AN1018
应用笔记

如何使用 PY32F030_003_002A 微控制器的 RCC 模块

前言

RCC(Reset Clock Control，复位和时钟控制)模块主要负责芯片的复位和时钟控制功能。时钟是单片机运行的基础，如同人的脉搏心跳，时钟信号推进单片机执行指令，其重要性不言而喻。

本应用笔记主要介绍了 RCC 模块的复位和时钟，提供了含有配置时钟的代码例程。

在本文档中，PY32 仅指表 1 中列出的产品系列。

表 1. 适用产品

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

目录

1 RCC 功能简介

RCC 模块主要分为复位和时钟两个部分。

- 复位：芯片内共有两种复位，电源复位和系统复位。
- 时钟：外部高速时钟 HSE，外部低速时钟 LSE，内部高速时钟 HSI，内部低速时钟 LSI，锁相环 PLL 倍频功能时钟。

2 RCC 注意事项

- 用户在使用 HSE 和 LSE 时注意时钟稳定的时间，具体请参照表 1-1。

表 1-1 HSE 和 LSE 时钟稳定时间

时钟源	时钟稳定时间
HSE	大于 200ms
LSE	大于 2000ms

- 用户在使用 PLL 时钟时注意不同型号的芯片对于 PLL 的时钟源有要求。具体请参照表 1-2。

表 1-2 不同型号产品对 PLL 输入时钟频率要求

产品型号	输入频率最小值	输入频率最大值	单位
PY32F030Fxxx	16	24	MHz
PY32F030Exxx	16	24	MHz
PY32F030Gxxx	16	24	MHz
PY32F030Kxxx	16	24	MHz
PY32F030Lxxx	24	24	MHz

3 RCC 应用例程

3.1 使用 HSE 作为系统时钟

HSE 时钟使能后需要大约 200ms 的稳定时间，为此我们准备了两种 HSE 作为系统时钟的初始化方式。

- 第一种是 HSE 使能后一直等待，直到 HSE 稳定后，再配置系统时钟或其它时钟。打开我们的 RCC_HSE_Div 代码例程，在 main.c 中可以看到 SystemClock_Config 函数，此函数开启了 HSI，HSE，LSE，LSI，PLL 时钟，选择 HSE 作为系统时钟源，并初始化 AHB，APB 总线时钟。

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    //开启 HSI,HSE,LSE,LSI,PLL 所有时钟
    RCC_OscInitStruct.OscillatorType= RCC_OSCILLATORTYPE_HSE| //
                                      RCC_OSCILLATORTYPE_LSE| //
                                      RCC_OSCILLATORTYPE_LSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV4;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_8MHz;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEFreq = RCC_HSE_16_32MHz;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.LSEState = RCC_LSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

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

    //初始化 CPU,AHB,APB 总线时钟
    RCC_ClkInitStruct.ClockType= RCC_CLOCKTYPE_HCLK| //
                                 RCC_CLOCKTYPE_SYSCLK| //
                                 RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV4;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
    {
        Error_Handler();
    }
```

```
}
```

注意：打开 py32f030_hal_rcc.c 文件，HAL_RCC_OscConfig 就是开启并等待各个时钟稳定的函数，其中 HSE_TIMEOUT_VALUE 是等待 HSE 稳定的最大时间，建议用户不要更改，否则可能对初始化时钟有影响。

- 第二种方式是先使能 HSI 作为系统时钟，待 HSE 稳定后再选择该时钟作为系统时钟源。第二种方式没有 200ms 等待的时间影响。

1. 首先先选择 HSI 作为系统时钟，需要注意的是因为没有等待 HSE 稳定，所以不要初始化 HSE 作为 PLL 时钟，用户如果需要用到 HSE 作为 PLL 时钟源，可在 HSE 稳定后再配置。

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    //开启 HSI,HSE,LSE,LSI,PLL 所有时钟
    RCC_OscInitStruct.OscillatorType= RCC_OSCILLATORTYPE_HSE| //
                                      RCC_OSCILLATORTYPE_LSE| //
                                      RCC_OSCILLATORTYPE_LSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV4;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_8MHz;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEFreq = RCC_HSE_16_32MHz;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.LSEState = RCC_LSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
    //因为没有等待 HSE 稳定，所以 PLL 时钟源不能选择 HSE

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

    //初始化 CPU,AHB,APB 总线时钟
    RCC_ClkInitStruct.ClockType= RCC_CLOCKTYPE_HCLK| //
                                 RCC_CLOCKTYPE_SYSCLK|
                                 RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_PLLSOURCE_HSI;
    //先选择 HSI 作为系统时钟，等 HSE 稳定后再切换系统时钟为 HSE
    RCC_ClkInitStruct.AHBCLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
    {
        Error_Handler();
```

```
    }
}
```

2. 因为不需要一直等待 HSE 稳定，所以我们需要把 HAL 库的 py32f030_hal_rcc.c 文件，初始化时钟的 HAL_RCC_OscConfig 函数等待 HSE 时钟稳定的代码注释掉。

```
HAL_StatusTypeDef HAL_RCC_OscConfig(RCC_OscInitTypeDef *RCC_OscInitStruct)
{
……
//HSE 稳定需要 200ms，这里不等待 HSE 稳定直接往下执行
#if 0
                /* Check the HSE State */
                if (RCC_OscInitStruct->HSEState != RCC_HSE_OFF)
                {
                    /* Get Start Tick*/
                    tickstart = HAL_GetTick();

                    /* Wait till HSE is ready */
                    while (READ_BIT(RCC->CR, RCC_CR_HSERDY) == 0U)
                    {
                        if ((HAL_GetTick() - tickstart) > HSE_TIMEOUT_VALUE)
                        {
                            return HAL_TIMEOUT;
                        }
                    }
                }
                else
                {
                    /* Get Start Tick*/
                    tickstart = HAL_GetTick();

                    /* Wait till HSE is disabled */
                    while (READ_BIT(RCC->CR, RCC_CR_HSERDY) != 0U)
                    {
                        if ((HAL_GetTick() - tickstart) > HSE_TIMEOUT_VALUE)
                        {
                            return HAL_TIMEOUT;
                        }
                    }
                }
#endif
……
}
```

3. 代码进入 main 函数的 while 循环中后，我们开始判断 HSE 时钟是否稳定，若稳定切换系统时钟为 HSE 时钟，否则进行其它任务处理。

```
while (1)
{
  if ((READ_BIT(RCC->CR, RCC_CR_HSERDY) != 0U)&&(HSE_SYSCLK_Ready ==
0))
```

```
    {
        SetSysClock(RCC_SYSCLKSOURCE_HSE);
        HSE_SYSCLK_Ready = 1;
    }
    //其它任务处理
}
```

3.2 使用 LSE 作为系统时钟

LSE 时钟使能后需要大约 2000ms 的稳定时间，为此我们准备了两种 LSE 作为系统时钟的初始化方式。

- 第一种是 LSE 使能后一直等待，直到 LSE 稳定后，再配置系统时钟或其它时钟。打开我们的 RCC_LSE_Div 代码例程，在 main.c 中可以看到 SystemClock_Config 函数，此函数开启了 HIS，LSE 时钟，并选择 LSE 作为系统时钟源，初始化 AHB，APB 总线时钟。

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    //开启 HIS，LSE 所有时钟
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_8MHz;
    RCC_OscInitStruct.HSEState = RCC_HSE_OFF;
    RCC_OscInitStruct.LSIState = RCC_LSI_OFF;
    RCC_OscInitStruct.LSEState = RCC_LSE_ON;
    RCC_OscInitStruct.LSEDriver = RCC_LSE_DRIVER2;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_OFF;

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

    //初始化 CPU,AHB,APB 总线时钟
    RCC_ClkInitStruct.ClockType =   RCC_CLOCKTYPE_HCLK| //
                                    RCC_CLOCKTYPE_SYSCLK| //
                                    RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_LSE;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
    {
        Error_Handler();
    }
}
```

注意：打开 py32f030_hal_rcc.c 文件，HAL_RCC_OscConfig 就是开启并等待各个时钟稳定的函数，其中 LSE_TIMEOUT_VALUE 是等待 LSE 稳定的最大时间，建议用户不要更改，否则可能对初始化时钟有影响。

- 第二种方式是先使能 LSI 作为系统时钟，待 LSE 稳定后再选择该时钟作为系统时钟源。第二种方式没有 2000ms 等待的时间影响。

1. 首先先选择 LSI 作为系统时钟。

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    //开启 HSI,LSE,LSI,PLL 所有时钟
    RCC_OscInitStruct.OscillatorType =  RCC_OSCILLATORTYPE_LSE| //
                                        RCC_OSCILLATORTYPE_LSI;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
    RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_8MHz;
    RCC_OscInitStruct.HSEState = RCC_HSE_OFF;
    RCC_OscInitStruct.LSIState = RCC_LSI_ON;
    RCC_OscInitStruct.LSEState = RCC_LSE_ON;
    RCC_OscInitStruct.LSEDriver = RCC_LSE_DRIVER2;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_OFF;

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

    //初始化 CPU,AHB,APB 总线时钟
    RCC_ClkInitStruct.ClockType =  RCC_CLOCKTYPE_HCLK| //
                                   RCC_CLOCKTYPE_SYSCLK|//
                                   RCC_CLOCKTYPE_PCLK1;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_LSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

    if(HAL_RCC_ClockConfig(&RCC_ClkInitStruct,FLASH_LATENCY_0)!= HAL_OK)
    {
        Error_Handler();
    }
}
```

2. 因为不需要一直等待 LSE 稳定，所以我们需要把 HAL 库的 py32f030_hal_rcc.c 文件，初始化时钟的 HAL_RCC_OscConfig 函数等待 LSE 时钟稳定的代码注释掉。

```
HAL_StatusTypeDefHAL_RCC_OscConfig(RCC_OscInitTypeDef
*RCC_OscInitStruct)
{
.....
```

```
//LSE 稳定需要 2000ms，这里不等待直接往下执行
#if 0
        /* Check the LSE State */
        if (RCC_OscInitStruct->LSEState != RCC_LSE_OFF)
        {
            /* Get Start Tick*/
            tickstart = HAL_GetTick();

            /* Wait till LSE is ready */
            while (READ_BIT(RCC->BDCR, RCC_BDCR_LSERDY) == 0U)
            {
                if ((HAL_GetTick() - tickstart) > RCC_LSE_TIMEOUT_VALUE)
                {
                    return HAL_TIMEOUT;
                }
            }
        }
        else
        {
            /* Get Start Tick*/
            tickstart = HAL_GetTick();

            /* Wait till LSE is disabled */
            while (READ_BIT(RCC->BDCR, RCC_BDCR_LSERDY) != 0U)
            {
                if ((HAL_GetTick() - tickstart) > RCC_LSE_TIMEOUT_VALUE)
                {
                    return HAL_TIMEOUT;
                }
            }
        }
#endif
……
}
```

3. 代码进入 main 函数的 while 循环中后，我们开始判断 LSE 时钟是否稳定，若稳定切换系统时钟为 LSE 时钟，否则进行其它任务处理。

```
while (1)
{
 if ((READ_BIT(RCC->BDCR, RCC_BDCR_LSERDY) != 0U)&&( LSE_SYSCLK_Ready == 0))
   {
        SetSysClock(RCC_SYSCLKSOURCE_LSE);
        LSE_SYSCLK_Ready = 1;
   }
   //其它任务处理
}
```

4 版本历史

版本	日期	更新记录
V0.1	2021.10.18	初版
V1.0	2022.06.21	修改了应用例程
V1.1	2023.08.16	增加 002A 内容
V1.2	2023.08.24	更新声明

Puya Semiconductor Co., Ltd.