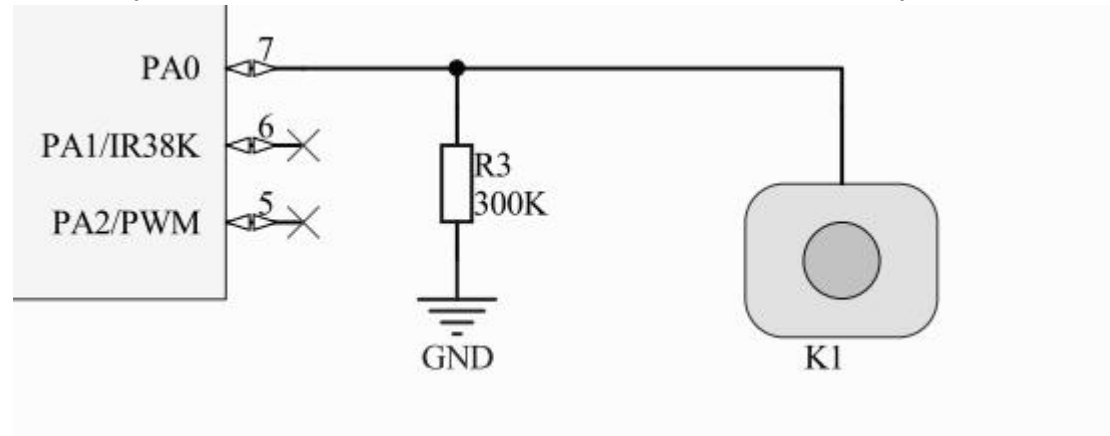


## NucleoF429 GPIO Touch Key

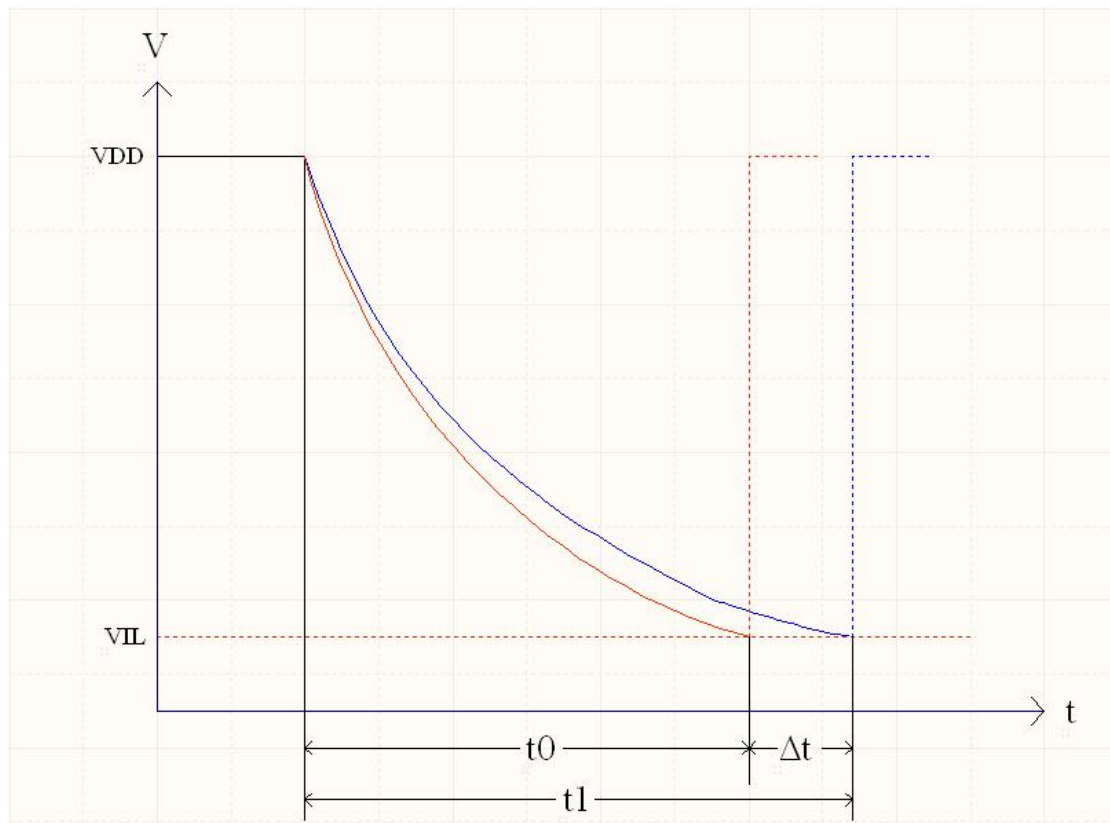
这里先不说触摸（Touch）按键的优缺点，只从 GPIO 实现 Touch Key 的方法上来讲下这个 Demo。Touch Key 的实现方式有多种，效果和稳定性也相差很大。有些是硬件上专门做了设计，且有不少是有专利的。记得 ST 有支持 Touch Key 的型号，也有对应的库，但没用过。进入正题-->

1、Touch Key 硬件原理：

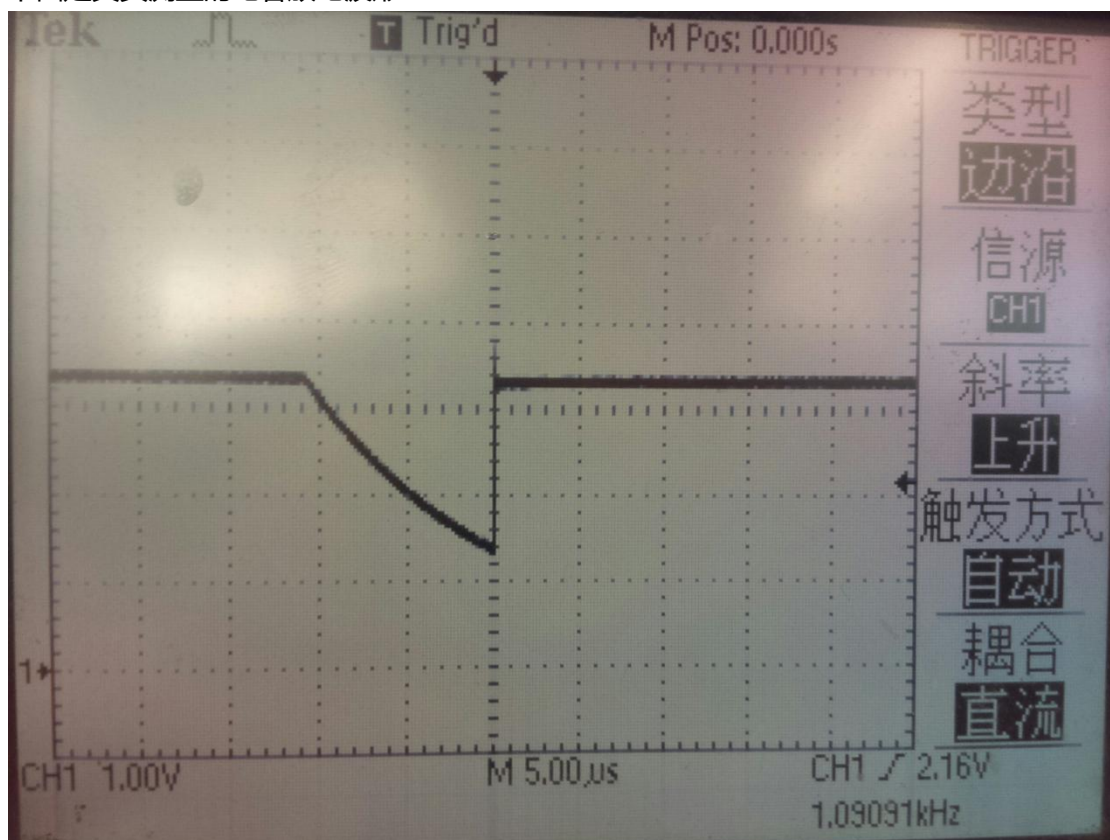
Touch Key 检测的原理网络上也有许多，方法各异，此文中的 Touch Key 如下图：



实现原理是将 Touch Pad 和 人体 等效为一个电容，Touch Pad 的电容固定，和一个电阻并联到 GND，IO 输出 High 后，再转为 Input，Touch Pad 上的电荷通过电阻放电，只要 Touch Pad 的等效电容和电阻阻值不变，放电时间就是固定的。当有触摸时，人体等效电容相当于并联到了 Touch Pad 上，总的等效电容增大，放电时间增加，通过判断这个增加的“电容”大小，就能判断是否有触摸。如下图：红色的曲线为无触摸时的放电曲线，放电时间为  $t_0$ ，蓝色曲线为有触摸时的放电曲线，放电时间为  $t_1$ ，两者的差值为  $\Delta t$ 。由  $\Delta t$  的大小可判断是否有触摸。VIL 是 IO 读到低电平的阈值

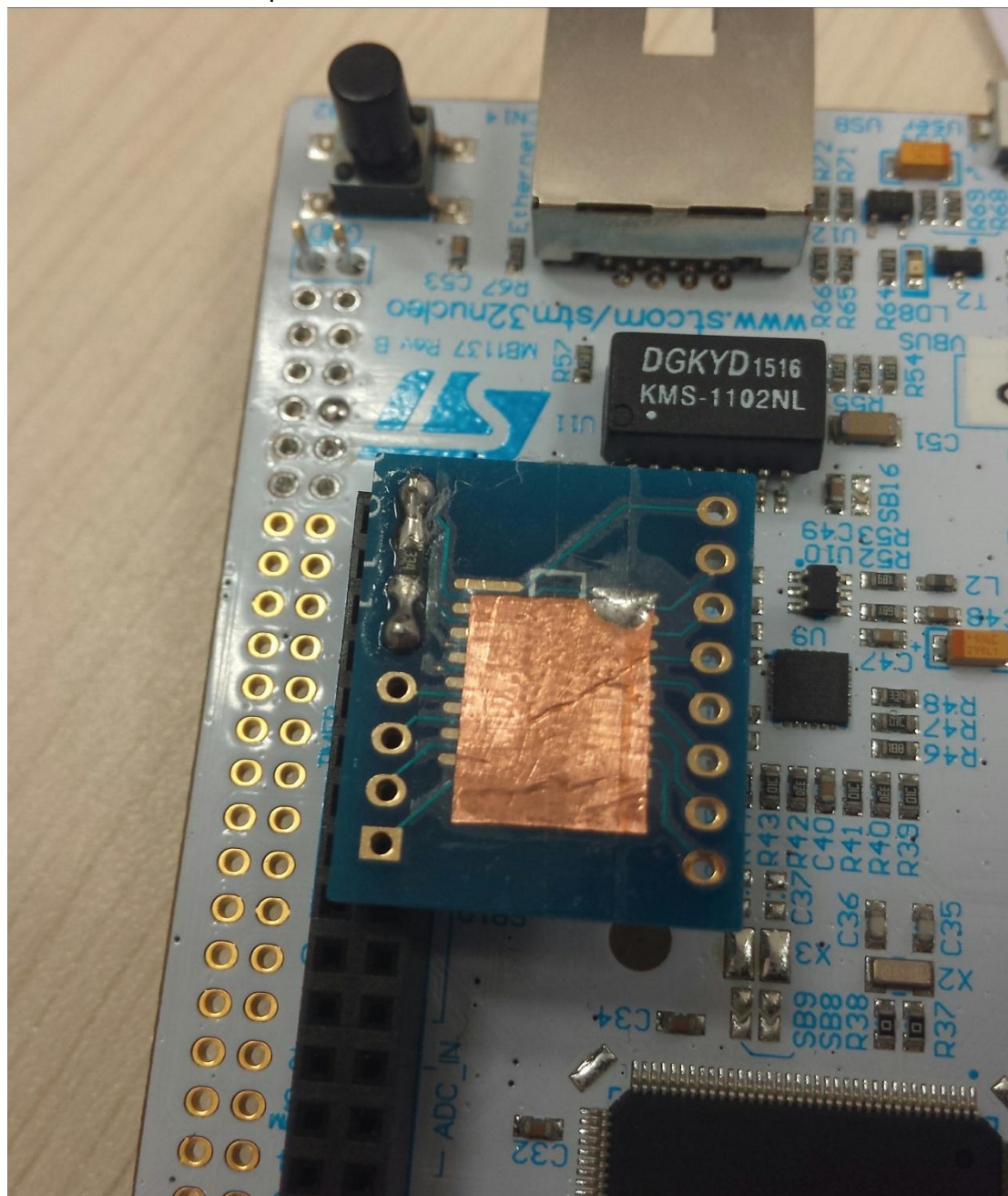


下面是真实测量的电容放电波形：



可以看到，放电时间在 10us 左右（330K 下拉电阻，有触摸时大概增加 5us 左右），这还是加了示波器探头的放电时间，如果不加探头，时间大概小一半，

下图是一个简易的 Touch Pad。铜箔大小越  $1\text{cm}^2$ ，覆盖了透明胶带。下拉 330K 电阻，铜箔等效电容估计个位数 pF。



以上是硬件方面和检测的原理。下面来看如何检测这个电容变化量？

## 2、Touch Key 软件原理：

2.1、电路原理知道了，电容放电的波形也看到了，如何检测变化量呢？用 ADC？有无触摸的电压变化量在 mV 级别，时间为 us 级别，用多大速度和分辨率的 ADC 暂且不管，主要是有很多 MCU 并不具备 ADC 模块，如果外接满足这个要求的 ADC 还不如用一颗触摸芯片来的简单。测量电压不行，那测量时间吧。上面有提到，有无触摸，放电波形会有 5us 的变化量。5us 对于几十 MHz 的 MCU 来说能做太多事了。我这里采用的方法并不是采用中断，因为 5us 的变化量测量得需要上 MHz 的中断且获得的数据变化也不明显，所以

我采用了 do while ( Loop ) 方式，IO 输出 High 充电完成转为 Input，通过电阻放电，直到 IO 读到低电平，在放电其间，放一个计数器，基本是以一条指令的时间来计数的，而且获得的数据变化量够大，便于后期计算，比如 10MHz 主频执行单周期指令计数，1us 计数 10 次，5us 计数 50 次，还可以连续做多次充放电，将计数累加，有无触摸就会有上百次的计数值变化，完全可以被检测出来。这是我的计数函数，在 IO 设置为 Input 后，查询 IO 是否为低，并且做计数，函数中连续放了 16 次累加，因为整个 while 循环中，执行 while 判断的时间是最长的，所以减少 while 判断可以计数更多的值，我的 TouchKey 是在 PE0 上，如果为 High，计数值递增为 1，如果换作其他 IO，在读到值之后做相应的位移，使每次递增为 1 即可，

```
/*-----*/
uint16_t Get_Touch_Value(void)
{
    //HAL_GPIO_WritePin(GPIOC,GPIO_PIN_10,GPIO_PIN_SET);
    Touch_Count = 0;
    GPIO_InitTypeDef GPIO_InitStructure;

    GPIO_InitStructure.Pin = GPIO_PIN_0;
    GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
    GPIO_InitStructure.Pull = GPIO_NOPULL;

    HAL_GPIO_Init(GPIOE, &GPIO_InitStructure);

    //while(HAL_GPIO_ReadPin(GPIOE,GPIO_PIN_0)) //2
    while(GPIOE->IDR & GPIO_PIN_0) //1
    {
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);

        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);

        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
        Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
    }
}
```

```

Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);

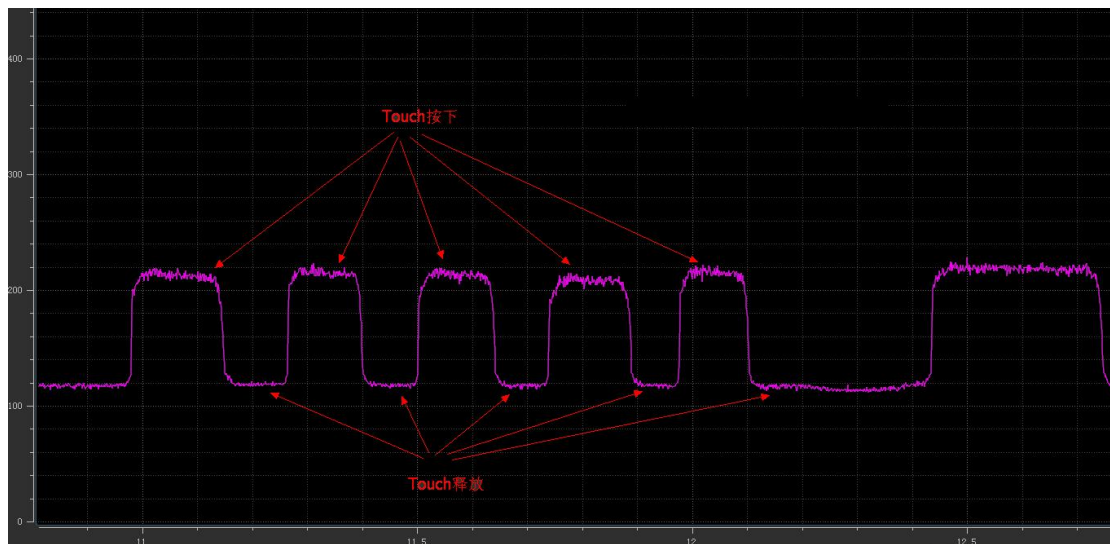
Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
Touch_Count +=(GPIOE->IDR & GPIO_PIN_0);
}
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

HAL_GPIO_WritePin(GPIOE,GPIO_PIN_0,GPIO_PIN_SET);

//HAL_GPIO_WritePin(GPIOC,GPIO_PIN_10,GPIO_PIN_RESET);
return Touch_Count;
}

```

2.2、因为手指本身有干扰，并且手指与 Touch Pad 的接触面积也不固定，导致增量时间一直在变化，必须要经过滤波处理，如下图中的 Touch 按下部分，波形有杂波，在 Touch 隔离物较厚时，干扰问题会更严重且数据变化量较小，不经过滤波处理，误触发的几率很大。



我做了两次滤波：

1、间隔 100us 进行一次充放电采集，连续采集 12 个点，将这 12 个点进行从大到小排序，之后去掉队首最大的 4 个值和队尾最小的 4 个值，取中间的 4 个值做平均，通过这样将一段时间内的数据累加平均为一个点，能有效去掉脉冲干扰。

```

/*-----*/
uint16_t Touch_Heap(void)
{
    uint8_t i,j;
    uint8_t DataTemp;
    uint32_t DataSum = 0;
    for(i=0; i<Heap_Num-1;i++)
    {
        for(j=0;j<Heap_Num-i;j++)
        {
            if(Data_Buffer[j]<Data_Buffer[j+1])
            {
                DataTemp = Data_Buffer[j];
                Data_Buffer[j] = Data_Buffer[j+1];
                Data_Buffer[j+1] = DataTemp;
            }
        }
    }
    for(i=Dis_Num;i<Heap_Num-Dis_Num;i++)
    {
        DataSum += Data_Buffer[i];
    }
    DataSum = DataSum/(Heap_Num-2*Dis_Num);

    return (uint16_t) DataSum;
}

```

2、间隔 10ms 进行一次连续的采集，采集后的点再进行平滑滤波，我这里采用了一个一阶低通滤波器，通过调整滤波系数能调节波形的延时和滤波效果。（ST 有滤波函数库，效率高效果好，有兴趣的同学可以深入研究一下）

```

/*-----*/
/*
    LPF:  $Y(n) = \alpha * X(n) + (1-\alpha) * Y(n-1)$ 
     $Y[n] = Y[n-1] + \alpha * (X[n] - Y[n-1])$ 
    LPF_Yn[0:1] = Yn-1:Yn
*/

uint16_t IIR_LowPass_Filter(uint16_t LPF_Data)

```

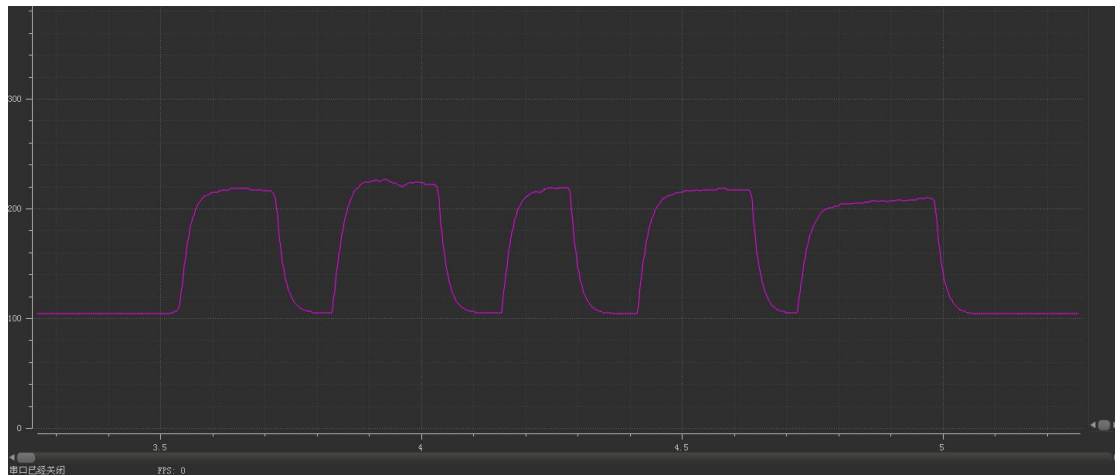


```

{
    float LPF_Temp;
    LPF_Temp = LPF_Pre_Data+ LPF_Alpha*(LPF_Data - LPF_Pre_Data);
    LPF_Pre_Data = LPF_Temp;
    return (uint16_t)LPF_Temp;
}

```

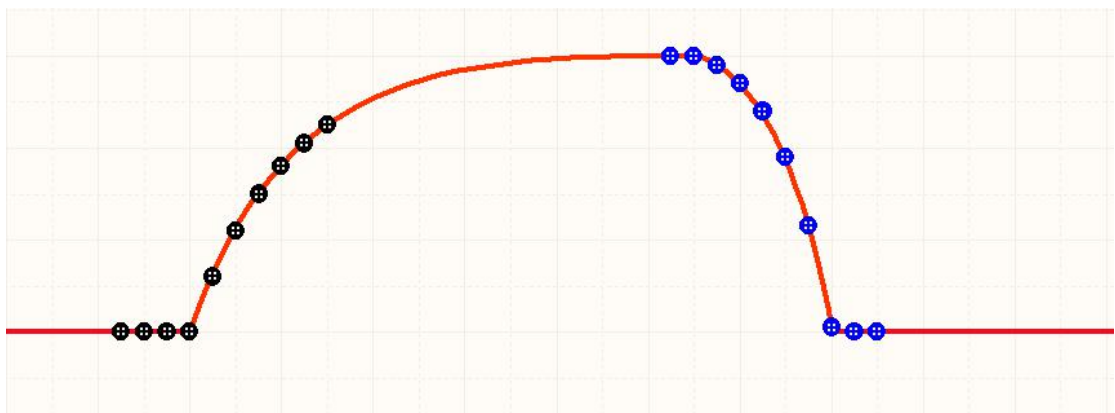
经过两次滤波，波形数据改善很多。



### 3、Touch Key 状态判断：

这里是指判断按下和释放，通常的做法是设置一个阈值，高于多少或者低于多少就认为是按下或释放，有些程序会做到动态追踪最低点等等，对于元件一致性好的产品，误差都在可控范围内，这样做通常没有太大问题。但对于一些成本敏感，元器件误差较大的产品，问题就会比较明显，因为每一个产品的元件都有误差，这些误差的范围有可能已超过了你的阈值范围，无论你阈值设置过大或过小，总有一部分产品会有问题，即使是同一个产品，不同的环境都会导致阈值超出范围，比如 Touch Pad 受到灰尘、磨损等改变了自身的等效电容，电阻阻值随温度进行变化等

我这里采用数据变化的“趋势”来判断 Touch 按下或释放，如图：



把数据点放入队列，依次计算前后两个点的差值，判断数值是增大还是减小，通过计数符合差值的点的次数，判断 Touch 按下还是释放，这也能有效排除干扰，最重要的是，这种方法受外界因素的影响比较小，元器件之间的差异被排除，因为这里只判断增量而非绝对值。

```

/*-----*/
// 按下时：符合差值的计数增减，释放时，符合差值的计数减小。
void Check_Touch_State(void)
{
    uint8_t i;
    uint8_t Touch_DebCnt = Data_Num; //数据点个数
    int8_t Touch_D_Value = 0;
    for(i=0;i<Data_Num;i++)
    {
        Touch_Data_Buffer[i] = Touch_Data_Buffer[i+1];    //队列左移
    }
    Touch_Data_Buffer[i] = IIR_LowPass_Filter(Touch_Heap()); //最新数据点补在队
列尾部
    if(Touch_POR_Deb > 200)                                // 上电先做
采集不判断按键状态，防止上电时数据为 0，被误判为按下。
    {
        for(i=0;i<Data_Num;i++)
        {
            Touch_D_Value = Touch_Data_Buffer[i] - Touch_Data_Buffer[i+1];
            if(Touch_D_Value > Release_Sen_Value)
            {
                Touch_DebCnt--;
            }
            else if(Touch_D_Value < Press_Sen_Value )
            {
                Touch_DebCnt++;
            }
        }

        if(Touch_DebCnt > (Data_Num+3))                    //判断递增
次数 ( Press )
        {
            Press_Flag = 0x01;
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_SET);
        }
        else if(Touch_DebCnt < (Data_Num-3))                //判断递减次数
( Release )

```



```
    {
        Press_Flag = 0xF1;
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,GPIO_PIN_RESET);
    }
}
else
{
    Touch_POR_Deb++;
}
}
```

以上是 Touch Key 的实现过程，作为一个原理性的 Demo，还有很多细节工作未作，如果采用这种方式到产品中，还需要仔细优化和调试，如防止放电超时，功耗等问题，以上 Demo 是用 NucleoF429 上完成的，原理较为简单，可以方便移植到其他 MCU 上完成，（事实上我首先是用 NucleoF031K6 完成之后再移植到 F429 上的）。

以上 Demo 没有写关于 CubeMX 的部分，因为只需要一个或者两个中断，并不局限于具体的 MCU。另外此文中的所有参数都是针对我的电路的，可以按照具体的电路和应用来调整参数。

--此文来自 [ARM 中文社区](#)