

### PY32F030\_003\_002A 内部 RC 振荡器 (HSI) 校准

#### 应用笔记

#### 前言

PY32F030\_003\_002A 微控制器具有内部 RC 振荡器(档位 4/8/16/22.12/24MHz HSI) , 在 25°C时, HSI 的典型精度为 7‰。在 -40 到 85 °C, HSI 的精度值扩大为  $\pm 3\%$ 。因此, 温度对 RC 精度有影响。

为补偿应用中的温度等环境影响, 用户可使用运行时校准程序, 进一步微调 RC 振荡器的输出频率, 提高 HSI 的频率精度。对通信外设来说, 这可能是至关重要的。

本应用笔记给出了校准内部 RC 振荡器的方法: 通过提供精确的参考源, 设置 RCC\_ICSCR 寄存器中的 HSI\_TRIM 位, 找到具有最小误差的频率档位。

表 1. 适用产品

类型	产品系列
微型控制器系列	PY32F030、PY32F003、PY32F002A

---

## 目录

<b>1</b>	<b>内部时钟： HSI 时钟</b> .....	<b>3</b>
1.1	校准.....	3
<b>2</b>	<b>RC 校准</b> .....	<b>5</b>
2.1	校准原理 .....	5
2.2	硬件实现 .....	5
<b>3</b>	<b>代码示例</b> .....	<b>7</b>
<b>4</b>	<b>版本历史</b> .....	<b>18</b>

PUYA CONFIDENTIAL

## 1 RC 振荡器

PY32F030/PY32F003 系列内部有一组 RC 振荡器，HSI 时钟信号由内部 RC 振荡器生成，可分为 4M/8M/16M/22.12M/24M 共 5 档，可直接用作系统时钟，或者用作 PLL 输入（仅支持 PY32F030 系列）。RC 振荡器的优点是成本较低（无需使用外部组件）。它还比 HSE 晶振具有更快的启动时间。但即使校准后，频率也不如外部晶振或陶瓷谐振器的频率精度高。

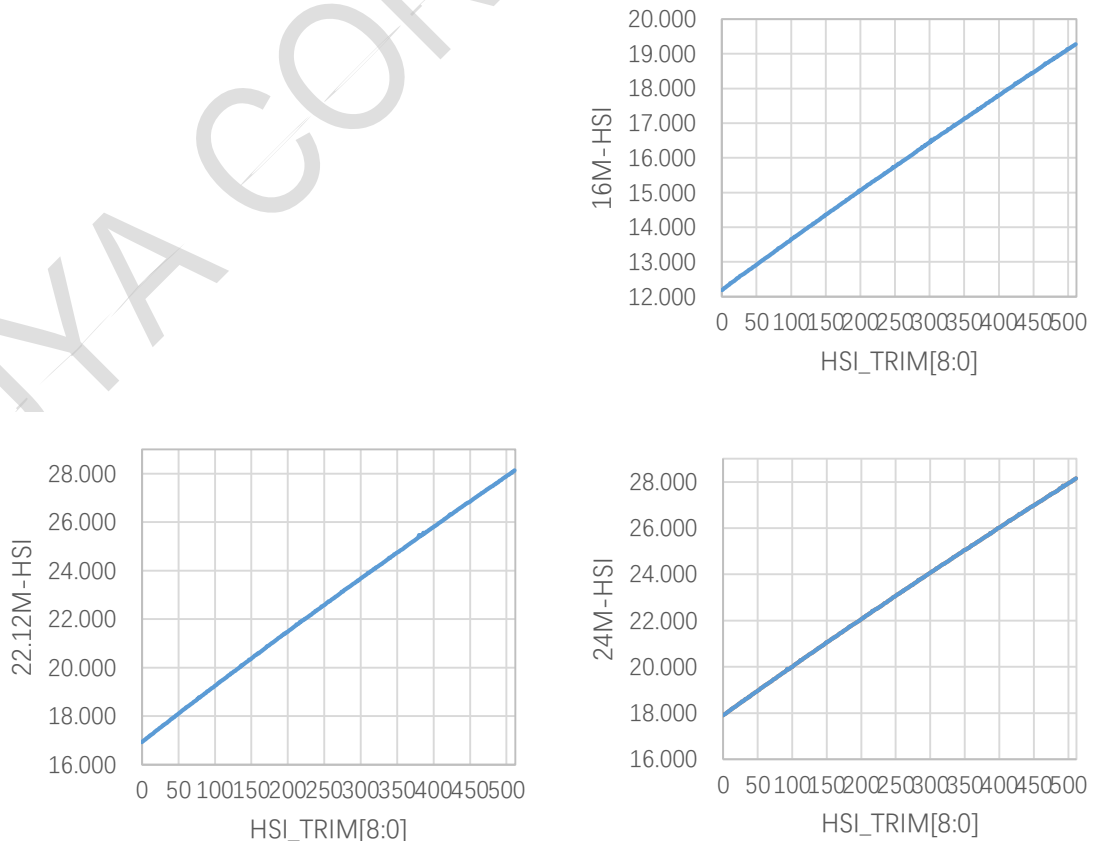
### 1.1 校准

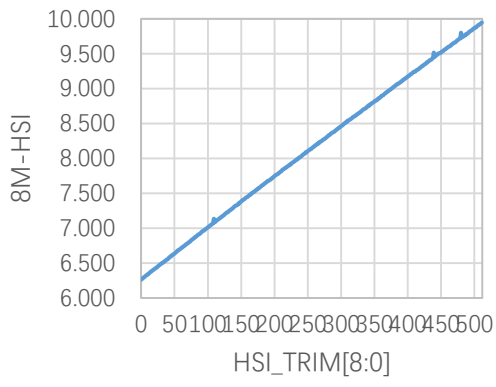
由于生产过程的不同，每个芯片的 RC 振荡器的频率都可能不同。因此，每个器件都在出厂前做工厂校准，在  $T_A = 25\text{ }^\circ\text{C}$  时达到 0.7% 精度。

复位后，工厂校准值自动加载到时钟控制寄存器 RCC\_ICSCR 的 HSI\_TRIM[12:0] 位中。其中高 4bit 中 HSI\_TRIM[12:9] 为粗调，低 9bit 中 HSI\_TRIM[8:0] 为细调，粗调数据不需要改动，保留上电后的初始值即可，通过设置 RCC\_ICSCR 寄存器中的细调 HSI\_TRIM[8:0] 位进行用户校准。可对这些位编程，以考虑电压和温度变化对内部 HSI RC 振荡器频率的影响。前后两个 HSI\_TRIM 步进之间的微调步长约为 0.1%。

图 2 显示了 HSI 频率与细调节校准值 HSI\_TRIM[8:0] 之间的关系，可以看出，两者间为线性关系，HSI 振荡器频率随校准值而增加。

图 2 HSI 频率和细调校准值





PUYA CONFIDENTIAL

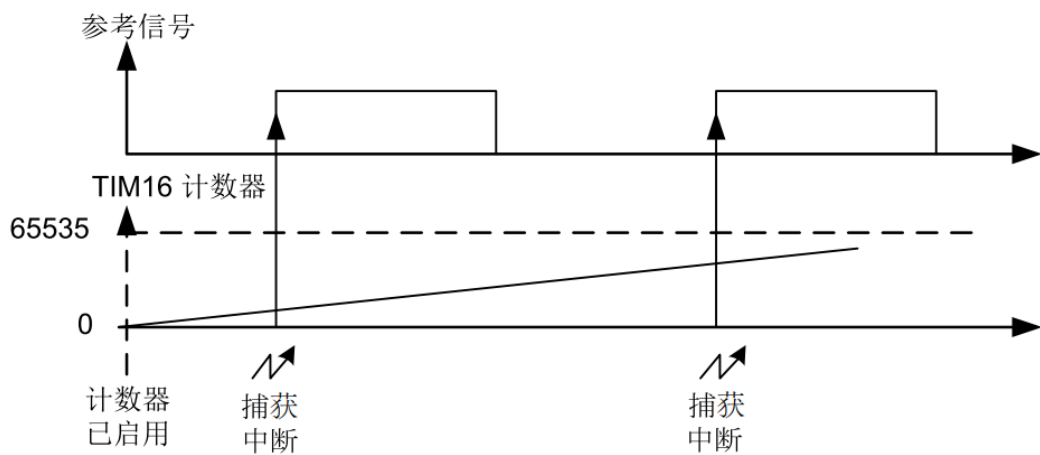
## 2 RC 校准

### 2.1 校准原理

校准的原理为首先测量 HSI 频率，然后计算频率误差，RCC\_ICSCR 的 HSI\_TRIM[12:9]保留初始值。最后设置 RCC\_ICSCR 的 HSI\_TRIM[8:0]位。

HSI 频率并不是直接测量的，而是使用定时器对 HSI 时钟沿计数方式算出，然后与典型值比较。为此，必须有一个非常精确的参考频率，比如由外部 32.768 kHz 晶振提供的 LSE 频率或 50 Hz/60 Hz 外部信号。

图 1.内部 RC 振荡器校准测量时序图



启用定时器计数后，当基准信号的第一个上升沿出现时捕获定时器计数器周期，并存储在 IC1ReadValue1 中。在第二个上升沿，又捕获到定时器计数，存储在 IC1ReadValue2 中。在两个上升沿之间的时间 (IC1ReadValue2 - IC1ReadValue1) 表示了参考信号的整个周期。

因为定时器计数器的时钟由系统时钟（内部 RC 振荡器），因此与参考信号有关的内部 RC 振荡器生成的真正频率为：

- 测得频率 = (IC1ReadValue2 - IC1ReadValue1) \* 基准频率

误差为测得频率与典型值之差的绝对值。因此，内部振荡器频率误差表示为：

- 误差 = 测得频率 - 要求的典型值

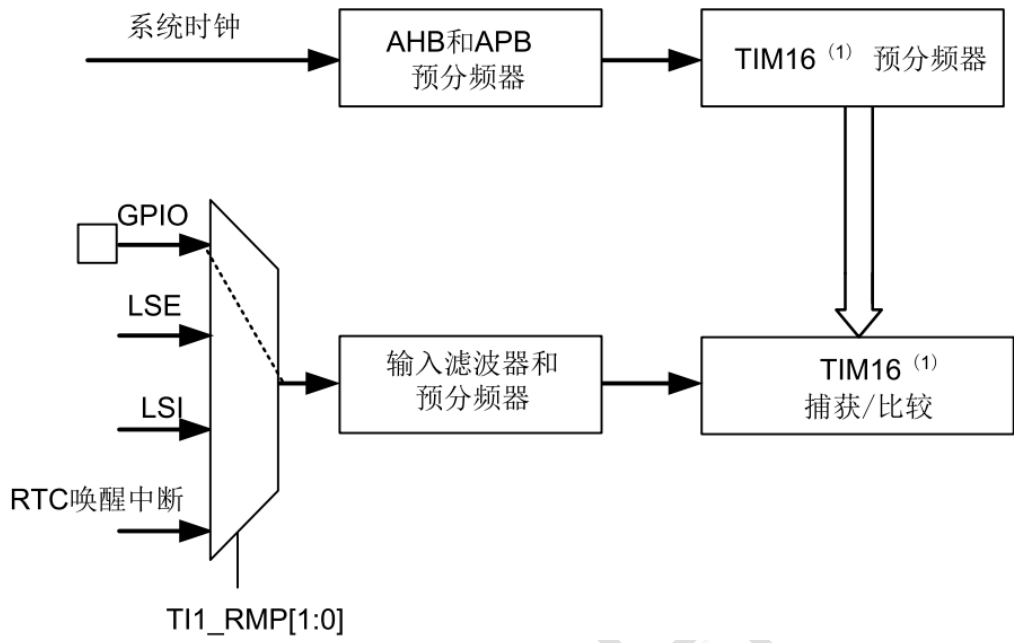
多次捕获后使用平均方法，可以使频率测量误差最小。

### 2.2 硬件实现

可以通过软件，将内部的 RC 振荡器连接到专用定时器（带输入捕获的均可）。

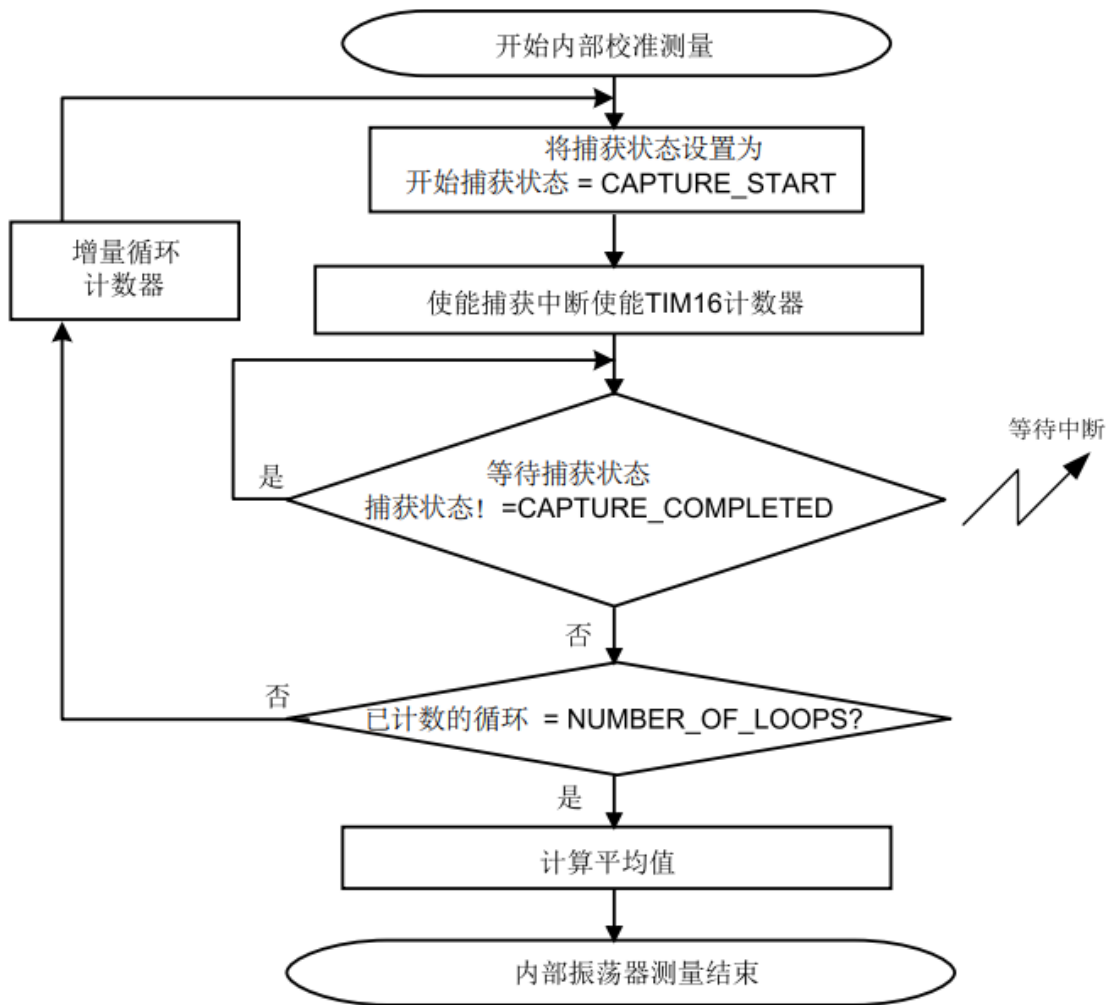
任何有精准频率的信号都可用于内部振荡器校准，例如外部信号。

下图显示连接到定时器 16 通道 1 的参考信号。



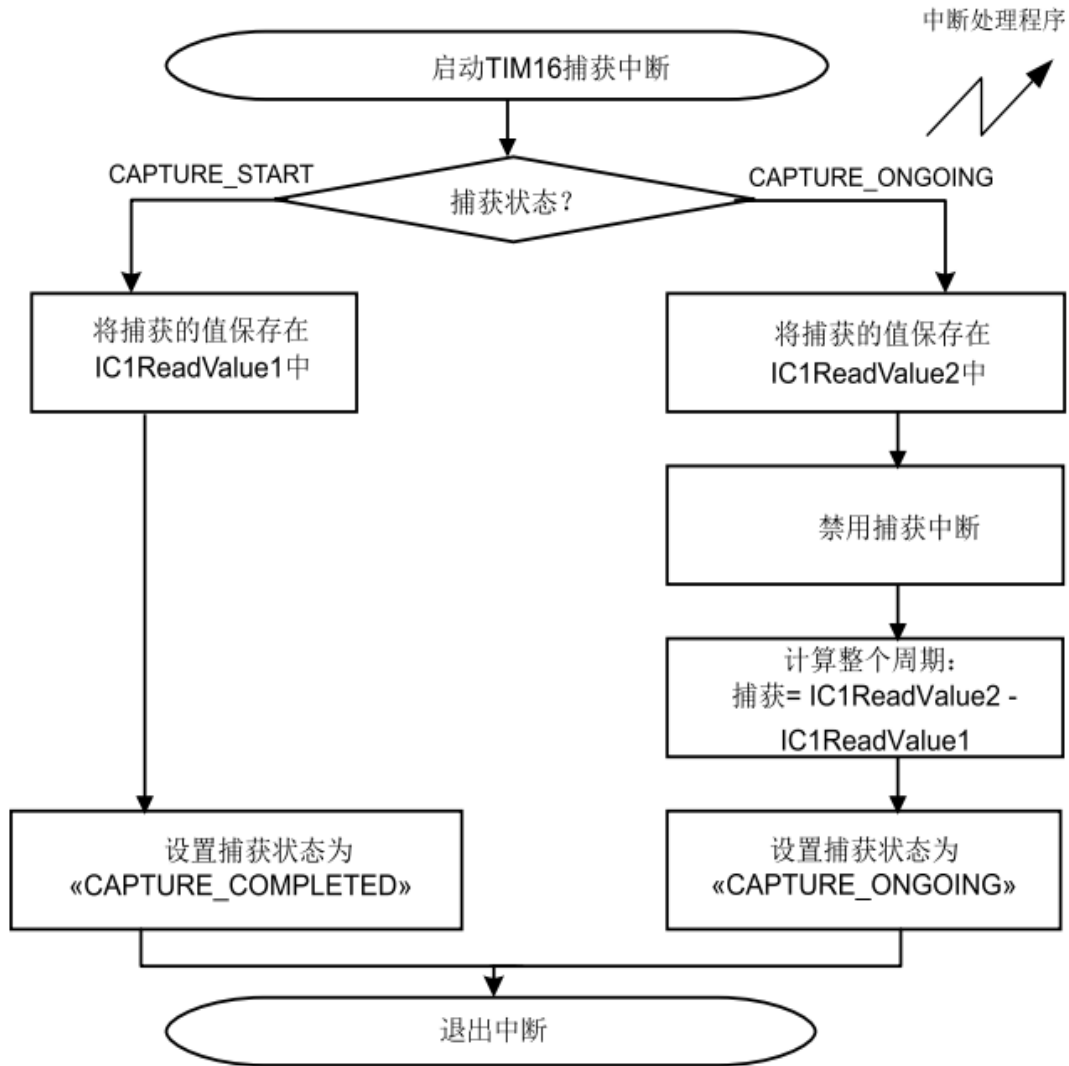
### 3 代码示例

PUYA CONFIDENTIAL



PUYAC





### 3.1 初始化

```

void HSI_MeasurementInit(uint32_t HSICLKSource_sel)
{
    SetSysClock_HSI(HSICLKSource_sel);

    /* HSI_Rough_Value default value */
    HSI_Rough_Value = ( READ_REG(RCC->ICSCR) & 0x00001fff ) >>9;//High 4 bits
    /* HSI_Fine_Value default value */
    HSI_Fine_Value = ( READ_REG(RCC->ICSCR) & 0x000001ff ); //Low 9 bits

    /* Configure the GPIO ports before starting calibration process */
    GPIO_ConfigForCalibration();

    /* Configure TIMx before starting calibration process */
  
```

```
HSI_TIMx_ConfigForCalibration();  
}
```

### 3.2 系统时钟初始化

```
void SetSysClock_HSI(uint32_t HSICLKSource_sel)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
    RCC_OscInitStruct.HSICalibrationValue = HSICLKSource_sel;
    RCC_OscInitStruct.HSEState = RCC_HSE_OFF;
    RCC_OscInitStruct.LSEState = RCC_LSE_OFF;
    RCC_OscInitStruct.LSIState = RCC_LSI_OFF;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;

    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /**Initializes the CPU, AHB and APB busses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
    RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,FLASH_LATENCY_1) != HAL_OK)
    {
        Error_Handler();
    }
}
```

```
}  
}
```

### 3.3 IO 初始化

```
void GPIO_ConfigForCalibration(void)  
{  
  
    GPIO_InitTypeDef gpio_init;  
  
    /* GPIOA clock enable */  
    TIMx_CHANNEL_GPIO_PORT();  
  
    /* TIMx channel 1 pin (PA4) configuration */  
    gpio_init.Pin = GPIO_PIN_4;  
    gpio_init.Mode = GPIO_MODE_AF_PP;  
    gpio_init.Speed = GPIO_SPEED_FREQ_HIGH;  
    gpio_init.Pull = GPIO_NOPULL;  
    gpio_init.Alternate = GPIO_AF4_TIMx;  
    HAL_GPIO_Init(GPIOA, &gpio_init);  
}
```

### 3.4 定时器 14 初始化

```
void HSI_TIMx_ConfigForCalibration(void)  
{  
    TIM_IC_InitTypeDef      ic_config; /* Timer Input Capture Configuration Structure  
    declaration */  
  
    /* Enable TIMx clock */  
    __TIMx_CLK_ENABLE();  
  
    /* Set TIMx instance */  
    TimHandle.Instance = TIMx;  
  
    /* Reset TIMx registers */  
    HAL_TIM_IC_DeInit(&TimHandle);
```

```
/* Connect input signal */
HSI_Timer_ConnectInput();

/* Initialize TIMx peripheral as follows:
+ Period = 0xFFFF
+ Prescaler = 0
+ ClockDivision = 0
+ Counter direction = Up
*/
TimHandle.Init.Period          = 0xFFFF;
TimHandle.Init.Prescaler       = HSI_TIMx_COUNTER_PRESCALER;
TimHandle.Init.ClockDivision   = 0;
TimHandle.Init.CounterMode     = TIM_COUNTERMODE_UP;
if (HAL_TIM_IC_Init(&TimHandle) != HAL_OK)
{
    /* Initialization Error */
    Error_Handler();
}

/*##-2- Configure the Input Capture channel
#####*/
/* Configure the Input Capture of channel 2 */
ic_config.ICPolarity   = TIM_ICPOLARITY_RISING;
ic_config.ICSelection  = TIM_ICSELECTION_DIRECTTI;
ic_config.ICPrescaler  = HSI_TIMx_IC_DIVIDER;
ic_config.ICFilter     = 0;
if (HAL_TIM_IC_ConfigChannel(&TimHandle, &ic_config, TIM_CHANNEL_2) != HAL_OK)
{
    /* Configuration Error */
    Error_Handler();
}

/*##-2- Configure the NVIC for TIMx */
HAL_NVIC_SetPriority(TIMx_IRQn, 0, 1);

/* Disable the TIMx global Interrupt */
```

```
HAL_NVIC_DisableIRQ(TIMx_IRQn);  
}
```

### 3.5 时钟细调校准

```
uint32_t Hsi_Fine_Trimming()  
{  
    uint32_t i;  
    uint32_t trim_Dac;//细调最终值, 低 9bit  
    uint32_t dac_Index;  
    uint32_t dac_Array[511];  
    uint32_t first_Index=255;  
    uint32_t binary_Cyc=9;  
    uint32_t sysclockfrequency = 0;  
    uint32_t measuredfrequency = 0;  
    uint32_t trim_Final_Value=0;  
    uint32_t Fine_trim_Final_freq =0;  
    uint32_t Fine_trim_Final_Dac = 0;  
  
    for(i=0;i<511;i++)  
    {  
        dac_Array[i]=i+1;  
    }  
  
    dac_Index=first_Index;  
    trim_Dac=dac_Array[dac_Index];  
  
    /* Get system clock frequency */  
    sysclockfrequency = HAL_RCC_GetSysClockFreq();  
  
    //二分法主体  
    do  
    {  
        if (StartCalibration != 0)  
        {  
            /* Set the Intern Osc trimming bits to trimmingvalue */
```

```

        HSI_RCC_AdjustCalibrationValue(_HAL_RCC_GET_SYSCLK_SOURCE(),
(HSI_Rough_Value<<9) + trim_Dac);
    }

    /* Get actual frequency value */
    measuredfrequency = HSI_FreqMeasure();

    if(ABS_RETURN((int32_t)(measuredfrequency-
sysclockfrequency))<ABS_RETURN((int32_t)(Fine_trim_Final_freq-sysclockfrequency))) //选择
最优 DAC
    {
        Fine_trim_Final_freq = measuredfrequency ;
        Fine_trim_Final_Dac = trim_Dac ;
    }
    if(measuredfrequency < sysclockfrequency)//根据当前 DAC 测量的频率和目标频率关
系, 选择下一个 DAC
    {
        dac_Index = dac_Index +(uint32_t)pow(2,binary_Cyc-2);
        trim_Dac = dac_Array[dac_Index ];
    }
    else
    {
        dac_Index = dac_Index -(uint32_t)pow(2,binary_Cyc-2);
        trim_Dac = dac_Array[dac_Index];
    }
    binary_Cyc-=1;
}while(binary_Cyc>0);
return Fine_trim_Final_Dac;
}

```

### 3.6 细调档位调节

```

void HSI_RCC_AdjustCalibrationValue(uint8_t InternOsc, uint32_t TrimmingValue)
{
    MODIFY_REG(RCC->ICSCR, RCC_ICSCR_HSI_TRIM, (TrimmingValue) <<
RCC_ICSCR_HSI_TRIM_Pos);
}

```

## 3.7 频率测量

```
uint32_t HSI_FreqMeasure( void )
{
    uint32_t measuredfrequency;
    uint32_t loopcounter = 0;
    uint32_t timeout = HSI_TIMEOUT;

    /* Start frequency measurement for current trimming value */
    measuredfrequency = 0;
    loopcounter = 0;

    /* Start measuring Internal Oscillator frequency */
    while (loopcounter <= HSI_NUMBER_OF_LOOPS)
    {
        CaptureState = CAPTURE_START;

        /* Enable capture 1 interrupt */
        HAL_TIM_IC_Start_IT(&TimHandle, TIM_CHANNEL_y);

        /* Enable the TIMx IRQ channel */
        HAL_NVIC_EnableIRQ(TIMx_IRQn);

        /* Wait for end of capture: two consecutive captures */
        while ((CaptureState != CAPTURE_COMPLETED) && (timeout != 0))
        {
            if(--timeout == 0)
            {
                return ERROR;
            }
        }

        /* Disable IRQ channel */
```

```

HAL_NVIC_DisableIRQ(TIMx_IRQn);

/* Disable TIMx */
HAL_TIM_IC_Stop_IT(&TimHandle, TIM_CHANNEL_y);

if (loopcounter != 0)
{
    /* Compute the frequency (the Timer prescaler isn't included) */
    measuredfrequency += (uint32_t) (REFERENCE_FREQUENCY * Capture);
}

/* Increment loop counter */
loopcounter++;
}
/* END of Measurement */

/* Compute the average value corresponding the current trimming value */
measuredfrequency = (uint32_t)((_HAL_GET_TIM_PRESCALER(&TimHandle) + 1) *
(measuredfrequency / HSI_NUMBER_OF_LOOPS));
return measuredfrequency;
}

```

### 3.8 定时器捕获中断回掉函数

```

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_y)
    {
        if (CaptureState == CAPTURE_START)
        {
            /* Get the 1st Input Capture value */
            IC1ReadValue1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_y);
            CaptureState = CAPTURE_ONGOING;
        }
        else if (CaptureState == CAPTURE_ONGOING)
        {

```



```
/* Get the 2nd Input Capture value */
IC1ReadValue2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_y);

/* Capture computation */
if (IC1ReadValue2 > IC1ReadValue1)
{
    Capture = (IC1ReadValue2 - IC1ReadValue1);
}
else if (IC1ReadValue2 < IC1ReadValue1)
{
    Capture = ((0xFFFF - IC1ReadValue1) + IC1ReadValue2);
}
else
{
    /* If capture values are equal, we have reached the limit of frequency
    measures */
    Error_Handler();
}
CaptureState = CAPTURE_COMPLETED;
}
}
}
```

# 1 版本历史

版本	日期	更新记录
V1.0	2022.07.10	初版
V1.1	2022.08.16	增加 002A
V1.2	2023.08.24	更新声明



Puya Semiconductor Co., Ltd.

### 声 明

普冉半导体(上海)股份有限公司 (以下简称: “Puya” ) 保留更改、纠正、增强、修改 Puya 产品和/或本文档的权利, 恕不另行通知。用户可在下单前获取产品的最新相关信息。

Puya 产品是依据订单时的销售条款和条件进行销售的。

用户对 Puya 产品的选择和使用承担全责, 同时若用于其自己或指定第三方产品上的, Puya 不提供服务支持且不对此类产品承担任何责任。

Puya 在此不授予任何知识产权的明示或暗示方式许可。

Puya 产品的转售, 若其条款与此处规定不一致, Puya 对此类产品的任何保修承诺无效。

任何带有 Puya 或 Puya 标识的图形或字样是普冉的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代并替换先前版本中的信息。