}

**实验现象：**

按下左侧按钮开始画正弦波，按下右侧按钮开始画三角波，其他地方正常触屏显示。

**六、实验体会与心得**

本次实验又让我学会了一个在日常生活中很实用的功能-------TFT触控，市面上大多有两种触控检测方法，一是根据电阻，二是根据电容。实验室提供的开发板上是利用电阻的变换来检测（具体原理见实验原理）所以只需要两个ADC引脚来检测分压即可判断触点的位置，当没有触摸时，ADC检测出电压为高电平，由于ADC为12位分辨率，所以显示值为4096，当按下时检测的x，y值0-4096，线性变换为TFT的长和宽即可得到对应的坐标。