应用笔记

ACM32F070 / A070 / FP001 / WB15 系列芯片
时钟说明

版本：V1.2

日期：2025-3-10

上海航芯电子科技股份有限公司

1. 系统时钟配置

本章节介绍了如何配置系统时钟 SYS_CLK，如图所示，系统时钟 SYS_CLK 来源包括 XTL、XTH、RCH、RC32K、PLL 五种方式，PLL 可以选择 XTH 或 RCH 的 16 分频后作为输入。

时钟来源通过系统寄存器中的时钟控制寄存器 1(CCR1)进行选择。

图 1–1 系统时钟来源

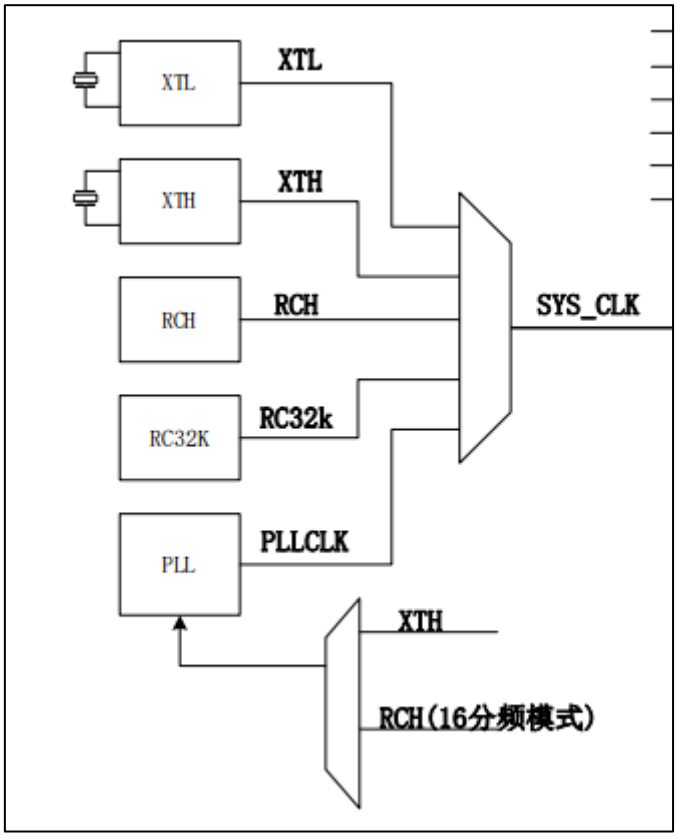


图 1–2 时钟控制寄存器

2.10.5. **时钟控制寄存器 1/CCR1(偏移：10h)**

比特	名称	属性	默认值	功能描述
31:3	RSV	-	-	保留
2:0	SYS_CLK_SEL	RW	000	系统时钟 SYS_CLK 选择： 000：来自 RCH 001：来自 RC32K 010：来自 XTH 011：来自 XTL 1xx：来自 PLLCLK

1.1. 通过 RCH 配置

系统时钟 SYS_CLK 源可选择内部高速 RC 产生的 RCH 时钟，最高 64M。在航芯提供的 HAL 库中，默认使用 RC64M 时钟作为时钟源，默认支持 64M/32M/16M/8M 频率，且 PCLK=HCLK，通过系统寄存器中的时钟控制寄存器 2（CCR2）进行分频配置。

图 1–3 时钟配置代码

System_ACM32F0x0.c

```
106
107 /*****************************************************************************************
108  * Function    : System_Clock_Init
109  * Description : Clock init
110  * Input       : fu32_Clock: System core clock
111  * Outpu       : 0: success, other value: fail reason
112  * Author      : xwl                          Date : 2021
113 *****************************************************************************************/
114 bool System_Clock_Init(uint32_t fu32_Clock)
115 {
116     uint32_t lu32_DIV, lu32_system_clk_source, lu32_result, lu32_timeout;
117
118     lu32_system_clk_source = CLK_SRC_RC64M;
119
120     SET_EFC_RD_WAIT(RD_WAIT_SET_DEFAULT)
121
122     switch (fu32_Clock)
123     {
124         /* 64MHz */
125         case 64000000: lu32_DIV = 1; break;
126
127         /* 32MHz */
128         case 32000000: lu32_DIV = 2; break;
129
130         /* 16MHz */
131         case 16000000: lu32_DIV = 4; break;
132
133         /* 8MHz */
134         case 8000000:  lu32_DIV = 8; break;
135
136         default: return false;
137     }
138
139     lu32_result = 0;
140
141     if (lu32_system_clk_source == CLK_SRC_XTH_PLL)
142     {
143         lu32_timeout = 0;
```

系若要更改时钟源，只需将宏 CLK_SRC_RC64M 更改为 CLK_SRC_XTH_PLL 即可使用 PLL 作为时钟，PLL 相关将在下一小节进行讲解。

图 1-4 时钟源选择

System_ACM32F0x0.h

```
25
26 #define CLK_SRC_RC64M                   (0x00)
27 #define CLK_SRC_XTH_PLL                 (0x01)
28
```

1.2. 通过 PLL 进行配置

系统时钟 SYS_CLK 源可选择 PLL 进行配置，在航芯提供的 HAL 中，当初始化代码中 lu32_system_clk_source=CLK_SRC_XTH_PLL 时，使用 PLL 进行配置，其中 PCLK=HLCK。使用 PLL 进行配置时可选外部 8M 或者 12M 晶振输入，通过宏进行切换，如图。

图 1–5 通过 PLL 配置代码

System_ACM32F0x0.c

```
140
141     if (lu32_system_clk_source == CLK_SRC_XTH_PLL)
142     {
143         lu32_timeout = 0;
144
145         SCU->XTHCR = SCU_XTHCR_XTH_EN | SCU_XTHCR_READYTIME_32768;
146         while (0 == (SCU->XTHCR & SCU_XTHCR_XTHRDY))
147         {
148             if (lu32_timeout == SYSTEM_TIMEOUT)
149             {
150                 lu32_result = 1;
151                 break;
152             }
153             lu32_timeout++;
154         }
155
156         if (0 == lu32_result)
157         {
158             SCU->PLLCR |=  SCU_PLLCR_PLL_EN;
159             SCU->PLLCR &= ~(SCU_PLLCR_PLL_SLEEP);
160             while(!(SCU->PLLCR & (SCU_PLLCR_PLL_FREE_RUN) )) {}
161
162 #ifdef XTH_8M_CRYSTAL
163             SCU->PLLCR = (SCU->PLLCR &(~(0x1FFFFU << 3))) | (15U << 3) | (1U << 12) | (0U << 16);
164 #endif
165
166 #ifdef XTH_12M_CRYSTAL
167             SCU->PLLCR = (SCU->PLLCR &(~(0x1FFFFU << 3))) | (15U << 3) | (2U << 12) | (0U << 16);
168 #endif
169
170             SCU->PLLCR = (SCU->PLLCR & (~(0x3U << 1)) ) | (3 << 1);
171             SCU->PLLCR |= SCU_PLLCR_PLL_UPDATE_EN;
172             while(!(SCU->PLLCR & (SCU_PLLCR_PLL_FREE_RUN) ) );
173
174             /* Division Config */
175             SCU->CCR2 = (SCU->CCR2 & (~0xFF)) | APB_CLK_DIV_0 | (lu32_DIV - 1);
176             while((SCU->CCR2 & (1UL << 31)) == 0x00);
177
```

1.2.1. 选择 PLL 时钟源

PLL 时钟源可以选择为片内 RCH 16 分频后的时钟或片外 XTH，通过系统寄存器中的 PLL 模块控制寄存器（PLLCR）进行使能和配置。

图 1-6 PLL 时钟源选择寄存器

2:1	PLL_SRC_SEL	RW	00	PLL 时钟源选择 00：选择片内 RCH（需设置 16 分频模式） 1x：选择片外 XTH
0	PLL_EN	RW	0	PLL 模块使能 0：不使能，PLL 模块处于 power down 状态 1：使能，PLL 模块使能

在航芯提供的 HAL 库代码中，默认使用外部 XTH8M，或者外部 XTH12M。

注:若使用的外部 XTH 为其他频率，可参照直接修改代码即可。

1.2.2. 配置系统时钟

在选择完 PLL 的时钟源后，即可根据输入配置 PLLCLK 作为系统时钟，通过系统寄存器中的 PLL 模块控制寄存器（PLLCR）的 PLL 分频因子进行配置，如图。

图 1–7 PLL 分频因子说明

17:16	PLL_M	RW	01	输出分频控制字段。PLL_M 含有输出分配控制字段。对应分频值为 00: 1 分频 01: 2 分频 10: 4 分频 11: 8 分频
15:14	RSV	-	-	保留
13:12	PLL_N	RW	01	降频因子分频器字段。PLL_N 含有 PLL 输入时钟的分频因子，该值的实际作用是参考频率的分频因子。对应分频值为（PLL_N+1）
11:7	RSV	-	-	保留
6:3	PLL_F	RW	0001	增频因子分频器字段。PLL_F 含有 PLL 反馈回路中分频器的分频因子，该值的实际作用是参考频率的倍频因子。对应倍频值为(PLL_F+1) PLL 输出公式为: Fpll= ((PLL_F+1)*Fin/(PLL_N+1))/ 2^{PLL_M} 其中输入频率 Fin 范围为 3~32MHz VCO 输入频率范围 Fin/(PLL_N+1)为 3~8Mhz VCO 的工作范围(PLL_F+1)*Fin/(PLL_N+1)为 32~64Mhz

例如当 PLL 时钟源为 XTH（8M）时，若需要配置为 64M 系统时钟，则可分别将 PLL_F=15,PLL_N=1,PLL_M=0,则 Fpll=8M*(15+1)/(1+1)/2^0=64M。

图 1-8 PLL 配置代码

```
161
162 #ifdef XTH_8M_CRYSTAL
163             SCU->PLLCR = (SCU->PLLCR &(~(0x1FFFFU << 3))) | (15U << 3) | (1U << 12) | (0U << 16);
164 #endif
165
166 #ifdef XTH_12M_CRYSTAL
167             SCU->PLLCR = (SCU->PLLCR &(~(0x1FFFFU << 3))) | (15U << 3) | (2U << 12) | (0U << 16);
168 #endif
```

2. RCH 时钟相关

本章节介绍了如何通过 NVR 获取 RCH 的实测值，因为在实际情况中，RC64M 时钟并不是刚好等于 64M，在需要准确配置波特率的场景，如 UART 通讯时，需要知道当前 RCH 的实际值，并进行 Trim 等操作，本章将做一一介绍。

2.1. RCH 实测值获取

以下地址记录了 RC64M 的实测值，为 CP 测试后写入，定义如下：

地址	位宽	说明	备注
0x0008022C	32bit	RC64M 除以 16 后的分频值，单位为 K	常温下 CP 测试实测值，封装以及焊接后 RC64M 实际值会改变

如：读出该地址 0x0008022C 的数值为 0xF7D，则实际的 RC64M 值=0xF7D*16000=63440K。

2.2. RCH 的 Trim

当 RC64M 实测值比理论值相差过多时，可自行进行 Trim，通过系统寄存器中的 RCH 模块控制寄存器（RCHCR）进行配置，Trim 值越高，频率越高，如下图所示。

图 2-1 RCH 的 Trim 说明

7:1	RCH_TRIM[6:0]	RW	0x0	RCH 时钟 TRIM 值 TRIM 值越高频率越高
0	RCH_EN	RW	1	RCH 模块使能

3. UART 中关于 RCH 的使用

本章节介绍了在 UART 中如何根据 RCH 的实测值和理论值之间的差异对波特率进行配置，由于 RCH 并不是准确等于 64M，故采用 RCH 配置的系统时钟也存在误差，若不对 RCH 进行 Trim 校准的情况下直接使用理论系统时钟进行计算，可能导致波特率不准确，无法进行通讯。

在航芯提供的 HAL 库中，在 UART 配置波特率时，将判断当前时钟源是否来自 RCH，若来自 RCH，则会利用第二章的方法，获取 RCH 的实测值，然后跟 64M 理论值进行比例换算，根据比例系数计算出当前准确的 PCLK 时钟频率，再继续比特率配置，如图。

图 3–1 UART 波特率设置方法

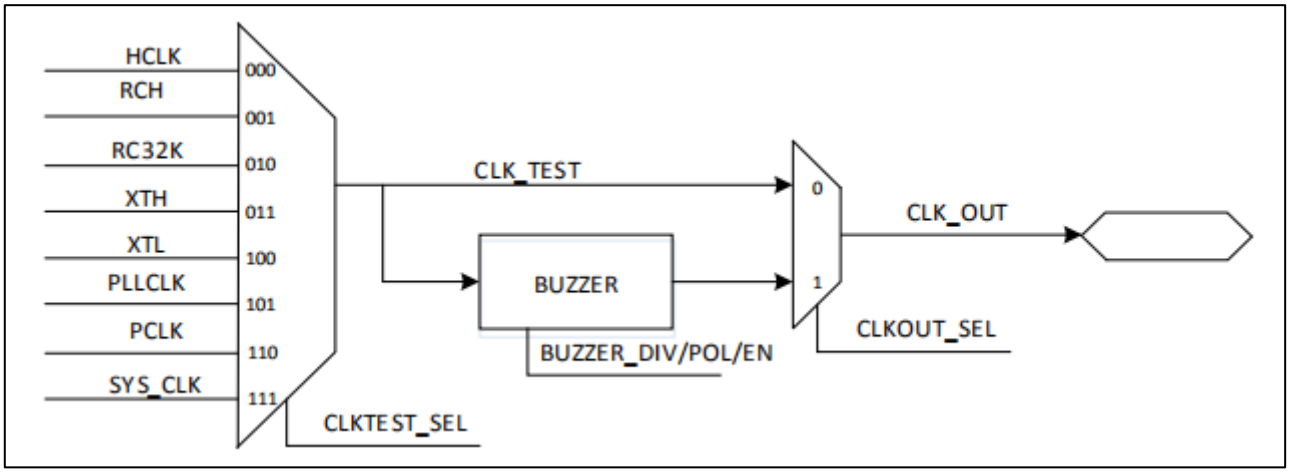
HAL_Uart.c

```
757
758 /*******************************************************************************************
759  * Function    : UART_Config_BaudRate
760  * Description : Config BaudRate
761  * Input       :
762  * Output      :
763  * Author      : Chris_Kyle                          Data : 2020
764 *******************************************************************************************/
765 static void UART_Config_BaudRate(UART_HandleTypeDef *huart)
766 {
767     uint32_t lu32_PCLK;
768     uint32_t lu32_IBAUD, lu32_FBAUD;
769     uint64_t lu64_TempValue;
770
771     if (CLK_SRC_RCH == System_Get_Clk_Source() )
772     {
773         uint32_t  RCH_Real= *((uint32_t *)0x8022c);
774         uint32_t  RCH_Ratio= (RCH_Real*16)/64;      // RCH_Ratio = RCH_Real/64
775         lu32_PCLK = (System_Get_APBClock()/1000)*RCH_Ratio;
776     }
777     else
778     {
779         lu32_PCLK = System_Get_APBClock();
780     }
```

4. CLKOUT 的使用

芯片的时钟输出可以通过 MCO 引脚。如图所示，输出的时钟信号可以通过 CLKTEST_SEL 选择，可以来自于系统时钟 SYS_CLK，也可以来自于其他时钟源如 XTL（低速晶振），然后可以直接输出，也可以通过一个 Buzzer 控制电路再输出。通过 Buzzer 可以将时钟进行分频，极性翻转后再输出。

图 4–1 CLKOUT 示意图



在航芯提供的 HAL 库中，内嵌了时钟输出函数，使用了 Buzzer 进行输出，默认输出时钟为 HCLK，若需要输出上文所述的其他时钟，则需用户自行修改此函数，设置 CLKOCR 的 CLK_TEST 即可。

图 4–2 时钟输出函数

System_ACM32F0x0.c

```
610 }
611 /*******************************************************************************
612 * Function    : System_Set_Buzzer_Divider
613 * Description : set buzzer divide factor
614 * Input       :
615               div: div factor, if div = 80 then output buzzer freq=HCLK/80
616               enable: FUNC_DISABLE and FUNC_ENABLE
617 * Output      : none
618 * Author      : xwl                          Date : 2021?¨o
619 *******************************************************************************/
620 void System_Set_Buzzer_Divider(uint32_t div, FUNC_DISABLE_ENABLE enable)
621 {
622     if (FUNC_ENABLE == enable)
623     {
624         SCU->CLKOCR = (SCU->CLKOCR & (~(0x1FFFFU << 5) ) ) | (div << 5);
625         SCU->CLKOCR |= BIT23;
626     }
627     else
628     {
629         SCU->CLKOCR &= (~BIT23);
630     }
631 }
632
```

在系统初始化时，默认将 CLKOUT 关闭，Buzzer 分频默认为 80，若需使能只需修改形参即可。如要输出原始的 HCLK，则将调用修改为：

System_Set_Buzzer_Divider(79, FUNC_ENABLE); //输出即为 HCLK/(79+1)的值

图 4–3 时钟输出函数调用

System_ACM32F0x0.c

```
76 /*******************************************************************************
77  * Function    : System_Init
78  * Description : Initialize the system clock
79  * Input       : none
80  * Outpu       : none
81  * Author      : Chris_Kyle                          Data : 2020年
82  *******************************************************************************/
83 void System_Init(void)
84 {
85     SCU->RCR |= SCU_RCR_REMAP_EN;
86     System_Set_Buzzer_Divider(80, FUNC_DISABLE); // disable clock out
87     /* Configure the Vector Table location add offset address ------------------*/
88 #ifdef VECT_TAB_SRAM
89     /* Vector Table Relocation in Internal SRAM */
90     SCU->VECTOROFFSET = SRAM_BASE | VECT_TAB_OFFSET | SCU_VECTOROFFSET_VOFFSETEN;
91 #else
92     /* Vector Table Relocation in Internal FLASH */
93     SCU->VECTOROFFSET = EFLASH_BASE | VECT_TAB_OFFSET | SCU_VECTOROFFSET_VOFFSETEN;
94 #endif
```

5. 版本历史

版本	日期	作者	描述
V1.0	2021-04-30	Hangxin	初始版
V1.1	2023-02-10	Hangxin	添加 A070 系列芯片支持
V1.2	2025-03-10	Hangxin	添加 WB15 系列芯片支持

6. 版权声明

联系我们

公司：上海航芯电子科技股份有限公司

地址：上海市闵行区合川路 2570 号科技绿洲三期 2 号楼 702 室

邮编：200241

电话：+86-21-6125 9080

传真：+86-21-6125 9080-830

Email： service@HangChip.com

Website： www.hangChip.com