软硬件全开源，航芯方案分享 | 智能电动牙刷方案

当代口腔问题频发，让人们越来越重视口腔卫生。传统的刷牙方式，由于个人习惯和刷牙方式的不同，会不同程度地导致牙龈受损，牙菌斑去除不彻底等问题。而电动牙刷设备，基于其相对程序化的刷牙方式，可根据个人口腔特性支持自主选择，调节刷牙力度。而且在刷牙过程中，不需要过多的手部动作，仅需要调节刷牙的角度，更多的清洁工作交付由牙刷本身的特性来完成，方便人们的同时也更能有效的减少口腔问题。

电动牙刷类型

现在市面上电动牙刷品类繁多，从刷头的方式可将其分为两大类型：旋转式和振动式（也叫声波式）。参考 https://zhuanlan.zhihu.com/p/...

图 1. 电动牙刷工作方式对比图

旋转式电动牙刷是由电机带动刷头旋转，牙面清洁度高，但牙缝清洁能力薄弱且相较于振动式，更易损伤牙釉质。而振动式，由电机带动刷头进行上下的高频振动，高频摆动的刷头能

高效完成洗刷牙齿的动作，可以让牙膏与水的混合物产生大量微小的气泡，气泡爆裂时产生的压力可以更深入牙缝达到深度的清洁效果。

振动式的实现有两种方式，一种由偏心振动电机实现，多用于中低档的电动牙刷方案。该种方式的电动牙刷振动感强，振动无序。另一种则是采用线性电机，业内也称之为磁悬浮电机。

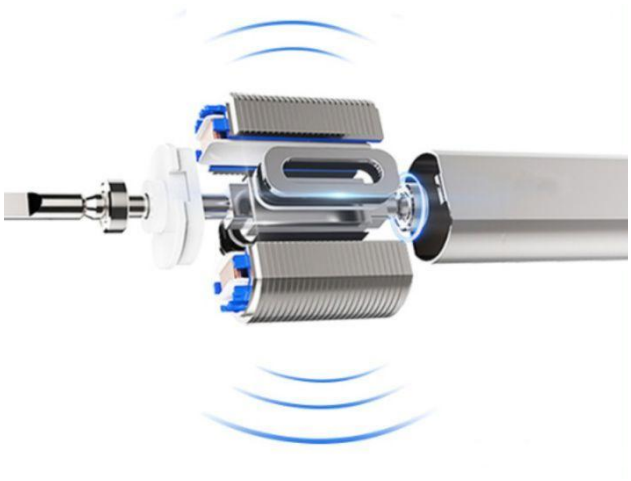

图 2. 磁悬浮电机示意图

磁悬浮电机的优点在于其在工作运行噪声小，机身振感低，振动能量集中，清洁效果佳。因此，本文采用 ACM32F030 作为主控芯片，基于磁悬浮电机提出一款电动牙刷的设计方案。

设计方案

本文描述的电动牙刷方案，是基于上海航芯 ACM32F030 系列的 MCU 进行设计，整体的方案框图如下所示：

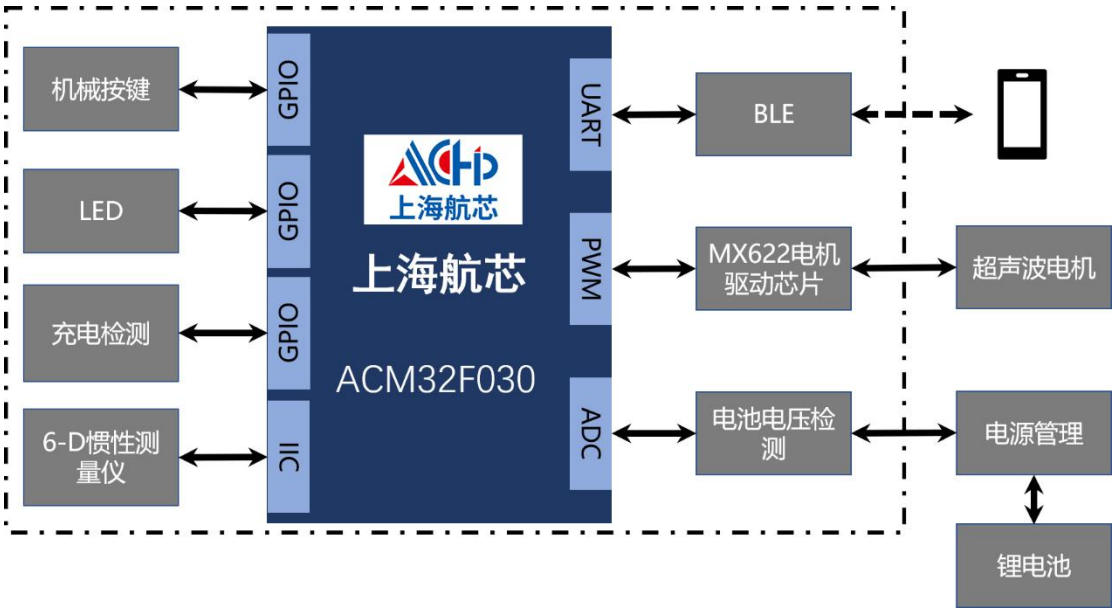

图 3. 基于 ACM32F030/070 电动牙刷设计方案框图

ACM32F0X0 系列是一款支持多种低功耗模式的通用 MCU。集成 12 位 1.6 Msps 高精度 ADC 以及比较器、运放、触控按键控制器、段式 LCD 控制器，内置高性能定时器、多路 UART、LPUART、SPI、I2C 等丰富的通讯外设，内建 AES、TRNG 等信息安全模块，支持多种低功耗模式，具有高整合度、高抗干扰、高可靠性的特点。本产品采用 ARM Cortex-M0 系列内核，最高工作频率 64MHz。足以满足一般的电动牙刷方案的需求。

软硬件下载链接如下：

https://gitee.com/acm32-mcu/electric-toothbrush

https://github.com/ACM32-MCU/electric-toothbrush

- **人机交互系统**

本文论述的设计方案中的人机交互功能是采用简单的 LED 和按键的方式进行实现。共有 1 个按键和 6 个 LED。按键需实现设备的开关机以及模式切换功能。设备会根据按键按下时

间的长短来判定当前的动作是需要切换模式或是开关机操作。6 个 LED 中有 3 个用于工作模式指示，最大可支持 7 种工作模式（2^3-1），本设计方案中仅提供了三种模式。另外 3 个 LED 用于系统状态指示，包括正常，欠压，充电，充满 4 种电压状态。

长短按识别程序：

```
void keyPressHandler(void)
{
   key.isPressed = Key_GetPressValue();
   switch(key.pressState)
   {
case 0:
        if(key.isPressed)
        {
           key.pressTime = 0;
           key.pressState = 1;
        }
        break;
case 1:      /*   eliminate jitter   */
        if(key.isPressed)
        {
           if(++key.pressTime > 10)
              key.pressState = 2;
```

```
        }
        else
            key.pressState = 0;
        break;
case 2:      /*   whether long press is existed   */
        if(key.isPressed)
        {
            if(++key.pressTime > LONG_PRESS_TIME)
                key.pressState = 3;
        }
        else
        {
            if(key.shortPressHandler != NULL)
                key.shortPressHandler();
            else
                DEBUG_KEY( “have no short press handler!!\r\n” );
            key.pressState = 0;
        }
        break;
case 3:
        if(key.longPressHandler != NULL)
            key.longPressHandler();
```

```
        else
            DEBUG_KEY( “have no long press handler!!\r\n” );
        key.pressState = 4;
        break;
case 4:      /*   wait for releasing key   */
        if(key.isPressed == 0)
            key.pressState = 0;
        break;
    }
}
```

工作指示程序：

```
void appMotorModeLedControl(void)
{
    static uint8_t state = 0xFF;

    if(sys.status == SYSTEM_RUNMODE)
    {
if(state != sys.motorStatus)
{
        state = sys.motorStatus;
        if(sys.motorStatus == 0)
        {
```

```
            ModeLed_Select(MODE_LED_1, MODE_LED_ON);
        }
        else if(sys.motorStatus == 1)
        {
            ModeLed_Select(MODE_LED_2, MODE_LED_ON);
        }
        else if(sys.motorStatus == 2)
        {
            ModeLed_Select(MODE_LED_3, MODE_LED_ON);
        }
}
  }
  else
  {
state = 0xFF;
ModeLed_Select(MODE_LED_UNKNOWN, MODE_LED_OFF);
  }
}
```

系统指示程序：

```
void appSysLedController(void)
{
  static uint8_t led_state = 0xFF;
```

```
   if(led_state != led.state)
   {
led_state = led.state;
if(led.state == LED_OFF)
{
        led.duty = 0;
        PowerLed_Select(PWR_LED_UNKNOWN, PWR_LED_OFF);
        PWM_dutySet(PWM_LED, led.duty);
}
else if(led.state == LED_TWINKLE)       // low power warning
{
        led.duty = 0;
        PowerLed_Select(PWR_LED_R, PWR_LED_ON);
        PWM_dutySet(PWM_LED, led.duty);
}
else if(led.state == LED_ON)
{
        led.duty = 0;
        PowerLed_Select(PWR_LED_R, PWR_LED_OFF);
        PWM_dutySet(PWM_LED, led.duty);
}
```

```
else if(led.state == LED_BREATHE)
{
        if(led.duty == PWM_DUTY_MAX)
          led.dir = LED_FADE;
        else
          led.dir = LED_BRIGHTER;
}
else
        led.state = LED_OFF;
  }
  else{
if(led.state == LED_BREATHE)
{
        PowerLed_Select(PWR_LED_UNKNOWN, PWR_LED_OFF);
        if(led.dir == LED_BRIGHTER)
        {
          if(led.duty < PWM_DUTY_MAX)
            led.duty += BREATHE_INTERVAL;
          else
          {
            if(++led.cnt > BREATHE_HOLD_TIME)
            {
```

```
                led.dir = LED_FADE;
                led.cnt = 0;
            }
        }
    }
    else
    {
        if(led.duty > BREATHE_INTERVAL)
            led.duty -= BREATHE_INTERVAL;
        else
        {
            led.duty = 0;
            if(++led.cnt > BREATHE_HOLD_TIME)
            {
                led.dir = LED_BRIGHTER;
                led.cnt = 0;
            }
        }
    }
    PWM_dutySet(PWM_LED, led.duty);
}
  }
```

}

- **电源及功耗管理**

电动牙刷产品的续航能力也是一直备受人们关注。本设计方案在低功耗的处理，摒弃了一般的休眠方式，直接采用关闭电源来避免设备在不工作状态下的设备功耗。整个设备的供电线路共有三种，如下图所示。

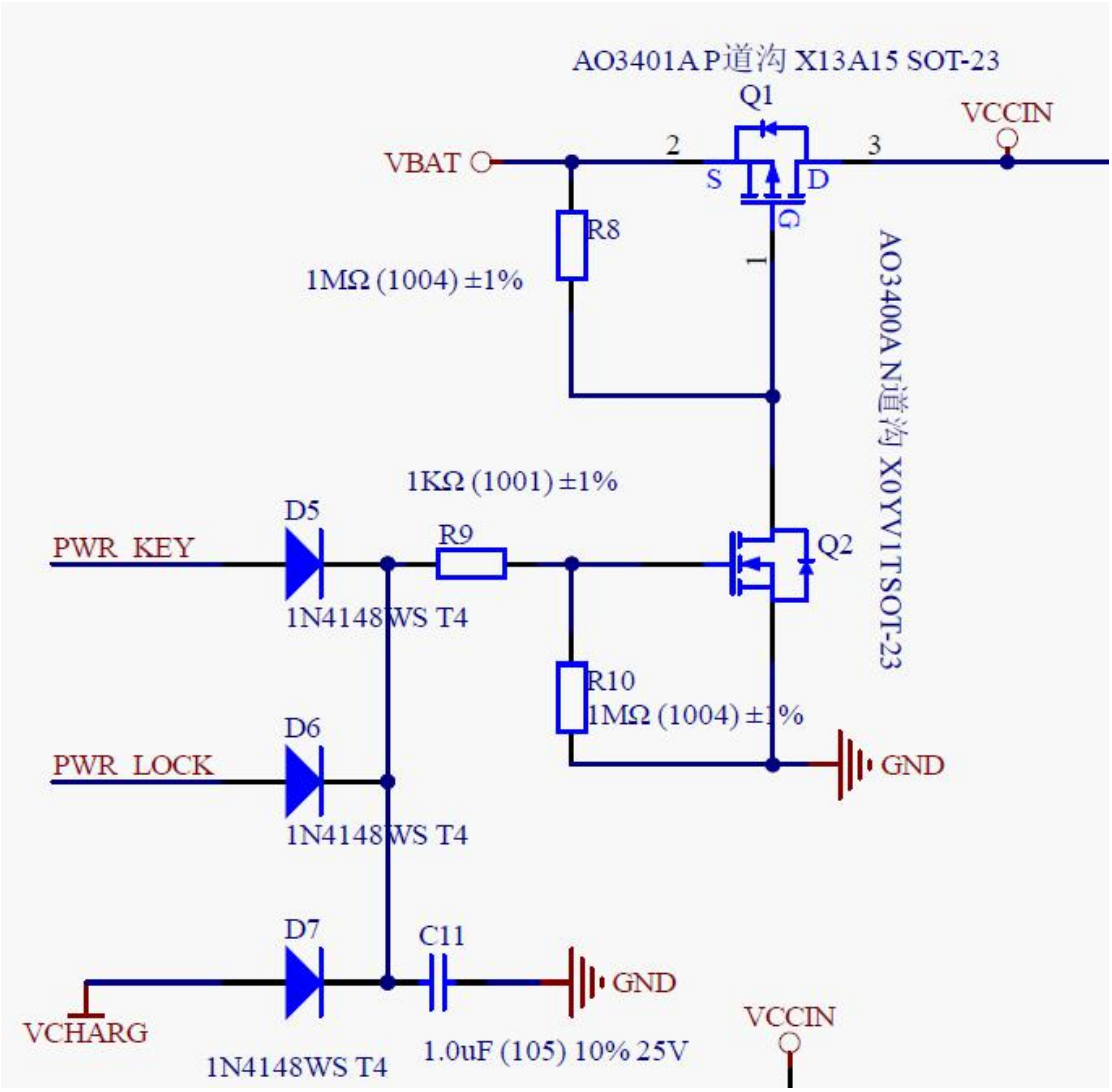

图 4. 基于 ACM32F030 的电动牙刷供电电路（部分）

正常情况下，设备不在充电时，VCHARG 电压为 0，需要关机时，按键弹开，PWR_KEY 为低电平，芯片内部程序也将 PWR_LOCK 拉低，此时 Q2 关断，Q2 的 D 极电压同 VBAT，从而引起 Q1 断开，VCCIN 断电，系统关机。而开机时，按键按下，PWR_KEY 先被拉至高电平，Q2 导通，Q2 的 D 极拉低，则 Q1 导通，设备供电，程序检测到开机，拉高 PWR_LOCK，此时，尽管按键弹开，PWR_LOCK 仍然会提供 Q2 的导通电压，系统正常工作。充电时，Q2 的导通电压会由 VCHARG 提供，系统保持在工作状态，此时会程序会检测系统的运行状态，在不需要启动时，进入休眠状态。

电源管理部分，则通过锂电池充电芯片检测是否进行充电，同时通过一路 ADC 监测电池电压。为减少芯片工作负担，电池电压的欠压和满电通过 ADC 门限电压功能来实现。ADC 的门限电压初始化程序如下：

```
// ADC Watchdog config
  ADC_WDT_Handle.ITMode          = ENABLE;
  ADC_WDT_Handle.WatchdogMode    = ADC_ANALOGWATCHDOG_RCH_ALL;
  ADC_WDT_Handle.Channel         = channel;
  ADC_WDT_Handle.HighThreshold   = (HIGH_POWER_THS * 0x0FFF) / VREF ;
  ADC_WDT_Handle.LowThreshold    = (LOW_POWER_THS * 0x0FFF) / VREF ;
```

- **智能管理系统**

智能管理系统分为两个部分，一部分为上位机的数据处理，由云端处理，另一部分是电动牙刷数据记录和传输。整个的实现过程可简述为，电动牙刷通过惯性测量仪 QMI8658C 记录电动牙刷在使用过程中的运动轨迹，并实时将该部分数据以及整个系统的工作参数通过 BLE

发送到手机，手机连接云端，并将数据传输至云平台进行数据解析，分析用户刷牙的健康指数，并将相关建议反馈至手机。电动牙刷作为数据采集设备，需上报实时数据，结构如下：

```
typedef __packed struct{
   uint32_t time;                // This shows the relative time of each activity
   uint16_t location[3];         // This shows the acceleration of brush when using
   uint16_t pressure;            // This is the force between tooth and brush
   uint16_t angle[3];            // This shows the angle between brush
}BLE_RealTimeDataDef;            // This define the data structure about brushing tooth in real time
```

其中，location 为三轴的加速度，angle 为三轴的角度。定时上传电动牙刷的相关实时数据。上位机根据一系列点位数据进行建模计算可得到整个牙刷的运动轨迹。

- **电机驱动系统**

电动牙刷的驱动系统是通过 H 桥芯片 MX612E 进行处理，MX612E 的输入端连接芯片的 PWM 互补输出端口。如下图所示：

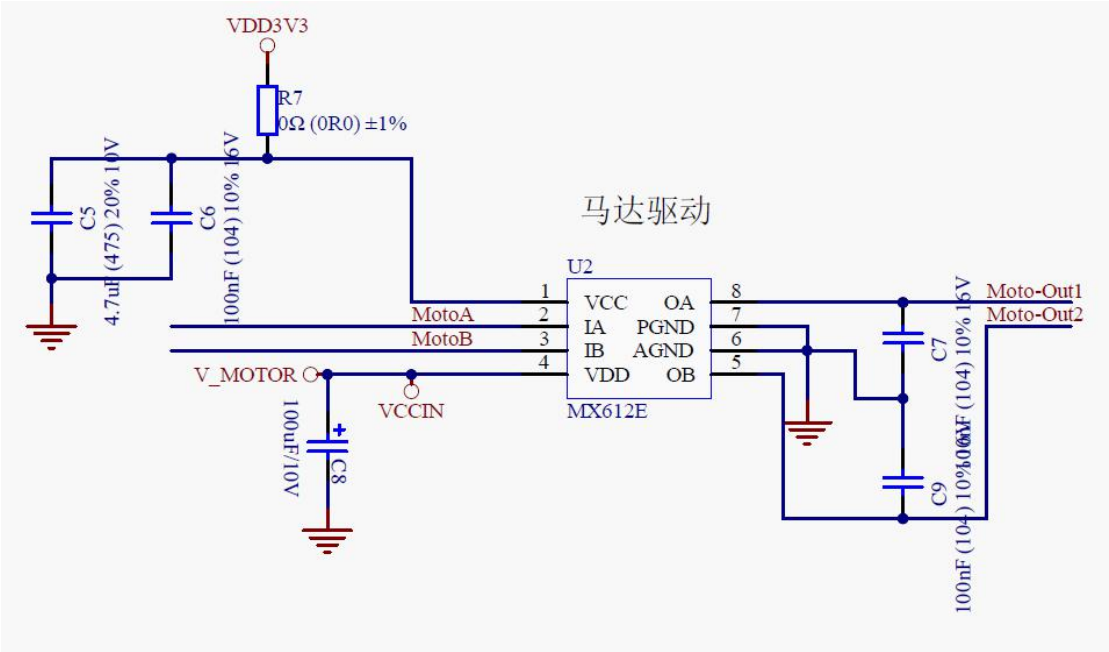

图 5. 电动牙刷电机驱动电路

本设计中的电动牙刷采用磁悬浮电机，内部构造和直流无刷电机相似，但相比于直流无刷电机，其仅有两相输入端。这也就造成该电机在通电后，正负极不变的情况下，电机旋转至某一角度形成平衡后将会停止旋转。切换正负极后则又会在另一个方向旋转形成平衡。在电动牙刷的正常工作中，是通过两相的正负极切换来使电机正反旋转从而带动刷头做高频运动的。因此，其电机速度的控制依靠于输出 PWM 的输出频率而非占空比。控制代码如下：

```
void PWM_freqSet(uint8_t PWMx, uint16_t freq)
{
  uint32_t arr;
  if(IS_PWM_INSTANCE(PWMx) == 0)   return;
  if(freq == 0)
  {
TIM15->ARR = 0;
```

```
return;
  }
  if(freq > PWM_FREQ_MAX)   freq = PWM_FREQ_MAX;
  if(freq < PWM_FREQ_MIN) freq = PWM_FREQ_MIN;
  arr = (PWM_TIMER_FRE / freq);
  if(PWMx == PWM_MOTOR)
  {
TIM15->ARR = arr-1;
TIM15->CCR1 = arr / 2;
  }
}
```

上例中，PWM 的占空比为 50%，使得在一个 PWM 周期内，电机可完成一次往返运动。

本文提出的设计方案的主旨是将电动牙刷智能化，在提高人们刷牙效率的同时，也能达到进一步保证人们刷牙质量的目的。通过电动牙刷对惯性的数据采集，实时上传至云端，并对数据进行处理，恢复用户的刷牙轨迹，给出合理建议，纠正用户不良的刷牙习惯。磁悬浮电机的高频振动也能有效清除口腔污渍。岁月恒久远，牙齿永相随 ^-^。

了解更多航芯产品&方案：www.aisinochip.com