AN0107
入门指南

AT32 MCU ACC使用指南

前言

AT32 MCU 拥有 HICK 自动时钟校准器，其作用是当系统内部使用 HICK 作为时钟源时，若外界环境改变等因素导致 HICK 时钟差生偏差时，可利用 HICK 自动时钟校准器将 HICK 校准到合理范围内，从而确保系统运行频率的精度。

支持型号列表：

支持型号	具备 ACC 的型号

目录

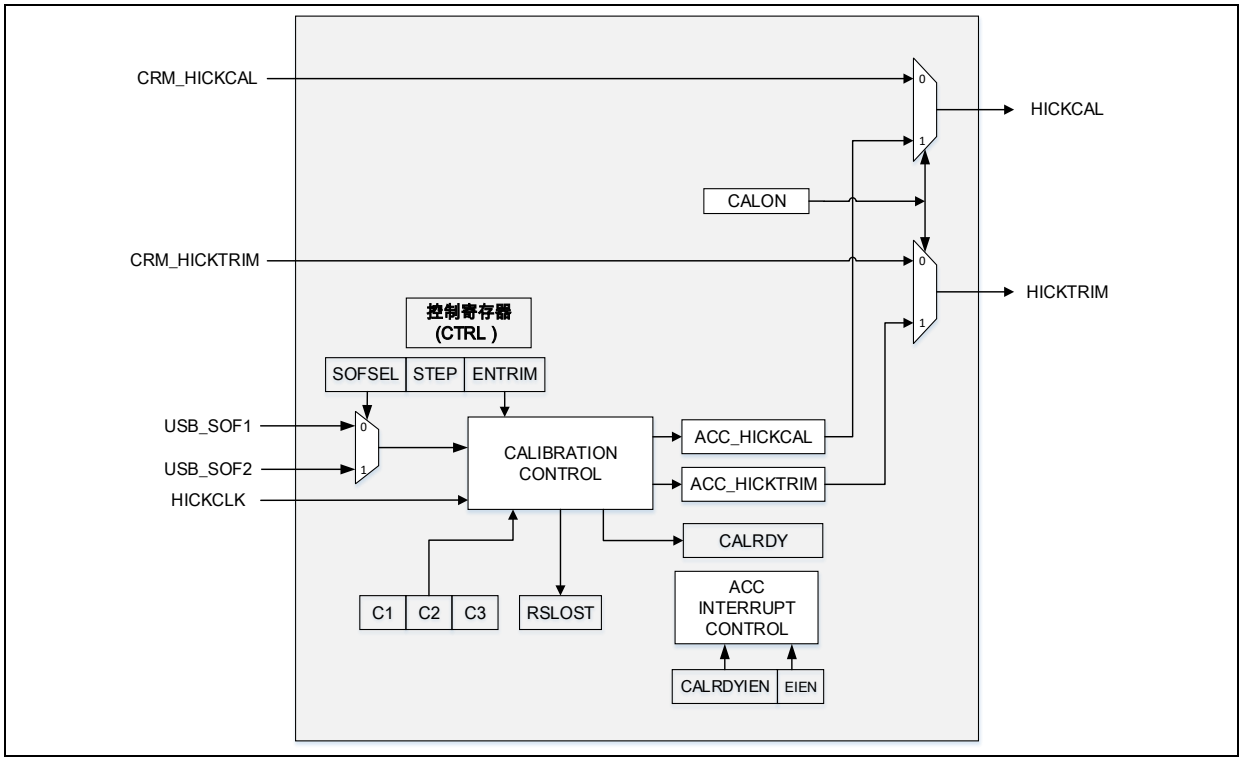
表目录

图目录

1 ACC 简介

HICK 时钟校准器（ACC）利用 OTGFS 作为设备时产生的 SOF 信号作为参考信号达到校准 HICK 的目的，SOF 信号为主机发给设备其周期为 1ms 的脉冲信号。ACC 控制器采用“跨越回归”算法，可以将 HICK 频率尽可能校准到靠近目标频率。

图 1. ACC 控制器架构



2 ACC 功能解析

2.1 主要特性

ACC 控制器具备如下特性：

- 校准 HICK，已达到对 OTGFS 设备提供 48MHz±0.25%精度的时钟
- SOF 标志可选择来源：OTGFS1 或者 OTGFS2
- 可配置的触发校准功能的边界频率
- 两种校验方式：粗校验和精校验
- 状态标志：校准就绪标志和 SOF 参考信号丢失标志
- 带标志的中断源：校准就绪标志中断源和 SOF 参考信号丢失标志中断源

2.2 中断请求

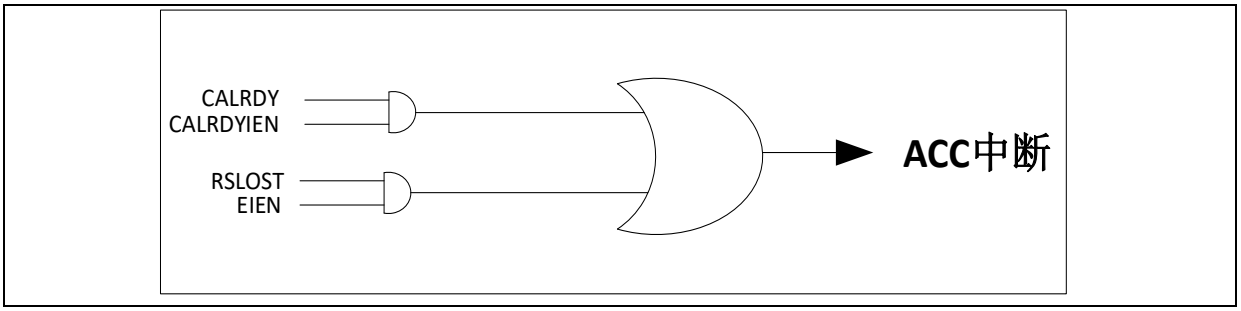
拥有两个中断请求源：校准就绪中断源和 SOF 参考信号丢失中断源

表 1. ACC 中断源

中断事件	事件标志	使能位
校准就绪	CALRDY	CALRDYIEN
参考信号丢失	RSLOST	EIEN

当设置了对应的使能位，当产生了对应的中断，就会进入对应的中断处理函数。

图 2. ACC 中断示意图



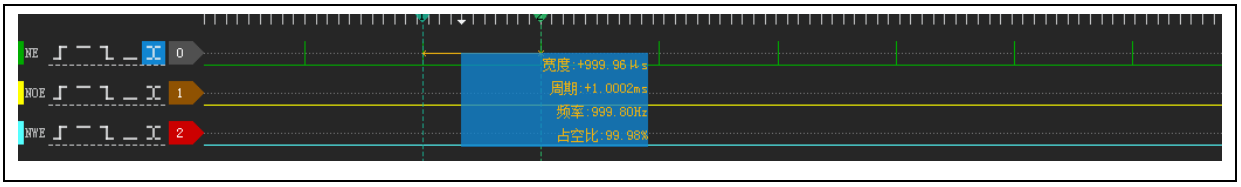
2.3 校准原理

首先需要了解到，如果系统使用的是 HICK 作为系统时钟源，而本身 ACC 模块又是对 HICK 进行校准，那么此时整个系统是不具备校准条件的，因为没有一个准确的参考信号作为校准的基准，所以就引入了 OTGFS 的 SOF 信号。SOF 信号是外部主机提供的，主机将准确的 SOF 信号（1ms 周期）给到设备（待校准系统），然后 ACC 模块采样 SOF 信号，并进行一系列的计算达到判定 HICK 是否准确，如若发现 HICK 不准确那么就会进行校准动作。

SOF 周期信号：1 毫秒的周期性必须是准确的，是自动校准模块能够正常工作的前提条件；

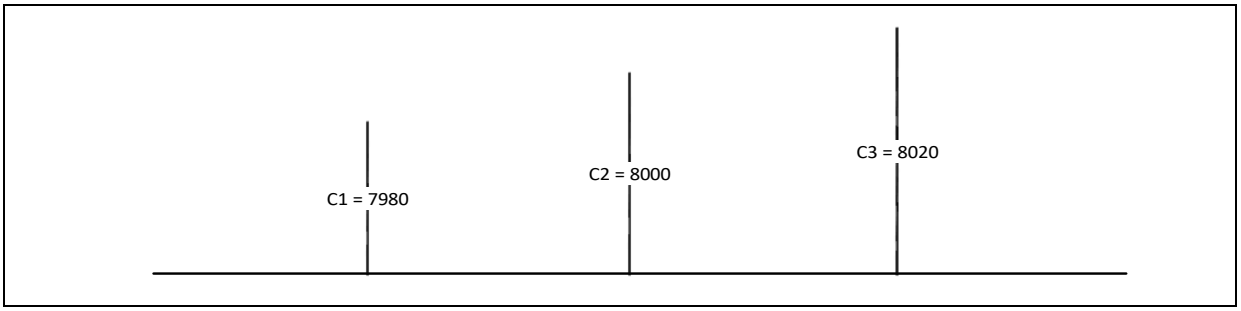
SOF 信号通过 PA8 输出，如下图：

图 3. SOF 信号示意图



cross-return 策略（跨越回归算法）：计算出离理论值最近的校准值；从理论上来说，可以将校准后的实际频率调校到离目标频率（8MHz）约 0.5 个 step 的精度范围以内。

图 4. 跨越回归算法示意图



如上图所示，一旦触发自动校准的条件满足，自动校准就会按照 step 所规定的步长调整 HICKCAL 或者 HICKTRIM。

跨越（cross）：

在满足自动校验的条件后的第一个 1 毫秒采样周期内的实际采样值要么小于 C2，要么大于 C2。当这个值小于 C2，自动校准按照 step 的定义，增加 HICKCAL 或者 HICKTRIM，直到实际采样值比 C2 大，实现实际采样值由小到大对 C2 的跨越。

当这个值大于 C2，自动校准按照 step 的定义，减少 HICKCAL 或者 HICKTRIM，直到实际采样值比 C1 小，实现实际采样值由大到小对 C2 的跨越。

回归（return）：

在跨越完成后，比较在跨越前后的实际采样值和 C2 之间的差值（按绝对值计算），得到离 C2 最近的实际采样值，从而得到最佳的校验值 HICKCAL 或者 HICKTRIM。

若跨越后的实际采样值和 C2 之间的差值小于跨越前的实际采样值和 C2 之间的差值，则以跨越后的校验值为准，并结束校验流程，直到满足下一个满足自动校验的条件。

若跨越后的实际采样值和 C2 之间的差值大于跨越前的实际采样值和 C2 之间的差值，则以跨越前的校验值为准，那么校验值会退回一个 step，并返回到跨越前的那个校验值，并结束校验流程，直到满足下一个满足自动校验的条件。

按照 cross-return 策略，在理论上，可以得到离中心频率约 0.5 个 step 所对应的频率精度。

如下四种情形会启动自动校准：

- CALON 的上升沿（从 0 到 1）；
- 当 CALON=1 时，参考信号丢失之后又恢复；
- 当采样计数器的值小于 C1；

- 当采样计数器的值大于 C3。

在 CALON 的上升沿，即便采样计数器的值大于 C1 并小于 C3,也会启动自动校准，其目的在于，在 CALON 之后，能够尽快将 HICK 的频率调整到中心频率的 0.5 个 step 以内。

以上四种情形的自动校准的结果均能将 HICK 的频率调整到中心频率的 0.5 个 step 以内。所以为了获得最佳的校准精度，建议将 step 保持为默认值 1。若将 step 设为 0，则 HICKCAL 或者 HICKTRIM 将无法改变，也即，无法校准。

3 ACC 配置解析

以下对 ACC 的配置接口及流程进行说明。

3.1 函数接口

表 2. 配置函数列表

/* ACC 校准使能：粗检验或者精校验 */ void acc_calibration_mode_enable(uint16_t acc_trim, confirm_state new_state);
/* ACC 校 step 设置 */ void acc_step_set(uint8_t step_value);
/* sof 信号源选择：OTG1 或者 OTG2 */ void acc_sof_select(uint16_t sof_sel);
/* 使能 ACC 中断 */ void acc_interrupt_enable(uint16_t acc_int, confirm_state new_state);
/* 获取精校验值 */ uint8_t acc_hicktrim_get(void);
/* 获取粗校验值 */ uint8_t acc_hickcal_get(void);
/* 配置 C1 值 */ void acc_write_c1(uint16_t acc_c1_value);
/* 配置 C2 值 */ void acc_write_c2(uint16_t acc_c2_value);
/* 配置 C3 值 */ void acc_write_c3(uint16_t acc_c3_value);
/* 读取 C1 值 */ uint16_t acc_read_c1(void);
/* 读取 C2 值 */ uint16_t acc_read_c2(void);
/* 读取 C3 值 */ uint16_t acc_read_c3(void);
/* 获取 ACC 标志 */ flag_status acc_flag_get(uint16_t acc_flag);
/* 清除 ACC 标志 */ void acc_flag_clear(uint16_t acc_flag);

3.2 配置流程

- 系统以 HICK 作为系统时钟源，并打开 ACC 时钟；
- OTG_ FS 以 HICK 为时钟源并初始化 OTGFS；
- 使能 ACC 相关中断；
- 配置 C1/C2/C3 值；
- 选择 SOF 源；
- 使能 ACC 并选择粗校验或者精校验。

4 案例 ACC 校准 HICK

4.1 功能简介

实现了使用 ACC 模块将 HCIK 校准在要求的精度内。

4.2 资源准备

1) 硬件环境:

对应产品型号的 AT-START BOARD

2) 软件环境

project\at_start_f4xx\examples\acc\calibration

4.3 软件设计

1) 配置流程

- 开启 ACC/OTGFS 外设时钟
- 配置 OTGFS 设备和 ACC 模块
- 开启 ACC

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 初始化系统时钟 */
  uint16_t data_len;
  uint8_t send_zero_packet = 0;
  uint32_t timeout;
  /* 设置优先级分组 */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  /* 系统时钟初始化 */
  system_clock_config();
  /* 板载硬件初始化 */
  at32_board_init();
  /* USB 管脚初始化 */
  usb_gpio_config();
  #ifdef USB_LOW_POWER_WAKUP
    usb_low_power_wakeup_config();
  #endif
  /* enable otgfs clock */
  crm_periph_clock_enable(OTG_CLOCK, TRUE);

  /* 选择 OTG 时钟源为 48M 并来自 HICK */
  usb_clock48m_select(USB_CLK_HICK);
  /* enable otgfs irq */
  nvic_irq_enable(OTG_IRQ, 0, 0);
  /*初始化 OTG */
```

```
  usbd_init(&otg_core_struct,
                USB_FULL_SPEED_CORE_ID,
                USB_ID,
                &class_handler,
                &desc_handler);

  /* 开启 ACC 时钟 */
  crm_periph_clock_enable(CRM_ACC_PERIPH_CLOCK, TRUE);
  /* 开启 ACC 校准完成中断 */
  acc_interrupt_enable(ACC_CALRDYIEN_INT, TRUE);
  /* 开启 ACC 参考信号丢失中断 */
  acc_interrupt_enable(ACC_EIEN_INT, TRUE);
  /* config nvic for acc int    */
  nvic_irq_enable(ACC_IRQn, 1, 0);
  /* 配置 c1\c2\c3 值 */
  acc_c2_value = 8000;
  #ifdef ACC_CAL
  acc_write_c1(acc_c2_value - 20);
  acc_write_c2(acc_c2_value);
  acc_write_c3(acc_c2_value + 20);
  #else
  acc_write_c1(acc_c2_value - 10);
  acc_write_c2(acc_c2_value);
  acc_write_c3(acc_c2_value + 10);
  #endif
  /* 选择 SOF 信号来源 */
#if (USB_ID == 0)
    acc_sof_select(ACC_SOF_OTG1);
#else
    acc_sof_select(ACC_SOF_OTG2);
#endif

  /* 开启 ACC */
  #if defined(ACC_CAL)
  acc_calibration_mode_enable(ACC_CAL_HICKCAL, TRUE);
  #elif defined(ACC_TRIM)
  acc_calibration_mode_enable(ACC_CAL_HICKTRIM, TRUE);
  #endif

  while(1)
  {
    /* get usb vcp receive data */
    data_len = usb_vcp_get_rxdata(&otg_core_struct.dev, usb_buffer);

    if(data_len > 0 || send_zero_packet == 1)
    {
```

```
      /* bulk transfer is complete when the endpoint does one of the following
         1 has transferred exactly the amount of data expected
         2 transfers a packet with a payload size less than wMaxPacketSize or transfers a zero-length
           packet
      */
      if(data_len > 0)
        send_zero_packet = 1;

      if(data_len == 0)
        send_zero_packet = 0;

      timeout = 50000;
      do
      {
        /* send data to host */
        if(usb_vcp_send_data(&otg_core_struct.dev, usb_buffer, data_len) == SUCCESS)
        {
          break;
        }
      }while(timeout --);
    }
  }
```

■ 中断处理函数

```
/* OTG 中断处理函数 */
void OTG_IRQ_HANDLER(void)
{
  usbd_irq_handler(&otg_core_struct);
}

/* ACC 中断处理函数 */
void ACC_IRQHandler(void)
{
  if(acc_flag_get(ACC_CALRDY_FLAG) != RESET)
  {
    at32_led_toggle(LED2);
    /* clear acc calibration ready flag */
    acc_flag_clear(ACC_CALRDY_FLAG);
  }
  if(acc_flag_get(ACC_RSLOST_FLAG) != RESET)
  {
    at32_led_toggle(LED3);
    /* clear acc reference signal lost flag */
    acc_flag_clear(ACC_RSLOST_FLAG);
  }
```

```
}
```

4.4 实验效果

- 如若 HICK 时钟偏离正常值，ACC 将自动启动校准（前提是 OTGFS 设备与主机成功连接），校准完成后在中断函数内会翻转 LED2；产生 SOF 信号丢失后也会进入对应中断函数内翻转 LED3。

5 文档版本历史

表 3. 文档版本历史

日期	版本	变更
2021.11.3	2.0.0	最初版本
2023.3.24	2.0.1	修改文档名称，AT32F435_437改为AT32 MCU

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A) 对安全性有特别要求的应用，如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。