电子设计2实验报告

- **实验题目**：低频信号的存储与回放
- **实验日期**：2023.9.19~2023.10.17
- **实验成员**：李佳承、江育圃、童艺辰

一、实验目的

1. 学习规范的C语言编程方法
2. 学会使用C8051F020单片机开发板
3. 弄清A/D、D/A的使用方法
4. 正确使用LCD显示模块

二、实验内容

幅度0～1V、频率小于1KHz的信号通过C8051F020的内置ADC转换成数字信号存储在单片机的内存中，存储深度为3K，并由C8051F020的DAC输出到示波器上显示，同时在LCD屏上显示稳定的波形。

要求：

1. 单次存储、连续稳定显示
2. 实时存储、实时显示

三、实验思路

- 本次实验需要使用的资源：IO口、ADC、DAC、Timer3。
- “单次存储，连续显示”实现：使用AD连续采样 (每次Timer3定时器溢出开启AD转换)，采样数据放入内存，共计存储3K $Byte$ (ADC为12位，每个采样值需要两个字节存储，所以实际存储深度为1500个采样值)，当数据存满3K之后，将数据送入DAC (即将1500个采样值一次送入DAC)，并将这1500个采样值循环显示。
- “实时存储、实时显示”实现：不再使用多余的缓存，将AD采样后的值直接送入DAC，以实现实时的效果。
- “LCD屏显示波形”：查阅资料得到LCD屏的横向有128格，纵向有64格，同样使用AD连续采样，将采样数据放入内存，存储深度为128个采样值，对于每个采样值根据转换公式

  $$转换代码 = Vin \times \frac{Gain}{VREF} \times 2^n$$ ：单端方式时 n = 12；差分方式时 n = 11。

  得到Vin公式即为LCD屏上每一个点的纵坐标值。

四、代码设计

1. **主函数体**

先进行设备的初始化，包括看门狗、内外部时钟、IO口、ADC模块、DAC模块、定时器3配置、LCD屏等，配置完成后进入应用。

```
void main ()
{
        Device_Init();//包括看门狗、时钟、IO口初始化
        DAC0_Init ();
        ADC0_Init ();
        Timer3_Init(10);
        LCD_Init();
        LCD_Clear();//LCD清屏
        Show();//主应用
}
```

2. **重要模块配置**

```
//DA配置
void DAC0_Init(void)
{
   DAC0CN = 0x80;//打开DAC0使能位，DAC0处于工作状态
}
```

```
//AD配置
void ADC0_Init(void)
{
    AMX0CF = 0; //配置AIN1单通道输入
    AMX0SL = 1;

    ADC0CF = ((SYSCLK/SAR0_CLK - 1) << 3);   //设置ADC转换时钟
    ADC0CF &= ~(BIT2 + BIT1 + BIT0);         //PGA增益为1

    ADC0CN &= ~(BIT6 + BIT5 + BIT3 + BIT1 + BIT0); //当ADC被使能时，除了转换期间之外一直处于跟踪方式
    ADC0CN |= BIT2;//定时器3溢出启动ADC0转换

    REF0CN &= ~BIT4; //参考电压取自VREF0
    REF0CN |= BIT1 + BIT0; //内部偏压发生器和内部电压基准缓冲器工作

    EIE2 |= BIT1; //允许ADC转换结束中断
    EA = 1;

    ADC0CN |= BIT7; //使能ADC0
}
```

定时器3定义了两个，一个用于实时转换，关闭了定时器3中断，另一个用于单次转换，打开了定时器3中断，这样对于不同的需要 (单次还是实时) 选择不同的定时器3配置，可以节省CPU资源，提高运行速度。

```
//实时转换定时器3配置，关闭了定时器3中断
void Timer3_Init(unsigned int counts)
{
    TMR3CN &= ~(BIT7 + BIT2 + BIT0);  //清零定时器3溢出标志 暂时关闭定时器3 定时器3时钟由T3M定义
    TMR3CN &= ~BIT1;//定时器3时钟为系统时钟/12，方便设置更多的采样频率
    TMR3RLL = ((65536 - counts) & 0xff);
    TMR3RLH = (((65536 - counts) >> 8) & 0xff); //计算定时器3重载寄存器
    TMR3L = TMR3RLL;
```

```
    TMR3H = TMR3RLH; //设定计数初值
    EIE2 &= ~BIT0; //close
    EA = 1;
    TMR3CN |= BIT2; //启动定时器3
}
//单次转换定时器3配置，打开了定时器3中断
void Timer3_Init2(unsigned int counts)
{
    TMR3CN &= ~(BIT7 + BIT2 + BIT0);  //清零定时器3溢出标志 暂时关闭定时器3 定时器3时钟由T3M定义
    TMR3CN |= BIT1;//定时器3时钟为系统时钟
    TMR3RLL = ((65536 - counts) & 0xff);
    TMR3RLH = (((65536 - counts) >> 8) & 0xff); //计算定时器3重载寄存器值
    TMR3L = TMR3RLL;
    TMR3H = TMR3RLH; //设定计数初值
    EIE2 |= BIT0; //open
    EA = 1;
    TMR3CN |= BIT2; //启动定时器3
}
```

3. **两个中断函数设置**

```
//AD转换中断
void ADC0_INT_ISR(void) interrupt 15
{
    if(AD0INT == 1)//判断中断标志
    {
        if( RT_flag == 1)//实时存储，实时输出

        {
            if(datanum <= 127&& Wait_flag == 0)
            {
                Voltage[datanum] = (ADC0H << 8) + ADC0L;
                datanum++;
            }
            else if (datanum >127)
            {
                Wait_flag = 1;//存储数据已满标志位
                datanum = 0;
            }
            DAC0L = ADC0L;//AD采样值直接送入DA
```

```
            DAC0H = ADC0H;
            }
        else//单次存储，连续稳定输出
        {
            if(datanum <= 1500)
            {
                Voltage[datanum] = (ADC0H << 8) + ADC0L;
                datanum++;
            }
        }
     }
}
```

```
//定时器3中断
void Timer3_INT_ISR(void) interrupt 14
{
    if(TMR3CN & BIT7)
    {
        TMR3CN &= ~BIT7;
        DAC0L = Voltage[datanum2] & 0xFF;//将内存采样值送入DA
        DAC0H = (Voltage[datanum2] >> 8) & 0xFF;
        datanum2++;
        if(datanum2 >= 1500) datanum2 = 0;//清零方便循环显示
    }
}
```

4. **应用函数设计**

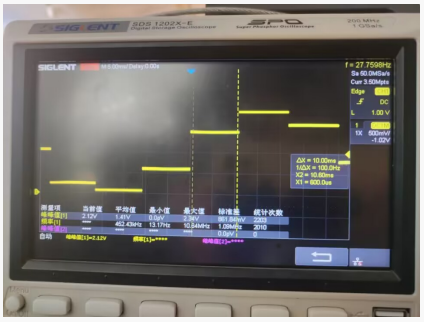
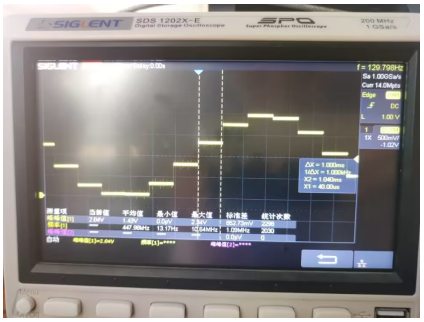
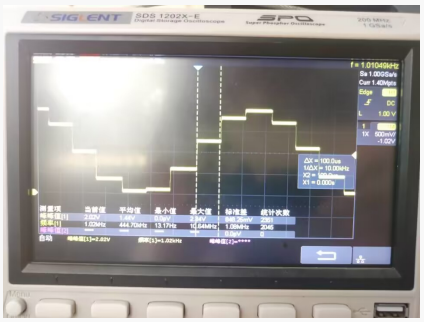
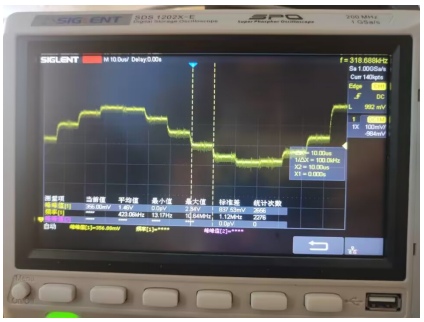
```
void Show(void)
{
    int i = 0;
    if( RT_flag == 1)//实时显示
    {
        while(1)
        {
            key = KeyScan();
            if(key != 0)
            {
                switch(key)
                {
                    case 1: Timer3_Init(10000);break;
                    case 2: Timer3_Init(1000);break;
```

```
                    case 3: Timer3_Init(100);break;
                    case 4: Timer3_Init(10);break;
                }
            }
            while(!Wait_flag);
            for(i=1;i<127;i++)
            {
                LCD_DrawPoint(i,Voltage[i]*0.01875);
            //LCD=Voltage*VREF(2.4)/Gain(1)/4096(单端)/2*64(0~2v 纵向
为64格)
            }
            LCD_Clear();
            Wait_flag = 0;
        }
    }
    else//单次显示
    {
        while(!Voltage[1500]);
        ADC0_Disable();//关闭AD，不再进行AD转换
        Timer3_Init2(15);
        while(1)
        {
            for(i=0;i<127;i++)
            {
                LCD_DrawPoint(i, Voltage[i]*0.01875);
            }
        }
    }
}
```

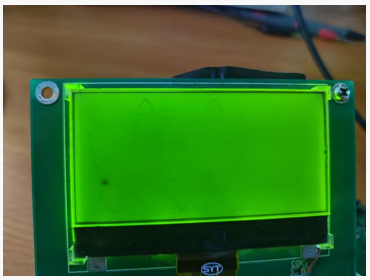
对于按键选择采样频率的实现方法，应用按键扫描函数，采样频率用定时器3控制，定时器3的时钟设为1MHz（`TMR3CN &= ~BIT1;//定时器3时钟为系统时钟/12`），要获得100Hz、1KHz、10KHz、100KHz的采样率只要将定时器3的counts分别设为10000、1000、100、10。

五、实验现象

测试功能	测试要求	测试结果

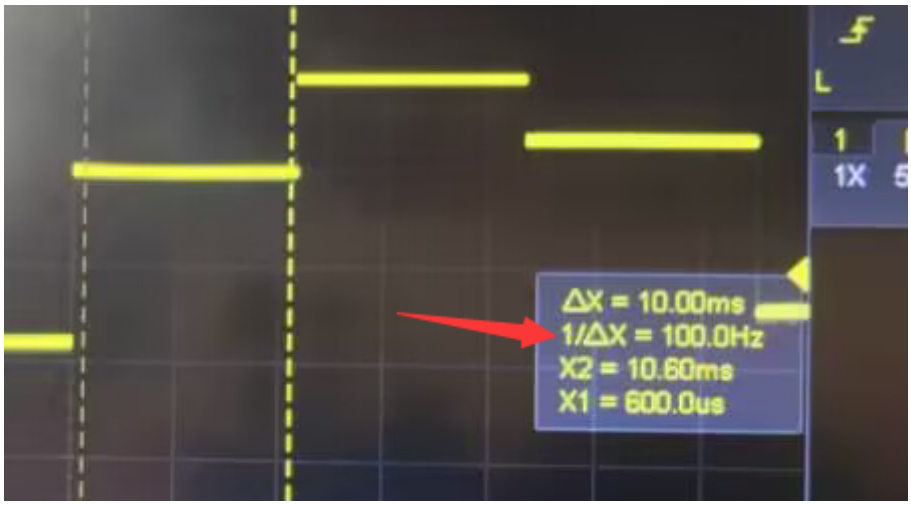
1、AD采样频率	100Hz	
	1KHz	
	10KHz	
	100KHz	
2、实时存储，实时显示	改变输入信号频率，示波器上输出波形频率是否跟着改变?	是
	示波器显示的频率是不是信号的频率?	是
	改变输入信号频率，LCD上输出波形频率是否跟着改变?	是
	是否能同时在LCD上显示完整波形?	是

3、单次存储，连续稳定显示	示波器显示的频率是不是信号的频率?	是
	改变输入信号频率，示波器上输出波形频率是否跟着改变?	否
	是否能同时在LCD上显示完整波形?	是

LCD实时显示		
		

六、实验中遇到的问题及解决办法

1. 如何用定时器调节采样率以及如何在示波器上反映采样频率?
    - 由于起初定时器3的时钟设置为12MHz，而如果要得到100Hz的采样率，定时器值就要设为120000，可这远远超过了最大值65535，于是将定时器时钟设为1MHz (系统时钟/12) 解决的这个问题。在此之前曾考虑提高ADC0的时钟频率(`#define SAR0_CLK 4000000`)，进而提高AD转换速率(`ADC0CF = ((SYSCLK/SAR0_CLK - 1) << 3);`)，ADC0CF寄存器的位7~位3为转换周期控制位，AD的转换周期影响AD转换速率和AD转换精度，但二者不是兼得的，转换周期越短，AD转换速率越快，但转换精度越低。当时选择修改AD转换周期是为了提高采样速率，现在看来是不必要的。
    - 在示波器上用两个光标表明两个采样点，要测量采样频率即是测量两个采样点之间的时间间隔，如图:



2. LCD屏画点的值如何计算?
    - 根据公式:

$$转换代码 = Vin \times \frac{Gain}{VRET} \times 2^n$$

    得到 $Vin = 转换代码 \times \frac{VREF}{Gain} \times \frac{1}{2^n}$ ，而由于LCD纵向只有64格，信号源输入为0~ 2V，即Vin为0~ 2V，为使画点能尽量铺平整个LCD屏，将Vin转化成0~64V，即 $LCD = \frac{Vin}{2} \times 64$ 得到画点的值，进入到函数 `LCD_DrawPoint` 运算，在LCD屏上得到较为优美的波形。

3. 实验中发现，在每次AD中断中直接通过函数 `LCD_DrawPoint(i,Voltage[i]*0.01875)` 输出在LCD不可行（理论上，这样实时性最好），因为LCD_DrawPoint函数执行速率过慢，远慢于定时器溢出和AD转换的速度，导致大量中断嵌套，输出波形异常。
    - 引入Wait_flag，通过大量单次显示来代替实时显示，因为AD转换速度很快，存满单次显示所需的数据耗时很短，大量的单次显示与实时显示效果相近。

七、实验需要改进的地方

1. 实验中单次与实时的转换是通过修改代码中的标志位 `RT_flag` 来实现的，不是很方便，可以改为按键转换会更为便捷和实用。
2. 实验中通过按键切换采样率时，由于按键本身加延时来消抖，而这段时间AD又在反复申请中断，使主函数执行得很慢，导致按键要长按才能生效。