

前言

本应用指南主要是介绍一种针对内部高速时钟的校准方法。

支持型号列表：

支持型号	AT32F403Axx
------	-------------

目录

1	简介	5
2	校准及原理	6
2.1	校准	6
2.2	原理	6
2.3	硬件实现	6
2.4	校准方法	6
3	使用说明	8
3.1	函数说明	8
3.2	宏定义说明	8
3.3	校准流程	8
3.4	校准演示说明	9
4	注意事项	11
5	文档版本历史	12

表目录

表 1. 文档版本历史	12
-------------------	----

图目录

图 1. 参考信号测量	6
图 2. 硬件连接	6
图 3. 流程图	9

1 简介

AT32 系列 MCU 内部都有提供适合运行的内部高速时钟（HICK），其本质就是内置于芯片的 RC 振荡器。在 25℃ 下，其典型值频率 8MHz 的精度由工厂校准到 $\pm 1\%$ ，在 -40 到 105℃，该内部高速时钟的精度达到 $\pm 2.5\%$ ，可见精度会受到温度的影响。

为了降低环境温度对精度造成的影响，用户可在运行时间隙调用校准程序来进行校准。

2 校准及原理

2.1 校准

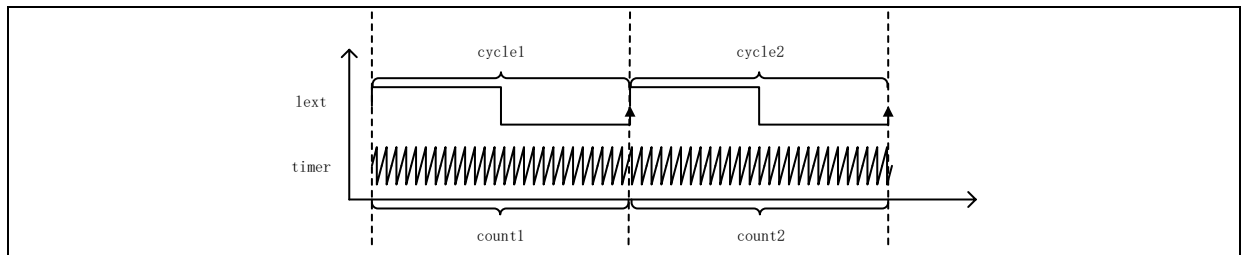
每颗 AT32 MCU 芯片的内部高速时钟在出厂时都有被进行校准，待芯片复位后该校准值会自动加载到 CRM_CTRL 寄存器的 HICKCAL[7:0]位，与 CRM_CTRL 寄存器的 HICKTRIM[4:0]位一起作用于 HICK 的校准，HICKTRIM[4:0]的复位值为 0x20（不同系列该复位值可能不同），在外部电压和温度变化对内部高速时钟频率产生影响时，可通过软件对 HICKTRIM[4:0]这些位进行编程，对 HICK 进行微调，以达到满足要求的频率。

2.2 原理

校准的原理就是对当前的 HICK 频率进行较为准确的测量，参考实际的测量值与典型值的比较结果，判断是否达到校准的目的。此处用到的 HICK 测量方法不是采用外部设备来进行的，而是使用片上定时器来对外部精准的时钟源周期进行计数，因定时器的计数时钟源于 HICK，这样就可以通过精确的外部时钟源周期来推算出当前 HICK 的频率值。

在本应用示例及文档中，精准时钟源采用的是 LEXT（通常 RTC 使用的 32kHz 晶振），图示显示了如何使用定时器计数个数来测量参考信号周期。

图 1. 参考信号测量



如上所示，为提高计数测量的准确性，实际应用中可连续对多个 LEXT 周期进行计数，再用求平均值的方式来减小误差。频率计算公式：

$$\text{Frequency}_{\text{timer}} = (\text{count1} + \text{count2} + \dots + \text{countN}) / N * \text{Frequency}_{\text{l_ext}}$$

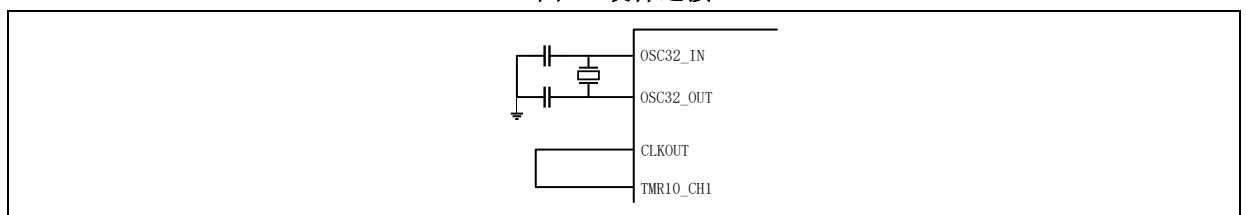
因 HICK 直接或间接的提供给系统时钟，再通过 timer 频率和主频的关系，推算出此时 HICK 的频率，当前 HICK 的频率值减去 HICK 典型值就是此时 HICK 的误差频率：

$$\text{Error (Hz)} = \text{Frequency}_{\text{hick}} - 8000000$$

2.3 硬件实现

由校准原理可知，要想计算出 HICK 的频率就必须得有一个准确的校准源，该文档示例推荐的是采用 LEXT，并且该校准源需要连接到定时器的输入捕获通道。AT32 MCU 可通过 CLKOUT 功能将 LEXT 校准源输出，再通过内部配置功能或外围连线将 CLKOUT 与定时器的输入通道连接

图 2. 硬件连接



2.4 校准方法

前文提到设置 CRM_CTRL 寄存器的 HICKTRIM 位可调整 HICK 输出，校准流程首先按 HICKTRIM

的默认值为基点配置，准确的测量出此时 HICK 实际的频率值，然后按实际频率与典型值计算得到频率误差，判断该误差是否在可接受误差值范围内，如果是，则返回成功，如果不是，则轮询下一个 HICKTRIM 设置点，再进行测量判断，直到轮询完毕返回失败。

3 使用说明

3.1 函数说明

与本文档对应的示例代码中包含了三个主要的函数

<pre>/* clkout 输出配置 */ void clkout_config(void);</pre>
<pre>/* timer 输入捕获配置 */ void tmr_input_config(void);</pre>
<pre>/* hick 校准接口函数 */ error_status hick_trimming(void);</pre>

a) clkout 输出配置

主要进行校准源 LEXT 时钟的开启，并将它配置由 **clkout** 来进行输出。由于示例的 MCU 型号支持 **clkout** 与 **timer10 channel1** 进行内部连接设置，如果该函数内并未进行 **clkout** 对应 GPIO Pin 脚的初始化，并且采用外部连接，请加入 GPIO Pin 脚初始化即可。

b) timer 输入配置

主要进行 **timer10** 的基础时钟配置、输入捕获配置及中断配置，默认采用校准源时钟的上升沿捕获。由于示例的 MCU 型号支持 **clkout** 与 **timer10 channel1** 进行内部连接设置，如果该函数内并未进行 **timer10 channel1** 对应 GPIO Pin 脚的初始化，并且采用外部连接，请加入 GPIO Pin 脚初始化即可。

c) HICK 校准接口

该函数为程序运行时的校准函数接口，可在一定时间周期或条件下调用该函数进行 **HICK** 的校准，校准到满足误差范围的频率时，该函数返回成功，反之返回失败。

需注意：在校准过程中 **HICK** 会有一个调整的过程，故此过程中的主频或外设频率可能非预期值，所以通讯类接口或时序强相关的外设需暂停使用，待校准完成后再开启使用。

3.2 宏定义说明

示例代码中有两个配置宏定义可以由用户按实际使用情况进行修改。

<pre>#define ERROR_VALUE_MAX</pre>	<pre>10000 /* 10 KHz, accuracy ±10 KHz */</pre>
<pre>#define CAPTURE_NUM</pre>	<pre>10 /* must greater than 1 */</pre>

ERROR_VALUE_MAX 定义了可接受的最大误差范围（单位 Hz），如上值（10000）即表示校准成功后的频率精度为±10kHz，用户可自行修改。需注意：由于 **HICK** 的校准特性关系，精度要求越高（即 **ERROR_VALUE_MAX** 值越小）时校准失败的概率越大。

CAPTURE_NUM 定义了 **LEXT** 校准源的捕获次数，再用累计求平均的方式减小误差。需注意：由于捕获计数时间点比较随机的关系，第一个计数值不准确，故读取的 **timer** 计数值需丢弃第一个计数，所以该宏定义值需大于 1。

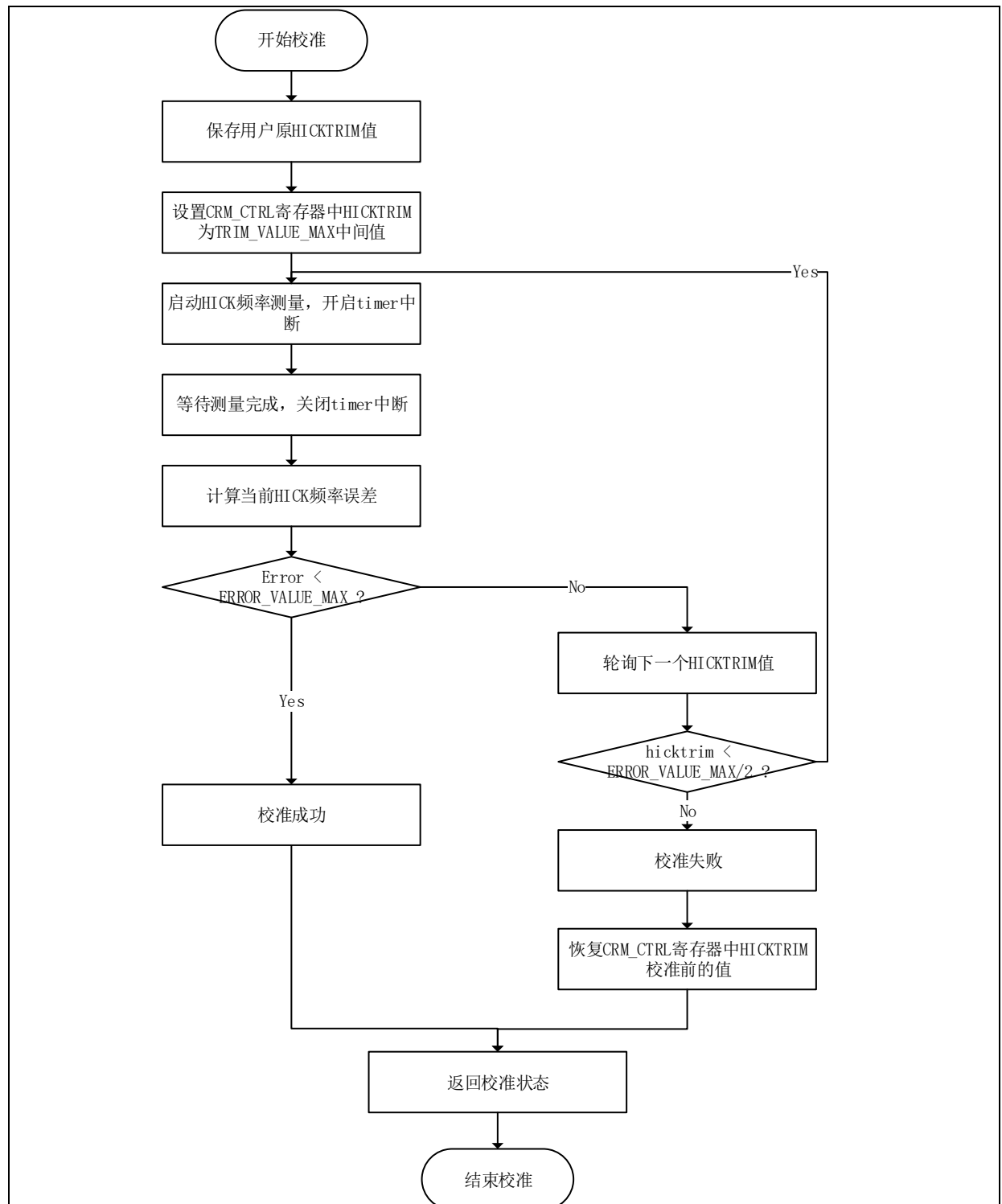
3.3 校准流程

可在系统运行时状态下调用校准函数接口 **hick_trimming** 进行校准。为防止非预期情况的发生，在调用校准前请确保各外设处于空闲未使用的状态。

首先会将校准前 **CRM_CTRL** 寄存器中的 **HICKTRIM** 值进行保留，初始校准值从 **TRIM_VALUE_MAX/2** 开始，对写入校准值后的 **HICK** 频率进行测量，量得误差精度是否满足

ERROR_VALUE_MAX 设定要求，如果是返回校准成功，如果未满足则继续轮询下一 HICKTRIM 值，直到轮询完所有，回写校准前的 HICKTRIM 值并返回校准失败。校准流程图如下：

图 3. 流程图



初始校准值选择从 HICK_VALUE_MAX/2 开始的原因是 HICKTRIM 的复位值大致就与 HICK_VALUE_MAX/2 相当，这样更接近频率漂移的中心点，由中间往外轮询的查找方式，通常情况下采用这种方式进行轮询时可以更快的找出符合要求的校准值，减少了校准时间。

3.4 校准演示说明

示例代码是基于 AT-START 进行编写，为了更好的查看到校准效果，可采用示波器量测 CLKOUT

（PA8）的输出频率，频率值为 HICK48（是 8MHz 典型值的 6 倍时钟），代码中有故意将 HICK48 调偏，当按下 **user button** 后开始校准，校准完毕串口 1 会输出校准结果（成功或失败），并可通过 CLKOUT 量测到校准后的 HICK48 频率值，示例演示的代码如下

```
int main(void)
{
    system_clock_config();

    at32_board_init();
    uart_print_init(115200);

    clkout_config();
    tmr_input_config();

    /* set a wrong trim value */
    crm_hick_clock_trimming_set(0x08);

    /* config clkout hick48 */
    crm_clock_out_set(CRM_CLKOUT_HICK);
    /* wait till user button pressed */
    while(at32_button_press() == NO_BUTTON);

    /* config clkout lext as calibration clock source */
    crm_clock_out_set(CRM_CLKOUT_LEXT);
    /* hick trimming */
    if(hick_trimming() == SUCCESS)
    {
        printf("trimming success\r\n");
    }
    else
    {
        printf("trimming fail\r\n");
    }

    /* config clkout hick48 */
    crm_clock_out_set(CRM_CLKOUT_HICK);

    while(1)
    {
    }
}
```

4 注意事项

在使用该校准方法时需注意以下几点。

1. 系统时钟应直接或间接的由 HICK 提供。
2. timer 计数时钟频率越高测量到的 HICK 频率越精确。
3. 示例 demo 所采用的方式是系统时钟与 timer 计数时钟同频，如果因应用场景需求有修改后不同频时，需注意调整频率对应关系部分的代码。

```
/* calculate hick frequency, based sysclk = tmr_freq */  
mult = (clocks.sclk_freq * 10) / HICK_VALUE;  
hick_freq = (tmr_freq * 10) / mult;
```

4. 在校准过程中可能导致主频或外设频率的变化，需注意此时如有外设还在运行时可能会出错。
5. 如所使用的系列型号不支持 clkout 与 timer 的输入通道内部连接时，需在配置函数中增加相应的 GPIO Pin 脚初始化。
6. 捕获次数宏定义 CAPTURE_NUM 需大于等于 2。
7. 最大误差宏定义 ERROR_VALUE_MAX 越大越容易校准成功，越小越可能校准失败。

5 文档版本历史

表 1. 文档版本历史

日期	版本	变更
2023.03.21	2.0.0	最初版本

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。

© 2023 雅特力科技 保留所有权利